J Intell Manuf (2017) 28:1189–1201
DOI 10.1007/s10845-015-1074-0

Fast GA-based project scheduling for computing resources allocation in a cloud manufacturing system

Yang-Kuei Lin[1] · **Chin Soon Chong**[2]

Received: 30 November 2014 / Accepted: 25 March 2015 / Published online: 8 April 2015

Abstract Cloud manufacturing is becoming an increasingly popular enterprise model in which computing resources are made available on-demand to the user as needed. Cloud manufacturing aims at providing low-cost, resource-sharing and effective coordination. In this study, we present a genetic algorithm (GA) based resource constraint project scheduling, incorporating a number of new ideas (enhancements and local search) for solving computing resources allocation problems in a cloud manufacturing system. A newly generated offspring may not be feasible due to task precedence and resource availability constraints. Conflict resolutions and enhancements are performed on newly generated offsprings after crossover or mutation. The local search can exploit the neighborhood of solutions to find better schedules. Due to its complex characteristics, computing resources allocation in a cloud manufacturing system is NP-hard. Computational results show that the proposed GA can rapidly provide a good quality schedule that can optimally allocate computing resources and satisfy users' demands.

Keywords Resource allocation · Cloud manufacturing · Project scheduling · Genetic algorithm

✉ Yang-Kuei Lin
yklin@mail.fcu.edu.tw

Chin Soon Chong
cschong@SIMTech.a-star.edu.sg

[1] Department of Industrial Engineering and Systems Management, Feng Chia University, P.O. Box 25-097, Taichung 40724, Taiwan, ROC

[2] Planning and Operations Management Group, Singapore Institute of Manufacturing Technology, A-star, 7 Nanyang Avenue, Singapore 638075, Singapore

Introduction

Cloud manufacturing is a service-oriented, customer-centric, demand-driven manufacturing model that aims at providing low-cost, resource-sharing and effective coordination. According to Wu et al. (2013), cloud manufacturing requires interaction between three groups: the users, application providers, and physical resource providers. Users are the consumers in cloud manufacturing. They have the need to manufacture a product, but do not possess the capabilities to do so, or they possess the capabilities but can also gain a competitive advantage by using cloud manufacturing. The application providers are responsible for managing all aspects of the cloud manufacturing environment and transfers user requirements into data required for manufacturing the desired products. Physical resource providers own and operate manufacturing equipment, such as forms of machining, finishing, inspection, packaging technology, testing resources, etc. Moreover, physical resource providers have the expertise and experience to utilize these machines effectively and efficiently. Computing resources (such as CPU, processors, I/O, networks, servers, storage, applications, services, etc.) at the physical resource layer is an important part of cloud manufacturing. Through centralized management, the application providers transfer users' demands into virtual access, and assign various heterogeneous computing resources to process those demands.

The current computing resource allocation in cloud manufacturing faces several difficult problems. Based on Wu and Yang (2010), in order to cope with the quickly changing market requirements in an increasing globalized economic environment, manufacturing enterprises need to develop extensive collaboration among themselves. This includes many kinds of manufacturing resources sharing. These resources are geographically distributed, with morphology diversity and autonomy, which makes the resource-sharing and management very complicated. Further, the computing resources are made available on-demand and dynamically allocated to the user as needed, to realize scalable integration of all kinds of distributed resources for effective use. Some of the previous studies pointed out that in the cloud manufacturing system, the resource demands are time varying and often has large spikes. How to appropriately select the price and allocate resources for each type of virtual machine services in order to best match the interests of the customers is a key issue of cloud manufacturing (Zhang et al. 2011). Another key issue is how to minimize the response time of data processing queries to satisfy customer demands (Lee et al. 2011). Further, Laili et al. (2012) mentioned that the optimal allocation of computing resources is the core part of implementing cloud manufacturing. High heterogeneity, high dynamism, and virtualization make the optimal allocation of computing resources problems more complex than the traditional scheduling problems in the grid system or cloud computing system. Motivated by these studies, this paper proposed a new and fast hybrid GA method on the proposed model to solve the optimal allocation of computing resources problems.

In this work, we are assuming that there are multiple project planning problems that require computing resources from cloud manufacturing platforms at the same time. Hence, large amount of tasks with different resource requirements that are submitted to the cloud manufacturing platform need to be scheduled heterogeneously under layer clusters. The cloud-based application providers need to centrally manage those demands, and allocate computing resources in order to process them and respond to users promptly.

Figure 1 shows computing resources allocation in a cloud manufacturing system for resource-constrained project scheduling problems (RCPSPs). We focus on studying the problem of computing resources allocation in a cloud manufacturing system. A GA has been proposed to tackle the problem. The rest of this work is organized as follows. Section 2 provides the related works; Sect. 3 introduces resource-constrained project scheduling; the proposed GA is presented in Sect. 4; in Sect. 5, the computational results are reported; Sect. 6 presents our conclusions and suggestions for future research.

