J Supercomput (2015) 71:87–120
DOI 10.1007/s11227-014-1289-4

Distributed optimization Grid resource discovery

Mohammad Hasanzadeh · Mohammad Reza Meybodi

Published online: 25 September 2014

Abstract Grid computing is a framework for large-scale resource sharing and indexing that evolves with the goal of resource provisioning. In this paper, we develop a distributed learning automata (DLA) based on multi-swarm discrete particle swarm optimization (PSO) approach for Grid resource discovery, called distributed optimization grid (DOG) resource discovery algorithm. This algorithm makes use of swarms of particles for different computational resource metrics while a group of DLA is the control unit of each swarm of particles. The algorithm takes advantage of the PSO solution diversity to optimize the quality of delivered resource. Moreover, the recommended algorithm uses DLA as a fully distributed model for imitating the Grid infrastructure topology. Our experimental results show that DOG is fast as well as efficient and accurate.

Keywords Grid resource discovery · Learning automata · Distributed learning automata · Discrete particle swarm optimization

1 Introduction

Grid computing simplifies the way scientific workflows execute and delivers plain services by automating and managing different resources across the globe. The resource discovery service plays a critical role in Grid sites by delivering automated resource management capabilities. Different sites of the Grid have different administrative

M. Hasanzadeh (✉) · M. R. Meybodi
Soft computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
e-mail: mdhassanzd@aut.ac.ir

M. R. Meybodi
e-mail: mmeybodi@aut.ac.ir

rules, where each of them forms an independent virtual organization (VO) [1]. Each VO accompanies with a Grid information service (GIS) [2–4] leveraging the resource management issues. For instance, an implementation of such a service is the monitoring and discovery system (MDS) [1] of the Globus Grid Toolkit [5].

Highly dynamic networks such as Grid must have a functionality to manage and monitor resource allocation and condition processes. The Grid is responsible for processing the incoming tasks. It utilizes a scheduler to fulfil these tasks. Also, The resource discovery is the process of finding the requested resources of tasks with respect to one or more user criterion performance, price, etc. Furthermore, the resource discovery overlaps with Grid scheduler in terms of resource scavenging and reserving.

Ad Hoc Grid illustrates a highly dynamic computing environment. In [6], a resource selection scheme for Ad Hoc Grid is proposed. Coordinate load management enables Grid to efficiently manage computing resources. In [7], a coordinated load management approach is applied to Grid resources to perform resource brokering. Preventing the communication bottleneck is one of the major challenges of Grid. In [8], a direction aware approach for resource discovery in Grid and Cloud is introduced. Resource allocation is a complex problem in Grid and Cloud environments. A task-oriented resource allocation model proposed for Cloud environments in [9]. Resource allocation mechanisms aim for satisfying the user preferences and Grid policies. A Multi-agent based negotiation strategy is proposed in [10] for studying the interactions between Grid providers and consumers.

Machine Learning (ML) is a field of Artificial Intelligence (AI), aiming to construct a model that can learn from data. Focusing on the ML algorithms, one can view Reinforcement Learning (RL). RL studies the learning behavior of Agents while interacting with an unknown environment. Learning Automata (LA) [11] is one of the RL's tools which embodies an autonomous unit placed in an unknown environment trying to interact with the environment by taking a series of actions and earning a series of reward or penalty signals. Moreover, DLA [12] is a newfound network of LA that cooperatively collaborates to solve a particular problem.

Computational Intelligence (CI) is a multidisciplinary topic in AI which utilizes different computational methodologies to address the complicated real-world problems. Evolutionary Computation (EC) is a subfield of CI that is inspired from evolution biology of species. Moreover, PSO [13] is an EC algorithm that is inspired from social behavior of flocks of birds and schools of fish.

LA enable the versatility of AI with a simple integration of probabilistic control parameters into the adaptive control systems. Target monitoring and lifetime scheduling are among challenging issues of wireless sensor networks (WSNs). Moreover, LA models are employed in [14] for solving target coverage problem of WSNs. Classification algorithms try to determine the observations labels based on the set of training data. A Cellular Automata (CA) model is proposed in [15] for binary classification problem. Also, the mobile Ad Hoc networks (MANETs) are decentralized networks constituting of independent groups of nodes. In [16], LA theory is used for designing a routing protocol for MANET. Furthermore, in [17], a cellular learning automata (CLA) model is introduced for simulating the investment behavior of stock market dealers. Finally, designing a resource discovery protocol for Grid environment is a challenging task. In [18], a distributed learning automata (DLA) framework is implemented for Grid resource discovery.

Efficient tasks' mapping of highly dynamic Grid resources leads to high resource utilization of Grid. In [19], a task mapping algorithm based on LA is proposed for Grid resources. Also, Grid has a dynamic environment and includes distributed resources. In [20], the PSO algorithm is utilized to solve the independent task scheduling problem of dynamic Grid systems. Furthermore, the uncertain nature of Grid needs an efficient scheduling algorithm for its dynamic environment. In [21], a scheduling algorithm is introduced that processed the synthetic uncertainty of tasks to schedule the Grid tasks.

The DOG algorithm is a Grid computing initiative which allows Grid on-demand multipath resource discovery. Also, the DOG Algorithm utilizes both DLA and PSO algorithms for different computational resource metrics. A DLA constructs a graph of LA by sampling distribution of Grid topology where the vertices are interconnected LA and the links between them are actions associated with selecting the endpoints of vertices (LA). Each particle of PSO population is coupled with a DLA initializes its position. Different PSO populations facilitate different groups of DLA on behalf of their position. The objective of DOG is to use DLA and PSO to find a policy that determines a diverse set of routes from initiator VOs to supplier VOs with high hit-count and optimized hop-count.

The remainder of this paper is organized as follows: Section 2 reviews the related works. Section 3 presents the LA methodology. Section 4 devotes to a brief study of PSO methodology. Section 5 introduces the proposed model methodology and its hierarchy view. In Sect. 6, we present the empirical analysis of proposed model. Finally in Sect. 6, we draw the conclusion of the proposed model.

2 Resource discovery in Grid systems

Since emerging computational Grids, the new research topic of resource discovery has been introduced. The Grid could be discussed from various points of view users, applications, administration, security, scale, Quality of Service (QoS), etc. Also Grid has various problems like infrastructure deployment, architectural design, node management, job scheduling, job monitoring, resource discovery, resource reservation, resource scavenging, etc. Moreover, Grid needs meta-scheduling systems [22] to execute types of jobs that require special attention because an increasing number of jobs are being executed among limited number of Grid resources.

The Grid system is responsible for sending a job to a given machine to be executed [23]. Three main concepts of executing a job on Grid are: (1) job scheduling, (2) resource reservation, and (3) resource scavenging. To find the suitable resources for user tasks, resource discovery is vital for all three mentioned concepts. The relationship between resource discovery and aforementioned concepts is shown in Fig. 1.

In the following, some well-known Grid resource discovery strategies will be described. Please note that request forwarding strategy is the basis of the proposed architecture. Also, a comprehensive survey of resource discovery in Grids is reported in [24].

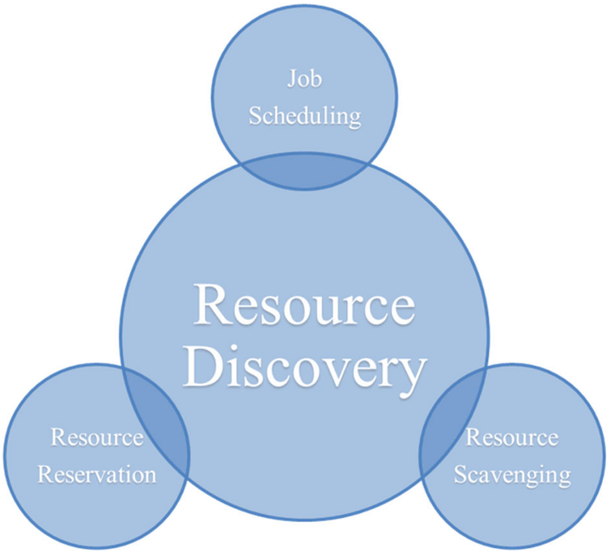

Fig. 1 Resource discovery overlaps with job scheduling, resource scavenging and resource reservation

2.1 Request forwarding strategies

Imantchi and Foster [25,26] propose four request forwarding strategies. Each participant of the proposed architecture has a VO that stores and provides access to local resources for other nodes. Users send their requests to a locally known node. The node responds with the matching resource if it has them locally, otherwise it forwards the request to another node. In these strategies in addition to neighbors' addresses, each node can maintain a history of its neighbors' responses. These strategies choose the appropriate node to forward the resource request. The discussed strategies are listed below:

(1) In the *random walk strategy* the next node to that request which is forwarded is selected randomly. This strategy does not need extra memory. (2) In the *experience-based +random strategy* nodes forward the resource request based on the past experience of their neighbors' answers. In this strategy, a repetitive request will forward to a node that successfully answered similar requests previously. If no coherent experience exists, the request is forwarded by random walk strategy. (3) The *best-neighbor strategy* records the number of successful answers of each neighbor and forwards the resource request to the peer that answered the largest number of requests. (4) The *experience-based +best-neighbor strategy* is similar to experience-based strategy except that if no matching neighbor exists, the request is forwarded based on best neighbor strategy.

The experimental results showed that the learning-based strategy is the best one. This strategy uses the advantage of similarity learning in requests responding. The random walk strategy obtained the worst results, but it takes advantage of no additional memory to save the history of neighbors.

Tangpongprasit et al. [27] propose a flat decentralized P2P architecture for resource discovery in Grid environments. The feature of this mechanism is to find an appropriate matching resource using time-to-live (TTL) value as the deadline of resource discovery. In this strategy, users connect to a locally known node and the resource discovery is initiated. The resource discovery mechanism will forward the resource request to other neighbors until request TTL expires or resource is found.

A node can forward a resource request using one of four request forwarding strategies of [25,26]. Simulation results show that experience-based + best-neighbor has the best performance. In [25,26] resource is chosen based on first-found-first- served (FFFS) schema. This schema immediately sends back the resource to the user, if it finds a matching resource type. During the same TTL, with the replacement of FFFS schema by a reservation algorithm, more available matching resources can be found. The mechanism decides which matching resource should be informed back to the user. The tradeoff between the FFFS schema and the TTL-based reservation algorithm generates four request propagation strategies which automatically suggest a resource to the user:

(1) In the *TTL + closest-attribute strategy* inter nodes forward the resource request until the TTL value reaches zero. The resource reservation algorithm is tried to find the resource which has the best similar attributes. (2) The *TTL + highest-performance strategy* is similar to previous strategy except the reservation algorithm looks for the resource which has the highest performance. (3) In the *FFFS + closest-attributes strategy*, the node which has the first available resource will immediately offer service to the user. If there are multiple available resources in the same node, the node is assigned the resource whose attributes are closest to the request description. (4) The *FFFS + highest-performance strategy* is similar to FFFS + closest-attributes strategy, but in the case of multiple available resources, the resource which has the highest performance is chosen for the user.

The experiments perform on a 1,000-node network with duplex links and no link failure. The TTL-based strategies need more hops to obtain a better solution than FFFS schema. In the resource discovery phase, the algorithm does not cause much traffic in the network. Moreover, the algorithm maintains the tradeoff between resource utilization and turn-around time.

