Energy-Aware Service Composition Algorithms for Service-Oriented Heterogeneous Wireless Sensor Networks

Tao Wang, Lianglun Cheng, Ke Zhang, and Jun Liu

1 School of Automation, Guangdong University of Technology, Guangzhou 510006, China
2 School of Computer, Guangdong University of Technology, Guangzhou 510006, China

Correspondence should be addressed to Tao Wang; wangtaosea@gmail.com

Received 1 November 2013; Revised 13 February 2014; Accepted 17 February 2014; Published 30 March 2014

Wireless sensor networks are evolving with increasingly device heterogeneity, while the problem of effectively composing the platform-specific functionalities provided by heterogeneous sensor nodes to achieve specific goals still remains a challenge. Because of the constrained resources and unreliable communication in sensor networks, traditional service composition techniques in web services with adequate resources are insufficient. In this paper, focusing on the limited energy of sensor nodes, we propose an energy-aware service composition framework for developing various applications in heterogeneous wireless sensor networks. With both the energy-aware metrics: energy-aware load-balancing factor and overall energy consumption, we formulate the process of energy-aware sensor service composition into a combinatorial optimization problem; furthermore, an improved discrete particle swarm optimization (IDPSO) algorithm with inertia weights adjustment and extreme perturbation scheme is proposed to solve the combinatorial optimization problem. The experiment results have shown that the performance of the service route given by IDPSO is approximately equal to the best service route searched out by the exhaustive algorithm. Meanwhile, our proposed energy-aware service composition method is able to reduce the energy consumption and prolong the lifetime of the sensor network when providing stable service composition for various applications.

1. Introduction

In recent years, wireless sensor networks (WSNs) are evolving with new features such as larger scale, higher device heterogeneity and the capability of supporting multiple applications [1]. For example, a WSN deployed for intelligent buildings contains a variety of different sensors, for example, video, audio, temperature, humidity, smoke, and light sensors, and it should be able to support video surveillance, accident location and tracking, environmental monitoring, and other applications. For example, as is shown in Figure 1, the target identification and tracking application is provided by combining acoustic sensing, target localization, target identification, and camera-based target tracking functionalities. In such environments, the device heterogeneity presents a great challenge for application programmers because it requires them to master the details of the underlying software and hardware platform [2, 3]. In order to develop diverse applications rapidly and efficiently, it is really necessary to provide programmers with adequate abstractions of the low-level platform-specific software and hardware details.

Existing service-oriented architecture (SOA) [4–6] has been considered as a promising means to provide higher level abstraction for the development of applications, and it helps enable applications to be platform-independent and adapt to network dynamics. Among the SOA technologies, service composition, that is, the process through which kinds of complex applications are developed by composing a set of primitive services with various restrictions (e.g., QoS [7], fault-tolerant [8]), has been well studied in the context of web services.

Service composition in wireless sensor networks is faced with more difficulties and has gradually gained attention. The existing service models and composition techniques for web services have limitations when directly applied to wireless sensor networks. The main reason is that the software-based web services are often supported with adequate resources. However, the service capabilities of the heterogeneous sensor
nodes are affected by their intrinsic properties, such as the limited and dynamic computational ability, power supply, and residual memory and communication bandwidth. Meanwhile, too many service invocations in the same node will easily cause excessive computation and storage usage, energy consumption, and network traffic. More seriously, it will exhaust the battery of the involved service nodes and then create the energy holes that split the network into isolated islands. Therefore, it is critical to bind service requests to sensor nodes according to their available resources and current service load.

Considering that the network lifetime and energy spent at the node and system level are much important in WSNs, the objective of the work presented herein is to propose an energy-aware service composition framework for sensor services in the service-oriented heterogeneous wireless sensor networks.

With this service-oriented approach, nodes’ sensing, data processing and transmission, and actuation capabilities are encapsulated as a collection of services. Conceptually, a service provided by a sensor node can be understood as a computational component that inherits the specific contexts of the related physical node. Meanwhile, a service can be provided by multiple different sensor nodes, although each service implementation may have different spatial, temporal, and QoS characteristics.

Then, based on resolving the application request into a workflow graph, we are able to develop new applications rapidly by combining a suitable set of available service nodes with service composition. Here, the process of choosing the appropriate set of service nodes is called service routing. Within the process of service routing, not only the service functionalities provided by the candidate node is concerned, but also its available energy is checked. We have exploited two different energy-aware metrics: energy-aware load-balancing factor and overall energy consumption. Based on both the metrics, we formulate the process of energy-aware sensor service routing into a combinatorial optimization problem. Considering that the traditional exhaustive algorithm suffers from the high computational complexity, we then propose an improved discrete particle swarm optimization algorithm to compute the optimal service route. In summary, we make the following contributions.

1. In Section 3, we provide a framework to model the problem of energy-aware sensor service composition for developing sensor network applications.
2. In order to optimize for the energy balance and total energy minimization goals, we formulate the energy-aware sensor service composition process as a combinatorial optimization problem.
3. To deal with the high computational complexity of exhaustive algorithm, we propose an improved discrete particle swarm optimization algorithm to find out the approximately optimal service route effectively.

With our proposed methods, the workload of the service nodes is well balanced and the total energy cost is minimized as much as possible. Therefore, our proposal is able to reduce the energy consumption and prolong the lifetime of the sensor network when providing stable service composition for various applications.

2. Related Works

In recent years, service composition for pervasive computing such as distributed real-time system, MANETs, CPSs, and WSNs has been received more and more attention. In this section, we will discuss the related works on modeling and composition of services in these systems.

2.1. Related Works of Service Composition for Pervasive Computing

Excepting for web services, service composition for pervasive computing has also been the subject of intense study over the past several years. Brønsted et al. [9] discussed that service composition mechanisms for pervasive computing should address the following four goals including context awareness, managing contingencies, leveraging heterogeneous devices, and empowering users. Their analysis suggests that there are indeed opportunities and reasons to focus on service composition for pervasive computing.
The work of Kalasapur et al. [10] presented a dynamic service composition mechanism in pervasive computing. They modeled and represented resources as services, maintained a repository of service graphs, and dynamically combined multiple basic services into complex services. However, the scheme of context-aware (e.g., residual energy and location) service selection for dependable service composition was not deeply studied in this paper.

The work of Estévez-Ayres et al. [11] presented a model for QoS-aware service composition in distributed systems with real-time and fault-tolerance requirements. However, they selected out the appropriate service nodes mainly based on their QoS properties, but it is not enough for dependable service provision in WSNs. The work developed by Wang [12] studied a dependable service composition problem in MANET. They proposed to achieve dependable service composition by taking the mobility prediction of the service providers into consideration. The work of Jiang et al. [13] proposed a new service composition and recovery framework designed to achieve minimum service disruptions for mobile ad hoc networks. However, the common WSNs do not have the mobility characteristics; moreover, energy issues should be more important for service composition because of the fewer available resources in sensor nodes.

In [14], the authors proposed OSGi-based service architecture for Cyber-Physical Home Control Systems, which supports service-oriented control methods. Users can control appliances in the physical environment by intuitive operation through a virtual home on the network. In [15], Huang et al. developed an ontology model for physical entity specification. Based on this model, OWL-S is extended to accurately model the characteristics of service providers in the context of cyber-physical worlds. These proposed SOAs in CPS are helpful for our studies in WSNs; however, the WS standards such as OWL and XML exploited in CPS are too much complex for the low-capacity sensor nodes.

