Information Sciences 192 (2012) 228–243

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments

Hongbo Liu [a,b,c,*], Ajith Abraham [c,d], Václav Snášel [c,d], Seán McLoone [e]

[a] School of Information, Dalian Maritime University, 116026 Dalian, China
[b] School of Computer, Dalian University of Technology, 116023 Dalian, China
[c] Machine Intelligence Research Labs, Auburn, WA 98071, USA
[d] Department of Computer Science, VŠB-Technical University of Ostrava,708 33 Ostrava-Poruba, Czech Republic
[e] Department of Electronic Engineering, National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland

ARTICLE INFO

Article history:
Available online 9 January 2012

Keywords:
Swarm intelligence
Particle swarm
Scheduling problem
Work-flow
Security constraints
Distributed data-intensive computing environments

ABSTRACT

The scheduling problem in distributed data-intensive computing environments has become an active research topic due to the tremendous growth in grid and cloud computing environments. As an innovative distributed intelligent paradigm, swarm intelligence provides a novel approach to solving these potentially intractable problems. In this paper, we formulate the scheduling problem for work-flow applications with security constraints in distributed data-intensive computing environments and present a novel security constraint model. Several meta-heuristic adaptations to the particle swarm optimization algorithm are introduced to deal with the formulation of efficient schedules. A variable neighborhood particle swarm optimization algorithm is compared with a multi-start particle swarm optimization and multi-start genetic algorithm. Experimental results illustrate that population based meta-heuristics approaches usually provide a good balance between global exploration and local exploitation and their feasibility and effectiveness for scheduling work-flow applications.

1. Introduction

Advances in high performance computing, computational grid and cloud computing platforms are enabling researchers to explore increasingly computationally complex problems in domains such as chemistry, meteorology, high-energy physics, astronomy, biology, human brain planning and sensor networks [31,24,62,5,23]. A key consideration when developing applications for these platforms is how to schedule application tasks in order to optimize performance with respect to the available resources. However, scheduling in distributed data-intensive computing environments differs substantially from conventional scheduling. Jobs and resources in data-intensive applications have to meet specific requirements including process flow, data access/transfer, security constraints, completion cost, flexibility and availability, adding greatly to the complexity of the scheduling problem. In addition, all the components in an application can interact with each other directly or indirectly. Scheduling algorithms in traditional computing paradigms rarely consider the data transfer problem when mapping computational tasks, but such an omission can be very costly in the case of distributed data-intensive applications [23,14].

The particle swarm paradigm [27,11,30] is inspired by the social behavior patterns of organisms that live and interact within large groups. In particular, it incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms

* Corresponding author at: School of Information, Dalian Maritime University, 116026 Dalian, China.
E-mail addresses: lhb@dlut.edu.cn (H. Liu), ajith.abraham@ieee.org (A. Abraham), vaclav.snasel@vsb.cz (V. Snášel), sean.mcloone@eeng.nuim.ie (S. McLoone).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.12.032

of bees, and even human social behavior. It can be easily implemented and applied to solve various function optimization problems and, by extension, to problems that can be transformed into function optimization problems. As an algorithm its main strength is its fast convergence, which compares favorably with many other global optimization algorithms [15,6,1,13,34,2,3,46].

In this paper, several meta-heuristics adaptations to the particle swarm optimization algorithm are investigated for scheduling work-flow on distributed data-intensive computing environments. In particular, a novel variable neighborhood search strategy is introduced that helps prevent particle swarms getting trapped in local minima.

The remainder of the paper is organized as follows. Related research on scheduling algorithms in distributed data-intensive computing environments is presented in Section 2. The scheduling problem and security constraint model are formulated in Section 3. Section 4 then introduces the particle swarm based schedule optimization algorithm and a number of extensions targeted at enhancing performance. Experimental results and discussions are provided in Section 5 and finally in Section 6 conclusions are presented.

2. Related work

The job scheduling problem in distributed computing systems has been drawing researchers' attention worldwide, not only because of its practical and theoretical importance, but also because of its complexity. It is an NP-hard optimization problem [7,38], which means that the amount of computation required to find optimum solutions increases exponentially with problem size. Different approaches have been proposed to solve this problem. Ranganathan and Foster [40] defined a general and extensible scheduling framework within which a wide variety of scheduling algorithms are instantiated. Khoo et al. [29] proposed a distributed resource-scheduling algorithm capable of handling multiple resource requirements for jobs that arrive in a grid computing environment. Yu and Marinescu [58] proposed a set of divisible load scheduling algorithms for different data staging strategies. Their algorithm generates the optimal mapping from process groups to parallel systems, computes the data allocated to each system, and guarantees the shortest makespan. Venugopal and Buyya [52] considered the problem of scheduling an application composed of a set of independent tasks, each of which requires multiple data sets that are each replicated on multiple resources. They break this problem into two parts: (1) to match each task (or job) to one resource for executing the job and one storage resource for accessing each data set required by the job; and (2) to assign the set of tasks to the selected resources. They extend the MinMin and Sufferage algorithms to schedule the set of distributed data-intensive tasks. Roshanaei et al. [41] proposed a variable neighborhood search algorithm for the problem which has also proven to be effective.

Zhang and Wu [60] proposed a hybrid simulated annealing algorithm based on the immune mechanism for the job shop scheduling problem. The hybrid optimization algorithm is subsequently tested on a number of job shop instances. Computational results for different-sized instances show that the proposed hybrid algorithm performs effectively and converges quickly to satisfactory solutions. Chung et al. [10] proposed a modified genetic algorithm approach to deal with distributed scheduling models with maintenance consideration, aiming to minimize the makespan of the jobs. They also tested the influence of the relationship between maintenance repair time and machine age on the performance of scheduling of maintenance during distributed scheduling in the studied models. Wen et al. [53] investigate a heuristic-based hybrid genetic-variable neighborhood search algorithm for the minimization of makespan in the heterogeneous multiprocessor scheduling problem. The empirical results on benchmark task graphs of several well-known parallel applications show that the hybrid algorithm significantly outperforms several related algorithms in terms of the schedule quality.

Work-flows play an important role in distributed computing environments. For the job scheduling problem, the work-flow constraints need to be considered. Tao et al. [47] presented a grid work-flow scheduling based on reliability cost. The performance evaluation results demonstrate that the approach improves the dependability of work-flow execution and success ratio of tasks with low reliability cost.

With the development of large scale distributed data-intensive computing systems, issues of reliability and security have increasingly become the focus of attention. Song et al. [45] proposed a space–time genetic algorithm for trusted job scheduling. According to their security model, a job can possibly fail if the site security level is lower than the job security demand. They further proposed six risk-resilient scheduling algorithms to assure secure grid job execution under different risky conditions. These risk-resilient job scheduling schemes can upgrade grid performance significantly at only a moderate increase in extra resources or scheduling delays in a risky grid computing environment [43]. Xie and Qin [55] proposed a security-aware real-time heuristic strategy for clusters, which integrates security requirements into the scheduling for real-time applications on clusters. Experimental results show that their approach significantly improves security. They also introduced the concept of security heterogeneity for their scheduling model in the context of distributed systems. Based on the concept, they proposed a heuristic scheduling algorithm, which strives to maximize the probability that all tasks are executed without any risk of being attacked [56]. Wu and Sun [54] proposed a genetic algorithm for job scheduling to address the problem of heterogeneity of fault-tolerance mechanisms in a computational grid. They assume that the system supports four different fault-tolerance mechanisms, including job retry, job migration without checkpointing, job migration with checkpointing and job replication mechanisms. The risky nature of the grid environment is taken into account in the algorithm.

Recently, swarm intelligence and multi-agent techniques have attracted the attention of parallel computing researchers. Liu et al. [36] presented a complete multi-agent framework for dynamic job shop scheduling considering robustness and

adaptability and provided experimental justification for their approach using computational experiments on dynamic job arrivals. Chang et al. [8] proposed a balanced ant colony optimization algorithm for job scheduling in the grid environment. The approach is used to balance the entire system load while trying to minimize the makespan of a given set of jobs. They demonstrate the superiority of their approach to other job scheduling algorithms using experimental results.