2.2 Genetic algorithm strategy

Noghabi et al. [28] propose a genetic algorithm (GA) based resource discovery mechanism. Their proposed architecture uses three units to locate the requested resource with optimal value of hop-count and success rate. (1) The *statistical table unit* is the basic data structure of this forwarding algorithm. Each peer has two statistical tables in which the columns and rows are associated with peer neighborhood and requested resources, respectively. These tables store information about number of successful results or average number of hops for the previous queries through each peer in the neighborhood. (2) The *resource checking and query evaluation unit* is a local unit of peer which is used to observe peer resources whenever a query is created. (3) The *genetic operation unit* performs if the resource checking and query forwarding unit could not locate requested resource in local resources. Every peer in the Grid has its own population associated to its neighborhood that genetic operations apply to them. The GA uses the information of statistical tables to locate the requested resource on the Grid.

The simulation performed in a 10,000-peer network while the maximum number of neighbor peers is 50. In addition to compare the GA-based resource discovery strategy with flooding and random walks strategies, in the simulations the convergence of GA was also observed. However, there is no information about the distribution of resources across the network, but the GA-based algorithm finds the requested resource within 3 or 4 steps with the success rate of 95 %. Moreover, the algorithm does not support multi-range queries.

2.3 Ant colony strategies

Multi-agent system (MAS) is a computational system where a set of autonomous agents interact within an environment [29]. Nowadays, ant colonies are used as MDSs to solve different problems. Deng et al. [30] propose a large-scale P2P Grid system which employs an ant colony optimization (ACO) algorithm to locate the required resources. The system constitutes an overlay network which is built on the top of the existing network. Each node of the system is modeled as one nest that can initiate ant. Each nest is composed of four modules: (1) The *Grid service module* monitors and manages the Grid services that will be located by ants. (2) The *routing table module* saves the routing information of ants. (3) The *ant management module* is responsible for generating and killing ants. (4) The *communication module* manages the ants' movements between nests. Also, each ant consists of two modules: (1) The *life cycle management module* keeps the TTL value of ant. (2) The *memory management module* stores the requested resource and the path which it has passed.

The MDS is a key component of resource discovery in Grid environments. When a user request is generated, the nearest MDS will issue one or several ants to locate the corresponding resource. Since the requested resource is not found and the TTL is larger than zero, the ant will randomly select an entry from the routing table of its resident MDS. If the required resource is registered to the current MDS, the ant will travel back to its original nest and the routing table will be updated.

The network topology is dynamic and nodes join and leave the system at any time. The bandwidth of links is the same and the TTL of each ant is set as 256. The method is effective for locating highly duplicated resources. The routing information disseminated quickly and most of the ants learned the shortest path. Therefore, the algorithm keeps network resource consumption low. Moreover, by increasing the number of ants, searching efficiency is improved.

Brocco et al. [31] propose a fully distributed ant-inspired self-organized overlay network for large-scale Grid systems. Each peer in the proposed overly contributes to rearrange its local links. Different classes of ant-like mobile agents are used for the management of overlay network: (1) A *discovery ant* collects information about the network. (2) A *construction link ant* connects a new peer to the overlay. (3) An *optimization-link ant* creates links between peers. (4) An *unlink ant* removes existing links between peers. (5) An *update neighbors ant* updates the information of peer neighborhood.

The resource discovery is performed using a limited and selective flooding algorithm. Limited flooding avoids forwarding queries that have already been processed and selective flooding forward the queries to a subset of all neighbors. The aim of this approach is to exploit from similar nodes for matching redundant queries, whereas there are many situations where an exact match is not available. Simulations consider both performance and efficiency of resource discovery strategy. The strategy improves hit rate with little impact on the network traffic compared to traditional flooding approaches.

2.4 Super-peer strategy

A super-peer operates both as a centralized server at the cluster level, where cluster size is the number of nodes in the cluster and a regular node at the super-peer level [32]. Mastroianni et al. [33] adopt a super-peer Grid resource discovery protocol. The Grid overlay network is partitioned into several Physical Organizations (POs). Each PO (cluster) included a set of Grid nodes within one administrative domain. Also, a set of high available Grid nodes in each PO can be chosen as super-peers.

The algorithm works as follows: generating a query message in the Grid node of PO, then the message will be forwarded to the local super-peer. If the queried resource is not presented in the local PO, the super-peer forwards the query to a set of selected neighbor super-peers. Whenever a resource matching the query criteria is found in a remote PO, a query hit is generated. The set of neighbors to which a query is forwarded is determined based on statistical information about previous query hits received from the neighboring super-peers. Performance compassions showed that the super-peer model outperforms decentralized and hierarchical models in terms of response time and expected number of results. Moreover, the super-peer is more effective in very large-scale Grids; therefore super-peer is a scalable model.

2.5 Comparison summary

When a user queries a resource, a resource discovery algorithm should search for the requested resource. There are different resource discovery heuristics in Grid environment. Decentralized resource discovery strategies [25–27] suffer from initial high messaging overhead of blind request forwarding and terminal single path convergence of elite request forwarding. Another method for resource discovery problem would be GA-based query forwarding [28]. This optimization approach performs genetic operations on the population of each node which increase the computational load of query forwarding on each node. Ant-inspired agent-based resource discovery [30,31] enables parallel and distributed resource discovery, while multiple ants increase load on central nodes. Super-peer approach utilizes both hierarchical and decentralized architectures, but creates a single point of failure which is associated with super-peer.

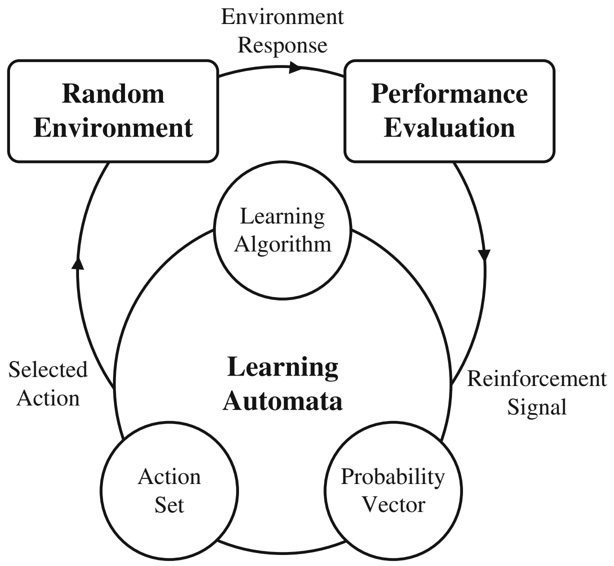

Fig. 2 Schematic view of a learning automaton in a random environment

3 Learning automata methodology

Learning automaton [11] also called variable structure learning automaton (VSLA) is an area of reinforcement learning in machine learning surveyed by Narendra and Thathachar in 1970s. Learning automata supporting a wide range of applications included evolutionary computation [34–36], wireless sensor networks [37], economic Grids [38], etc.

A learning automaton [11] can be modeled as an abstract entity with finite number of actions. Learning automaton operates by selecting one of its actions and employing it to the environment. The action is evaluated by the environment and the learning automaton uses the environment response to select its next action. Along this procedure learning automata gradually learns to select optimal action. The interactions between learning automaton and the environment are depicted in Fig. 2.

3.1 Variable structure learning automata

Variable structure learning automata (VSLA) [11] can be modeled as a quadruple $\{\alpha, \beta, P, T\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of actions, $\beta = \{\beta_1, \beta_2, \ldots, \beta_r\}$ is the set of inputs, $p = \{p_1, p_2, \ldots, p_r\}$ is the set of action probabilities and $p(n+1) = T\,[\alpha(n), \beta(n), p(n)]$ is the learning algorithm. If $p(n+1)$ is a linear function of $p(n)$ the schema is called linear, otherwise it is nonlinear. Assume that the action α_i is selected in nth step; a typical linear leaning algorithm of VSLA is shown in Algorithm 1 [11].

Algorithm 1 Variable Structure Learning Automata with Linear Learning Algorithm

```
define
Initialize r-dimensional action set: $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ where $r$ is the number of actions.
Initialize r-dimensional action probability vector: $p = \left\{\left(\frac{1}{r}\right)_1, \left(\frac{1}{r}\right)_2, ..., \left(\frac{1}{r}\right)_r\right\}$
while (the automaton converge to one of its action)
    The learning automaton selects an action based on the probability distribution of $p$.
    The environment evaluates the action and calculates the reinforcement signal $\beta \in \{0,1\}$.
    The environment feedbacks $\beta$ to the learning automaton.
    Consider $i$ as the selected action of the automaton and $j$ as the current checking action and
    $n$ as the $n^{th}$ step of evolution.
    Update the probability Vector:
    for each action $j \in [1, ..., r]$ do
        if $\beta = 0$        \\ positive response
            $p_j(n+1) = \begin{cases} p_j(n) + a.(1 - p_j(n)) & if\ i = j \\ p_j(n).(1-a) & if\ i \neq j \end{cases}$   (1)
        else if $\beta = 1$   \\ negative response
            $p_j(n+1) = \begin{cases} p_j(n).(1-b) & if\ i = j \\ \frac{b}{r-1} + (1-b).p_j(n) & if\ i \neq j \end{cases}$   (2)
        end if
    end for
end while
```

In (1) and (2), a and b are called *learning parameters* and they are associated with the reward and penalty responses. Considering the values of a and b, there are three types of learning algorithms. In Linear Reward-Penalty algorithm (L_{R-P}), it is considered that a and b are equal. In Linear Reward-Inaction (L_{R-I}) the learning parameter b is set to 0. And finally in Linear Reward-epsilon-Penalty (L_{R-eP}) the learning parameter b is much smaller than a.

3.2 Learning automata with variable number of actions

LA have finite number of actions but in some applications LA with variable number of actions are necessary [39]. Similar to VSLA the action set, input set and probability vector of each variable action set learning automaton is equal to: α, β and p. In nth step, this automaton selects its action from a non-empty set of actions ($V(n)$) which is nominated as active actions. The process of this kind of automaton is as follows: for choosing an action in nth step, at first the sum of probabilities of active actions is computed ($k(n)$) and after that $\hat{p}(n)$ vector is calculated based on (3). Then automaton selects an action based on $\hat{p}(n)$ probability vector of active actions set and deploys it to the environment. If α_i is the selected action, after receiving the environment response the automaton will update $\hat{p}(n)$ probability vector through (4) in case of getting the reward or (5) in case of getting the penalty. Similar to VSLA, in (4) and (5) a and b are reward and penalty parameters. Finally, $p(n+1)$ probability vector of actions is scaled using $\hat{p}(n)$ and $p(n)$ in (6) and (7). The procedure of probability update of a learning automaton with changing number of actions is shown in Algorithm 2 [39].

3.3 Distributed learning automata

A DLA [12] is a network of LA which collectively cooperates to solve a particular problem. A DLA can be modeled by a directed graph in which the set of nodes of graph constitutes the set of LA and the set of outgoing edges for each node constitutes the set of actions for corresponding learning automaton. When a learning automaton selects one of its actions, another learning automaton on the other end of edge corresponding to the selected action will be activated.