2.2. Related Works of Service Composition for WSNs. In recent years, some researchers have begun to focus on service composition for service-oriented wireless sensor networks. In order to handle the device heterogeneity, some kinds of service-oriented architectures and middleware have been proposed for a flexible service provision. In [5], the authors presented a service-centric WSN model, which focused on services provided by a WSN and viewed a WSN as a service provider. In [6], the authors presented a service-oriented, flexible, and adaptable middleware for QoS configuration and management of wireless sensor networks. Both proposed service-oriented models are meaningful for our studies; however, no details on automated service composition are studied in them.

Amundson et al. [16] have developed OASIS, which is an object-centric, service-oriented programming model and middleware for ambient-aware sensor network applications; however, it did not include automated composition with cost measures. In [17], the authors introduced the WSN-SOA suite which enables SOA-based communications in networks of low-capacity sensors. Service-oriented middleware TinySOA [18] allows programmers to access wireless sensor networks from their applications by using a simple service-oriented API via the language of their choice. However, both the studies did not address automatic composition construction. In [19], the authors presented a Distributed Operating System (DOS) based on the service-oriented architecture (SOA) to manage all embedded devices in a home network at high level of interoperability. Another service provision middleware, Servilla[20], enables applications to be both platform independent and energy efficient by facilitating in-network collaboration among heterogeneous devices. With Servilla, the available energy in the candidate nodes has been taken into account in the service binding process; however, the common service composition model and service selection algorithm had not been presented.

Few approaches have been proposed for modeling service composition problem and the related resolution algorithms. In [21], the authors provided service composition solutions during a persistent query’s lifetime such that the involved routing update cost and transmission cost are minimized. Another approach [22] proposed a probabilistic context-free grammar (PCFG) based modeling technique to construct service compositions. In [23], the authors proposed a policy-aware sensor service composition framework, which can automate the process of combining simple sensor services into larger ones with pre-specified policies. To the best of our knowledge, energy-aware automatic service composition in our study is novel in the sensor network domain.

3. Mathematical Formulations for Energy-Aware Service Composition

3.1. Network and Application Model

3.1.1. Service Model Description in Sensor Networks. Currently, there are many modeling languages, protocols, and frameworks in web services, for example, OWL-S [24], WSDL [25], BPEL [26], and SOAP [27]. However, these technologies are heavyweight for the limited resources of sensor nodes. Meanwhile, they view services as explicit entities and the resources that provide services as implicit entities.

In contrast with traditional resource-implicit web service description model, our proposed model is resource explicit, and the state of node services is affected by the dynamic properties of the sensor nodes. In addition, each heterogeneous node $N_i$ has its own contexts, such as its location, residual battery, available storage, and computation ability and bandwidth. Note that these contexts will affect the service capability of the node, and we only concentrate on the residual battery in this paper. Then, a sensor service $s_i$ in sensor networks can be described as follows:

$$s_i = \langle \text{Name}_i, \text{Input}_i, \text{Output}_i, \text{Op}_i, ER_i \rangle .$$  (1)

The service $s_i$ is defined by the name, input data that it accepts, the operation function that it applies to its input, the output data that it produces, and the residual energy of the sensor node. The service functionality of $s_i$ is defined
3.1.2. Network Model Description. Assume that \( n \) heterogeneous nodes \( N = \{N_1, N_2, \ldots, N_n\} \) are deployed in a specific area, and the topology of the whole sensor network can be described as \( TG = \{N, L\} \); here, \( L \subseteq N \times N \) represents the connection set. The set of service classes provided by the whole sensor network is denoted by \( S = \{S_1, S_2, \ldots, S_m\} \), and the syntactic parameters of a service class define service name, input, output, the operation function, and some additional information of service functionality. Assume that the set of service classes provided by a node \( N_i \) is denoted by \( NSL_i \subseteq S \). Therefore, we can use a \( n \times m \) matrix \( R \) to describe the service capability of each node, and \( R_{ij} = 1 \) denotes that the node \( N_i \) is able to provide the service class \( S_j \in S \). Then, the service classes provided by \( N_i \) can be denoted by \( NSL_i = R[i, :] * S^t \), and \( R[i, :] \) is the corresponding row vector of \( N_i \). Meanwhile, for a service class \( S_i \), the set of available service nodes can be denoted by \( SNL_i = N * R[:, i] \), and \( R[:, i] \) is the corresponding column vector of \( S_i \). Then, a heterogeneous sensor network can be described as \( NG = \{N, L, R\} \).

3.1.3. Application Model Description. In a SOA environment, an assigned application \( A_a \) can be decomposed into a workflow graph, which contains a set of services with the control and data flow among them. Note that the service in the workflow graph refers to a service class rather than a specified service instance. Then we are able to describe the application as a workflow graph \( AG_a = \{S_a, O_a\} \); here, \( S_a = \{S_1^a, \ldots, S^t_a\} \subseteq S \) represents the set of the requested service classes and the dependency relation \( O_a \subseteq S_a \times S_a \) describes the dependencies between them. Make sure that the requested service classes should be covered by the whole sensor network.

3.1.4. Description of Service Composition Problem. The service composition process can be achieved with a centralized or distributed method. In this paper, for simplicity, we use a centralized fashion. A sink node or a gateway can act as a central decision node receiving information from all service nodes and dispatch the service requests to the selected sensor nodes. The centralized fashion should be only suitable for service composition in small-scale sensor networks because of the dramatically increasing communication cost along with the scale of the wireless sensor network.

As is shown in Figure 2, we can implement a service composition for a given application \( AG_a = \{S_a, O_a\} \) by mapping the requested service classes in its workflow graph into the sensor nodes. An application can be constructed as a set of autonomous services that perform independent operations on the data stream passing through them. A service link between two service nodes is defined by \( O_a \), and it is an overlay link that may consist of several wireless links in the network. The communication path for a service link is built with the shortest path first routing method. Note that the arrow represents the direction of data flow, and \( f_i \) is the invocation times of each combined service, and \( w_{ij} \) represents the amount of data transferred between both the services.

Note that if the service capability of a node \( N_i \) can satisfy the requirements of a requested service class \( S^i \), it will meet the following conditions:

\[ NSL_i \supseteq S^i. \quad (2) \]

This formula denotes that certain service class provided by \( N_i \) should meet the functionality requirements specified by \( S^i \).

Now based on the wireless heterogeneous sensor network \( NG = \{N, L, R\} \) and the received application request \( A_a \) with the workflow graph \( AG_a = \{S_a, O_a\} \), the key step of service composition for the application \( A_a \) is to select the appropriate service nodes to serve the service classes specified by the graph \( AG_a \), and the entire selection process is called service routing. The goal of our study is to find out an approximately optimal service route \( G_{opt} \) such that (1) the total energy consumption should be minimized as much as possible and (2) the workload of the service nodes should be well balanced to avoid excessive energy consumption of certain service nodes. We then summarize the symbols used in our model in Table I.

3.2. Energy Cost. The set of the candidate service routes can be denoted by \( \overline{G}_{an} = \{\overline{G}^1_{an}, \overline{G}^2_{an}, \ldots\} \), and for each candidate
3.2.1. Calculate the Overall Communication and Computation Cost of a Candidate Service Route. The overall communication cost contains not only the communication cost of each involved service node, but also the communication cost of the forwarding nodes within the communication paths for the service links. Note that the communication cost in this paper refers to the energy used to transmit and receive the data specified by the requested application. For two requested adjacent service classes \( S_j^a, S_k^a \) which are connected by a service link in the workflow graph, two nodes \( N_p, N_q \in \tilde{G}_a \) are assumed to provide these two service classes, respectively. For simplicity, we exploit the shortest path network routing scheme when transferring data between two service nodes and then the communication cost from the service node \( S_j^a(N_p) \) to \( S_k^a(N_q) \) can be calculated as follows:

\[
E_{tk}^j = f_j \times w_{jk} \times (E_{tx} + E_{rx}) \times \text{hops}(P_{tk}^j)
\]

here: \( ER(N_i) > f_j \times w_{jk} \times (E_{tx} + E_{rx}) \),

for each \( N_i \in P_{tk}^j \).