Related works

In recent years, cloud manufacturing has been studied by many researchers. Some researchers focused on studying computing resource allocation in a cloud manufacturing system (Wu and Yang 2010; Lee et al. 2011; Zhang et al. 2011). Laili et al. (2011) used a GA to solve the problem of optimal allocation of computing resources in cloud manufacturing systems. In a subsequent study, Laili et al. (2012) proposed a new and improved niche immune algorithm (NIA) for optimal allocation of computing resources in cloud manufacturing systems. They showed that NIA outperforms other intelligent algorithms in solving computing resources allocation problems. Tao et al. (2012) proposed a novel parallel intelligent algorithm, namely, full connection based parallel-adaptive chaos optimization with reflex migration (FC-PACO-RM) for service composition optimization-selection in a cloud manufacturing system. Laili et al. (2013) proposed a ranking chaos algorithm for dual scheduling of cloud service and computing resources in a private cloud. Cheng et al. (2013) established comprehensive utility models, which consider energy consumption, costs, and risks for the three sides (provider, consumer, and

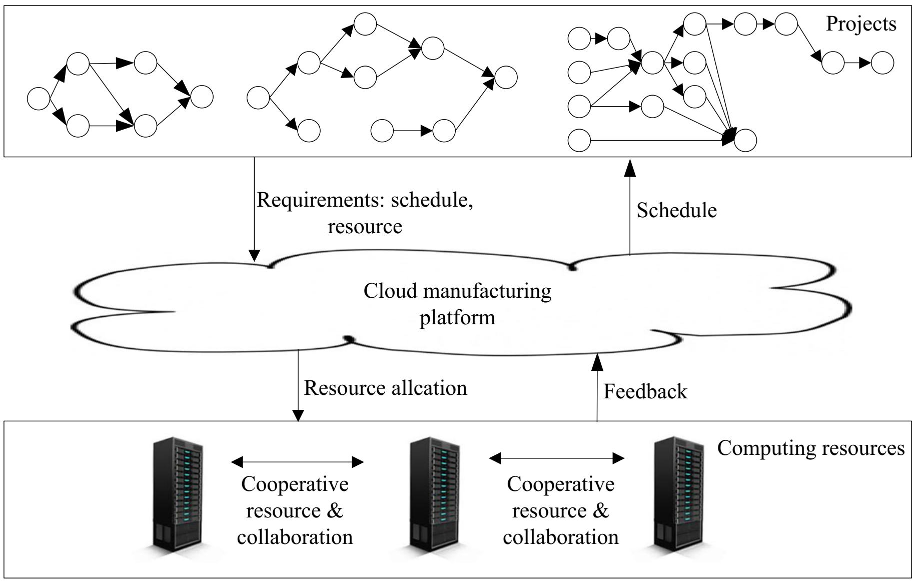
Projects
Requirements: schedule, resource
Schedule
Cloud manufacturing platform
Resource allcation
Feedback
Computing resources
Cooperative resource & collaboration
Cooperative resource & collaboration

Fig. 1 Computing resources allocation in cloud manufacturing system for project scheduling problems

operator) in the resource service scheduling process in a cloud manufacturing system.

Some researchers focused on studying workflow scheduling in cloud computing environment. A cloud workflow system is a type of platform service which facilitates the automation of distributed applications based on the novel cloud infrastructure. Kaur et al. (2011) provided a comparison of workflow scheduling algorithms in cloud computing. Varalakshmi et al. (2011) proposed an optimal workflow based algorithm for scheduling workflows in a cloud environment. Bardsiri and Hashemi (2012) provided a review of workflow scheduling in the cloud computing environment. Kim et al. (2012) developed an adaptive workflow scheduling scheme that optimized the allocation ratio of computing elements to the different datasets in order to minimize the total makespan under certain resource constraints. Abrishami and Naghibzadeh (2012) proposed an algorithm for workflow scheduling in the software as a service cloud, which minimizes the total execution cost, while meeting user-defined deadlines. Rahman et al. (2013) proposed an adaptive workflow scheduling algorithm for dynamic grid and cloud computing environments. Lodha and Wadhe (2013) provided a study on different types of workflow scheduling algorithm in cloud computing. Yassa et al. (2013) proposed a multi-objective approach for energy-aware workflow scheduling in cloud computing environments.

From the literature review, we concluded that the optimal allocation of computing resources is one of the urgent issues in cloud manufacturing. Also, prompt response to customers' demands is a requirement in cloud manufacturing. Hence, in this study, we proposed a fast hybrid GA to solve the optimal allocation of computing resources problems.

Resource-constrained project scheduling problem

According to Laili et al. (2011), cloud manufacturing is a service-oriented networked manufacturing model which aims at achieving low cost resource sharing and coordination. By way of virtual access, various heterogeneous computing resources at the physical layer of cloud manufacturing are assigned to users on-demand. The demand of coordination and computing power on the cloud manufacturing platform is high and varied as manufacturing tasks are usually complex and multi-disciplinary collaborative tasks. To ensure efficient running of manufacturing tasks and to fully utilize computing resources, the optimal allocation of computing resources to manufacturing tasks in cloud manufacturing is important. This poses challenges for optimal allocation of computing resources as computing resources have the characteristics of large scale, high heterogeneity, dynamic interconnection and group collaboration. The practical issues and constraints related to resource allocation are discussed in detail in Laili et al. (2011). To tackle the challenges, Laili et al. (2011) formulated a new optimal allocation of computing resources model based on the uncertain resource topology. The model considered the constrain conditions from all aspects and is closer to reality.

Similar to Laili et al. (2011, 2012), we are assuming that there are multiple project planning problems that require computing resources from cloud manufacturing. Due to the large amount of tasks, there are different resource requirements submitted to a cloud manufacturing platform that need to be scheduled heterogeneously under layer clusters. There is an insufficient amount of practical activity time, so we made some assumptions and set a range for them. Hence, this study only follows one project scheduling. For a given project scheduling problem, we use graphs and record the successors and predecessors of each node by having adjacent lists. Since many tasks are submitted to a resource for service, and the resource is needed to handle the tasks sequentially, we vary the activity times to reflect the situations of resource allocation in cloud manufacturing. The task duration of each node is randomly generated within a certain range. When we complete evaluations, we start from the source (the first) node, and then find its successors step by step, which is followed by recording the whole task durations at the end of the graph (sink node). Project scheduling problem in cloud manufacturing is one type of RCPSPs.