An example of DLA is given in Fig. 3, in which every automaton has two actions. If automaton A_1 selects action α_3, then automaton A_3 will be activated. The activated automaton A_3 chooses one of its actions which in turn activates one of the automata connected to A_3. At any time only one automaton in the network will be activated.

A DLA is represented by a directed graph where the set of graph vertices constitutes the set of LA and the set of outgoing links for each node constitutes the set of actions for corresponding LA. When a learning automaton chooses an action, the corresponding learning automaton of selected endpoint will be activated. Typically, a DLA with n LA is a graph $G = (A, E)$ containing a set $A = \{A_1, A_2, \ldots, A_n\}$ of automata together with a set $E \subset A \times A$ of edges, where edge (i, j) is a 2-element subset of A corresponds to action α_j of automaton A_i.

Since introducing DLA by Beigy and Meybodi [12], DLA has received a lot of applications in solving various problems such as: Stochastic shortest path problem [12], traveling salesman person (TSP), Steiner tree problem, minimum spanning tree, vehicle routing problem, connected dominating set (CDS) problem in backbone formation of wireless sensor networks [40], web page recommendation [41], and Grid computing [18].

4 Particle swarm optimization methodology

The PSO algorithm [13] is a population based stochastic optimization strategy developed by Kennedy and Eberhart in 1990s. It has applications in Financial option [42] pricing, Bitmap join index problem in data warehouses [43], Grid workflow scheduling [44], etc.

4.1 Standard particle swarm optimization

In PSO, a swarm of particles fly in a $D \in [1, 2, \ldots, d]$ dimensional space. The position and velocity vectors of ith particle of swarm are $X_i = (x_i^1, x_i^2, \ldots, x_i^d)$ and

Algorithm 2 Variable Action-set Learning Automata with Linear Learning Algorithm

```
define
Initialize r-dimensional action set: $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ where r is the number of actions.
Initialize r-dimensional action probability vector: $p = \left\{ \left(\frac{1}{r}\right)_1, \left(\frac{1}{r}\right)_2, ...., \left(\frac{1}{r}\right)_r \right\}$
while (the automaton converge to one of its action)
  Calculate active actions of learning automaton: $V(n) = \{A_1, A_2, ..., A_m\} \mid V(n) \subseteq \alpha$
  Calculate sum of probabilities of active actions: $K(n) = \sum_{\alpha_i \in V(n)} p_i(n)$
  Calculate action probability vector of active actions:
  for each available action $k \in [1, ..., r]$ do
    if $\alpha_i \in V(n)$        \\ active action
        $\hat{p}_j(n) = prob\left[\alpha(n) = \alpha_i \mid V(n), \alpha_i \in V(n)\right] = \frac{p_k(n)}{K(n)} \mid j \in [1, ..., m]$   (3)
    end if
  end for
  Construct a hash function which maps: $k \in [1, ..., r] \xrightarrow{F} j \in [1, ..., m]$
  The learning automaton selects an action based on the probability distribution of $\hat{p}$.
  The environment evaluates the action and calculates the reinforcement signal $\beta \in \{0, 1\}$.
  The environment feedbacks $\beta$ to the learning automaton.
  Consider i as the selected action and j as the current checking action and n as the $n^{th}$ step of evolution.
  Update the probability vector of available actions:
  for each active action $j \in [1, ..., m]$ do
    if $\beta = 0$        \\ positive response
        $\hat{p}_j(n+1) = \begin{cases} \hat{p}_j(n) + a.\left(1 - \hat{p}_j(n)\right) & if\ i = j \\ \hat{p}_j(n).(1-a) & if\ i \neq j \end{cases}$   (4)
    else if $\beta = 1$   \\ negative response
        $\hat{p}_j(n+1) = \begin{cases} \hat{p}_j(n).(1-b) & if\ i = j \\ \frac{b}{r-1} + (1-b).\hat{p}_j(n) & if\ i \neq j \end{cases}$   (5)
    end if
  end for
  Rescale the probability vector of available actions:
  for each action $k \in [1, ..., r]$ do
    if $\alpha_k \in V(n)$        \\ active action
      $j = F(k)$          \\ map $k^{th}$ action of $\alpha(n)$ to $j^{th}$ action of $V(n)$
        $p_k(n+1) = \hat{p}_j(n+1).K(n)$   (6)
    else if $\alpha_j \notin V(n)$      \\ inactive action
        $p_k(n+1) = p_j(n)$   (7)
    end if
  end for
end while
```

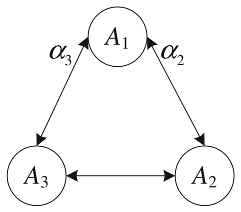

Fig. 3 Distributed learning automata with three LA

$V_i = (v_i^1, v_i^2, \ldots, v_i^d)$, respectively. The velocity V_i^D and position X_i^D of the ith particle in Dth dimension are changed according to the following equations [13]:

$$v_i^D = w \times x_i^D + c_1 \times \text{rand1}_i^D \times (\text{pbest}_i^D - x_i^D) + c_2 \times \text{rand2}_i^D \times (\text{gbest}^D - x_i^D) \tag{8}$$

$$x_i^D = x_i^D + v_i^D \tag{9}$$

where in (8), $\text{pbest}_i = (\text{pbest}_i^1, \text{pbest}_i^2, \ldots, \text{pbest}_i^d)$ and $\text{gbest} = (\text{gbest}^1, \text{gbest}^2, \ldots, \text{gbest}^d)$ are the best personal position of ith particle and the best global position of swarm, respectively. c_1 and c_2 are acceleration constants, rand_1 and rand_2 are two uniform random numbers in range [0,1] and w is inertia weight.

4.2 Discrete particle swarm optimization

Discrete PSO [45] is emerged from continuous version of PSO to work in discrete binary space. The discrete binary space can be viewed as a hypercube. In hypercube search space, particle trajectory changes as a function of hypercube corners value. In this context, particle velocity defines as the number of changes in bits (0/1). If none of bits change, the particle will standstill and if all of bits change, the particle will move the farthest.

In discrete PSO the velocity of particle is updated through (8), but the position of particle is computed based on the changes in the probability that a bit will be 0 or 1. This context is applied to the Dth bit of ith particle through the sigmoid transformation of (10) [45]:

$$x_i^D = \frac{1}{1 + e^{-v_i^D}} \tag{10}$$

4.3 Learning automata based discrete particle swarm optimization

LA-based discrete PSO [46] utilizes a set of learning automata as the brain of particles to determine their position. In this algorithm, a bi-action learning automaton is assigned to each dimension of particle. In each dimension, a learning automaton initializes the particle's position by selecting one of its actions (0/1). Then the learning automaton will receive the reinforcement signal which is calculated form the particles' fitness. This process is like the sigmoid transformation of (10) which initializes the position of particle, except for the LA that are not associated in (10).

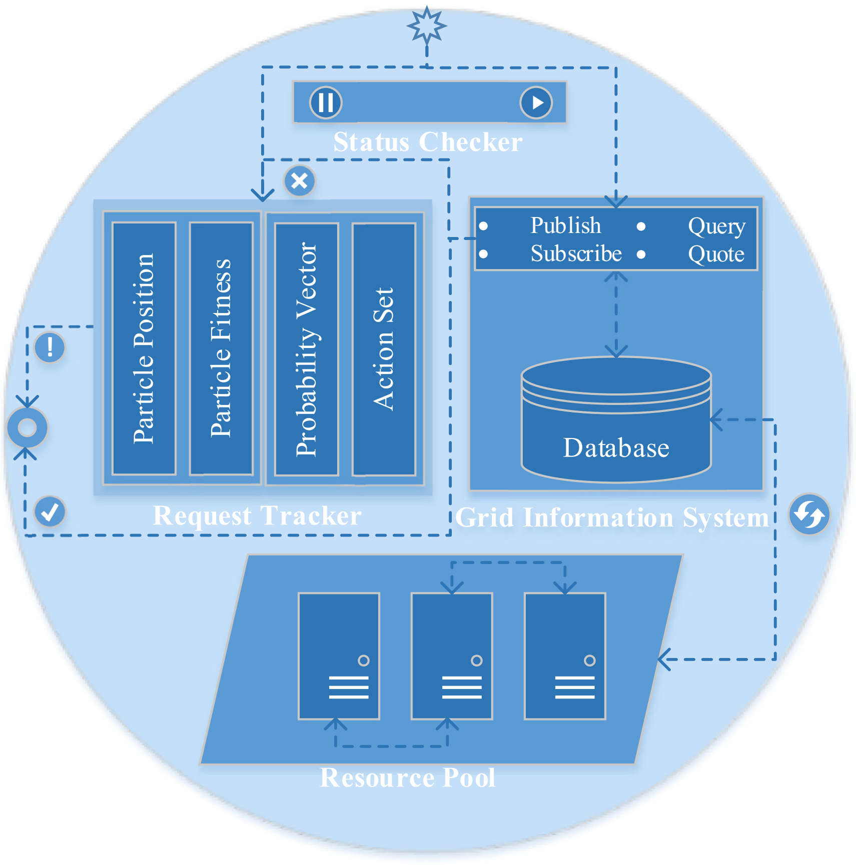

Fig. 4 A VO consists of status checker, resource pool, GIS and request tracker units. The *star* icon represents the origination point of incoming query while the *doughnut* icon represents the termination point of it. The *play* icon represents the availability of VO, while the *pause* icon represents the unavailability of it. The *multiply* icon represents an unsuccessful query, while the *tick* icon represents a successful one. The *exclamation mark* represents query forwarding and the *refresh* icon represents the information flow between resource pool and GIS

5 Distributed optimization Grid resource discovery

In this section the DOG algorithm is introduced and discussed. The proposed algorithm consists of multiple components that logically interacts with each other to literally meet the designated criteria of Grid resource discovery process.

5.1 Overall model structure

Like other computational intelligence techniques, PSO is a population-based nature inspired algorithm that aims to solve some real-world complex problems. Recently, PSO and LA successfully combined [34,35] to enhance the optimization capability of PSO and boast the learning ability of LA. Also in this paper, the DOG algorithm combines Discrete PSO and DLA and applies them to resource discovery problem. Consider a Grid with a distributed set of resources which lies under different administrative domains named as VO. Figure 4 shows the four-component regulatory mechanism that embedded in each VO. The definition of these components are as follows:

1. A Grid monitoring unit for determining whether the VO is available or not. This monitoring task is done by status checker unit which periodically tests the status of VO.
2. A resource pool unit which physically aggregates the computational power of a Grid-datacenter.
3. A resource management unit (GIS unit) which provides logical links for the status of resource pool members and processes the incoming queries by searching the local database.
4. A request tracker unit which forwards the unsupported queries to another VO.

Discrete PSO and LA constitute the aforementioned four-component VOs, where each of them should has a distributed Grid-datacenter. The Grid-datacenters connect together via Local Area Network (LAN) and VOs connect with each other via continental fiber-optic networks. Multiple VOs with local Grid-datacenters form a Grid-cell. The Grid cells connect together via global Backbone networks.