Here, \( P_{tk}^j \) denotes the shortest path between the service nodes \( N_p \) and \( N_q \), and the residual energy of each node in the path should be greater than the necessary communication cost. \( E_{tx} \) represents the energy spent by transmitting one unit of data, and \( E_{rx} \) represents the energy spent by receiving one unit of data. The communication cost is proportional to the number of hops between both the involved service nodes. Meanwhile, there is no communication energy consumption when both the requested services are implemented in the same node.

The overall communication cost of the candidate service route \( G_a^3 \) is given by

\[
E_{\text{comm}}(G_a^3) = \sum_{(S_j^a, S_k^a) \in \Omega_a} E_{tk}^j.
\]

The overall computation cost of the candidate service route \( G_a^3 \) is given by

\[
E_{\text{comp}}(G_a^3) = \sum_{S_j^a \in \Omega_a} f_i E_{\text{inv}}(S_j^a).
\]

Based on formula (5), we can conclude that the overall computation cost has no relation with the specific service routes.

3.2.2. Calculate the Communication and Computation Cost of Each Involved Service Node. For each service node involved in a candidate service route \( G_a^3 \), we first calculate its communication and computation cost. Note that a specific node may have the capabilities to serve multiple requested services, so we denote the set of requested services assigned to node \( N_p \in G_a^3 \) as follows:

\[
NS_p = \{ S_{\alpha}^a, \text{ where } \tilde{G}_a^3(S_{\alpha}^a) = N_p \}.
\]

Here, \( \tilde{G}_a^3(S_{\alpha}^a) = N_p \) represents that \( S_{\alpha}^a \) is provided by \( N_p \) in the candidate service route \( G_a^3 \). Then the computation cost for the service node is given by

\[
E_{\text{comp}}(N_p) = \sum_{S_j^a \in NS_p} f_i E_{\text{inv}}(S_j^a).
\]
Then we compute the communication cost of the service node and before this, we assume that the amount of input and output data of a requested service $S_a$ is given by

\[
\begin{align*}
\text{input}(S_a) &= \sum_{(s_a, s_a') \in O_a} f_{xa} \cdot w_{xa}, \\
\text{output}(S_a) &= \sum_{(s_a, s_a') \in O_a} f_{ia} \cdot w_{ya}.
\end{align*}
\] (8)

Assume that there are no adjacent relationships among the services provided by node $N_p$; then, the communication cost of $N_p$ can be calculated by

\[
E_{\text{comm}}(N_p) = \sum_{S_a \in NS_p} \left[ E_{rx} \cdot \text{input}(S_a) + E_{tx} \cdot \text{output}(S_a) \right].
\] (9)

However, in some practical applications, a node may provide multiple services which are adjacent. For example, a specific node may provide perception and data fusion services at the same time, and these two services are usually adjacent since the perception data is directly sent to the data fusion service. Here, the data exchange between these two services is implemented within a node and there is no communication cost. Therefore, formula (10) can be rewritten as follows:

\[
E_{\text{comm}}(N_p) = \sum_{S_a \in NS_p} f_i \cdot \left[ E_{rx} \cdot \text{input}(S_a) + E_{tx} \cdot \text{output}(S_a) \right] - \sum_{S_a, S_a' \in NS_p} f_{p} \cdot \sum_{(s_a, s_a') \in O_a} (E_{rx} + E_{tx}) f_{p} \cdot w_{pq} \cdot (E_{rx} + E_{tx}).
\] (10)

Here, $S_a \in NS_p$ denotes that $S_a$ is served by $N_p$ and $(S_a, S_a') \in O_a$ denotes that $S_a$ and $S_a'$ are adjacent in the workflow graph.

### 3.3. Service Routing with Two Energy-Aware Metrics

As we mentioned above, the optimal service route should balance the workload on the service overlay network. Assume that $N_i$ is the service node for $S_a$, without exhausting its energy; the residual energy in $N_i$ should be more than the energy required for the service $S_a$. Meanwhile, if too many requested services are concentrated on a specific node, it will tend to consume the node's energy excessively and increase the response time of applications. To avoid these from happening, the service routing process should choose the appropriate service nodes by considering their residual energy and current service loads.

In this paper, we implement the service composition with two proposed energy-aware metrics. The first metric is energy-aware load-balancing factor, which we consider to be achieved when the maximal fraction of energy spent by any service node is minimized. With this goal, we define $\text{LBF}(G_a)$ as the energy-aware load-balancing factor of the candidate service route $G_a$, as follows:

\[
\text{LBF}(G_a) = \max_{N_p \in O(G_a)} \frac{\text{LBF}(N_p)}{N_p},
\] (11)

where $\text{LBF}(N_p)$ is the energy-aware load-balancing factor of node $N_p$. $\text{LBF}(N_p)$ denotes the residual energy of the node $N_p$. The second energy-aware metric is the total energy consumption for implementing the application. Although we believe that energy balance is a better metric to measure the service routes, it is also necessary to use the goal of minimizing the total energy consumption to enhance our energy-aware service composition method. The total energy spent for a candidate service route is given by

\[
E(G_a) = \sum (E_{\text{comm}}(G_a) + E_{\text{comp}}(G_a)).
\] (12)

Our service routing algorithm then selects the smallest value of a weighted sum of total energy consumption $E(G_a)$ and energy-aware load-balancing factor $\text{LBF}(G_a)$ of the candidate $G_a$. Note that the values of both the metrics are normalized in advance. Hence, the goal of service routing can be expressed as follows:

\[
G_a^\text{opt} = \min_{G_a} \left\{ \mu \cdot E(G_a) + \lambda \cdot \text{LBF}(G_a) \right\}.
\] (13)

### 4. Exhaustive Algorithms for Service Routing

To formulate the problem, the candidate service route $G_a$ is denoted by a $1 \times 1$ dimensional matrix and $G_a[j]$ denotes the selected service node providing the requested service $S_a$. Then, the process of searching out the optimal service route can be formulated into a combinatorial optimization problem, which is expressed as follows.

**Inputs:**

(i) $AG_a = \{S_a, O_a\}$: workflow graph of a required application;

(ii) $NG = \{N, L, R\}$: network description;

(iii) $\text{ER}(N_i)$: residual energy in node $N_i$;

(iv) $w_{ij}$: amount of data transferred from $S_a$ to $S_a'$ each time $S_a'$ is invoked;

(v) $f_i$: invocation times of each required service $S_a \in S_a$;

(vi) $E_{\text{inv}}(S_a)$: energy spent per invocation to the service $S_a$;

(vii) $E_{rx}$, $E_{tx}$: energy spent by transmitting/receiving one unit of data.