Here, we give a brief literature review of RCPSPs. In recent years, more and more researchers have studied the RCPSPs (Brucker et al. 1999; Hartmann and Briskorn 2010). Kolisch and Hartmann (2006) have summarized and categorized a large number of heuristics that have recently been proposed in the literature for the RCPSPs. Some researchers have proposed sampling-method based on heuristics (Kolisch 1995, 1996a, b; Kolisch and Drexl 1996; Shue and Zamani 1999; Grèze et al. 2014). Amongst the sampling methods, the heuristics are referred to as sampling random (S) and sampling-random (P) corresponding to pure random sampling with serial and parallel scheduling generation scheme (SGS). Some researchers have developed meta-heuristic based algorithms to tackle the RCPSPs, which include GA (Leon and Ramamoorthy 1995; Hartmann 1998; Coelho and Tavares 2003; Debels and Vanhoucke 2007; Valls et al. 2008; Agarwal et al. 2011; Ji and Yao 2014), ant colony optimization (ACO) (Merkle et al. 2002, Chen et al. 2010a, b), simulated annealing (SA) (Bouleimen and Lecocq 2003), filter-and-fan approach (Ranjbar 2008), bee colony (Shi et al. 2010; Jia and Seo 2013b), shuffled frog-leaping algorithm (Fang and Wang 2012) and particle swarm optimization (PSO) (Jia and Seo 2013a).

A brief introduction of RCPSP is provided here. The RCPSP can be described as follows. A single project consists of a set $J = \{0, 1, ..., n, n + 1\}$ of tasks (activities), which have to be processed. Fictitious tasks 0 and $n + 1$

correspond to the “project start” and to the “project end”, respectively. The tasks are subjected to two kinds of constraints. First, precedence constraints enforce task j to start only after all its immediate predecessors have been completed. Second, performing the tasks requires resources that are limited. Resources may be renewable or non-renewable. Renewable resources are available for each period without being depleted, such as labor and computer memory. Non-renewable resources are depleted as they are used up, such as capital and raw materials. We have K renewable resource types, given by the set $\kappa = \{1, \ldots, K\}$. While being processed, task j requires $r_{j,k}$ units of resource type $k \in \kappa$ during every period of its non-preemptable duration p_j. Resource type k has a limited capacity of R_k at any point in time. For the project start and end tasks, we have $P_j = 0$ and $r_{j,k} = 0$ for all $k \in \kappa$. The objective of the RCPSP is to find precedence and resource feasible completion times for all tasks so that the makespan of the project is minimized. We use an instance j301_5 (we call it Example 1) that is taken from the project scheduling problem library (PSPLIB, Kolisch and Sprecher 1997) to illustrate the ideas of the studied problem. Table 1 gives an example of a project comprising $n = 32$ tasks, which have to be scheduled subject to the $K = 4$ renewable resource type with a capacity of $R_1 = 11$, $R_2 = 11$, $R_3 = 9$, $R_4 = 11$ units. A feasible schedule with a makespan of 40 periods is represented in Fig. 2. Figure 3 shows resource allocation of Example 1 where a number inside a parenthesis indicates the resource consumption of the task.

Genetic algorithms

GAs are stochastic search techniques that are inspired by the principles of evolution and heredity (Holland 1975). The usual form of a GA is described by Goldberg (1989). GAs have been proven to be a very robust algorithm for solving NP-hard global optimization problems, including scheduling problems. GAs are very effective for its climbing ability and the diversification in search. Besides that, GAs are easy to adapt, implement, and enhance. In the last two decades, there are lots of researchers applied GA-based approaches to solve the RCPSPs successfully (Leon and Ramamoorthy 1995; Hartmann 1998; Alcaraz and Maroto 2001; Hartmann 2002; Coelho and Tavares 2003; Kochetov and Stolyar 2003; Alcaraz et al. 2004; Valls et al. 2008; Mendes et al. 2009; Debels and Vanhoucke 2007; Khanzadi et al. 2011; Agarwal et al. 2011; Gonçalves et al. 2011; Zamani 2013). GAs are population based algorithms that work iteratively. A single iteration is called a generation. When applying them to scheduling problems, GAs consider a schedule as an individual of a population. A fitness value derived from the objective value of the schedule is assigned to each single individual, called a genome. The individuals of the new generation are obtained from the individuals of the previous generation by applying crossover and mutation procedures. The fittest among populations are selected, and are then carried into the next generation. In this work, we present a GA for optimal allocation of computing resources in a cloud manufacturing system. The proposed GA are described as follows.