As shown in Fig. 4, an incoming query is delivered to the Grid monitoring unit which checks the availability of the VO. If the VO is available, the query will deliver to the GIS unit for checking the local resource pool, otherwise it will pass to the request tracker unit for query forwarding. The GIS unit sends the attached query information to the local database and checks whether the requested resource exists or not, while the request tracker unit forwards the query to another VOs, either because the VO or because the query is unavailable.

5.2 Model workflow

In the proposed model, at the verge of the VO the status checker unit is invoked and the state of VO is checked. Each VO contains a request tracker unit which forwards the query based on LA and discrete PSO algorithm. The cumulative view of LA forms a DLA which the set of VOs in the Grid constitutes the set of automata and the set of outgoing edges for each VO constitutes the actions of corresponding automaton. Each query is equipped with a population of particles and a group of DLA. By receiving an incoming query, GIS unit checks the resource pool whether it can service the query or not. In case of an unsupported query, the request tracker unit will enable the corresponding automaton. The designated learning automaton will choose an action based on its action set and determines the next VO to forward the query. Eventually, the chosen action is used as the position of the corresponding particle. Each query is allowed to transmit till its TTL reaches the end and the dimensionality of each particle is equal to this TTL value. The sequence of actions that occurs through the initiator VO to terminator VO initializes the position of particle and forms a solution

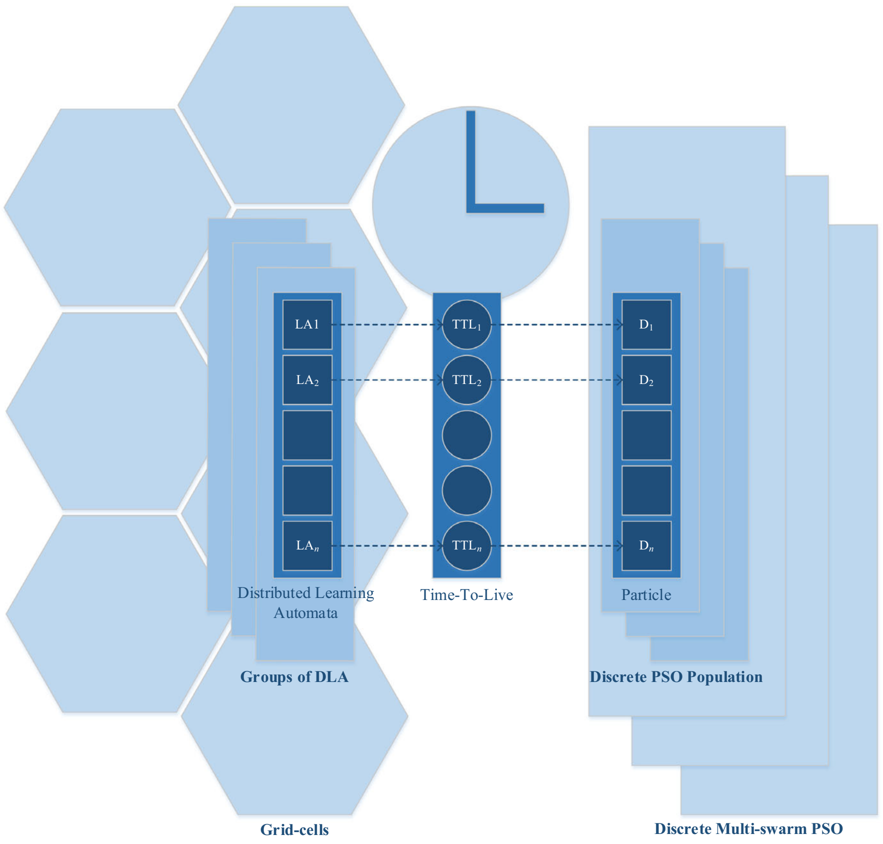

Fig. 5 DLA Interacts with Discrete PSO population by initializing the position of each particle

vector. Figure 5 shows the resource discovery model from the view of one particle of discrete PSO population and its corresponding DLA. Of course, from request tracker view (Fig. 4) learning automaton and the corresponding dimension of particle are the decision maker and query actuator components of VO, respectively.

In each TTL, a learning automaton forwards the request to another VO and stimulates the particle's position in the corresponding dimension. The selection of another VO is based on the action set of learning automaton which is designated to current VO. This action set constitutes a list of all adjacent VOs. The learning automaton chooses a VO from the aforementioned action set with respect to the learning automaton probability vector.

As long as each particle and its corresponding DLA have independent data structures, the model is capable of executing in both serial and parallel modes. When the maximum TTL value reaches, the fitness of particles is evaluated through a Grid-enable fitness function. Then, for each DLA a reinforcement signal is calculated based on the fitness of corresponding particle. Finally, the reward or penalty signal is deployed to the probability vectors of DLA.

Table 1 The relationship between resources types and their designated metrics with number of DLA groups and PSO populations

Resource type	Metric	Grid nodes				
		N_1		…	N_n	
	r_1	LA_{R1r1N1}	P_{R1r1N1}	…	LA_{R1r1Nn}	P_{R1r1Nn}
R_1	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	r_m	LA_{R1rmN1}	P_{R1rmN1}	…	LA_{R1rmNn}	P_{R1rmNn}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	…	$\vdots$	$\vdots$
	r_1	LA_{Rtr1N1}	P_{Rtr1N1}	…	LA_{Rtr1Nn}	P_{Rtr1Nn}
R_t	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	r_m	LA_{RtrmN1}	P_{RtrmN1}	…	LA_{RtrmNn}	P_{RtrmNn}

5.3 Imitating the population distribution of multi-swarm discrete particle swarm optimization for distributed learning automata grouping

From coarse-grain distributed computing viewpoint, each DLA conveys a particle while a group of DLA conveys a population of particles. In other words, a PSO population is assigned to the entire computing resource metrics of VOs resource pools. An instance of an attribute for a specific resource type is called a metric; but from fine-grain distributed computing viewpoint, groups of DLA convey swarms of particles while each swarm is assigned to a specific computing resource metric of VOs resource pool. This approach enables a finer degree of precision for query tracking. For example, assume r resources from t resource types, $g = r/t$ is the resource granularity. Moreover, each resource type is divided into m resource metrics. So, in this configuration $t * m$ swarms of particles and groups of DLA are used in the DOG algorithm. Moreover, there is a tradeoff between the granularity of metrics per resource and the accuracy of the DOG algorithm. Picking a coarser value for g that is aligned with the computational time of the DOG algorithm can remarkably accelerate the computing time rather than a finer value which only leads a future bloatware that will need to be removed from Grid. Table 1 shows the relationship between groups of DLA and resource types.

5.4 Designing the objective function for discrete particle swarm optimization

In DOG, a population of particles is allocated to each metric of computational resources. Also, each particle is supplied with a DLA that acts as the brain of particle. Each hop of DOG is equal to an action of learning automaton. The dimensionality of particles is equal to the TTL value. However, if the query satisfied before the TTL value reaches; the rest of dimensions (or actions) will not take place in calculating reinforcement signal.

One of the most challenging issues of optimization engineering is the degree of concern for optimization parameters. In population-based algorithms, the eligibility of populations is evaluated through a fitness function. In a Grid-enabled environment, the hit-count (Hit) and hop-count (Hop) are two significant performance parameters. If the query is successfully processed the hit-count will set to one (Hit = 1), otherwise it will be set to zero (Hit = 0). Generally, the hit-count is a member of {0, 1}. Also, if the query is successfully processed in the initiator VO, the hop-count will be set to zero (Hop = 0), otherwise it will be set to the number of hops between the initiator VO and the terminator VO. Generally, the hop-count is a member of {0, …, TTL}.

In designing the objective function, the hit-count and hop-count are the first and the second priority parameters, respectively. So, a particle or route with lower hop-count will gain higher fitness.

$$f(x_i) = \mathrm{Hit}_i \times \left(\frac{\mathrm{Hop}_i}{\mathrm{TTL}}\right)^{\mathrm{TTL}} + (1 - \mathrm{Hit}_i) \times \left(1 + \left(\frac{\mathrm{Hop}_i}{\mathrm{TTL}}\right)\right) \tag{11}$$

In (11) x_i, f(x_i), Hit$_i$ and Hop$_i$ represent the position, fitness, hit-count and hop-count of ith particle, respectively. The objective function can be considered as the minimization of the hit-count and hop-count parameters. If the particle finds the requested resource (Hit = 1), the fitness of particles is a member of [0, 1]. A particle with lower hop-count will gain better fitness than a particle with higher hop-count. Also, if the particle could not find the requested resource (Hit = 0), the fitness of particles is a member of (1, 2].

5.5 Defining the reinforcement signal for distributed learning automata

The reinforcement signal is modeled from convergence characteristics of Discrete PSO population. By receiving an incoming query, the request actuator unit of VO enables its corresponding learning automaton. The learning automaton chooses its action based on its probability vector and initializes the particle's position in the current dimension (TTL). The query is forwarded till it matches to a proper resource or maximum TTL value reaches. Then, the fitness of the particle is evaluated through (6). Now it's time to construct a reinforcement signal which mediated the behavioral output of a particle to the actuator output of a DLA.

The position and best personal position of ith particle are $x_i = (x_i^1, x_i^2, \ldots, x_i^{\mathrm{TTL}})$ and $\mathrm{pbest}_i = (\mathrm{pbest}_i^1, \mathrm{pbest}_i^2, \ldots, \mathrm{pbest}_i^{TTL})$, respectively. Also, $\mathrm{gbest} = (\mathrm{gbest}^1, \mathrm{gbest}^2, \ldots, \mathrm{gbest}^{\mathrm{TTL}})$ is the global best position of population. The effective reinforcement signal of DLA is calculated from x_i, pbest$_i$ and gbest$_i$ positions. In this case, each DLA isomorph to Grid topology needs multiple reinforcement signals for its activated learning automata. So, the reinforcement signal of jth learning automata (A_j) is calculated through (12):

$$\beta_i^j = \begin{cases} 0 & \text{if } x_i^j = \mathrm{pbest}_i^j = \mathrm{gbest}^j \\ 1 & \text{Otherwise} \end{cases} \tag{12}$$

In (12), the reinforcement signal of ith particle is emitted to ith DLA in form of $\beta_i = (\beta_i^1, \beta_i^2, \ldots, \beta_i^{\mathrm{TTL}})$. If in jth hop of the ith route that establishes from ith DLA, the position and personal previous position of ith particle and the global best position of population are all the same, the reinforcement signal will be positive otherwise it will be negative. The reinforcement signal will deploy on the probability vectors of activated learning automata of ith DLA. These activated LA are that which forward the query along ith route and constitute it.

5.6 Modeling the information exchange pattern between discrete particle swarm optimization and distributed learning automata

Figure 6 shows the semantic representation of a group of DLA interacting with a PSO population. DLA determines the position of particles and reinforcement signals reflect the particles' fitness after they fly over Grid cells. The reinforcement signals that the DLA receives encoded the success of actions' outcome of activated LA. The DLA indirectly learns from particle's fitness rather than being explicitly taught. Each particle dedicated to a resource query forms a query route based on DLA actions. There are groups of DLA associating with swarms of particle while answering different queries with different resource metrics, thereby enabling better accuracy than a single metric solution.