**Outputs:**

(i) $G_a^\text{opt}$: the optimal service route.
Evaluation Factor:

\[
\text{Minimize } Z = \left\{ \mu \cdot E \left( \hat{C}_a \right) + \lambda \cdot \text{LBF} \left( \hat{C}_a \right) \right\}. \tag{14}
\]

Constraint:

\[
E_{\text{comp}} \left( N_p \right) + E_{\text{comm}} \left( N_p \right) \leq \text{ER} \left( N_p \right), \text{ where } N_p \in \hat{C}_a. \tag{15}
\]

As is shown in Algorithm 1, according to the evaluation factor and constraint, we then propose an exhaustive algorithm to explore the optimal service route for a given application graph. A service route is constructed by selecting an available service node for each requested service class of the application graph. The exhaustive algorithm traverses all combinations with a depth first search algorithm (line (20)–(23) of Algorithm 2). Each valid service route found by the algorithm is scored by calculating its evaluation factor (line (9)–(11) of Algorithm 1). During the execution, the algorithm stores each valid service route and its value of evaluation factor and replaces them every time it finds a service route with a smaller value.

Algorithm 2 shows the whole process of calculating the evaluation factor. It mainly includes two steps: first, it calculates the expected communication and computation energy cost of every involved service node (line (5)–(27) of Algorithm 2) and it should make sure that the candidate service route is valid by checking if the available energy of every involved service node is enough for providing the requested services (line (29)–(31) of Algorithm 2). Meanwhile, for the valid service node, the algorithm stores its value of load-balancing factor and replaces it every time a service node with larger value of load-balancing factor is found (line (34)–(39) of Algorithm 2). Second, it calculates its evaluation factor by a weighted sum of its load-balancing factor and overall energy cost (line (43)–(46) of Algorithm 2).

Algorithm 3 shows the process of calculating the overall energy cost of each candidate service route, and it is called by Algorithm 2 (line (41) of Algorithm 2). This algorithm mainly contains two steps of calculating computation energy cost and communication energy cost of the whole service route. The latter is more complex, and it needs to obtain the hops of the shortest path between both the service nodes (line (10)–(13) of Algorithm 3), which depends on the specific topology of the sensor network.

4.1. Computational Complexity. The proposed exhaustive algorithm suffers from the relatively high computational complexity, because it evaluates all the candidate service routes. Given the \( n \times m \) matrix \( R \) described in Section 3, the number of associated service instances for each service \( S_j \) can be calculated as \( N(S_j) = \sum \{ R(i, j) \} \), and the number of requested services of the application \( A_a \) is \( m \). The algorithm has to evaluate \( N(A_a) = \prod_{j=1}^{m} N(S_j) \) paths, and the computational complexity increases exponentially with the number of service nodes.

5. Improved Discrete Particle Swarm Optimization for Service Routing

Considering the high computational complexity of the exhaustive algorithm, we propose to adopt the evolutionary computing techniques which are more suitable to solve the combinatorial optimization problem. PSO [28] is an adaptive evolutionary computing technique proposed by Kennedy and Eberhart in 1995, which maintains and exploits both the position and velocity information in the evolution process, while other evolutionary algorithms only use the position information. In other words, PSO performs a kind of conscious mutation, which improves the convergence speed greatly. Meanwhile, existing studies show [29] that PSO has the merits of high speed, high quality solution, and good robustness in terms of multimodal optimization in multidimensional space and dynamic optimization. PSO has been widely applied in the complex optimization areas including multiobjective optimization, and pattern recognition. Therefore, we propose to use the PSO algorithm to solve the combinatorial optimization problem in this paper.

5.1. Basic Particle Swarm Optimization. PSO is a population-based optimization tool and inspired by the social behavior of natural individuals. It searches the optimal solution for a difficult optimization problem through the collaboration between the individuals. Assume that the size of swarm is \( P \), the coordinate position of each particle \( p_i \) (\( i = 1 \sim P \)) at time \( t \) is denoted by \( x_i(t) = (x_{i1}(t), x_{i2}(t), \ldots, x_{iD}(t)) \), and the particle velocity is defined as the moving distance of each search iteration, which is represented by \( v_i(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{iD}(t)) \). At each search iteration, the current position and velocity of each particle are updated by two best values: (1) \( p_{\text{best}} \), individual best position that has achieved so far, and (2) \( g_{\text{best}} \), another best value representing the global best position among all particles in the swarm. Then the particle \( p_i \) updates its \( d \)th dimensional velocity and position according to the following formulas:

\[
v_{id}(t + 1) = w v_{id}(t) + c_1 r_1 \left( p_{\text{best}d}(t) - x_{id}(t) \right) + c_2 r_2 \left( g_{\text{best}d}(t) - x_{id}(t) \right)
\]

\[
x_{id}(t + 1) = x_{id}(t) + v_{id}(t + 1)
\]

Here, \( w \) is the inertia weight factor which represents the impact of previous values of particle velocities on its current ones. A larger inertia weight factor is suitable for global exploration and a smaller inertia factor is suitable for fine tuning in the current search area. Parameters \( c_1 \) and \( c_2 \) are positive constants called acceleration coefficients; in general, \( c_1 = c_2 = 2 \). Parameters \( r_1 \) and \( r_2 \) are two random variables with range \([0, 1] \).

5.2. Improved Discrete Particle Swarm Optimization for Energy-Aware Service Composition. In this section, we present a formalized description for energy-aware service composition with an improved discrete particle swarm optimization (IDPSO) algorithm.
5.2.1. Particle Representation. As we defined in Section 3, a requested application $A_d$ can be decomposed into a workflow containing a set of requested service classes $S_a = \{s_1^d, \ldots, s_l^d\} \subseteq S$. The candidate service route $G_a^d[i]$ contains the set of involved service nodes. Each candidate service route can be viewed as a certain position of a particle $p_i$ in the $l$-dimensional space. Therefore, the particle position will change if any service node in the candidate service route is substituted. Then, the $i$th particle can be represented as $p_i = \{p_{i1}, p_{i2}, \ldots, p_{id}, \ldots, p_{il}\}$, where $p_{id}$ indicates the index of the allocated service node for the $d$th requested service $s_d^a$ in the application graph. The set of service node for $S_a^d$ is $SNL_d = \{SN_1^d, \ldots, SN_j^d, \ldots, SN_{M_d}^d\}$, $SN_j$ represents a node $N_j$ with the capability of providing the requested service class $s_d^a$, and $M_d$ is the number of candidate service nodes. Each dimension of the particle has a discrete set of possible limit values $\{p_{id} : 1 \leq p_{id} \leq M_d\}$. 

5.2.2. Fitness Function. The fitness function $\text{fitness}(p_i)$ measures to what extent the particle $p_i$ satisfies the objective and constraints (which are defined by formula (14) and (15)) of the energy-aware service composition problem. According to the fitness function, the swarm evolution is navigated towards the optimal solution by the best particles. The fitness function $\text{fitness}(p_i)$ at the $t$th iteration is defined as follows:

$$\text{fitness}(p_i(t)) = Z(p_i(t)) .$$

Constraint:

$$E_{\text{comp}}(p_{id}(t)) + E_{\text{comm}}(p_{id}(t)) \leq ER(p_{id}(t)), \quad p_{id}(t) \in p_i(t).$$

Here, $p_i(t)$ contains a set of service nodes providing services for the requested application and $p_{id}(t)$ represents a service node. The objective function $Z(p_i(t))$ measures the merit of the solution $p_i(t)$ under the assumption that every service node in $p_i(t)$ satisfies the constraint. However, if any service node in $p_i(t)$ violates the constraint, the solution is infeasible. Meanwhile, the value of $\text{fitness}(p_i)$ is invalid and goes to infinity.