Table 1 Data of Example 1

Task j	Successors	Durations p_j	$r_{j,1}$	$r_{j,2}$	$r_{j,3}$	$r_{j,4}$	Task j	Successors	Durations p_j	$r_{j,1}$	$r_{j,2}$	$r_{j,3}$	$r_{j,4}$
1	2, 3, 4	0	0	0	0	0	17	26	3	0	2	0	0
2	11, 12, 15	6	0	0	7	0	18	19, 23	2	7	0	0	0
3	5, 8, 16	4	0	0	1	0	19	22	3	0	8	0	0
4	27	2	0	0	0	2	20	29	3	0	0	5	0
5	6, 12, 14	1	0	0	6	0	21	31	7	0	0	0	2
6	7	5	2	0	0	0	22	24	4	0	0	0	1
7	9, 28	1	0	0	0	6	23	24	8	0	0	0	5
8	10, 20	3	0	9	0	0	24	29	1	0	0	0	5
9	13, 21	5	0	8	0	0	25	30	8	0	9	0	0
10	17, 25	6	8	0	0	0	26	28	8	0	5	0	0
11	25, 28	6	0	0	7	0	27	30	5	0	9	0	0
12	25, 29	4	7	0	0	0	28	30	1	0	4	0	0
13	22, 23, 31	1	8	0	0	0	29	32	5	0	4	0	0
14	17	6	0	1	0	0	30	32	2	0	9	0	0
15	24	5	0	0	0	2	31	32	1	0	0	0	10
16	18	3	0	0	5	0	32	–	0	0	0	0	0

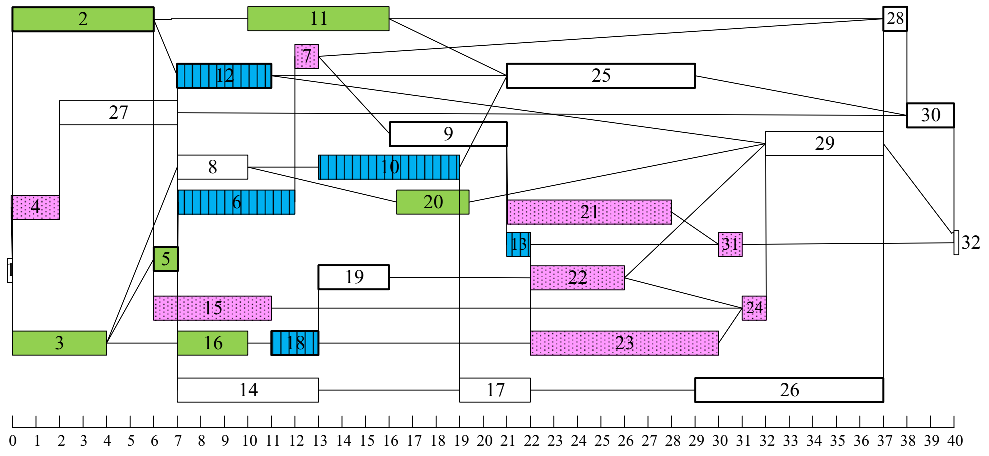

Fig. 2 A feasible solution of Example 1

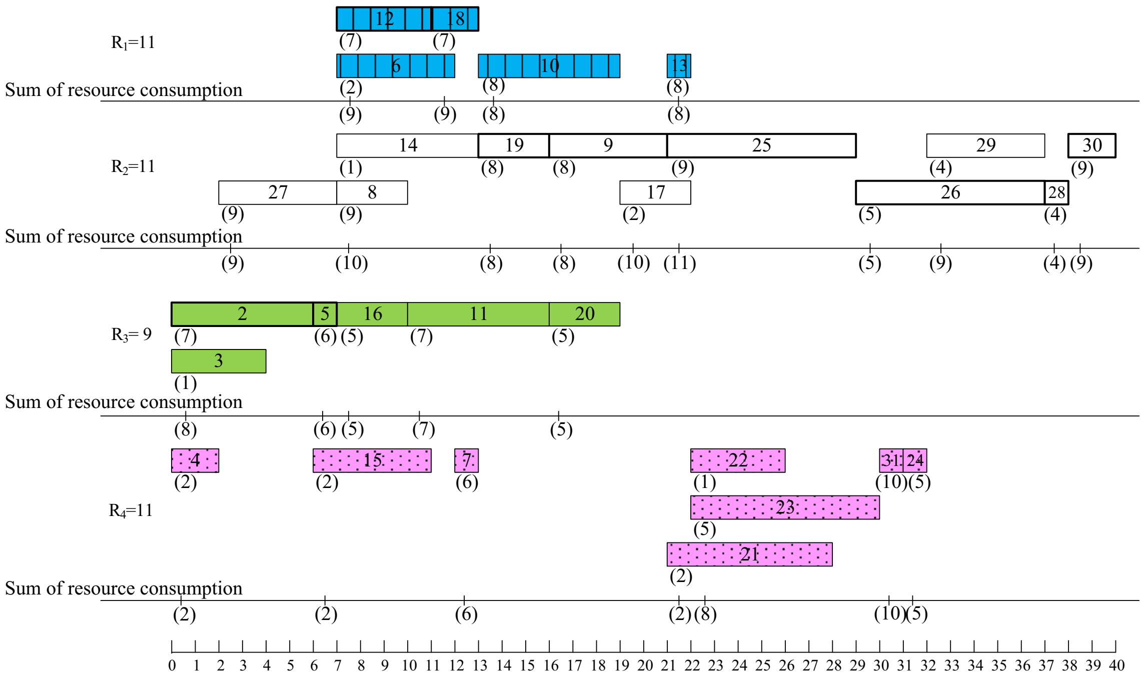

Fig. 3 Resource allocation of Example 1

Basic operations of the GA

Genetic representation

Wall (1996) developed a coding scheme that can generate a feasible solution for project scheduling problems. Here, we use the Wall (1996) coding scheme to represent a solution to the problem at hand, as a genome. A single genome is a single array of integer start times. Each element in the array corresponds to a task in the project plan. The times represent delay times relative to the latest finish time of the task's predecessors. Unlike traditional solution methods which are typically sequence-based, each genome uses representation that encodes scheduling information as an array of relative start times. Each time represents the duration from the latest finish of all predecessor tasks, to the start time of the corresponding task. The times represent delay times relative to the finish time of the predecessors. If the precedence constraints for a task is allowed to overlap, then negative values are permitted for the start time for this particular task. If overlap is not modeled, values are truncated to zero. The elements in the array correspond to the tasks in the project plan, but the order of elements relative to each other is insignificant. This encoding scheme for scheduling is a minimal repre-