5.7 Dynamic route correction by distributed learning automata action refinement

In the proposed resource discovery framework each swarm of particles is devoted to a resource metric and each particle of swarm is facilitated with a DLA. In other words, a specific group of DLA is reserved for query tracking of a specific resource metric. DLA is a network of LA which is isomorphic to the Grid overlay. Each action of learning automaton is equal to forward the query to an adjacent VO. Since a learning

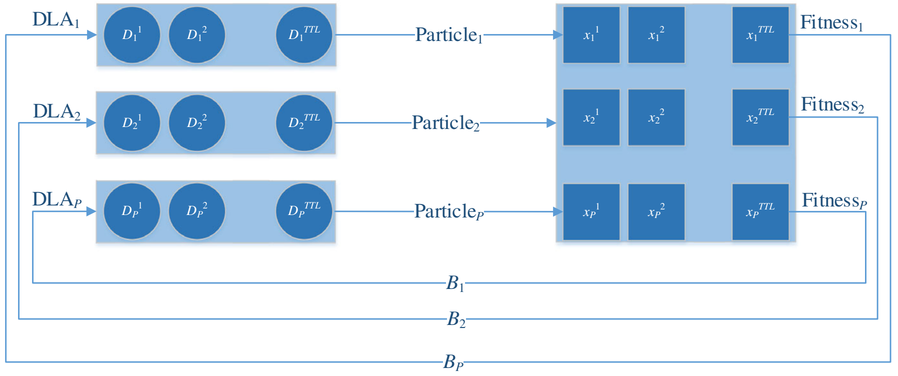

Fig. 6 DLA and PSO population interact together to answer resource queries. The reinforcement signal (B) of each DLA calculated through associated particle's fitness

automaton has fixed action set that is imprinted from neighborhood configuration of its corresponding VO, the final path that is created from DLA may contain loop. To avoid bloating the final path with duplicate VOs, utilizing learning automaton with variable number of actions [39] will be beneficial. Since the action set of learning automaton is generated form its corresponding VO's network connections, the action set of learning automaton could be manipulated through changing its configuration. This could be done through disabling actions corresponding to the visited VOs if any.

To implement this approach, when an automaton chooses an action, all other LA of DLA will disable the action corresponding to this selected action. After the query is completely processed, the reinforcement signal will be deployed on the activated LA of traversed path and all disable actions will be enabled.

5.8 Convergence analysis of distributed optimization Grid

The DOG algorithm has one main property which significantly impacts on its performance. The DOG is a *loop free* search heuristic because it uses LA with a variable action set. The mentioned property is established in Theorem 1. The following discussion demonstrates the convergence of this feature of DOG when $L_{\text{R-I}}$ learning algorithm is used for each peer.

Let π be the traversed path that is generated from the DOG algorithm based on $L_{\text{R-I}}$ learning algorithm. Let us assume that each GIS corresponds to a node of network graph and it has an automaton for request forwarding. Let $V = \{1, 2, \ldots, n\}$ be the set of network peers where n is the number of nodes. Let $E \subset V \times V$ be the set of edges and let $d \in [1 \ldots n-1]$ be the connectivity degree of each peer in the graph. Lastly, let k be the current step of π, h be the current step of DOG and let K be the length of traversed path. Let α_k be the action set of A_k, P_k be the associated probability vector of A_k, $\bar{\alpha}$ be the set of disabled actions and $\bar{P}$ be the associated probability vector of disabled actions.

Theorem 1 *DOG with $L_{\text{R-I}}$ learning algorithm is a loop free algorithm.*

Proof If automaton A_2 selects action α_2^i with the probability of $p_2^i | p_2^i \geq 0$, $i \in [1, \ldots, l]$, then $\bar{\alpha} \cap \alpha_2 = \emptyset$ because activating automaton A_1 has already disabled all actions associated with its selection in first step of π. Therefore, $\alpha_2^i \notin \bar{\alpha}$ and second peer of π is not duplicate. Hence, the DOG algorithm cannot result in loop in second step. □

Now, assume that the DOG algorithm does not have loop for first k steps. If automaton A_k selects action α_k^i with the probability of $p_k^i | p_k^i \geq 0$, $i \in [1, \ldots, l]$, then $\bar{\alpha} \cap \alpha_k = \emptyset$ because activating automaton A_{k-1} has already disabled all actions associated with its selection in $k-1$th step of π. Therefore, $\alpha_k^i \notin \bar{\alpha}$ and $k+1$th peer of π is not duplicate. Hence, the DOG algorithm cannot result in loop in $k+1$th step.

So, if during $k+1$th step of path π a dead-end occurs with the probability of $q_k > 0$, after taking one step backward in $h+3$th step of DOG which is corresponded to $k+1$th step of π with the probability of $q_{h+2} | q_k > q_h + 2 \geq 0$ the dead-end may occur again. Therefore, by reducing the probability of action that leads to the dead-end in kth step of π, DOG avoids the dead-end zone.

5.9 Complexity analysis of distributed optimization Grid

The DOG algorithm provides an iterative resolution for multi-attribute range queries. Also, it can be implemented on a scale-free network which uses the connectivity degree parameter $d \in [1, \dots, n-1]$. In this algorithm for each peer with $O(d)$ neighbors a query message is sent to one of these neighbors. Each peer contains t types of resources with m resource metrics. In this configuration $t * m$ swarms of particles and groups of DLA are used in DOG. Since each query message is processed using First Found First Served (FFFS) policy and an m-fold query requires m classes of DLA that each of them is equipped with a population of p particles. The query message results in cost of $O(t)$ in time and m-fold query resolution leads to the lookup cost of $O(m)$ in time. Furthermore, O(q) and $O(u)$ computation time are also needed, respectively, for q requests and u users.

By receiving an incoming query, its associated resource is specified in $O(t)$ time. Then the request is forwarded to one of its d neighbors, which leads to a lookup cost of $O(n^{1/d})$ for each step. Lastly, local resource metrics processing requires $O(m)$ time. Therefore, in a case with u users and q queries for each of the users, the DOG algorithm results in complexity of $O\left(\sum_{i=1}^{u} \sum_{j=1}^{q} (t + (n^{1/d} \times m))\right)$, where i and j, respectively, correspond to user and request.

5.10 Pseudocode of distributed optimization Grid (DOG)

5.10.1 Grid overlay

An Overlay network could be structured (Mesh topology) or unstructured (Random topology) [24]. Random topology is a near-reality choice for overlay network. A random topology implies that the out degree of nodes followed a random distribution to form a random graph. Moreover, a random graph G is defined by a $G = (V, E)$, where $V = \{1, 2, \dots, n\}$ is set of nodes, $E \subset V \times V$ is set of edges, n is the number of nodes. In random graph G, each node n is connected to other nodes with a *connectivity degree* parameter (d), in which $d \in [1 \dots n-1]$. Finally, each peer $p_i | p \in [1 \dots n]$ maintains multiple connections to $d_i | d \in [1 \dots n-1]$ neighbor peers of Grid overlay network.

5.10.2 Resource characteristics

In Grid systems, different resources are registered to various GISs. Since the resource membership of each GIS is done randomly, some nodes may contain multiple resources while others may have none. The emulated Grid environment supports five types of resources including: processor, memory, storage, operating system and networking. The distribution of each resource type is shown in Table 2. These resources draw from Poisson distribution with lambda equal to seven. The Poisson distribution applies to various phenomena of discrete properties whenever the probability of the phenomenon happening is constant in time. Also, according to the Moore's law, the number of transistors in a dense integrated circuit double approximately every 2 years. In conclu-

Table 2 Resource distribution of all supported resource type

Resource type	Distribution	Attributes	Lambda	Unit
Processor	Poisson	Size	7	GFLOPS
Memory	Poisson	Size	7	Gigabyte (GB)
Storage	Poisson	Size	7	Terabyte (TB)
Operating system	Poisson	Type	7	N/A
Networking	Poisson	Size	7	Megabyte (MB)

sion, using the Poisson distribution for modeling the computing resources (especially processors) in a specific period of time establishes a real-world choice.

The lambda parameter controls different dimensions of different resource types of Table 2. This parameter contributes to the processor through specifying the Floating-point Operations Per Second (FLOPS). FLOPS could be calculated using (13):

$$\text{FLOPS} = \text{cores} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}} \tag{13}$$

As λ grows larger the Poisson distribution can be modeled by a normal distribution with $\mu = \lambda$ and $\sigma^2 = \lambda$. Although this approximation is used to calculate confidence interval for $\lambda > 100$, here we used it as a rule of thumb for calculating the Poisson bounds using (14):

$$\begin{aligned} &\text{Poisson}(\lambda = 7) \overset{\text{for}\, \lambda > 100}{\longrightarrow} \text{Normal}(\lambda, \lambda) \\ &\text{Poisson Bounds } : 0 < \lambda \leq 2\lambda \end{aligned} \tag{14}$$

Based on (14), the upper and lower bounds of Poisson distribution with $\lambda = 7$ are [1, 14]. These bounds confirm the formulation of (13) with the following example: Consider a single-core 2.0 GHz processor with 4 FLOPS per clock cycle. From (13), this processor has a peak performance of 8 GFlOPS (2*1*4).

Also, lambda parameter contributes to memory, storage, operating system, and networking through specifying the memory size, storage size, operating system type, and networking transfer speed.

5.10.3 Pseudocode of distributed optimization Grid resource discovery

Algorithm 3 shows the distributed optimization Grid (DOG) resource discovery algorithm for Grid environments.

5.11 Out of the box features of distributed optimization Grid resource discovery

The distributed optimization Grid (DOG) is designed around 'MDS' [1] principals. The DOG algorithm aims to launch a solution that acts as a part of Grid resource monitoring and provisioning. The DOG algorithm gives the agility, scalability and automation to the process of resource discovery that complex heterogeneous Grids need to be addressed. The following are the key characteristics of DOG:

Algorithm 3 Distributed Optimization Grid (DOG) Resource Discovery

```
define
Construct a graph $G = (V, E)$ where $V$ is a set of Grid Information Services (GISs) and $E$ is a set of network links
Consider start node $v_s$, end node $v_e$, hop-count $k = 0$ and maximum hop-count $K$ = TTL
Consider $r$ computing resources with $t$ types that constitute from $m$ metrics
Initialize $r * m$ swarms of Discrete PSO: $X_{r,m} = (P, D)$ where $P$ is the population size, $D$ is the dimension number and
$X_{r,m} = \{x_1, x_2, ..., x_D\}$ is the population of $m_{\text{th}}$ metric of $r_{\text{th}}$ computational resource
Initialize pbest position of each particle and gbest position of each swarm
Consider fitness evaluation $F$ and reinforcement signal $\beta$
Construct $r * m * p$ groups of DLA from graph $G$

begin
  for each user query $i$ do
    Select the corresponding PSO population and group of DLA based on query$_i$ resource metrics
    for each particle $j \in [1,PS]$ do
      Set $k = 0$
      Disable action $\alpha_s$ from all inactivated automata
      let $v_u$ be source node $v_s$
      while (resource in not find AND the number of traversed edges is less than $|V|$ AND $k$ is lower than $K$) do
        if ($A_u$ associated action set $\alpha_u$ is not empty) then
          Automaton $A_u$ Selects an adjacent node to $v_u$ from its action set $\alpha_u$ according to its action
          probability vector $P_u$ and denote it $v_w$
          if (GIS$_u$ is enable) // Status Checker Unit
            Database Synchronization between GIS$_u$ and Resource Pool$_u$ // Resource Pool Unit
            Query GIS$_u$ Database based on FFFS policy // Grid Information Service Unit
          end if
          Disable action $\alpha_{A_u}$ corresponding to automaton $A_u$ from all inactivated automata
          Add $(v_u, v_w)$ to the traversed path and set $v_w$ to $v_u$ // Query Tracker Unit
          $X_j^k = v_w$ // Initialize particle's position in $k_{th}$ dimension
          $D_j = k{+}{+}$ // Update particle's hop-count
        end if
      end while
    end for
    for each particle $j \in [1,P]$ do // Update pbest and gbest positions
      Calculate $x_j$ fitness through (11)
      $F{+}{+}$ // Increase fitness evaluation
      if $f(x_j) < f(pbest_j)$ then
        $pbest_j = x_j$
      end if
      if $f(pbest_j) < f(gbest)$ then
        $gbest = pbest_j$
      end if
    end for
    if $gbest \in [0,1]$
      Set $gbest_{\text{end}}$ as the provider VO for Query$_i$ // Successful query
    else
      Drop Query$_i$ // Unsuccessful query
    end
    for each particle $j \in [1,P]$ do // Calculate reinforcement signal through (12)
      for each hop-count $k \in [1,D_j]$ do
        if ($gbest_j = pbest_j^k = x_j^k$)
          $\beta_j^k = 0$
        else if
          $\beta_j^k = 1$
        end if
      end for
    end for
    set $v_w$ to $v_e$ // Build traversed path
    Let $\pi = \{v_s, \ldots, v_e\}$ be the traversed path
    if ($\beta_j^k = 0$) then // Emit reinforcement signal
      Reward the selected actions of activated automata along path $\pi$
    else if ($\beta_j^k = 1$) then
      Penalized the selected actions of activated automata along path $\pi$
    end if
  end for
end
```

1. Centralized access to VOs
   - A. The Status Checker Unit placed at the edge of the VO's entrance to accelerate the resource discovery process
   - B. Agile status-aware VO by status checker unit
   - C. Flexible architecture for resource discovery across VOs
2. Resource management tool based on GIS taxonomy
   - A. Support key operations of resource indexing such as: publish, subscribe, query, quote
   - B. Coordinated sync connector between GIS and resource pool
3. Managing resource pools by GIS
   - A. Centralized infrastructure management
   - B. Control delegation of grid-datacenters to GIS
4. Fully-automated resource discovery solution
   - A. Learning-based query forwarding strategy
   - B. Lightweight probabilistic query tracker
   - C. Distributed memory-less approach
5. Optimized routing protocol for resource discovery
   - A. Multi-swarm cooperative query engine
   - B. Diverse set of suggested solutions for resource discovery
6. Bottom-up view of Grid by Grid datacenters and Grid cells elements
7. Scalable performance by processing concurrent resource queries
8. Fully Distributed architecture using DLA and discrete PSO
9. Grid-relay integration of resource management and provision units

6 Experimental results

To implement the Distributed Optimization Grid (DOG) resource discovery algorithm, the GridSim toolkit [47] is used. The GridSim is a Java-based Grid simulation toolkit. The simulator provides a set of predefined classes such as user, resource, job, etc. In the following, the key settings of simulation are discussed.

6.1 Initial topology

The Grid topology is drawn from Waxman model [30] that follows the *Waxman's probability model* for connectivity degree of nodes. As discussed in Sect. 5.3, the key parameter of a random topology is the connectivity degree parameter. This parameter indicates the outdegree and indegreee of each node. For the sake of simplicity, each Grid node is assumed as a VO. All links between VOs are full-duplex and have equal bandwidth. Also, different Grid scales are used to investigate the performance of DOG:

- small-scale Grid: 500–1,000 VOs
- medium scale Grid: 1,500–2,000 VOs
- large-scale Grid: 2,500–3,000 VOs

6.2 Resource types

The simulated Grid constitutes 500 resources ($R = 500$) from 5 different types including processor, memory, storage, operating system and networking ($t = 5$). So, the resource granularity or the number of resource instances is 100 ($g = 500/5 = 100$). Each resource type has 2 resource metrics ($m = 2$), and draw from Poisson distribution with parameter lambda equal to 7 ($\lambda = 7$).Therefore, 10 groups of DLA and swarms of particles ($t* \ m = 5 * 2 = 10$) are generated to fulfil the resource discovery task. Finally, observing the impact of resource distribution in different scales, the resource frequency is not adjusted proportional to Grid scale and remained fixed in all scales.

6.3 Resource queries

The simulated Grid hosted 30 users with the same set of 250 resource queries from 5 different types (50 queries per each resource type) and Poisson distribution with parameter lambda equal to 7. Furthermore, in each experiment the query initiator VO is selected randomly from a set of 10 VOs and the same set of request is sent to these VOs. Moreover, the TTL value of requests is set to 100 hops.

6.4 Algorithms description

The DOG algorithm is compared with other resource discovery algorithms including *random walk* [48], *best neighbor* [26] and *flooding* [49]. The description of these algorithms are as follows:

- The random walk resource discovery (RWRD) [48] that randomly forwards the resource query to another VO. This algorithm will randomly choose a peer to forward the resource request and also it does not store any additional information.
- The best neighbor resource discovery (BNRD) [26] that forwards the resource query to the most responsive VOs. This algorithm records the number of successful requests which is answered from each peer. Next time the request will be forwarded to the peer who answered the largest number of requests.
- The flooding resource discovery (FRD) [49] that forwards the resource query based on the flooding protocol. This algorithm spreads out the request from all peers of starting node and broadcasts it till finding the requested resource or meeting maximum TTL value.

6.5 Performance criteria

The following criteria are used for performance evaluation of the resource discovery algorithms:

Table 3 The parameters are used for robustness analysis

Scale	Domain	Threshold
500	250	190
1,000	250	150
1,500	250	110
2,000	250	100
2,500	250	90
3,000	250	70

The column "Domain" and "Threshold" indicate the total number of user queries and stopping criterion value, respectively

6.5.1 *Hop-count*

The percentage of hops a query is traversed with respect to TTL value. This criterion provides a time constraint for time critical queries.

$$\text{Hop} - \text{count} = \frac{\#\text{Hops}}{\#\text{TTL}} \times 100 \tag{15}$$

6.5.2 *Hit-count*

The percentage of queries which are satisfied before expiring their TTL value. A query is satisfied if GIS could suggest a proper resource for it. This criterion shows the optimal ability of algorithm for meeting high availability QoS conditions.

$$\text{Hit} - \text{count} = \frac{\#\,\text{Hits}}{\#\,\text{Queries}} \times 100 \tag{16}$$

6.5.3 *Resource utilization*

The percentage of resources which is used to service the queries. This criterion deals with the maximization of different aspects of service level agreement (SLA) constraints.

$$\text{Resource utilization} = \frac{\#\,\text{Matched Resources}}{\#\,\text{Resources}} \times 100 \tag{17}$$

6.5.4 *Robustness*

The term *robustness* indicates that the algorithm is succeeded in answering a certain number of queries based on a threshold. A *robust* algorithm manages to reach this threshold consistently. This criterion deals specifically with the negotiation of SLA. The main objective of this phenomenon ensures that mission critical applications meets certain level of QoS. Table 3 lists the parameters used for this performance metric. Threshold value for different scales is calculated through experimental equations of (18,19,20). In each scale, the threshold θ is calculated via two terms of ω_1 and ω_2

- ω_1 is the average result of all algorithms
- ω_2 is the maximum standard deviation of all results obtained from all runs.

Finally, in (20) these two terms are subtracted from each other to produce the threshold value.

$$\omega_1 = \left(\frac{1}{n} \sum_{i=1}^{n} \text{Algorithm}_i (\bar{M}_i) \right), \bar{M} = \frac{1}{m} \sum_{j=1}^{m} \text{Run}_j \tag{18}$$

$$\omega_2 = \frac{1}{l} \sum_{k=1}^{l} \text{MAX}_k \left(\text{Algorithm}_i \left(\left(\frac{1}{m} \sum_{j=1}^{m} (Run_j - \bar{M}_i)^2 \right)^{\frac{1}{2}} \right) \right), \ i \in [1, \ldots, m] \tag{19}$$

$$\theta = \omega_1 - \omega_2 \tag{20}$$

In (18) and (19), m and n are the number of algorithms and the number of independent runs, respectively. Also in (19), l denotes the number of different network scales. For having a fair comparison, ω_1 is computed in reversed order by gathering simulation results and then calculating ω_1 value. Furthermore, $\omega_1 = 10$ is obtained from the experiments and used for generating threshold values of Table 3 by (20).

6.6 Learning algorithm

In [50] different learning algorithms of LA are described in detail and their applications are introduced. The LA has three conventional learning algorithms: $L_{\text{R-P}}$, L_{RI} and $L_{\text{R-eP}}$. All mentioned learning algorithms have two learning parameters α (reward signal) and β (penalty signal) which are initialized by arbitrary values relative to the problem structure. Due to dynamicity and heterogeneity of Grid these learning parameters should be set to proper values. Moderate parameter adjusting enables the algorithm to easily alter the potential worst or non-optimal traversed paths. In DOG algorithm, the α and β parameters of $L_{\text{R-P}}$ learning algorithm are considered as $\alpha = \beta = 0.1$. This experimental configuration lets the DOG algorithm to refine DLA actions in steady state.

6.7 Particle swarm optimization properties

The PSO algorithm as an optimization heuristic contains a group of particles. The population size of PSO is a relatively critical parameter. Indeed, parameter adjusting experiments of DOG algorithm showed that selecting small population sizes deteriorate the performance of DOG algorithm while selecting large population sizes not only improves the performance of DOG algorithm slightly, but also increases the computation time. So, the population size of PSO is set to 30 as a moderate choice of simulation.

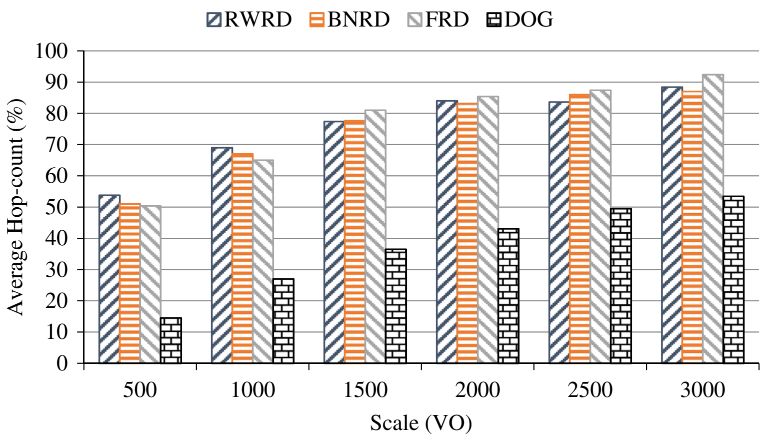

Fig. 7 Average hop-count at different scales. The hop-count is reported based on the percentage of TTL value that the algorithm is taken. Also, the scale of Grid is varied from 500 to 3,000 VOs

In DOG algorithm, the TTL value of query initialized the number of dimensions of each particle. Since the TTL value is 100, the number of dimensions of each particle is set to 100. Furthermore, since the number of user queries is 250 and the population size of PSO is 30, the maximum number of fitness evaluations is set to 750 (with reference to population size $\times$ number of user queries). As discussed before, there are 5 types of computational resource each of them has 2 different resource metrics. Therefore, DOG is facilitated by 10 swarms of PSO and 10 groups of DLA. Each swarm of PSO contains 30 particles. Each particle is equipped with an independent DLA.