5.2.3. Improved Discrete Particle Swarm Optimization Algorithm. As the structure of PSO algorithm is relatively simple, it is running fast. However, PSO is easy to fall into premature

---

**Algorithm 1: Exhaustive composition algorithm.**

Given: $AG_a = [S_a[i], O_a[i][i]], f[i], w[i][i], S[m]$, $NG = \{N[n], L[n][n], R[n][m]\}$, $ER(n)$, $E_{\text{in}}(l)$, $E_{\text{ex}}$, $E_{\text{rx}}$
Output: $G_a^d[i]$

1. /* the overall service selection optimization process */
2. [success, $G_a^d$, $Z_{opt}$] = Combine($i$, $j$, $G_a^d$, $AG_a$, $NG$, $Z$)
3. 
4. /* a node $j$ is assigned to the requested service class $S_a[i]$ in $AG_a$ */
5. if $i \neq 0$ then
6. $G_a^d(i) = j$
7. end if
8. /* if the last service node is chosen */
9. if $i = l$ then
10. $G_a^d = G_a^d$
11. [success, $Z_{opt}$] = calcSchedZ($G_a^d$, $AG_a$, $NG$, $Z$)
12. else
13. if $i = 0$ then
14. /* initialize the parameters */
15. $G_a^d[i] = []$; $Z_{opt} := \infty$; success := FALSE
16. end if
17. /* obtain the set of service nodes for each requested service class $S_a[i + 1]$ in $AG_a$ */
18. $idx = \text{GetIndex}(S_a[i + 1], S[m])$
19. $SNL_{i+1} = N[n] * R[.i]idx$
20. /* for each available service node for the next level */
21. foreach $SNL_{i+1}(k)$ in $SNL_{i+1}$ do
22. if $SNL_{i+1}(k) \neq 0$ then
23. $[e, \text{comb}, z] = \text{Combine}(i + 1, SNL_{i+1}(k), G_a^d, AG_a, NG, Z)$
24. /* if success and the value of evaluation factor is smaller */
25. if $e$ and $z < Z_{opt}$ then
26. $G_a^d = \text{comb}; Z_{opt} = z$
27. end if
28. end if
29. end foreach
30. end if
31. return(success, $G_a^d$, $Z_{opt}$)
32. }
success, Z_{opt} = calcSchedZ(G_{opt}^a, AG_{a}, NG, Z)
(2) {
(3) /* firstly initialize the parameters*/
(4) E_{comp} [n] := [] ; E_{comm} [n] := [] ; LBF[n] := [] ; max LBF := 0
(5) /* obtain the index of requested services provided by each involved node
(6) foreach N_{p} in G_{opt}^a do
(7) Sl_{p} = getServiceIndex(G_{opt}^a, N_{p})
(8) /* firstly, compute the computation cost*/
(9) foreach j in Sl_{p} do
(10) E_{comp}(p) = E_{comp}(p) + f[j] * E_{inv}(j)
(11) endforeach
(12) /
(13) /* compute the amount of input and output data for each requested service in AG_{a} */
(14) foreach S_{a}[i] in AG_{a} do
(15) foreach k <= l and k != i do
(16) Input[i] = Input[i] + w[k][i] * f[k]
(17) Output[i] = Output[i] + w[i][k] * f[i]
(18) endforeach
(19) endforeach
(20) /
(21) /* compute the communication cost*/
(22) foreach j in Sl_{p} do
(23) E_{comm}(p) = E_{comm}(p) + E_{rx} * Input(j) + E_{tx} * Output(j)
(24) endforeach
(25) /
(26) /* check if there exist adjacent services*/
(27) foreach k in Sl_{p} and k != j do
(28) if S_{a}[j] -> S_{a}[k] in O_{a} then
(29) E_{comm}(p) = E_{comm}(p) - (E_{tx} + E_{rx}) * f[j] * w[j][k]
(30) end if
(31) endforeach
(32) /
(33) /* check if the residual energy of N_{p} is enough*/
(34) if E_{comp}(p) + E_{comm}(p) > ER(p) then
(35) success = False
(36) return (False, ∞)
(37) Else
(38) /
(39) /* compute the load-balancing factor*/
(40) success = True
(41) LBF[p] = [E_{comp}(p) + E_{comm}(p)] / ER(p)
(42) ER(p) = ER(p) - [E_{comp}(p) + E_{comm}(p)]
(43) if max LBF < LBF[p] then
(44) max LBF = LBF[p]
(45) end if
(46) end if
(47) end foreach
(48) /
(49) if success then
(50) eg = calcOverallCost(G_{opt}^a, AG_{a}, NG)
(51) /* calculate the evaluation factor*/
(52) z = µ * eg + λ * max LBF
(53) return (TRUE, z)
(54) end if
(55) }

Algorithm 2: Evaluation factor calculation algorithm.

convergence and encounter local optimization. To overcome this problem, we propose an improved discrete PSO algorithm in this paper.

Adjustment of Inertia Weights. In this paper, we exploit the fuzzy logic approach to dynamically adjust the value of inertia weight w in the different convergence stages [30]. Then, the PSO algorithm is able to adjust the global parameter adaptively taking into account both the search efficiency and precision. In the fuzzy logic approach, the variables selected as inputs to the fuzzy system are the current best performance evaluation (CBPE) and current inertia weight, whereas the
process and accelerate the convergence speed with a low exploration ability with a high.

Here, improved based on this adjustment strategy of the inertia weights. Therefore, in addition to the swarm reoptimization with extreme perturbation scheme.

When PSO algorithm falls into evolutionary stagnation, the phenomenon of particle accumulation happens, and the particles will scatter until the stagnation is broken. This phenomenon is theoretically explained well in [31]. The definition of particle convergence is shown in formula (21), and $p(t)$ represents a particle’s position at the $t$th iteration. This definition describes that the particles will converge to a stable position $p^*$ finally, which is determined by the weighted sum of individual best position $p_{best}$ and the global best position $g_{best}$. If a position better than $g_{best}$ is not found when all the particles are converging to the position $p^*$, then the evolution will fall into stagnation and the particles gradually gather to $p^*$, 

$$\lim_{t \to \infty} p(t) = p^* = \frac{c_1 r_1}{c_1 r_1 + c_2 r_2} p_{best} + \frac{c_2 r_2}{c_1 r_1 + c_2 r_2} g_{best}. \quad (21)$$

An extreme perturbation scheme is employed to break the stagnant state. The basic idea of this scheme is that we will randomly adjust the individual/global best position when the evolution of both the best positions is stagnant for several iterations. And the extreme perturbation operator is defined as follows:

$$p_{best} = r_{1}^{t \xrightarrow{3} T_{0}} p_{best}; \quad g_{best} = r_{4}^{t \xrightarrow{4} T_{0}} g_{best}. \quad (22)$$

Here, $t_0$, $T_{0}$, $T_{g}$ represent the steps of evolutionary stagnation for individual and global best positions respectively, and $T_{0}$, $T_{g}$ represent the corresponding thresholds of triggering the extreme perturbation. And the random variables $r_3$ and $r_4$ are defined as follows:

$$r_{3} \xrightarrow{3} T_{0} = \begin{cases} 1, & t_0 \leq T_0, \\ U(0, 1), & t_0 > T_0; \end{cases} \quad (23)$$

$$r_{4} \xrightarrow{4} T_{g} = \begin{cases} 1, & t_g \leq T_g, \\ U(0, 1), & t_g > T_g. \end{cases}$$

Then, based on formula (16), the improved PSO algorithm with dynamic adjustment of inertia weight and extreme perturbation is given as follows:

$$v_{id} (t + 1) = w (t) v_{id} (t) + c_1 r_1 \left( r_{3}^{t \xrightarrow{3} T_{0}} p_{best_{id}} (t) - p_{id} (t) \right) \quad + c_2 r_2 \left( r_{4}^{t \xrightarrow{4} T_{g}} g_{best_{id}} (t) - p_{id} (t) \right),$$

$$p_{id} (t + 1) = p_{id} (t) + v_{id} (t + 1). \quad (24)$$

Here, the values of $T_{0}$ and $T_{g}$ determine the activation delay of extreme perturbation, generally in the range of [3, 8].