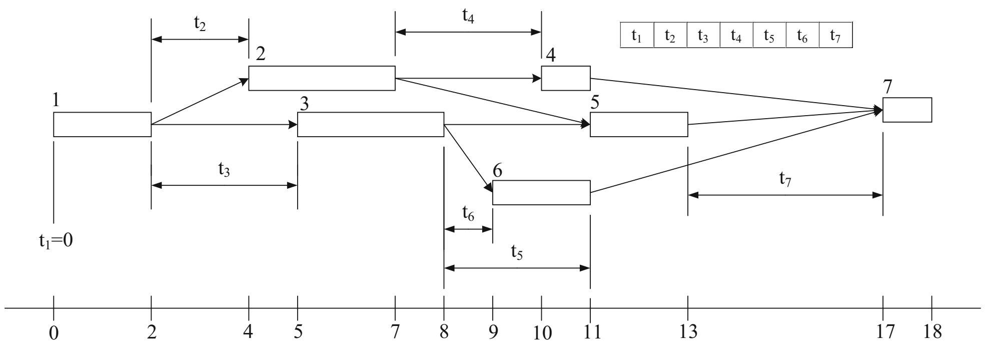

Fig. 4 Representation of a genome and its mapping to the schedule

sentation that can only represent resource-feasible solutions. Unlike order-based representation, the encoding scheme will assure precedence feasibility for every schedule after any crossover operator, with no schedule repair required. Figure 4 gives an example of a single project consisting of seven tasks. A possible representation for Fig. 4 could be $[t_1, t_2, t_3, t_4, t_5, t_6, t_7] = [0, 2, 3, 3, 3, 1, 4]$.

Initialization

We use random initialization. The integer start times of the genome are initialized with random numbers. The range of possible values is based upon the average estimated task durations. We sum up all of the task durations and divide them by the number of tasks, excluding the first and last virtual tasks. Based on this average duration, we get the lower and upper durations by multiplying them with certain factors, currently set at 0.1 and 3.0, and then use a random function to generate a random value for each element in the time component of the genome.

Crossover

We use blend crossover (Eshelman and Schaffer 1992) to generate new offsprings from two parent values. Each genome is encoded as an array of relative start times. Each time represents the duration from the latest finish of all the predecessor tasks to the start time of the corresponding task. This representation is different from traditional solution methods, which are typically sequence based. By combining the array of relative start times encoding and Blend Crossover, the location of the offspring will depend on the difference in the parent solutions. If the differences between the parent solutions are small, the difference between the offspring and parent solution will be correspondingly small. This property of the search operator constitutes an adaptive search and needs no tuning parameter. The operator enables searching of an entire space, at the initial stage, when a random population over the entire space is initialized, and then it concentrates the search more on the later stages, when the population tends to converge in some region of the search space. The adaptive search property was highlighted in a study by Mehra et al. (2014). They showed that Blend Crossover performed better than Arithmetic Crossover in their problem domain. The crossover operator is a kind of linear combination of the two parents that uses the following equations for each gene:

$$\text{Offspring 1} = \text{Parent1} - b \times (\text{Parent1} - \text{Parent2})$$
$$\text{Offspring 2} = \text{Parent2} + b \times (\text{Parent1} - \text{Parent2})$$

where b is a random value between 0 and 1.

Mutation

Mutation is performed by applying Gaussian noise to each element in the array. A single element in the genome was mutated by replacing it with a number chosen based upon a Gaussian distribution defined by a mean and a standard deviation. The mean and standard deviation are both set equal to the gene's previous value. The mutation probability is gene-based, not genome-based.

Evaluation and selection

During each generation, genomes are evaluated using a measure of fitness based on the original objective function. Since we deal with a minimization problem, we convert the genome i objective function value into its fitness value by using a $f_i = 1/\text{makespan}$, so that a fitter genome has a larger fitness value. In each generation, we select a portion (replacement percentage) of the worst individuals, and then replace them with newly generated good individuals in order to create a new population for the next generation.

Termination

There are two terminating conditions. The first one is based on the number of generations, and the second one is based on the number of generations that cannot achieve a better solution, whichever is earlier.

Parameterization

We studied the effects of several important parameters on the performance of our proposed GA. Since a rapid response and optimal resource allocation is required for cloud manufacturing, extensive trial runs have been performed to provide practical values for the GA-related parameters. The selected GA parameters are the number of generations = 20, population size = 30, crossover rate = 0.6 and mutation rate = 0.2, generations that cannot achieve a better solution = 9, the replacement percentage = 0.2, local search level = 5, and number of best local search at each level = 3.