In this paper several meta-heuristics inspired by the particle swarm optimization algorithm are introduced to deal with the scheduling problem and their effectiveness is illustrated using a number of case studies.

3. System architecture

Before we formulate the scheduling problem for work-flow applications in distributed data-intensive computing environments, the environment components and states need to be introduced. The security constraint model is also presented for the job scheduling problem. Lastly, performance metrics are also discussed.

3.1. Environment description

Scheduling problems have been an active research topic with the result that many terminologies have been suggested. Unfortunately, some of these technical terms are neither clearly stated nor consistently used in the literature, which can lead to confusion. Consequently, for clarity, the key terms employed in formulating the scheduling problem considered in this paper are now defined.

- Machine (computing unit): Machine (computing unit) is a set of computational resources with limited capacity. It may be a simple personal machine, a workstation, a super-computer, or a cluster of workstations. The computational capacity of the machine depends on the number of CPUs, amount of memory, basic storage space and other specializations. In other words, each machine has its processing speed, which is normally expressed as the number of Cycles Per Unit Time (CPUT). If there are m available machines in the computing environment, they are denoted as $\{M_1, M_2, \ldots, M_m\}$. Each machine, M_i ($i = 1, \ldots, m$), can offer its computing service with a certain level of reliability referred to as its security rank sr_{M_i}.
- Data resource: Data resources are the datasets which effect the scheduling. They are commonly located on various storage repositories or data hosts. Data resources are connected to the computational resources (machines) by links of different bandwidths. Fig. 1 shows a simplified data-intensive computing environment consisting of four computing units and four data hosts. The numbers on the connecting links indicate the corresponding connection bandwidths. If there are k available data resource sites in the distributed environment, they can be denoted as $\{D_1, D_2, \ldots, D_k\}$. D_i ($i = 1, \ldots, k$) can offer the data service with security rank sr_{D_i}.
- Job and operation: A job is considered as a single set of multiple atomic operations/tasks with each operation typically allocated to execute on a single machine without preemption. Each operation has input and output data and processing constraints. One of the most important constraints is the work-flow, which is the ordering of a set of operations for a specific application. Each operation can only be started after the completion of the previous operation in the sequence, which is the so-called work-flow constraint. A second constraint is the number of cycles needed to complete an operation (i.e. processing length). The other key constraints on operations are security constraints. Suppose a work-flow application comprises of q jobs $\{J_1, J_2, \ldots, J_q\}$ with the jth job J_j consisting of a set of operations $\{O_{j,1}, O_{j,2}, \ldots, O_{j,p}\}$. For convenience, we will decompose all jobs to their atomic operations and re-sort the operations as $\{O_1, O_2, \ldots, O_n\}$. The computing security demand of the operation $O_i (i = 1, \ldots, n)$ is then denoted as sd_{c,O_i} and its security demand of data service is denoted as sd_{d,O_i}.

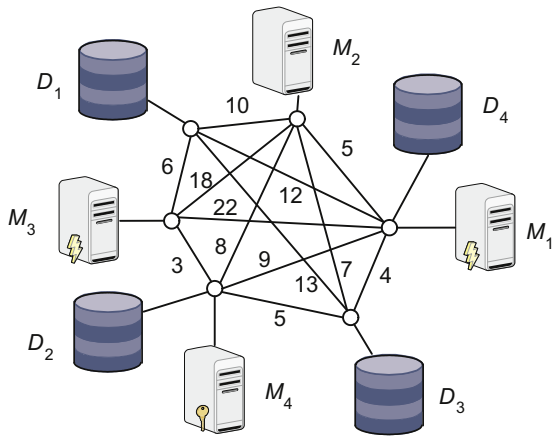

Fig. 1. A simplified data-intensive computing environment.

- Work-flow application: A work-flow application consists of a collection of interacting components that need to be executed in a certain partial order to successfully solve a certain problem. The components involve a number of dependent or independent jobs, machines, the bandwidth of the network, etc. and have specific control and data dependencies between them.
- Schedule and scheduling problem: A schedule is the mapping of the tasks to specific time intervals on specific machines. A scheduling problem is specified by a set of machines, a set of jobs/operations, optimality criteria, environmental specifications and various constraints.

3.2. Security constraint model

We present a security constraint model for data-intensive jobs running on distributed computing environments. Some related research on the security model can be found in [45,43,55,56,44,57].

There are three security modes for data-intensive job scheduling:

Secure mode: Schedule operations only on those computing units that can definitely satisfy the security requirements. A job is assigned to an available computing unit, only if the condition $sd \leqslant sr$ is met. A similar condition applies to the data service sites for the operation. Here *sd* is the security demand of the operation, and *sr* is the security rank of the computing machine or service data resource. The secure mode is considered a conservative approach to scheduling of jobs.
Risky mode: Schedule operations on any available computing unit (or using any available data resource) and thus take all possible risks. The risky mode is considered an aggressive way to schedule jobs.
γ-risky mode: Schedule operations to available computing units or data resources taking at most γ risk, where γ is a probability measure with the extremes $\gamma = 0$ and $\gamma = 1$ (i.e., 100%) corresponding to the secure and risky modes, respectively.

The secure mode is in general extremely challenging and costly to achieve, hence the risky mode and γ-risky mode are normally employed when scheduling jobs. In distributed computing environments security levels are also assessed in terms of a qualitative/fuzzy scale [44] consisting of five levels:

- Very high → 5
- High → 4
- Medium → 3
- Low → 2
- Very low → 1

A scheduled operation is considered secure if it is assigned to a completely safe machine or data service site (i.e. with $sd \leqslant sr$). If an operation is assigned to a machine or data service site with a failure risk (i.e. with $sd > sr$), the risk must be less than 50% in our security constraint model. If $0 < sd - sr \leqslant 1$ the scheduled operation will be executed, but when $1 < sd - sr \leqslant 2$ the scheduled operation will be delayed, but executed prior to the execution deadline [51]. If the deadline has passed or $2 < sd - sr \leqslant 5$, the operation cannot be completed and has to be re-scheduled. The risk probability for the security constraint model is defined in Eq. (1) and illustrated graphically in Fig. 2.

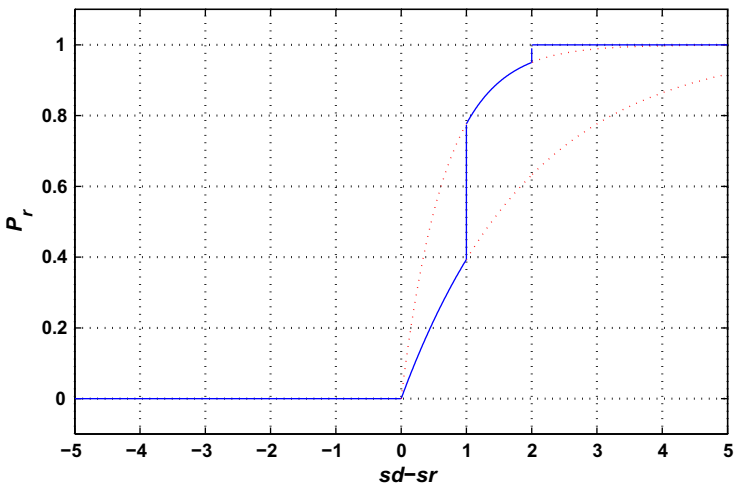

Fig. 2. Risk probability curve.

$$P(risk) = \begin{cases} 0 & \text{if } sd - sr \leqslant 0 \\ 1 - e^{-\frac{1}{2}(sd-sr)} & \text{if } 0 < sd - sr \leqslant 1 \\ 1 - e^{-\frac{3}{2}(sd-sr)} & \text{if } 1 < sd - sr \leqslant 2 \\ 1 & \text{if } 2 < sd - sr \leqslant 5 \end{cases} \quad (1)$$

3.3. Problem formulation

To formulate the scheduling problem, consider a work-flow application comprising of q Jobs $\{J_1, J_2, \ldots, J_q\}$, m machines $\{M_1, M_2, \ldots, M_m\}$ and k data hosts $\{D_1, D_2, \ldots, D_k\}$ with the processing speeds of the machines defined as $\{P_1, P_2, \ldots, P_m\}$. The jth job J_j consists of a set of operations $\{O_{j,1}, O_{j,2}, \ldots, O_{j,p}\}$. All the jobs are decomposed into atomic operations $\{O_1, O_2, \ldots, O_n\}$ with corresponding processing lengths in cycles denoted as $\{L_1, L_2, \ldots, L_n\}$. All the operations are in a specific work-flow, and will be completed in order on machines (each performing data retrieval, data input and data output) subject to the security constraint set $SC = \{sd, sr\}$. The operations in the work-flow can be represented as (or transformed to) a Directed Acyclic Graph (DAG), where each node in the DAG represents an operation and the edges denote control/data dependencies.