As we defined before, the particle position can be encoded as a $l$-dimensional integer array $p_i = (p_{i1}, p_{i2}, \ldots, p_{il})$, $1 \leq p_{id} \leq M_d$, $l$ is the number of requested service classes in the workflow graph, and $M_d$ is the number of candidate service nodes providing the requested service $s_{id}$. Meanwhile, the particle velocity can also be denoted by $l$-dimensional array $v_i = (v_{i1}, v_{i2}, \ldots, v_{il})$, $-M_d \leq v_{id} \leq M_d$. Each element in both the arrays is an integer and should be rounded when it is decimal. During IDPSO search process,
(1) Let
(2) $P$ is the size of the PSO swarm
(3) $p_i(t) = (p_{i1}(t), p_{i2}(t), \ldots, p_{il}(t))$, $v_i(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{il}(t))$ are the position and velocity of particle $p_i$ at the $t$th iteration.
(4) fitness($p_i$) is the fitness function of particle $p_i$
(5) $g_{best}(t)$ is the global best position so far during $t$ iterations
(6) $p_{best}(t)$ is the individual best position of particle $p_i$ so far during $t$ iterations
(7) $K$ is the total number of iterations
(8) Step 1. **(Initialization):** for each particle $p_i$ in the swarm
   1.1. initialize $P$, $K$, $w(1)$, $c_1$, $c_2$
   1.2. initialize $p_i(1)$, $v_i(1)$ randomly
   1.3. evaluate fitness($p_i$)
   1.4. initialize $p_{best}(1)$ with a copy of $p_i(1)$
   1.5. initialize $g_{best}(1)$ with the position of the particle with the best fitness among the swarm
   1.6. initialize CBPE with the current best fitness fitness($p_{best}(1)$)
   1.7. initialize CBPE$_{max}$ and CBPE$_{min}$
   1.8. calculate $\Delta w$ with the fuzzy logic approach
   1.9. initialize $T_0$ and $T_g$

Step 2. Repeat until a stopping criteria is satisfied
   for $t = 1 : K$
   2.1. for each particle $p_i$, update the velocity and position of each particle $p_i$ according to formula (24).
   \[ v_{id}(t + 1) = w(t) \cdot v_{id}(t) + c_1 \cdot r_1 \cdot (t_{1}(t) - p_{id}(t)) + c_2 \cdot r_2 \cdot (t_{2}(t) - g_{best}(t) - p_{id}(t)) \]
   2.2. reset $t_0$ or $t_g$ if the related extreme perturbation is executed:
   if $t_0 > T_0$, then $t_0 = 0$ end if
   if $t_g > T_g$, then $t_g = 0$ end if
   2.3. for each iteration, calculate $w(t + 1)$ according formula (20):
   \[ w(t + 1) = w(t) \cdot (1 + \Delta w) \]
   2.4. for each particle $p_i$, update its individual best position, the fitness calculation algorithm is given in Algorithm 2:
   if fitness($p_i(t + 1)$) $\leq$ fitness($p_{best}(t)$) then
   \[ g_{best}(t + 1) = p_i(t + 1) \]
   end if
   2.5. for each iteration, update the global best position:
   if fitness($p_i(t + 1)$) $\leq$ fitness($g_{best}(t + 1)$) then
   \[ g_{best}(t + 1) = p_i(t + 1) \]
   end if
   \end for
   2.6. for each particle $p_i$: calculate $t_0$ and $t_g$ for each particle $p_i$
   end for
   2.7. **Step 3. output**

**Algorithm 4: IDPSO for energy-aware service routing.**

5.3. **IDPSO Algorithm For Energy-Aware Service Routing.**

With the iteration formula, we have presented the detailed optimization process with IDPSO algorithm for energy-aware service routing, which is described as shown in Algorithm 4.

**Computational Complexity.** In the energy-aware service composition process with IDPSO algorithm, the size of swarm is $P$, the number of requested service classes in the workflow graph is $l$, and the number of iterations is $K$. The computational complexity of the main steps in Algorithm 4 is given as

\[
\text{if } p_{id}(t) > M_d, \text{ then } p_{id}(t) = M_d, \\
\text{else if } p_{id}(t) < 1, \text{ then } p_{id}(t) = 1, \\
\text{if } v_{id}(t) > M_d, \text{ then } v_{id}(t) = M_d, \\
\text{else if } v_{id}(t) < -M_d, \text{ then } v_{id}(t) = -M_d.
\]
follows: step 2.1: $O(P \times I)$; step 2.4: $O(P)$; step 2.5: $O(P)$. Then the total computational complexity of IDPSO is expressed as $O(K \times P \times I)$, which has no relation with the number of available service nodes for each requested service class.

6. Performance Evaluation

6.1. Simulation Environment. For evaluating the performance of our proposed energy-aware service composition algorithms, we apply them to the target identification and tracking application discussed in Section 1. The workflow graph of this application is shown in Figure 1. We build a sensor network in which there are $n$ ($n = 60$ in this simulation) heterogeneous nodes spreading across a specific area with the size $0.5 \text{ km} \times 0.5 \text{ km}$, and each node can be attached to an acoustic sensor, a temperature sensor, or a video sensor that can identify and track the target within its surveillance area. The underlying physical topology is generated by GT-ITM using $n$-node transit-stub models [32].

For simplicity, we assume that this network provides five service classes, including acoustic sensing, temperature sensing, target localization, identification, and tracking services. Each service class may have multiple service instances, which are randomly distributed over the network. The number of service instances is set to $0.6 \times n$, $0.6 \times n$, $0.4 \times n$, $0.2 \times n$, and $0.2 \times n$ in turn, and the value will be rounded if it is not an integer.

Furthermore, the fire rate $f_i$ of each service in the workflow graph is set as follows: for acoustic sensing service, $f_i$ is set to 5; for localization service, it would be awoken only after it has received all the acoustic sensing data. Consider that the sensing data arrives asynchronously because of the different network transmission paths and channel conditions. Therefore, the fire rate $f_i$ of localization service is 13. Then, the target’s location information will be sent to the following services. So the fire rate $f_i$ of the following services in the workflow graph is set to 13 too. The amount of data $w_{ij}$ transferred between the services is assumed to be of unit size ($w_{ij} = 1$).

A battery model simulates typical distributed embedded computing hardware, with a CPU and a radio module. For the parameters of our proposed IDPSO algorithm, the size of the swarm $P$ is set to 15, the total number of iterations $K$ is set to 150, and $c_1 = c_2 = 2$; $w(1) = 0.8$; $T_0 = 3$; $T_g = 5$. The weighting factors $\mu$, $\lambda$ in formula (14) are set to 0.5 if not specified differently.