6.8 Experiment 1: hop-count

The DOG algorithm follows geolocation resource discovery service paradigm that suggests resources with lowest hop-count. Figure 7 shows the hop number as a function of Grid scale. The DOG algorithm uses request tracker unit to achieve hop-count of less than half that of the counterpart algorithms, in all scales. Each VO in DOG utilizes a resource tracker unit that manages two subcomponents: DLA and PSO. DLA performs the query forwarding based on its past experience and initializes the PSO's position. Furthermore, PSO yields a set of candidate resources and offers the nearest one to the user. Traditional resource discovery algorithms like RWRD and FRD require large hop-count to resolve queries. These algorithms do not forecast the availability or unavailability of certain resources whilst BNRD partially remembers the path of formerly processed queries.

The scalability of an algorithm is a measure of its capability to effectively avoid performance degradation while increasing the number of nodes. From the scalability point of view, hop-count of DOG increases by a little margin at each Grid scale while the hop-count of other algorithms increases in larger amounts. Thus, DOG algorithm takes lowest hop-count in all scales followed by BNRD algorithm.

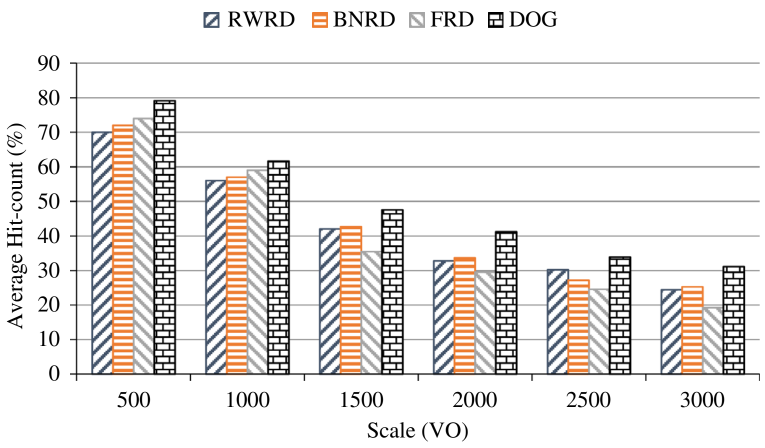

Fig. 8 Average hit-count at different scales. The hit-count is reported based on the percentage of successful queries that the algorithm is processed. Also, the scale of Grid is varied from 500 to 3,000 VOs

6.9 Experiment 2: hit-count

DOG is trying to create a competitive advantage in resource discovery by organizing the process of finding and then using resources. DOG bypasses unavailable VOs that drop the resource query and tries to suggest nearest resources. The particle responds the taken actions of DLA in turn with the fitness fluctuations. Then, the reinforcement signal approximates in terms of particle's position and experience and swarm's experience. Finally, the reinforcement signal transmits to the corresponding DLA.

Figure 8 shows the average hit-count of users' queries. Taking DOG's results into consideration, DOG dominates the hit-count measure in all Grid scales which also handles 80, 50 and 30 % of queries in small, medium and large Grid scales. FRD's hop-count, moved to 4th place from 2nd place in the medium and large-scale Grids where resource distribution is sparser than small-scale Grids. Since the TTL value affects the surf-zone of FRD, the algorithm definitely cannot find appropriate resources for some unique queries. Moreover, since each node of BNRD relays queries to those neighbors that answer the most of previous queries, in small and large-scale Grids whose resource profiles are dispersed their hit-count will be diminished.

Since the resource distribution is fixed through all Grid scales, performance of algorithms may deteriorate as a direct effect of low resource density of larger Grids. This issue may be resolved by considering the resource distribution as a function of Grids scale. Since initializing the larger Grids with denser resource distributions needs more memory capacity, further investigations about this issue are out of scope for this article.

6.10 Experiment 3: resource utilization

The DOG algorithm can recommend and provide minimum resource requirement alternatives as a decision support system for newly submitted queries by users. Granulation of resource space helps DOG to improve the resource utilization. The DOG algorithm

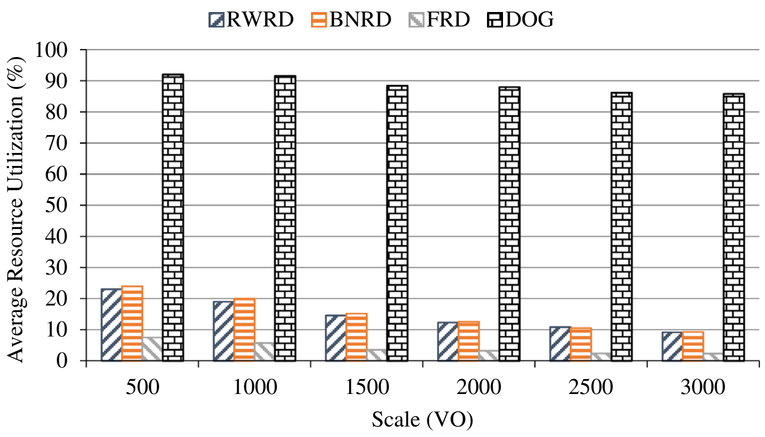

Fig. 9 Average resource utilization at different scales. The resource utilization is reported based on the percentage of utilized resources. Also, the scale of simulation is varied from 500 to 3,000 VOs

efficiently harvests the solution diversity of PSO and quality of DLA decision making toward increasing the resource utilization. Moreover, the action refinement heuristic of DLA avoids loops and increases resource utilization. Furthermore, the bottom-up structure of DOG which utilizes the concepts of Grid-datacenters and Grid cells helps the algorithm to efficiently lookup and allocate resources.

Figure 9 shows resource utilization of different Grid scales in the case of 100 resources per different types of computational resources. The DOG algorithm utilizes almost 90 % of resources in all scales. The DOG's high rate of resource utilization is due to the iterative advantage of PSO on optimizing the requested query. Although this level of utilization sounds odd, but since multiple swarms of particles are spread across the Grid, different resources may have chance to give service to the users. Also, since DOG executed in parallel mode, the algorithm's performance will not deteriorate during resource discovery process.

Also from Fig. 9, the FRD algorithm has the least resource utilization because it can recommend an only fixed set of resources for all resource queries. The resource utilization of RWRD and BNRD algorithms is approximately the same. By increasing the Grid scale, BNRD algorithm cannot record a rich history about VOs neighborhood and may converge to the utilization pattern of RWRD.

Combining DLA and PSO leads to a comprehensive resource discovery strategy with high rates of resource utilization in all Grid scales. These great results are due to the following factors:

1. The DLA learning strategy could easily adopt to the different changing factors of Grid such as scale, heterogeneity, and peer distribution. Thus the DOG algorithm consolidates resource utilization in a scale free infrastructure,
2. The PSO population avoids overloading the DOG algorithm in particle and population tires. In particle tire, the particles that are associated with less efficient resources have the chance to foster better resources by utilizing the population global best solution. Also in population tire, the information exchanging that occurs

between different particles by deployment of DLA's reinforcement signal leads to extend the capacity of resource utilization.

3. The DOG algorithm consists of a population of particles for each resource metric. The particles are executed in parallel mode that removes the bottleneck of traditional serial resource discovery algorithms and leverages the Grid resources more effectively. This type of execution concurrently checks different resources and finally proposes the optimal resource for further.
4. From the starting queries to the final ones, the DOG algorithm progressively tries to learn the request pattern of users by utilizing the DLA. Since the optimality of the proposed resource by DOG is related to the ability of DLA to locate it, the DOG algorithm proposes different set of resources from the initial steps to the last steps of simulation. By investigating this phenomenon, one may see that the DOG algorithm utilizes a diverse set of Grid resources.
5. The DOG algorithm launches multiple particles for each query depending on its associated metric. This configuration deals with both spatial and temporal dimension of resource discovery process. By resolving similar queries for the same metrics, DOG achieves a spatial perception for proposing common resources. Also, by learning request pattern of different users, DOG draws a temporal perception for responding singular resources.

6.11 Experiment 4: robustness analysis

There is a paradigm in Grid and Cloud computing that is named as Service Level Agreement (SLA). The SLA is a negotiated agreement between customer and service provider. An SLA contract encompasses QoS parameters such as reliability, delay, throughput, etc. Reliability is one of the key features of fault tolerance Grid. To perform reliability evaluation of resource discovery algorithms, a robustness analysis is conducted in this experiment. A robust algorithm is one that reaches a specified threshold consistently. Table 3 lists the parameters used for robustness analysis.

The robustness analysis of four resource discovery algorithms in six different scales is shown in Table 4. The DOG algorithm successfully manages to reach the threshold in all of six scales. The algorithm utilizes a DLA and PSO for each sub-range of each resource. This approach improves the robustness compared to BNRD and other search heuristics which do not support attribute-based resource query. Also, DOG with changing number of actions generates a loop free path. This unique characteristic of DLA lets DOG to efficiently search for requested resource while preventing to visit duplicate peers. Since DOG has successfully accomplished to cross the thresholds in all scales, the algorithm offers high reliability as a valuable QoS parameter of Grid.

In small-scale Grids, FRD algorithm is more robust than medium and large-scale Grids. By growing the Grid size, the distribution of resources is more dispersed. Therefore, FRD algorithm could not find the proper resources till the maximum TTL is reached. BNRD algorithm is more robust in medium scale Grids and RWRD algorithm is the least robust one. Finally, from the point of QoS constraints, Table 4 further shows

Table 4 Robustness analysis of different scales

Scale	500		1,000		1,500	
Function	Succeeded	Succ. Reqs.	Succeeded	Succ. Reqs.	Succeeded	Succ. Reqs.
RWRD	0	N/A	6	157	24	103
BNRD	6	182	24	153	30	104
FRD	24	192	25	155	0	N/A
DOG	30	201	25	161	30	119
Scale	2,000		2,500		3,000	
Function	Succeeded	Succ. Reqs.	Succeeded	Succ. Reqs.	Succeeded	Succ. Reqs.
RWRD	0	N/A	5	80	19	61
BNRD	7	94	0	N/A	23	65
FRD	0	N/A	6	82	0	N/A
DOG	30	103	25	96	30	79

The column *succeeded* lists the number of users (out of 30) that gets the service for more queries than threshold. The column *successful requests* (*succ. reqs.*) presents the number of resource requests needed on average to reach the threshold

that the DOG algorithm has the most satisfied users and could service all users without violating the SLA terms.