Definition 1. A work-flow graph for data-intensive work-flow applications can be represented as $G = (O, E)$, where the set of nodes $O = \{O_1, O_2, \ldots, O_n\}$ corresponds to the set of operations to be executed, and the set of weighted, directed edges E represents both the precedence constraints and the data transfer volumes among operations in O. An edge $(O_i, O_j) \in E$ implies that O_j can not start execution until O_i finishes and sends its result to O_j. We call operation O_i a predecessor of operation O_j and operation O_j a successor of operation O_i. Let $Pred(O_i)$ denote the set of all predecessors of operation O_i. Let $Succ(O_i)$ denote the set of all successors of operation O_i.

As mentioned above, scheduling is an NP-hard problem. Generally assumptions are made to simplify, formulate and solve scheduling problems. We also comply with the most common assumptions:

(1) A successor operation is performed immediately after its predecessor is finished (provided the machine is available).
(2) Each machine can handle only one operation at a time.
(3) Each operation can only be performed on one machine at a time.
(4) There is no interruption of operations or reworking once they have been processed successfully;
(5) Setup and transfer times are zero or have uniform duration.
(6) Tasks are independent.

The relation between operations can be represented by a flow matrix $F = [f_{i,j}]$, in which the element $f_{i,j}$ stores the weight of the edge (O_i, O_j) if the edge exists in the graph or is set to "−1" if it does not exist. Fig. 3 depicts a work-flow for nine operations. The recursive loop between O_1 and O_9 can be neglected when the scheduling is focused on the stages within the loop. The flow matrix F is thus given by:

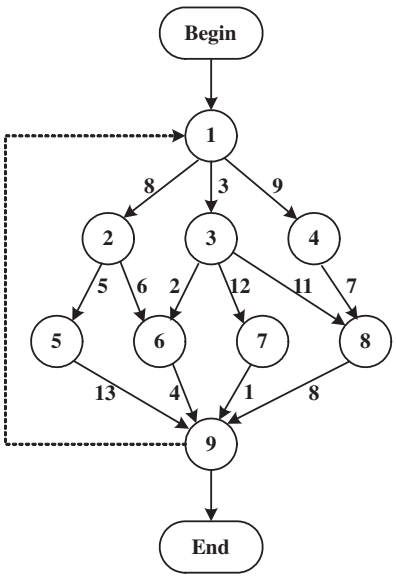

Fig. 3. A works-flow application with nine operations.

$$\begin{bmatrix} -1 & 8 & 3 & 9 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 5 & 6 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 2 & 12 & 11 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 7 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 13 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 4 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 8 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

The data host dependencies of operations are determined by the retrieval matrix $R = [r_{i,j}]$, where element $r_{i,j}$ is the retrieval time for O_i performing data retrieval from data host D_j. Throughput rates are determined by matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$, where the element $a_{i,j}$ in the former is the capacity of the link between the machine M_i and M_j, and the element $b_{i,j}$ in the latter is the capacity of the link between the machine M_i and the data host D_j. For each operation, its completion time is the sum of three components: the time needed to retrieve data, the time needed to input data and the time needed to execute the operation on the assigned machine. It is assumed that data retrieval for a given operation can be executed immediately after the completion of the previous operations in the work-flow. Given a feasible solution $S = \{S_1, S_2, \ldots, S_n\}$, where S_i is the serial number of the machine to which the operation O_i is assigned, and defining $C_{O_i} (i \in \{1, 2, \ldots, n\})$ as the completion time on machine M_{S_i} for operation O_i, C_{O_i} can be computed as:

$$C_{O_i} = \sum_{\substack{l=1 \\ f_{l,i} \neq -1}}^{n} f_{l,i} a_{S_l,S_i} + \sum_{h=1}^{k} r_{i,h} b_{S_i,h} + L_i / P_{S_i} \tag{2}$$

Denoting $\sum C_{M_i}$ as the time that the machine M_i takes to complete the processing of all the operations assigned on it, the maximum completion time (makespan) and the sum of the completion times (flowtime) of the candidate solution can be computed as $C_{max} = max\{\sum C_{M_i}\}$ and $C_{sum} = \sum_{i=1}^{m} (\sum C_{M_i})$, respectively. These two criteria are often used as performance criteria in scheduling problems. Minimizing C_{sum} seeks to reduce the average operation execution time, at the expense of the largest operation taking a long time, whereas minimizing C_{max}, ensures that no operation takes too long, at the expense of an increase in the average operation execution time and hence an increase in C_{sum}. Consequently, a weighted aggregation is often used to achieve a balance between these two metrics, that is:

$$f = w_1\{C_{max}\} + w_2\{C_{sum}\} \tag{3}$$

where w_1 and w_2 are non-negative weights and $w_1 + w_2 = 1$. The scheduling problem is thus to determine an assignment and sequence of operations on all machines/data service sites satisfying the security constraints and minimizing f. The weights in f can either be fixed or adapted dynamically during the optimization process [39]. Fixed weights, $w_1 = w_2 = 0.5$, are used in this article.

Definition 2. A scheduling problem for data-intensive work-flow applications can be defined as $\prod = (J(L), M(P), D, G, R, A, B, SC, f)$. If all the jobs are decomposed to atomic operations and the relationship between the operations are transformed to a flow matrix F, the scheduling problem can be represented as $\prod = (O(L), M(P), D, F, R, A, B, SC, f)$. The key components are operations, machines and data-hosts. For the sake of simplify, the scheduling problem can also be represented as triple $T = (O, M, D)$.

4. Particle swarm heuristics for scheduling problems

We introduce particle swarm heuristics for the scheduling problem in work-flow applications with security constraints in distributed data-intensive computing environments. The swarm models and their convergence are discussed in detail.

4.1. Canonical model

The canonical PSO model consists of a swarm of particles, which are initialized as a population of random candidate solutions. They move iteratively through the d-dimension problem space to search for new solutions with better fitness, f. Each particle has a position represented by a position-vector $\vec{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{x}_i^{\#}$, and its jth dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\vec{x}^*$, and its jth dimensional value is x_j^*. At each iteration t, the velocity update is determined by Eq. (4). The new position is then determined by the sum of the previous position and the new velocity by Eq. (5).

$$v_{ij}(t+1) = w v_{ij}(t) + c_1 r_1 \left(x_{ij}^{\#}(t) - x_{ij}(t)\right) + c_2 r_2 \left(x_j^*(t) - x_{ij}(t)\right) \tag{4}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{5}$$

Parameter w is called the inertia factor, c_1 is a positive constant called the coefficient of the self-recognition component and c_2 is a positive constant called the coefficient of the social component. r_1 and r_2 are random numbers, which are used to maintain the diversity of the population, and are uniformly distributed in the interval [0, 1]. The indexes i and j refer to the jth dimension of the ith particle.

From Eq. (4), a particle determines where to move next based on its own experience, which is the memory of its own best past position, and the experience of the most successful particle in the swarm. In the particle swarm model, particles search for solutions in the problem space within a range $[-s,s]$. (If the range is not symmetrical, it can be translated to the corresponding symmetrical range.) In order to guide the particles effectively in the search space, the maximum movement velocity during one iteration must be restricted to be within the range $[-v_{max}, v_{max}]$ and the particle positions are restricted to lie within the range of the search space, that is:

$$v_{ij} = sign(v_{ij})min(|v_{ij}|, v_{max}). \tag{6}$$

$$x_{i,j} = sign(x_{i,j})min(|x_{i,j}|, s) \tag{7}$$

The value of v_{max} is $p \times s$, with $0.1 \leqslant p \leqslant 1.0$ and is usually chosen to be s, i.e. $p = 1$. The pseudo-code for the particle swarm optimization algorithm is illustrated in Algorithm 1.