Enhancement

A newly generated offspring might not be feasible after crossover or mutation. This is because the encoding of genome has not considered resource constraints. In that case, we will perform enhancements to resolve those problems. After the crossover and mutation, and based on the delay times of genes of a genome, we construct an initial project schedule without considering the resource constraint, and in ascending order of task numbers and by using serial schedule generation scheme (serial SGS). Based on the start times of tasks in the constructed schedule, we sort the tasks in ascending order of their start times. The sorting also ensures the task precedence relationships of tasks satisfied. Based on the sorted task sequence, we then construct a final schedule that takes into consideration of task precedence and resource constraints by using serial SGS.

Local search

Two kinds of neighborhood search approaches are implemented in this research to exploit the neighborhood of solutions. Both neighborhood search approaches have used the concept of a block, in which a move is defined by inserting a task to a block on a critical path. Critical paths play an important part in a feasible solution. Critical paths are the longest routes from a start node to a sink node in a directed graph. We first try to locate the critical path blocks for a given solution. This is quite similar to job shop scheduling (Nowicki and Smutnicki 1996). From the last task, based on predecessor tasks and resource constraints, we traced them backwards. Take Example 1 in Fig. 3 as an example. Those rectangles highlighted with thicker borders are tasks in the critical path, which include (2, 5, 12, 18, 19, 9, 25, 26, 28, 30). A critical block (B) is a maximum subsequence of successive tasks, which contains tasks that are processed on the same machine in the critical path. For example, there are five blocks in the critical path, which are $B_1 = (2, 5)$, $B_2 = (12, 18)$, $B_3 = (19, 9, 25)$, $B_4 = (26, 28)$, $B_5 = (30)$.

Our first neighborhood search approach (N1) is based on Nowicki and Smutnicki (1996). For each critical path, we consider a set of moves that "swaps tasks near the borderline of blocks on a single critical path", as shown in Fig. 5. In the first block, we only swap the last two tasks, while in the last block, we only swap the first two tasks. For other blocks, besides the first and last blocks, we swap the first two tasks and the last two tasks, each of which contains at least two tasks.

Our second neighborhood search approach (N2) is based on Balas and Vazacopoulos (1998). For each critical path, we consider a set of moves that "moves internal tasks to either before the first or after the last task on a critical path" as shown in Fig. 6. The size of N2 is quite large and the approach is time consuming. However, our preliminary result shows that it tends to get better makespan than N1 since it considers more moves.

We test all of N1 (N2) moves, and choose the best n defined by the parameter (number of best local search) to go and find another critical path and then repeat the same procedure. This can go on indefinitely, as long as there are still tasks that can be swapped. Hence, we further use another parameter (local search level) to define how many levels (how many critical paths) we want to execute in a local search.

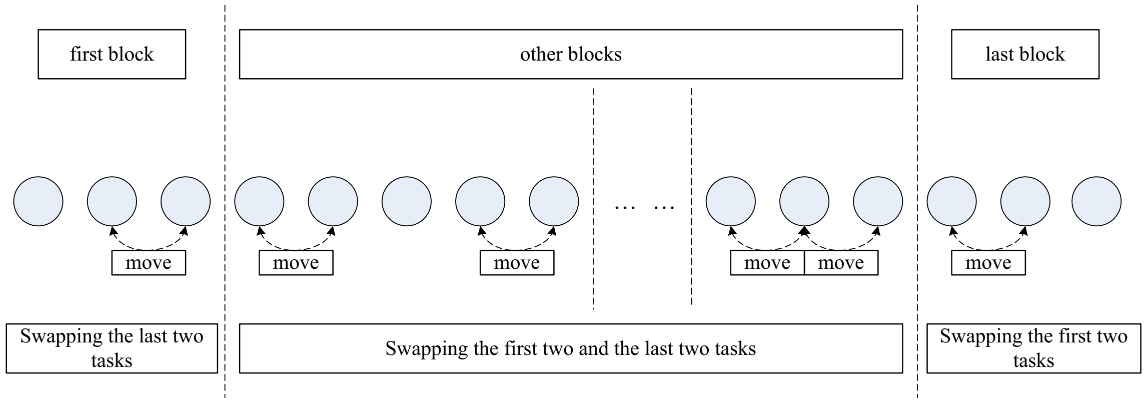
first block
other blocks
last block
move
move
move
move
move
move
… …
Swapping the last two tasks
Swapping the first two and the last two tasks
Swapping the first two tasks

Fig. 5 Neighborhood of moves (N1)

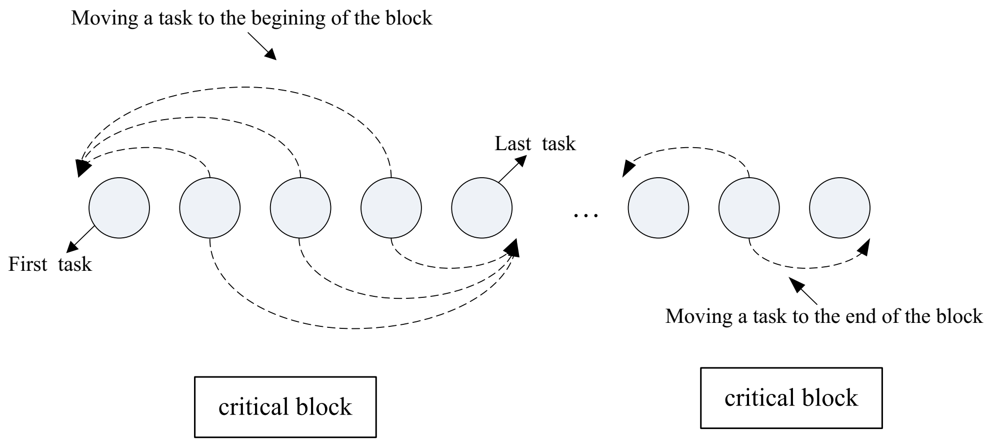
Moving a task to the begining of the block
Last task
First task
Moving a task to the end of the block
critical block
critical block