7 Conclusion

Placed at the heart of Grid systems, DOG resource discovery algorithm follows a concrete workflow that controls large pools of computational resources through a bottom-up structure. DOG offers on-demand computational resources, by monitoring (status checker unit), collecting (resource pool unit), provisioning (request tracker unit) and managing (Grid Information Service (GIS) unit) Grid sites. Based on Sect. 5, the description of four-component resource discovery model that exists in each VO of Grid is as follows:

1. Status checker unit: The status checker unit acts as a load balancer and is responsible for distributing incoming queries. For example, when a VO becomes unavailable, the status checker simply directs all new resource queries to request tracker unit for further processing. The status checker unit also improves VO's utilization and maximizes availability.
2. Resource pool unit: The resource pool unit is a set of computational resources that joins to a specific VO. The resource pool mediates the GIS unit to access Grid infrastructure information. The resource pool unit provides assignment information, cost models and availability for resources.
3. GIS unit: The GIS unit provides a high level of abstraction about computational resources. The GIS unit is a hot plug data collector responsible for gathering computational resource metrics. If the GIS unit becomes unavailable the

status checker unit transmits the incoming queries to the resource tracker unit. The GIS unit provides resource management and allocation across different Grid sites.

4. Request tracker unit: The request tracker unit forwards an unsupported query to another VO. The resource tracker automates the discovery of Grid-based resources by means of DLA and optimizes the resource queries by discrete PSO. This unit provides Grid with an energy-efficient resource optimization approach and a proactive resource discovery scheme.

Experimental results are shown that, DOG is superior to other algorithms in terms of hop-count, hit-count, resource utilization, and QoS constraints. Combining discrete PSO and DLA to efficiently locate the requested resources is the main advantage of DOG. In the DOG algorithm, a specific DLA acts as an actuator of a specific particle to consolidate the position of particle. The DOG algorithm utilizes groups of DLA leading swarms of particles to adapt to different resource metrics. Each group of DLA learns from the reinforcement signal that is generated based on the associated PSO population. Moreover, being embedded in a Grid operating system is another advantage of DOG resource discovery.

The Grid evolves from decentralized management and monitoring of heterogeneous computational resources. As a distributed and optimized heuristic, the DOG algorithm meets various dimensions of Grid necessities. Also, it is a suitable choice for Cloud environments. The monitoring unit of DOG algorithm monitors infrastructure, platform, and resources. This unit could track various QoS metrics of any Grid and Cloud based system. Also, the resource pool of DOG hosts clusters of different resources. Handling big data in a cluster is a promising aspect of future research of the DOG algorithm. Furthermore, resource management unit of DOG algorithm handles queries within an administrative domain. The resource allocation policy of this unit could be investigated in the future works of the DOG algorithm. Finally, the request tracker unit of DOG algorithm transmits unsupported queries to the most feasible areas. The adaptive forward planning of this unit could be enhanced by other learning strategies.

References

1. Czajkowski K, Fitzgerald S, Foster I, Kesselman C (2001) Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE international symposium on high performance distributed computing, pp 181–194
2. Keung HNLC, Dyson JRD, Jarvis SA, Nudd GR (2003) Performance evaluation of a grid resource monitoring and discovery service. Softw IEE Proc 150(4):243–251
3. Aloisio G, Cafaro M, Epicoco I, Fiore S, Lezzi D, Mirto M, Mocavero S (2005) iGrid, a novel grid information service. In: Advances in grid computing-EGC. Springer, Berlin, pp 506–515
4. Mirto M, Cafaro M, Aloisio G (2013) Peer-to-peer data discovery in health centers. In: 2013 IEEE 26th international symposium on computer-based medical systems (CBMS), pp 343–348
5. Foster I, Kesselman C (1997) Globus: a metacomputing infrastructure toolkit. Int J High Perform Comput Appl 11(2):115–128
6. Li C, Li L (2012) A resource selection scheme for QoS satisfaction and load balancing in ad hoc grid. J Supercomput 59(1):499–525
7. Ranjan R, Harwood A, Buyya R (2012) Coordinated load management in Peer-to-Peer coupled federated grid systems. J Supercomput 61(2):292–316

8. Chung W-C, Hsu C-J, Lai K-C, Li K-C, Chung Y-C (2013) Direction-aware resource discovery in large-scale distributed computing environments. J Supercomput 66(1):229–248
9. Ergu D, Kou G, Peng Y, Shi Y, Shi Y (2013) The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J Supercomput 64(3):835–848
10. Adabi S, Movaghar A, Rahmani AM, Beigy H (2013) Negotiation strategies considering market, time and behavior functions for resource allocation in computational grid. J Supercomput 66(3):1350–1389
11. Narendra KS, Thathachar M (1974) Learning Automata?: a survey. IEEE Trans Syst Man Cybern SMC (4):323–334
12. Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problems. Int J Uncertain FUZZINESS Knowl BASED Syst 14(5):591
13. Kennedy J (2010) Particle swarm optimization. In: Encyclopedia of machine learning. Springer, Berlin, pp 760–766
14. Mohamadi H, Ismail AS, Salleh S, Nodhei A (2013) Learning automata-based algorithms for finding cover sets in wireless sensor networks. J Supercomput 66(3):1533–1552
15. Piwonska A, Seredynski F, Szaban M (2013) Learning cellular automata rules for binary classification problem. J Supercomput 63(3):800–815
16. Misra S, Krishna PV, Bhiwal A, Chawla AS, Wolfinger BE, Lee C (2012) A learning automata-based fault-tolerant routing algorithm for mobile ad hoc networks. J Supercomput 62(1):4–23
17. Mozafari M, Alizadeh R (2013) A cellular learning automata model of investment behavior in the stock market. Neurocomputing 122:470–479
18. Hasanzadeh M, Meybodi MR (2013) Grid resource discovery based on distributed learning automata. Computing 96(9):909–922
19. Ghanbari S, Meybodi MR (2005) On-line mapping algorithms in highly heterogeneous computational grids: a learning automata approach. In: International conference on information and knowledge technology (IKT'05), vol 67
20. Pooranian Z, Shojafar M, Abawajy JH, Abraham A (2013) An efficient meta-heuristic algorithm for grid computing. J Comb Optim 1–22. doi:10.1007/s10878-013-9644-6
21. Huang D, Yuan Y, Zhang L, Zhao K (2009) Research on tasks scheduling algorithms for dynamic and uncertain computing grid based on a+ bi connection number of SPA. J Softw 4(10):1102–1109
22. Kashyap R, Vidyarthi DP (2013) Security driven scheduling model for computational grid using NSGA-II. J Grid Comput 11(4):721–734
23. Jacob B, I. B. M. C. I. T. S. Organization, S. B. O. (Firme) (2005) Introduction to grid computing. IBM, International Technical Support Organization
24. Trunfio P, Talia D, Papadakis H, Fragopoulou P, Mordacchini M, Pennanen M, Popov K, Vlassov V, Haridi S (2007) Peer-to-Peer resource discovery in Grids: models and systems. Future Gener Comput Syst 23(7):864–878
25. Iamnitchi A, Foster I (2001) On fully decentralized resource discovery in grid environments. In: Lee C (ed) Grid computing—GRID 2001, vol 2242. Springer, Berlin, pp 51–62
26. Iamnitchi A, Foster I, Nurmi DC (2003) A peer-to-peer approach to resource location in grid environments. In: International series in operations research and management science, pp 413–430
27. Tangpongprasit S, Katagiri T, Kise K, Honda H, Yuba T (2005) A time-to-live based reservation algorithm on fully decentralized resource discovery in Grid computing. Parallel Comput 31(6):529–543
28. Noghabi HB, Ismail AS, Ahmed AA, Khodaei M (2012) Optimized query forwarding for resource discovery in unstructured peer-to-peer grids. Cybern Syst 43(8):687–703
29. Campos J, Esteva M, López-Sánchez M, Morales J, Salamó M (2011) Organisational adaptation of multi-agent systems in a peer-to-peer scenario. Computing 91(2):169–215
30. Deng Y, Wang F, Ciura A (2009) Ant colony optimization inspired resource discovery in P2P Grid systems. J Supercomput 49(1):4–21
31. Brocco A, Malatras A, Hirsbrunner B (2010) Enabling efficient information discovery in a self-structured grid. Future Gener Comput Syst 26(6):838–846
32. Beverly Yang B, Garcia-Molina H (2003) Designing a super-peer network. In: Proceedings of the 19th international conference on data engineering, pp 49–60
33. Mastroianni C, Talia D, Verta O (2008) Designing an information system for Grids: comparing hierarchical, decentralized P2P and super-peer models. Parallel Comput 34(10):593–611
34. Hasanzadeh M, Meybodi MR, Ebadzadeh MM (2013) Adaptive cooperative particle swarm optimizer. Appl Intell 39(2):397–420

35. Hashemi AB, Meybodi MR (2011) A note on the learning automata based algorithms for adaptive parameter selection in PSO. Appl Soft Comput 11(1):689–705
36. Vafashoar R, Meybodi MR, Momeni Azandaryani AH (2011) CLA-DE: a hybrid model based on cellular learning automata for numerical optimization. Appl Intell 36(3):735–748
37. Esnaashari M, Meybodi MR (2013) Deployment of a mobile wireless sensor network with k-coverage constraint: a cellular learning automata approach. Wirel Netw 19(5):945–968
38. Shojafar M, Pooranian Z, Meybodi MR, Singhal M (2013) ALATO: an efficient intelligent algorithm for time optimization in an economic grid based on adaptive stochastic Petri net. J Intell Manuf 1–18. doi:10.1007/s10845-013-0824-0
39. Thathachar M (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern 17(6):1095–1100
40. Akbari Torkestani J, Meybodi MR (2010) An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata. Comput Netw 54(5):826–843
41. Forsati R, Meybodi MR (2010) Effective page recommendation algorithms based on distributed learning automata and weighted association rules. Expert Syst Appl 37(2):1316–1330
42. Sharma B, Thulasiram RK, Thulasiraman P (2013) Normalized particle swarm optimization for complex chooser option pricing on graphics processing unit. J Supercomput 66(1):170–192
43. Toumi L, Moussaoui A, Ugur A (2014) Particle swarm optimization for bitmap join indexes selection problem in data warehouses. J Supercomput 68(2):672–708
44. Garg R, Singh AK (2013) Multi-objective workflow grid scheduling using ε-fuzzy dominance sort based discrete particle swarm optimization. J Supercomput 68(2):709–732
45. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation, vol 5, pp 4104–4108
46. Rastegar R, Meybodi MR, Badie K (2004) A new discrete binary particle swarm optimization based on learning automata. In: Proceedings of the 2004 international conference on machine learning and applications, pp 456–462
47. Buyya R, Murshed M (2002) Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurr Comput Pract Exp 14(13–15):1175–1220
48. Jeanvoine E, Morin C (2008) RW-OGS: an optimized randomwalk protocol for resource discovery in large scale dynamic Grids. In: Proceedings of the 2008 9th IEEE/ACM international conference on grid computing, Washington, DC, USA, pp 168–175
49. Dimakopoulos VV, Pitoura E (2006) On the performance of flooding-based resource discovery. IEEE Trans Parallel Distrib Syst 17(11):1242–1252
50. Oommen BJ (2010) Recent advances in learning Automata systems. In: 2010 2nd international conference on computer engineering and technology (ICCET), vol 1. pp V1–724