Algorithm 1. Particle swarm optimization algorithm

```
01.  Initialize the size of the particle swarm n, and other parameters.
02.  Initialize the positions and the velocities of all the particles randomly.
03.  While (the termination criterion is not met) do
04.    t = t + 1;
05.    Calculate the fitness value of each particle;
06.    x*(t) = argmin_{i=1}^{n} (f(x*(t-1)), f(x_1(t)), f(x_2(t)), ..., f(x_i(t)), ..., f(x_n(t)));
07.    For i = 1 to n
08.      x_i^#(t) = argmin_{i=1}^{n} (f(x_i^#(t-1)), f(x_i(t)));
09.      For j = 1 to dimension
10.        Update the jth dimension value of x_i and v_i
10.          according to Eqs. (4)–(7);
12.      Next j
13.    Next i
14.  End While.
```

The PSO termination criterion is usually based on one or more of the following conditions:

- Maximum number of iterations: the optimization process is terminated after a fixed number of iterations, for example, 1000 iterations.
- Number of iterations without improvement: the optimization process is terminated after some fixed number of iterations without any improvement.
- Minimum objective function error: the error between the obtained objective function value and the best fitness value is less than a pre-fixed anticipated threshold.

The role of inertia weight w in Eq. (4) is considered critical for the convergence behavior of PSO. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Its value is usually initialized around 1.0 and gradually reduced towards 0 as the algorithm progresses. A better strategy is to use adaptive approaches in which parameters are adaptively fine tuned according to the problem under consideration [42,32,59]. Cognitive parameter c_1 and social parameter c_2 in Eq. (4) are not critical for the convergence of PSO. However, proper fine-tuning may result in faster convergence and reduce the impact of local minima. Usually, 2 is selected as the default value for c_1 and c_2, but recent research suggests that better results can be obtained by choosing $c_1 \geqslant c_2$ with $c_1 + c_2 \leqslant 4$ [12,33,25,37].

The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region, which is defined by each individual's previous best success and the success of some other particle. Some previous studies have discussed the trajectory of particles and their convergence [48,50,26]. Bergh and Engelbrecht [50] reviewed theoretical studies of PSO, and extended these studies to investigate particle trajectories for general swarms to include the influence of the inertia term. They also provided a formal proof that each particle converges to a stable point. It has been shown that the trajectories of the particles oscillate as different sinusoidal waves and converge quickly, sometimes prematurely. Various methods have been proposed for identifying which particles within a swarm have an influence on the trajectory of individual particles. Eberhart and Kennedy called the two basic methods "*g*best model" and "*l*best model" [27,28]. In the *g*best model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Eventually all particles will converge to

this position. In the *l*best model, particles have information only of their own and their nearest array neighbors' best (*l*best), rather than that of the whole swarm. Namely, in Eq. (4) *g*best is replaced by *l*best in the model. The *l*best model allows each individual to be influenced by some smaller number of adjacent members of the population array. The particles selected to be in one subset of the swarm have no direct relationship to the other particles in the other neighborhoods. Typically *l*best neighborhoods comprise exactly two neighbors. When the number of neighbors increases to all but itself in the *l*best model, the case is equivalent to the *g*best model. Unfortunately there is a large computational cost associated with exploring the neighborhood relation in each iteration when the number of neighbors is low relative to the swarm size. Some previous studies has shown that the *g*best model converges quickly on problem solutions but has a tendency to become trapped in local optima, while the *l*best model converges slowly on problem solutions but is able to "flow around" local optima, as the individuals explore different regions [35,4,17,61].

4.2. Variable neighborhood particle swarm optimization algorithm

Variable Neighborhood Search (VNS) is a relatively recent metaheuristic, which relies on iteratively exploring neighborhoods of growing size to identify better local optima using so called shaking strategies [19,21,20]. More precisely, VNS escapes from the current local minimum x^* by initiating other local searches from starting points sampled from a neighborhood of x^*, which increases its size iteratively until a local minimum that is better than the current one is found. These steps are repeated until a given termination condition is met. The metaheuristic method, Variable Neighborhood Particle Swarm Optimization (VNPSO) algorithm, was originally inspired by VNS. In PSO, if a particle's velocity falls below a threshold v_c, a new velocity is assigned using Eqs. (8) and (9):

$$v_{ij}(t) = w\hat{v} + c_1 r_1\left(x_{ij}^{\#}(t-1) - x_{ij}(t-1)\right) + c_2 r_2\left(x_j^{*}(t-1) - x_{ij}(t-1)\right) \tag{8}$$

$$\hat{v} = \begin{cases} v_{ij} & \text{if } |v_{ij}| \geqslant v_c \\ u v_{max}/\eta & \text{if } |v_{ij}| < v_c \end{cases} \tag{9}$$

where u is a random number drawn from a uniform random distribution in the interval $[-1,1]$, i.e. $u \sim U(-1,1)$. The VNPSO algorithm scheme is summarized as Algorithm 2. The performance of the algorithm is directly correlated to two parameter values, v_c and η. A large v_c shortens the oscillation period, and increases the probability that particles leap over local minima for the same number of iterations. However, a large v_c compels the particles to remain in the quick "flying" state, which prevents them from refining their search and converging to a solution. The value of η directly changes the variable search neighborhoods for the particles. It should be noted that this algorithm is different from the multi-start PSO technique [49,16], where the positions and velocities of all particles in the swarm are re-initialized (re-started) if the algorithm stagnates. For comparison purposed we also implement and evaluate the Multi-Start Particle Swarm Optimization (MSPSO) (illustrated in Algorithm 3) and the Multi-Start Genetic Algorithm (MSGA).

Algorithm 2. Variable neighborhood particle swarm optimization

```
01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. Set the flag of iterations without improvement Nohope = 0.
04. While (the end criterion is not met) do
05.   t = t + 1;
06.   Calculate the fitness value of each particle;
07.   x*(t) = argmin_{i=1}^{n}(f(x*(t-1)), f(x_1(t)), f(x_2(t)), ..., f(x_i(t)), ..., f(x_n(t)));
08.   If x* is improved then Nohope = 0, else Nohope = Nohope + 1.
09.   For i = 1 to n
10.     x_i^#(t) = argmin_{i=1}^{n}(f(x_i^#(t-1)), f(x_i(t)));
11.     For j = 1 to d
12.       If Nohope < 10 then
13.         Update the jth dimension value of x_i and v_i
14.           according to Eqs. (4), (6), (5), (7);
15.       else
16.         Update the jth dimension value of x_i and v_i
17.           according to Eqs. (9), (8), (5), (7).
18.       Next j
19.   Next i
20. End While.
```

Algorithm 3. Multi-start particle swarm optimization

01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. Set the flag of iterations without improvement *Nohope* = 0.
04. While (the end criterion is not met) do
05. $t = t + 1$;
06. Calculate the fitness value of each particle;
07. $\vec{x}^*(t) = argmin_{i=1}^{n}(f(\vec{x}^*(t-1)), f(\vec{x}_1(t)), f(\vec{x}_2(t)), \ldots, f(\vec{x}_i(t)), \ldots, f(\vec{x}_n(t)))$;
08. If $\vec{x}^*$ is improved then *Nohope* = 0, else *Nohope* = *Nohope* + 1.
09. For $i = 1$ to n
10. $\vec{x}_i^{\#}(t) = argmin_{i=1}^{n}\left(f\left(\vec{x}_i^{\#}(t-1)\right), f(\vec{x}_i(t))\right)$;
11. For $j = 1$ to d
12. If *Nohope* < 10 then
13. Update the jth dimension value of $\vec{x}_i$ and $\vec{v}_i$
14. according to Eqs. (4), (6), (5), (7);
15. else
16. Re-initialize the positions and the velocities
17. for all the particles randomly.
18. Next j
19. Next i
20. End While.