Fig. 6 Neighborhood of moves (N2)

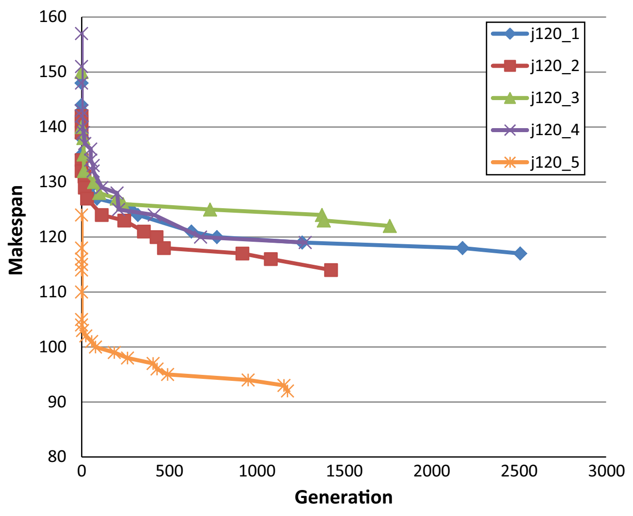

Fig. 7 The converge curves of the proposed GA

Computational results

In this section, we present several computational results on the performance of the proposed GA. The GA was implemented in C++ using GAlib, a C++ library of GA components developed by Wall (1996). Test instances include a variety of RCPSPs taken from PSPLIB (Kolisch and Sprecher 1997) and ran on an Intel (R) Core (TM) i7-3770 3.4 GHz processor and 16 GB RAM. We have selected the test sets J30, J60, J90, and J120. Each of these sets contains projects with 30, 60, 90, and 120 tasks, respectively, and four resources.

The convergence curves of the proposed GA

We examined the convergence curves of the proposed GA. We set the maximum generation to 10000 and disabled the terminating conditions for the number of generations that cannot achieve a better solution. Figure 7 shows five typical convergence curves for some of the benchmark instances of J120. From the curves, the GA has a steep descent for early generations. This is important for a fast GA applicable for resource allocation problems in cloud manufacturing. The GA, in average, takes 2703.4 generations to converge to the best solution for the 20 tested benchmark problem instances of J120.

The performance of the proposed GA

In order to examine the performance of the proposed GA, we first compare it with benchmark problem instances taken from PSPLIB. Two kinds of neighborhoods have been used to evaluate the quality of the proposed GA. GA_N1 represents GA incorporated with neighborhood search approach N1 while GA_N2 represents GA incorporated with neighborhood search approach N2. Four sets (J30, J60, J90, and J120) of benchmark instances have been used to evaluate the quality of the proposed GA. For each set of problem sizes, 20 instances have been selected. Since not all of the problem instances in PSPLIB have found optimal solutions, we have selected instances that have provided optimal solutions to be tested (lower bound = upper bound). Table 2 gives the computational results for benchmark problem instances. For single project scheduling that contains J30, J60, J90, and J120, the GA_N1 on average deviates 1.1, 2.9, 5.9, and 14.2 % from the optimal solution. The average computation times are 0.09, 0.24, 0.51, and 1.77 s for J30, J60, J90, and J120, respectively. The GA_N2 on average deviates 1.1, 2.4, 4.8, and 13.3 % from the optimal solution for J30, J60, J90, and J120, respectively. The average computation times are 0.09, 0.28, 0.81, and 2.88 s for J30, J60, J90, and J120, respectively. As the problem size increases, the average deviation from the optimal solutions increases. Also, as the problem size increases, the computation time increases. Our GA settings are fined tune for problem size 30 and we used the same GA

Table 2 The performance of GA for benchmark problem instances

Number of tasks	Opt.	GA_N1	(GA_N1-Opt.)/ Opt.	Computation time(s)	GA_N2	(GA_N2-Opt.)/ Opt.	Computation time(s)
J30	48.20	48.75	0.011	0.09	48.75	0.011	0.09
J60	71.45	73.55	0.029	0.24	73.20	0.024	0.28
J90	84.05	87.90	0.046	0.50	87.00	0.035	0.76
J120	98.00	111.90	0.142	1.77	111.00	0.133	2.88
Average	75.425	80.813	0.057	0.649	80.25	0.051	1.004

Table 3 Average percent deviations from the critical-path based lower bound for J30

Algorithm	Reference	Maximum no. of schedules	
		1000	5000
Neurogenetic (FBI)	Agarwal et al. (2011)	0.13	0.10
SFLA-FBI	Fang and Wang (2012)	0.36	0.21
PSO	Jia and Seo (2013a, b)	0.49	–
GA	This study	0.64	0.56
BSO	Ziarati et al. (2011)	0.65	0.36
GA-late join	Coelho and Tavares (2003)	0.74	0.33
Sampling-global	Coelho and Tavares (2003)	0.81	0.54
ABC	Ziarati et al. (2011)	0.98	0.57
Sampling-WCS	Kolisch (1996a, b)	1.40	1.28
Sampling-LFT	Kolisch (1996a, b)	1.40	1.29
Sampling-random (S)	Kolisch (1995)	1.44	1.00
ACO	Chen et al. (2010a, b)	1.57	–
Sampling-random (P)	Kolisch (1995)	1.77	1.48
GA-problem space	Leon and Ramamoorthy (1995)	2.08	1.59
Filter and fan	Ranjbar (2008)	13.93	13.37

settings for other problem sizes. If GA settings are changed depends on the problem sizes, better results can be achieved for larger problem sizes. Overall, the GA_N1 deviates 5.7 % from the optimal solution, and the average computation time is 0.649 s while the GA_N2 deviates 5.1 % from the optimal solution, and the average computation time is 1.004 s. The GA_N2 outperforms the GA_N1 in terms of makespan. However, the GA_N1 outperforms the GA_N2 in terms of computation time. This is a tradeoff between solution quality and the time response to the users that a manager needs to consider. Table 2 shows that the proposed GA can generate good quality solutions quickly, even in large problem instances.