6.2. Performance Evaluation for Our Proposed Algorithm. Firstly, we evaluate the capability of searching the optimal service route for our proposed methods. As shown in Figure 3, the IDPSO algorithm was evaluated by comparing it with the basic PSO algorithm and exhaustive composition algorithm. In this experiment, given the same initialization parameters, the requested application is executed continuously with 80 times for each algorithm. For each time, the optimal service route is refreshed by running each algorithm, respectively, and the fitness of the optimal solution of each algorithm is recorded.

It is shown that the exhaustive composition algorithm has gained the best performance with the lowest fitness, because it traverses all combinations of service nodes to find out the best one. The value of fitness increases stably because of the gradually increasing value of load-balancing factor. The basic PSO algorithm has the worst performance with the highest fitness, and the value of fitness fluctuates severely because the basic PSO algorithm easily falls into premature convergence and encounters local optimization. Our proposed IDPSO algorithm has gained better performance than the basic PSO algorithm. And it is able to search out a service route approximated with the best one, because the capability of global exploration of the IDPSO algorithm is enhanced with the extreme perturbation scheme.

Secondly, we measure the performance of our proposed methods on extending network lifetime, which is shown in Figure 4. Here, we also assume that all nodes started with an initial energy level and the requested application is executed continuously with 80 times for each algorithm. For the basic PSO algorithm, in Figure 4(a), we can notice that the variance of the residual energy of all the network nodes is the highest and increases most quickly; in Figure 4(b), the minimum residual energy is the lowest after the application is executed with 80 times. Therefore, some network nodes will run out of battery quickly with the basic PSO algorithm, and the network lifetime will be shortened because of the rapid emergence of energy hole. For our proposed IDPSO algorithm, it has gained better performance than the basic PSO algorithm on prolonging the network lifetime (lower variance of the residual energy and higher minimum residual energy), since it has searched out those service routes with better performance on load balancing. We can also notice that the performance of IDPSO approximates to the exhaustive composition algorithm.

With the proposed IDPSO algorithm, the performance of service composition is evaluated with using different energy-aware metrics, which is shown in Figure 5. Given the requested application above, we demonstrate the network...
Figure 4: The residual energy changes along with the execution times of the requested application: (a) the curve of the variance of the residual energy; (b) the curve of the minimum residual energy.

Figure 5: Network energy distribution after executing the requested application for 80 times with different energy-aware metrics: (a) $\mu = 1$, $\lambda = 0$; (b) $\mu = 0.5$, $\lambda = 0.5$; (c) $\mu = 0$, $\lambda = 1$. 
The number of total network transmission hops

![Graph showing the number of total network transmission hops](image)

**Figure 6:** The total network transmission hops of the optimal service route change along with the execution times of the requested application.

energy distribution after executing the service composition for 80 times under three circumstances: (a) only use the metric “overall energy consumption” with $\mu = 1$, $\lambda = 0$ in formula (14); (b) use both the metrics with $\mu = 0.5$, $\lambda = 0.5$; and (c) only use the metric “energy-aware load-balancing factor” with $\mu = 0$, $\lambda = 1$. From Figure 5(a), it is noticed that the energy of the sensor nodes in the central area with oxford blue has been consumed excessively. This is because the sensor nodes in the central area are exploited repeatedly in the service composition process when only using the overall energy consumption metric. In Figure 5(c), we can notice that the energy of many sensor nodes in a larger area has been exhausted; in other words, the sensor network is fallen into “malignant load-balancing.” When only the energy-aware load-balancing factor is concerned, the network routing paths will be too much long for many service routes. Further, the energy of many sensor nodes is exhausted just by forwarding data between the service nodes. In Figure 5(b), a better performance on network energy distribution has been gained when using both the metrics, and only few sensor nodes have nearly run out of battery. Therefore, in order to improve the performance of service composition, it is very important to achieve the balance between both the energy-aware metrics.

Further, to verify the experiment results in Figure 5, with different energy-aware metrics, we have monitored the number of total network transmission hops for the optimal service route of each execution of the requested application, which is shown in Figure 6. By the way, the number of total network transmission hops is obtained by accumulating the number of the network transmission hops between two adjacent service nodes in the service route; meanwhile, both the services nodes should have a service link specified by the workflow graph of the requested application. We notice that the number of total network transmission hops is 5 and remains almost unchanged while only using the overall energy consumption metric, and this is because the optimal service route is not refreshed until some involved sensor nodes run out of power. When only using the energy-aware load-balancing factor, the number of total network transmission hops for the optimal service route changes in a wide range mainly from 10 to 21; therefore, a greater number of sensor nodes in a larger area consume their battery quickly because of forwarding data frequently. By using both the metrics with $\mu = 0.5$ and $\lambda = 0.5$, the number of total network transmission hops decreases significantly and changes between 5 and 9 and this is because the number of network transmission hops is limited by the optimization objective of minimizing the overall energy consumption.

Further, to verify the experiment results in Figure 5, with different energy-aware metrics, we have monitored the number of total network transmission hops for the optimal service route of each execution of the requested application, which is shown in Figure 6. By the way, the number of total network transmission hops is obtained by accumulating the number of the network transmission hops between two adjacent service nodes in the service route; meanwhile, both the services nodes should have a service link specified by the workflow graph of the requested application. We notice that the number of total network transmission hops is 5 and remains almost unchanged while only using the overall energy consumption metric, and this is because the optimal service route is not refreshed until some involved sensor nodes run out of power. When only using the energy-aware load-balancing factor, the number of total network transmission hops for the optimal service route changes in a wide range mainly from 10 to 21; therefore, a greater number of sensor nodes in a larger area consume their battery quickly because of forwarding data frequently. By using both the metrics with $\mu = 0.5$ and $\lambda = 0.5$, the number of total network transmission hops decreases significantly and changes between 5 and 9 and this is because the number of network transmission hops is limited by the optimization objective of minimizing the overall energy consumption.
The number of sensor nodes

Running times (ms)

Figure 8: The running time versus the number of sensor nodes.

To analyze the global optimization capability of the IDPSO algorithm in depth, we then evaluate its performance on convergence of fitness function by comparing with the basic PSO algorithm. The convergence performance of both the algorithms for the 1st and 80th execution of the requested application is shown in Figure 7. In both the comparison experiments, the basic PSO algorithm has converged on local optimum quickly and fallen into evolutionary stagnation after a short iterative process, and the particles have to gather into the local optimum. For our proposed IDPSO, it has better global exploration ability because of the extreme perturbation scheme. Especially in Figure 7(b), IDPSO is able to search out better solutions at the 96th, 106th, and 139th iterations through a long-time global exploration. By adjusting the individual and global best position when the particle has fallen into evolutionary stagnation, the IDPSO algorithm can increase the particles’ searching field and then improve the probability of finding out better solutions.

For the running time of IDPSO, as we have analyzed in the last paragraph in Section 5, the computation complexity of IDPSO has no relationship with the number of sensor nodes. As shown in Figure 8, with the increase in the number of atomic services, the running time for finding out an optimal service route with IDPSO maintained steady. For the exhaustive algorithm, the running time increases dramatically because it traverses all the combinations to find out the optimal one.

We then evaluate the convergence and optimization performance of IDPSO with different swarm size $P$, which is shown in Figure 9. For each swarm size $P$, the IDPSO algorithm is run for 20 times independently, and then we obtain the convergence curve with the average values. The best fitness value of the optimal service route obtained by IDPSO descends with the increase of swarm size. This is because a larger swarm size can enlarge the overall searching field and improve the probability of finding out better service routes. Meanwhile, when the swarm size exceeds a certain value, for example, $P = 20$ and $P = 25$, both the best fitness values are almost the same and there is no significant improvement in performance of IDPSO. Therefore, an excessively large swarm size has little impact on improving the performance of IDPSO. However, it will bring higher computational complexity and delay.