To apply PSO successfully to scheduling problems, one of the key issues is mapping the problem solution to the particle space. We setup a search space of n dimensions for an (n – *Operations*, m – *Machines*) problem. Each dimension was limited to $[1, m + 1]$. For example, consider a small scale problem consisting of seven operations to be executed on three machines (7 – *Operations*, 3 – *Machines*). Fig. 4 shows a mapping between one possible assignment instance to particle position coordinates in the PSO domain for this problem. Each dimension of the particle's position maps one operation, and the value of the position indicates the machine number to which this task/operation is assigned to during the course of PSO. Since machine assignments are integer valued and particle positions are in general real valued after updating of their velocity and position of the particles, we round down the real valued outputs to their nearest integer number. In this way, we convert a continuous optimization algorithm to a scheduling problem. The particle's position is a series of priority levels of assigned machines based on the order of operations which remains fixed throughout the optimization process.

Since the position of a particle indicates a potential schedule, the position can be "decoded" to obtain a feasible solution. However, the potential schedule solution may violate the work-flow constraints. Therefore, as part of the decoding process operations are aligned by introducing delays so that the starting point of an operation occurs after the completion of the previous operation in the work-flow. The best situation is when the starting point of the operation is in alignment with the end point of the previous operation. After all the operations have been aligned, we obtain a feasible scheduling solution and then calculate the cost of the solution.

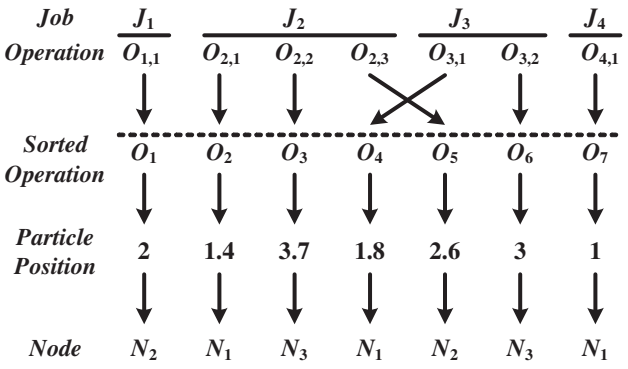

Fig. 4. The mapping between particle and scheduling problem.

4.3. Convergence analysis of the swarm models

To analyze the convergence of the algorithm we first introduce some definitions and lemmas [18,22,9], and then theoretically prove that our variable neighborhood particle swarm algorithm converges with probability 1 (or strongly) to the global optimum.

Consider the problem (P):

$$(P) = \begin{cases} minf(\vec{x}) \\ \vec{x} \in \Omega = [-s, s]^n \end{cases} \tag{10}$$

where $\vec{x} = (x_1, x_2, \ldots, x_n)^T$. Denoting $\vec{x}^*$ as the global optimal solution to the problem (P), and $f^* = f(\vec{x}^*)$, we can define

$$\begin{aligned} D_0 &= \{\vec{x} \in \Omega | f(\vec{x}) - f^* < \varepsilon\} \\ D_1 &= \Omega \setminus D_0 \end{aligned} \tag{11}$$

for every $\varepsilon > 0$.

Assume that the value of the ith dimension of the particle velocity falls below a threshold v_c and that the shaking strategy is activated. A variable neighborhood velocity is generated according to Eq. (9). In uv_{max}/η, u is a uniformly distributed random number in the interval $[-1, 1]$, and the scaling factor η is a positive constant that controls the domain of oscillation of the particle with respect to v_{max}. Therefore the variable neighborhood velocity $\hat{v}$ is uniformly distributed. If $v_{max} = s$, then $\hat{v} \sim U[-\frac{s}{\eta}, \frac{s}{\eta}]$. During the iterated procedure from the time t to $t + 1$, let q_{ij} denote that $\vec{x}(t) \in D_i$ and $\vec{x}(t+1) \in D_j$. Accordingly the positions of particles in the swarm can be assigned to one of four states: q_{00}, q_{01}, q_{10} and q_{01}. Obviously $q_{00} + q_{01} = 1, q_{10} + q_{11} = 1$.

Definition 3 (*Convergence in terms of probability*). Let ξ_n be a sequence of random variables, and ξ a random variable, all defined on the same probability space. The sequence ξ_n converges with a probability of ξ if

$$\lim_{n\to\infty} P(|\xi_n - \xi| < \varepsilon) = 1 \tag{12}$$

for every $\varepsilon > 0$.

Definition 4 (*Convergence with a probability of 1*). Let ξ_n be a sequence of random variables, and ξ a random variable, all defined on the same probability space. The sequence ξ_n converges almost surely or almost everywhere or with probability 1 or strongly to ξ if

$$P\left(\lim_{n\to\infty} \xi_n = \xi\right) = 1; \tag{13}$$

or

$$P\left(\bigcap_{n=1}^{\infty} \bigcup_{k\geqslant n} [|\xi_n - \xi| \geqslant \varepsilon]\right) = 0 \tag{14}$$

for every $\varepsilon > 0$.

Lemma 1 (*Borel–Cantelli Lemma*). *Let $\{A_k\}_{k=1}^{\infty}$ be a sequence of events occurring with a certain probability distribution, and let A be the event consisting of the occurrences of a finite number of events A_k for $k = 1, 2, \ldots$. Then*

$$P\left(\bigcap_{n=1}^{\infty} \bigcup_{k\geqslant n} A_k\right) = 0 \tag{15}$$

if

$$\sum_{n=1}^{\infty} P(A_n) < \infty; \tag{16}$$

and

$$P\left(\bigcap_{n=1}^{\infty} \bigcup_{k\geqslant n} A_k\right) = 1 \tag{17}$$

if the events are totally independent and

$$\sum_{n=1}^{\infty} P(A_n) = \infty. \tag{18}$$

Lemma 2 (*Particle state transference*). *$q_{01} = 0$; $q_{00} = 1$; $q_{11} \leqslant c \in (0,1)$ and $q_{10} \geqslant 1 - c \in (0,1)$.*

Proof. In our algorithm, the best solution is updated and saved during the complete iterated procedure. So $q_{01} = 0$ and $q_{00} = 1$.

Let $\vec{\hat{x}}$ be the position with the best fitness identified by the swarm up to and including time t, i.e. $\vec{\hat{x}} = \vec{p}^*$. In accordance with Eq. (11), $\exists r > 0$, when $\|\vec{x} - \vec{\hat{x}}\|_\infty \leqslant r$, we have $|f(\vec{x}) - f^*| < \varepsilon$. Denote $Q_{\vec{\hat{x}},r} = \{x \in \Omega | \|\vec{x} - \vec{\hat{x}}\|_\infty \leqslant r\}$. Accordingly

$$Q_{\vec{\hat{x}},r} \subset D_0 \tag{19}$$

Then,

$$P\left\{(\vec{x} + \Delta\vec{x}) \in Q_{\vec{\hat{x}},r}\right\} = \prod_{i=1}^{n} P\{|x_i + \Delta x_i - \hat{x}_i| \leqslant r\} = \prod_{i=1}^{n} P\{\hat{x}_i - x_i - r \leqslant \Delta x_i \leqslant \hat{x}_i - x_i + r\} \tag{20}$$

where x_i, Δx_i and $\hat{x}_i$ are the ith dimensional values of $\vec{x}, \Delta\vec{x}$ and $\vec{\hat{x}}$, respectively. Moreover, $\hat{v} \sim U\left[-\frac{s}{\eta}, \frac{s}{\eta}\right]$, so that

$$P((\vec{x} + \Delta\vec{x}) \in Q_{\vec{\hat{x}},r}) = \prod_{i=1}^{n} \int_{\hat{x}_i - x_i - r}^{\hat{x}_i - x_i + r} \frac{\eta}{2s} \tag{21}$$

Denote $P_1(\vec{x}) = P\{(\vec{x} + \Delta\vec{x}) \in Q_{\vec{\hat{x}},r}\}$ and $\mathbb{C}$ is the convex closure of level set for the initial particle swarm. According to Eq. (21), $0 < P_1(\vec{x}) < 1 (\vec{x} \in \mathbb{C})$. Again, since $\mathbb{C}$ is a bounded closed set, so $\exists \vec{\tilde{y}} \in \mathbb{C}$,

$$P_1(\vec{\tilde{y}}) = \min_{\vec{x} \in \mathbb{C}} P_1(\vec{x}), \quad 0 < P_1(\vec{\tilde{y}}) < 1. \tag{22}$$

Combining Eqs. (19) and (22) gives

$$q_{10} \geqslant P_1(\vec{x}) \geqslant P_1(\vec{\tilde{y}}) \tag{23}$$

Let $c = 1 - P_1(\vec{\tilde{y}})$, thus,

$$q_{11} = 1 - q_{10} \leqslant 1 - P_1(\vec{\tilde{y}}) = c \quad (0 < c < 1) \tag{24}$$

and

$$q_{10} \geqslant 1 - c \in (0,1). \qquad \square \tag{25}$$

Theorem 1. *Assume that the VNPSO algorithm provides a series of particle positions $\vec{p}_i(t)(i = 1, 2, \ldots, n)$ at time t by the iterated procedure. Let $\vec{p}^*$ is the best position among the swarm explored so far, i.e.*

$$\vec{p}^*(t) = arg \min_{1 \leqslant i \leqslant n} (f(\vec{p}^*(t-1)), \quad f(\vec{p}_i(t))) \tag{26}$$

Then,

$$P\left(\lim_{t \to \infty} f(\vec{p}^*(t)) = f^*\right) = 1 \tag{27}$$

Proof. For $\forall \varepsilon > 0$, let $p_k = P\{|f(\vec{p}^*(k)) - f^*| \geqslant \varepsilon\}$, then

$$p_k = \begin{cases} 0 & \text{if } \exists T \in \{1, 2, \ldots, k\}, \vec{p}^*(T) \in D_0 \\ \bar{p}_k & \text{if } \vec{p}^*(t) \notin D_0, t = 1, 2, \ldots, k \end{cases} \tag{28}$$

According to Lemma 2,

$$\bar{p}_k = P\{\vec{p}^*(t) \notin D_0, t = 1, 2, \ldots, k\} = q_{11}^k \leqslant c^k. \tag{29}$$

Hence,

$$\sum_{k=1}^{\infty} p_k \leqslant \sum_{k=1}^{\infty} c^k = \frac{c}{1-c} < \infty. \tag{30}$$

According to Lemma 1,

$$P\left(\bigcap_{t=1}^{\infty} \bigcup_{k \geqslant t} |f(\vec{p}^*(k)) - f^*| \geqslant \varepsilon\right) = 0 \tag{31}$$

Therefore, as defined in Definition 4, the sequence $f(\vec{p}^*(t))$ converges almost surely or almost everywhere or with probability 1 or strongly towards f^*. The theorem is proven. $\square$

5. Experimental results and algorithm performance

5.1. Experimental settings

To illustrate the effectiveness and performance of the particle swarm algorithm, three representative instances based on practical data were selected. In our experiments, the algorithms used for comparison were VNPSO, MSPSO (Multi-start PSO) and MSGA (Multi-start GA). In VNPSO, η and v_c were set to 2 and 1e−7 for the first 3/4 of the total number of iterations in each optimization run and to 5 and 1e−10 for the remaining iterations. Other specific parameter settings for the different algorithms are described in Table 1. The algorithms were run 20 times with different random seeds. Each trial had a fixed number of 2000 iterations. The average fitness values of the best solutions throughout the optimization run were recorded, as was the total computation time for the 20 trials.

5.2. Results and discussion

We begin by illustrating the algorithms on a small scale scheduling problem involving an application with nine operations, three machines and three data hosts ($O9, M3, D3$). The speeds of the three machines are 4, 3, 2 CPUT, respectively, i.e., $P = \{4,3,2\}$. The length of the nine operations are 6, 12, 16, 20, 28, 36, 42, 52 and 60 cycles, respectively, i.e., $L = \{6,12,16,20,28,36,42,52,60\}$. The flow matrix is F as given in Section 3. In terms of security constraints, $sd_{c,O_7} = 4$, $sr_{M_1} = 3$ and all other sd and sr are 2. The retrieval matrix and the machine to machine and machine to data host distance matrices for the problem are as follows:

$$R = \begin{bmatrix} 6 & 18 & 76 \\ 50 & 4 & 51 \\ 1 & 85 & 15 \\ 19 & 11 & 1 \\ 39 & 12 & 0 \\ 73 & 0 & 1 \\ 57 & 29 & 77 \\ 36 & 0 & 74 \\ 61 & 82 & 30 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 21 & 95 \\ 21 & 0 & 41 \\ 95 & 41 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 45 & 91 \\ 45 & 0 & 59 \\ 91 & 59 & 0 \end{bmatrix}$$

Fig. 5 illustrates the performance of the three algorithms during the search processes for the ($O9, M3, D3$) problem. The best scheduling solution obtained by the 20 MSGA runs is {3,1,2,3,1,1,1,3,1}, with a makespan of 23,654 and a flowtime of 34,075. 20 runs of MSPSO and VNPSO both yielded {3,1,2,3,1,1,2,3,2} as the best scheduling solution. This has a makespan of 15,011 and a flowtime of 30,647. While MSPSO provides the best results 8 times in 20 runs, VNPSO provides the best result 10 times in the same runs respectively. Taking the security constraints into account, the best scheduling solution was found to be {3,1,2,3,2,2,1,3,2} with makespan of 20,654 and a flowtime of 44175. Fig. 6 provides an optimal schedule for the ($O9, M3, D3$) problem, in which "W" means the waiting time. As depicted in Fig. 6a, the operations O_2 and O_3 both have to

Table 1
Parameter settings for the algorithms.

Algorithm	Parameter name	Parameter value
GA	Size of the population	20
	Probability of crossover	0.9
	Probability of mutation	0.09
	Swarm size	20
PSOs	Self-recognition coefficient c_1	1.49
	Social coefficient c_2	1.49
	Inertia weight w	0.9 → 0.1
	Clamping coefficient ρ	0.5

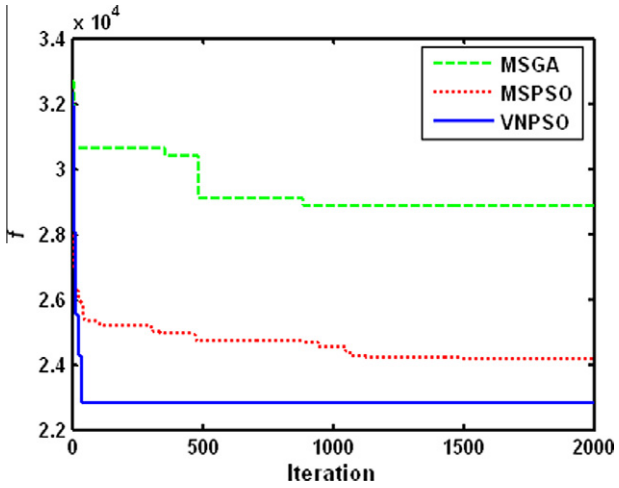

Fig. 5. Performance for the $(O9, M3, D3)$ problem.

wait for 1611 time units before they are processed in the scheduling solution. Because of the security constraints, O_7 has to be assigned to M_1. To minimize the objective, O_5 and O_6 are scheduled on M_2, as shown in Fig. 6b. Thus, facilitating the security constraints adds an extra overhead to the computing time.

The algorithms were tested on three further scheduling problems, namely, $(O10, M3, D3)$, $(O12, M4, D3)$ and $(O16, M6, D5)$. Empirical results are illustrated in Table 2, in which the unit of "average" is cycles per unit time (*CPUT*) and the unit of "time" is seconds (s). In general, VNPSO performs better than the other two approaches, although its computational time is worse than MSPSO.