7. Conclusion

In this paper, we have presented an energy-aware service composition method for service-oriented heterogeneous wireless sensor network, and this framework takes into account both the energy-aware metrics including overall energy consumption and energy-aware load-balancing factor. Based on this framework, we propose the IDPSO algorithm, which is able to break the evolutionary stagnation and improve the global exploration capability of the PSO algorithm with the extreme perturbation scheme. Our experiment results have shown that the IDPSO algorithm is capable of searching out better service routes with smaller fitness than the basic PSO algorithm. Meanwhile, the energy-aware performance of the service routes obtained by IDPSO is close to those optimal service routes computed by the exhaustive composition algorithm. Based on the service routes with balancing between both the energy-aware metrics, the lifetime of the sensor network can be prolonged to provide stable service composition for various applications.

Conflict of Interests

The authors declare that they have no conflict of interests regarding the publication of this paper.
Acknowledgments

Our work is supported by multiple funds in China, including the Key Program of NSFC-Guangdong Joint Funds (U1201251), Scientific Research Project of Guangzhou City (12242111582), Ph.D. Start-up Fund of Natural Science Foundation of Guangdong Province (2012040006666), and Foundation for Distinguished Young Talents in Higher Education of Guangdong (LYM11057).

References

[1] L. Mottola and G. P. Picco, "Programming wireless sensor networks: fundamental concepts and state of the art," ACM Computing Surveys, vol. 43, no. 3, article 19, pp. 1–51, 2011.
[2] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: a survey," ACM Transactions on Sensor Networks, vol. 4, no. 2, article 8, pp. 1–29, 2008.
[3] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," Service Oriented Computing and Applications, vol. 5, no. 2, pp. 71–85, 2011.
[4] M. Rouached, S. Chaudhry, and A. Koubaa, "Lowpans meet service-oriented architecture," Journal of Ubiquitous Systems and Pervasive Networks, vol. 1, no. 1, pp. 39–48, 2010.
[5] D. Gračanin, M. Ettoweissy, A. Wadaa, and L. A. Dasilva, "A service-centric model for wireless sensor networks," IEEE Journal on Selected Areas in Communications, vol. 23, no. 6, pp. 1159–1165, 2005.
[6] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, "A service-oriented architecture for QoS configuration and management of wireless sensor networks," in Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA ’10), pp. 1–8, Bilbao, Spain, September 2010.
[7] J. El Hadad, M. Manouvrier, and M. Rukoz, "TQoS: transactional and QoS-aware selection algorithm for automatic web service composition," IEEE Transactions on Services Computing, vol. 3, no. 1, pp. 73–85, 2010.
[8] A. Liu, Q. Li, L. S. Huang, and M. Xiao, "FACTS: a framework for fault-tolerant composition of transactional web services," IEEE Transactions on Services Computing, vol. 3, no. 1, pp. 46–59, 2010.
[9] J. Bronsted, K. M. Hansen, and M. Ingstrup, "Service composition issues in pervasive computing," IEEE Pervasive Computing, vol. 9, no. 1, pp. 62–70, 2010.
[10] S. Kalasapur, M. Kumar, and B. A. Shirazi, "Dynamic service composition in pervasive computing," IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 7, pp. 907–918, 2007.
[11] I. Estévez-Ayres, P. Basanta-Val, M. Garcia-Valls, J. A. Fisteus, and L. Almeida, "QoS-aware real-time composition algorithms for service-based applications," IEEE Transactions on Industrial Informatics, vol. 5, no. 3, pp. 278–288, 2009.
[12] J. Wang, "Exploiting mobility prediction for dependable service composition in wireless mobile ad hoc networks," IEEE Transactions on Services Computing, vol. 4, no. 1, pp. 44–55, 2011.
[13] S. Jiang, Y. Xue, and D. C. Schmidt, "Minimum disruption service composition and recovery in mobile ad hoc networks," Computer Networks, vol. 53, no. 10, pp. 1649–1665, 2009.
[14] C. F. Lai, Y. W. Ma, S. Y. Chang, H. C. Chao, and Y. M. Huang, "OSGi-based services architecture for cyber-physical home control systems," Computer Communications, vol. 34, no. 2, pp. 184–191, 2010.
[15] J. Huang, F. B. Bastani, I.-L. Yen, and W. Zhang, "A framework for efficient service composition in cyber-physical systems," in Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE ’10), pp. 291–298, June 2010.
[16] I. Amundson, M. Kushwaha, X. Koutsoukos, S. Neema, and J. Sztipanovits, "OASIS: a service-oriented middleware for pervasive ambient-aware sensor networks," Pervasive and Mobile Computing Journal on Middleware for Pervasive Computing, 2006.
[17] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN ’08), pp. 740–747, October 2008.
[18] E. Avilés-López and J. A. Garcia-Macias, "TinySOA: a service-oriented architecture for wireless sensor networks," Service Oriented Computing and Applications, vol. 3, no. 2, pp. 99–108, 2009.
[19] A. Sleman and R. Moeller, "SOA distributed operating system for managing embedded devices in home and building automation," IEEE Transactions on Consumer Electronics, vol. 57, no. 2, pp. 945–952, 2011.
[20] C. L. Fok, G. C. Roman, and C. Lu, "Servilla: a flexible service provisioning middleware for heterogeneous sensor networks," Science of Computer Programming, vol. 77, no. 6, pp. 663–684, 2012.
[21] X. Wang, J. Wang, Z. Zheng, Y. Xu, and M. Yang, "Service composition in service-oriented wireless sensor networks with persistent queries," in Proceedings of the 6th IEEE Consumer Communications and Networking Conference (CCNC ’09), pp. 368–372, January 2009.
[22] S. C. Geyik, E. Bulut, and B. K. Szymanski, "Utilizing PCFGs for modeling and learning service compositions in sensor networks," in Proceedings of the IEEE 9th International Conference on Services Computing (SCC ’12), pp. 282–289, 2012.
[23] R. Dilmaghani, S. Geyik, K. Grueenberg, and J. Lobo, "Policy-aware service composition in sensor networks," in Proceedings of the IEEE 9th International Conference on Services Computing (SCC ’12), pp. 186–193, 2012.
[24] D. Martin, M. Burstin, J. Hobbs et al., "OWL-S: semantic markup for web services," 2003, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
[25] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL)," 2001, http://www.w3.org/TR/wsd1.
[26] D. Box, D. Ehnebuske, G. Kakivay et al., "Simple object access protocol (SOAP)," 2007, http://www.w3.org/TR/soap/.
[27] T. Andrews, F. Curbera, H. Dholakia et al., "Business process execution language for web services (BPEL)," 2003, http://www.ibm.com/developerworks/library/specification/ws-bpel/.
[28] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, Perth, Australia, December 1995.
[29] R. C. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in Proceedings of the Congress on Evolutionary Computation, vol. 1, pp. 81–86, May 2001.
[30] P. Bajpai and S. N. Singh, "Fuzzy adaptive particle swarm optimization for bidding strategy in uniform price spot market," IEEE Transactions on Power Systems, vol. 22, no. 4, pp. 2152–2160, 2007.
[31] M. Clerc and J. Kennedy, “The particle swarm: explosion stability and convergence in a multi-dimensional complex space,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[32] K. L. Calvert, M. B. Doar, and E. W. Zegura, “Modeling internet topology,” *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, 1997.