In the swarm, each particle finds a solution represented by a position in the problem space. Since the positions are different, the swarm has the capability to explore the space searching for better solutions. We introduce the symbol, *Diversity*, to represent the diversity of the swarm and measure it as follows:

$$Diversity = \frac{1}{n}\sum_{i=1}^{n}\sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_{i,j}-\bar{x}_j)^2} \tag{32}$$

Here, n is the swarm population size, d is the particle dimension, $\bar{x}$ is the center point of the swarm, and its jth dimension is denoted by $\bar{x}_j$. Fig. 7 shows the typical evolution of swarm diversity for each of the three algorithms during an optimization

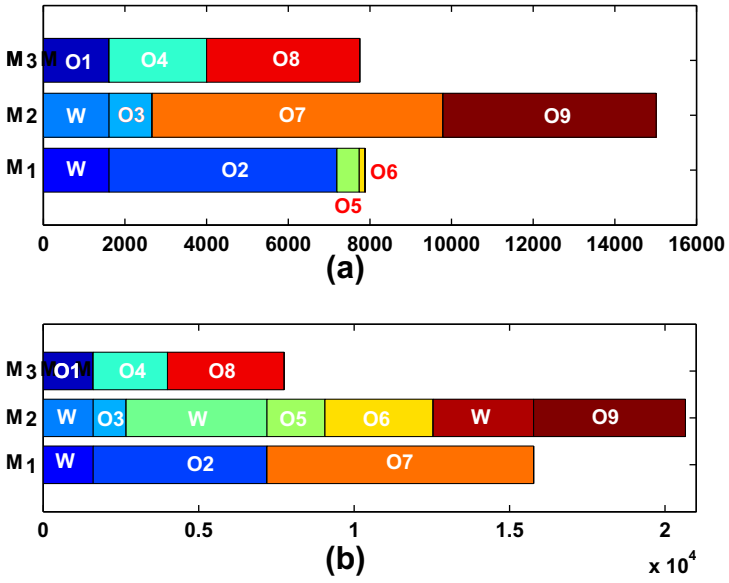

Fig. 6. Scheduling solutions for the $(O9, M3, D3)$ problem, (a) without security constraints, (b) with security constraints.

Table 2
Comparison of performance for different scheduling problems.

Instance	Items	MSGA	MSPSO	VNPSO
(*O*9,*M*3,*D*3)	Average time	34,020 210.8720	32,477 167.5360	32,464 193.6490
(*O*10,*M*3,*D*3)	Average time	24,712 232.7320	22,351 178.9520	19,872 187.4980
(*O*12,*M*4,*D*3)	Average time	18,580 257.5210	16,358 194.2150	15,214 207.3720
(*O*16,*M*6,*D*5)	Average time	39,582 362.7670	32,470 248.3540	30,852 252.5760

run. In PSO the diversity undergoes a fluctuating decrease phase before stabilizing at a very low value after 1500 iterations. In contrast three different phases can be observed in VNPSO. Initially the diversity decreases in much the same way as PSO, but stabilizes after 500 iteration at a high state of diversity. This is maintained until 1500 iterations where again the diversity begins to decrease. MSPSO maintains swarm diversity at its initial value throughout the optimization run.

Fig. 8 shows the evolution of the velocity of a randomly selected particle from each algorithm. In PSO particle velocities decrease rapidly over time. They are relatively constant for MSPSO and have an oscillating decay pattern in VNPSO that

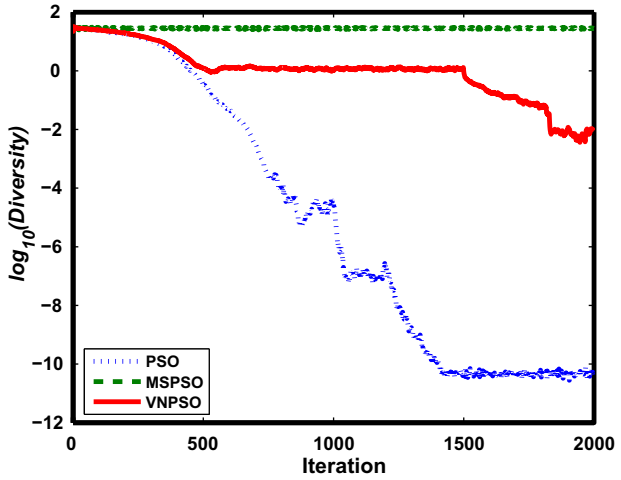

Fig. 7. Comparison of the evolution of swarm diversity.

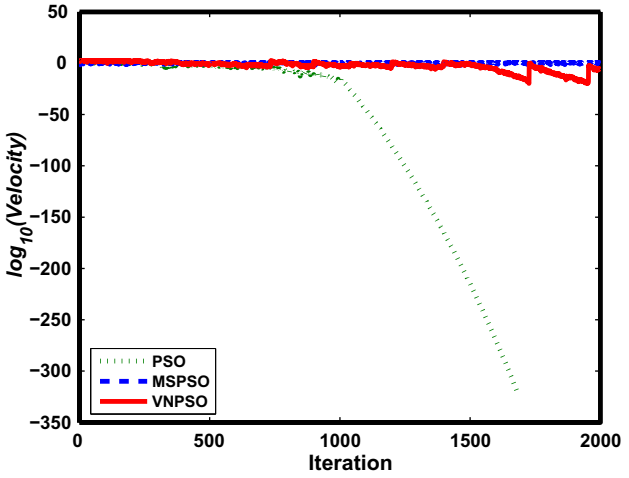

Fig. 8. Comparison about the velocity.

maintains the velocity at a relatively high level throughout the optimization run. Large diversity and velocity values facilitate global exploration (searching new areas), while smaller values tend to facilitate local exploitation, i.e. fine-tuning the solution in the current search area. The desirable outcome is a balance between global exploration and local exploitation abilities that results in a reduction of the number of iterations required to locate the optimum solution. Considering Figs. 7 and 8 together it can be seen that VNPSO initially provides large swarm diversity and particle velocities facilitating exploration of the global solution space and then focuses on the current best solution with a refining step to achieve local exploitation. When the algorithm gets trapped in a local solution the shaking strategy re-introduces the global exploration mode enabling the algorithm to escape the local minimum. Experimental results confirm that VNSPO achieves the best balance between global exploration and local exploitation.

6. Conclusions

In this paper, we modeled and formulated the scheduling problem for work-flow applications in distributed data-intensive computing environments. The environment components and states were defined and described. A security constraint model was also presented for the job scheduling problem and appropriate performance metrics discussed. The variable neighborhood search particle swarm optimization (VNPSO) algorithm is presented as a method for solving the resulting scheduling problem. Related swarm models and algorithm convergence were discussed in detail. Empirical results demonstrate that the VNPSO algorithm is feasible and effective. It usually provides a good balance between global exploration and local exploitation abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. VNPSO can be applied in distributed data-intensive applications to meet specified requirements, including work-flow constraints, security constraints, data retrieval/transfer, job interaction, minimum completion cost, flexibility and availability.

Acknowledgments

The first author would like to thank Drs. Shichang Sun, Bo Li, Ran He, Mingyan Zhao and Prof. Xiukun Wang for their scientific collaboration in this research work. This work is supported by the National Natural Science Foundation of China (Grant Nos. 60873054 and 61073056), the Fundamental Research Funds for the Central Universities (Grant No. 2011JC006), and Dalian Science and Technology Fund (Grant No. 2010J21DW006).

References

[1] A. Abraham, H. Guo, H. Liu, Swarm intelligence: foundations, perspectives and applications, Swarm Intelligent Systems, Studies in Computational Intelligence 26 (2006) 3–25.
[2] M.R. AlRashidi, M.E. El-Hawary, A survey of particle swarm optimization applications in electric power systems, IEEE Transactions on Evolutionary Computation 13 (4) (2009) 913–918.
[3] I. Babaoglu, O. Findik, E. Ülker, A comparison of feature selection models utilizing binary particle swarm optimization and genetic algorithm in determining coronary artery disease using support vector machine, Expert Systems with Applications 37 (4) (2010) 3177–3183.
[4] K. Binkley, M. Hagiwara, Balancing exploitation and exploration in particle swarm optimization: velocity-based reinitialization, Information and Media Technologies 3 (1) (2008) 103–111.
[5] A. Boccia, G. Busiello, L. Milanesi, G. Paolella, A fast job scheduling system for a wide range of bioinformatic applications, IEEE Transactions on NanoBioscience 6 (2) (2007) 149–154.
[6] D. Boeringer, D. Werner, Particle swarm optimization versus genetic algorithms for phased array synthesis, IEEE Transactions on Antennas and Propagation 52 (3) (2004) 771–779.
[7] P. Brucker, Scheduling Algorithms, Springer Verlag, 2007.
[8] R. Chang, J. Chang, P. Lin, An ant algorithm for balanced job scheduling in grids, Future Generation Computer Systems 25 (1) (2009) 20–27.
[9] K. Chung, P. Erdös, On the application of the Borel–Cantelli lemma, Transactions of the American Mathematical Society 72 (1952) 179–186.
[10] S. Chung, F. Chan, H. Chan, A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling, Engineering Applications of Artificial Intelligence 22 (7) (2009) 1005–1014.
[11] M. Clerc, Particle Swarm Optimization, ISTE Publishing Company, 2006.
[12] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[13] S. Das, A. Abraham, A. Konar, Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm, Pattern Recognition Letters 29 (5) (2008) 688–699.
[14] F. Dong, S. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, Tech. Rep., School of Computing, Queen's University, Kingston, Ontario, January 2006.
[15] R. Eberhart, Y. Shi, Comparison between genetic algorithms and particle swarm optimization, Lecture Notes in Computer Science 1447 (1998) 611–618.
[16] A. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, London, 2005.
[17] S. Ghosh, D. Kundu, K. Suresh, S. Das, A. Abraham, B. Panigrahi, V. Snášel, On some properties of the lbest topology in particle swarm optimization, in: Proceedings of the 2009 Ninth International Conference on Hybrid Intelligent Systems, vol. 3, IEEE Computer Society Press, Washington, 2009, pp. 370–375.
[18] C. Guo, H. Tang, Global convergence properties of evolution stragtegies, Mathematica Numerica Sinica 23 (1) (2001) 105–110.
[19] P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications, European Journal of Operational Research 130 (3) (2001) 449–467.
[20] P. Hansen, N. Mladenović, J. Moreno Pérez, Variable neighbourhood search: methods and applications, Annals of Operations Research 175 (1) (2010) 367–407.
[21] P. Hansen, N. Mladenović, D. Urošević, Variable neighborhood search and local branching, Computers and Operations Research 33 (10) (2006) 3034–3045.

[22] R. He, Y. Wang, Q. Wang, J. Zhou, C. Hu, Improved particle swarm optimization based on self-adaptive escape velocity, Chinese Journal of Software 16 (12) (2005) 2036–2044.
[23] T. Hey, S. Tansley, K. Tolle, The fourth paradigm: data-intensive scientific discovery, Microsoft Research (2009).
[24] W. Horn, AI in medicine on its way from knowledge-intensive to data-intensive systems, Artificial Intelligence in Medicine 23 (1) (2001) 5–12.
[25] M. Jiang, Y. Luo, S. Yang, Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm, Information Processing Letters 102 (1) (2007) 8–16.
[26] V. Kadirkamanathan, K. Selvarajah, P. Fleming, Stability analysis of the particle dynamics in particle swarm optimizer, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 245–255.
[27] J. Kennedy, R. Eberhart, Y. Shi, Swarm Intelligence, Springer, 2001.
[28] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Proceedings of the Evolutionary Computation on Evolutionary Computation, 2002, pp. 1671–1676.
[29] B. Khoo, B. Veeravalli, T. Hung, C. See, A multi-dimensional scheduling scheme in a grid computing environment, Journal of Parallel and Distributed Computing 67 (6) (2007) 659–673.
[30] A. Lazinica, Particle Swarm Optimization, InTech Education and Publishing, 2009.
[31] J. Lee, B. Tierney, W. Johnston, Data intensive distributed computing: a medical application example, Lecture Notes in Computer Science 1593 (1999) 150–158.
[32] H. Liu, A. Abraham, An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems, Journal of Universal Computer Science 13 (9) (2007) 1309–1331.
[33] H. Liu, A. Abraham, M. Clerc, Chaotic dynamic characteristics in swarm intelligence, Applied Soft Computing 7 (3) (2007) 1019–1026.
[34] H. Liu, A. Abraham, Z. Wang, A multi-swarm approach to multi-objective flexible job-shop scheduling problems, Fundamenta Informaticae 95 (2009) 1–25.
[35] H. Liu, B. Li, Y. Ji, T. Sun, Particle swarm optimisation from lbest to gbest, Applied Soft Computing Technologies: The Challenge of Complexity 34 (2006) 537–545.
[36] N. Liu, M. Abdelrahman, S. Ramaswamy, A complete multiagent framework for robust and adaptable dynamic job shop scheduling, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37 (5) (2007) 904–916.
[37] J. Martínez, E. Gonzalo, The generalized PSO: a new door to PSO evolution, Journal of Artificial Evolution and Applications 2008 (2008) 1–15.
[38] G. Mosheiov, A. Sarig, Scheduling with a common due-window: polynomially solvable cases, Information Sciences 180 (8) (2010) 1492–1505.
[39] K. Parsopoulos, M. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Natural Computing 1 (2) (2002) 235–306.
[40] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: Proceedings of the 11th International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Washington, 2002, pp. 352–361.
[41] V. Roshanaei, B. Naderi, F. Jolai, M. Khalili, A variable neighborhood search for job shop scheduling with set-up times to minimize makespan, Future Generation Computer Systems 25 (6) (2009) 654–661.
[42] Y. Shi, R. Eberhart, E. Team, I. Kokomo, Fuzzy adaptive particle swarm optimization, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, 2001, pp. 101–106.
[43] S. Song, K. Hwang, Y. Kwok, et al, Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling, IEEE Transactions on Computers 55 (6) (2006) 703.
[44] S. Song, K. Hwang, R. Zhou, Y. Kwok, Trusted P2P transactions with fuzzy reputation aggregation, IEEE Internet Computing 9 (6) (2005) 24–34.
[45] S. Song, Y. Kwok, K. Hwang, Security-driven heuristics and a fast genetic algorithm for trusted grid job scheduling, Proceedings of the 19th International Parallel and Distributed Processing Symposium, vol. 65, IEEE Computer Society, Washington, DC, USA, 2005, pp. 4–12.
[46] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (17) (2010) 3182–3191.
[47] Y. Tao, H. Jin, X. Shi, Grid workflow scheduling based on reliability cost, in: Proceedings of the 2nd International Conference on Scalable Information Systems, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Belgium, 2007.
[48] I. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (6) (2003) 317–325.
[49] F. Van den Bergh, An analysis of particle swarm optimizers, Ph.D. thesis, University of Pretoria, Pretoria, South Africa, November 2001.
[50] F. Van den Bergh, A. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Sciences 176 (8) (2006) 937–971.
[51] S. Venugopal, R. Buyya, A deadline and budget constrained scheduling algorithm for eScience applications on data grids, Lecture Notes in Computer Science 3719 (2005) 60–72.
[52] S. Venugopal, R. Buyya, An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids, Journal of Parallel and Distributed Computing 68 (4) (2008) 471–487.
[53] Y. Wen, H. Xu, J. Yang, A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system, Information Sciences 181 (3) (2011) 567–581.
[54] C. Wu, R. Sun, An integrated security-aware job scheduling strategy for large-scale computational grids, Future Generation Computer Systems 26 (2) (2010) 198–206.
[55] T. Xie, X. Qin, Scheduling security-critical real-time applications on clusters, IEEE Transactions on Computers 55 (7) (2006) 864–879.
[56] T. Xie, X. Qin, Performance evaluation of a new scheduling algorithm for distributed systems with security heterogeneity, Journal of Parallel and Distributed Computing 67 (10) (2007) 1067–1081.
[57] T. Xie, X. Qin, Security-driven scheduling for data-intensive applications on grids, Cluster Computing 10 (2) (2007) 145–153.
[58] C. Yu, D. Marinescu, Algorithms for divisible load scheduling of data-intensive applications, Journal of Grid Computing (2009) 1–23.
[59] Z. Zhan, J. Zhang, Y. Li, H. Chung, Adaptive particle swarm optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 39 (6) (2009) 1362–1381.
[60] R. Zhang, C. Wu, A hybrid immune simulated annealing algorithm for the job shop scheduling problem, Applied Soft Computing 10 (1) (2010) 79–89.
[61] X. Zhao, A perturbed particle swarm algorithm for numerical optimization, Applied Soft Computing 10 (1) (2010) 119–124.
[62] N. Zhong, J. Hu, S. Motomura, J. Wu, C. Liu, Building a data-mining grid for multiple human brain data analysis, Computational Intelligence 21 (2) (2005) 177–196.