Comparison with other algorithms

Next, we compare the performance of the GA_N2 developed in this work with the existing algorithms. We also compare it with benchmark problem instances taken from PSPLIB. We have selected the test sets J30, J60, and J120. Sets J30 and J60 consist of 480 instances. Set J120 consists of 600 instances. The effectiveness of the algorithm proposed is compared with a number of state-of-the-art algorithms for the RCPSP. In order to provide a basis for the comparison, the number of generated and evaluated schedules, has been fixed to 1000 and 5000. Tables 3, 4, 5 display the results obtained from our GA and other tested algorithms. In these tables, we present the type of algorithms, the reference of each heuristic and the average percent deviations from the critical-path based lower bound (from the optimal solution for J30 instances) for 1000 and 5000 schedules, respectively. In each table, the algorithms are sorted according to descending performance, with respect to 1000 schedules. Table 3 summarizes the average percentage deviations from the optimal solution for the instance set J30. The average deviation of the proposed GA from the optimal solution is 0.64 and 0.56 % for 1000 and 5000 schedules, respectively. Tables 4, 5 summarized the average percentage deviations from the critical-path based lower bound for the instance sets J60 and J120, respectively. The average deviation of the proposed GA from the critical-path based lower bound is 12.73 and 12.38 % for 1000 and 5000 schedules for J60 set. The average deviations are 41.2

Table 4 Average percent deviations from the critical-path based lower bound for J60

Algorithm	Reference	Maximum no. of schedules	
		1000	5000
SFLA-FBI	Fang and Wang (2012)	11.44	10.87
Neurogenetic (FBI)	Agarwal et al. (2011)	11.51	11.29
PSO	Jia and Seo (2013a, b)	12.12	–
GA	This study	12.73	12.38
ABC	Shi et al. (2010)	12.75	11.48
GA-late join	Coelho and Tavares (2003)	13.28	12.63
Sampling-LFT	Kolisch (1996a, b)	13.59	13.23
Sampling-WCS	Kolisch (1996a, b)	13.66	13.21
TS-schedule scheme	Baar et al. (1998)	13.80	13.48
Sampling-global	Coelho and Tavares (2003)	13.80	13.31
Sampling-LFT	Kolisch (1996a, b)	13.96	13.53
GA-problem space	Leon and Ramamoorthy (1995)	14.33	13.49
ABC	Ziarati et al. (2011)	14.57	13.12
GA-random key	Hartmann (1998)	14.68	13.32
Sampling-random (P)	Kolisch (1995)	14.89	14.30
Sampling-random (S)	Kolisch (1995)	15.94	15.17

Table 5 Average percent deviations from the critical-path based lower bound for J120

Algorithm	Reference	Maximum no. of schedules	
		1000	5000
Neurogenetic (FBI)	Agarwal et al. (2011)	34.65	34.15
SFLA-FBI	Fang and Wang (2012)	34.83	33.20
ACOSS	Chen et al. (2010a, b)	35.19	32.48
PSO	Jia and Seo (2013a, b)	37.22	–
GA	This study	41.20	39.52
Sampling-global	Coelho and Tavares (2003)	41.36	40.46
Sampling-adaptive	Kolisch and Drexl (1996)	41.37	40.45
SA-activity list	Bouleimen and Lecocq (2003)	42.81	37.68
Sampling-LFT	Kolisch (1996a, b)	42.84	41.84
GA-problem space	Leon and Ramamoorthy (1995)	42.91	40.69
ABC	Ziarati et al. (2011)	43.24	39.87
Sampling-random (P)	Kolisch (1995)	44.46	43.05
GA-random key	Hartmann (1998)	45.82	42.25
Sampling-random (S)	Kolisch (1995)	49.25	47.61
AS	Kolisch and Hartmann (2006)	49.25	–
s-ACO	Merkle et al. (2002)	–	39.82

and 39.52 % for 1000 and 5000 schedules for J120 set. These results are competitive with other techniques used in the literature.

Results for computing resources allocation in a cloud manufacturing system

As mentioned above, for a given RCPSP, graphs are used and the successors and predecessors of each node are recorded by adjacent lists. Since many tasks are submitted to a resource for service and the resource is needed to handle the tasks sequentially, we vary the task durations to reflect the situation of resource allocation in cloud manufacturing. The duration of each task is randomly generated from uniform distribution $[0.5p_j, 2.0p_j]$. Moreover, we vary the level of parameters to see how they influence the performance of the GA. Instead of comparing all combinations of parameter settings individually, we fix all parameters and change the level of a parameter in one setting at a time. For example, the second row of Table 6 shows the results of a changing

Table 6 Experiment results with different parameter settings of the GA (120 tasks)

Parameter settings		GA_N1			GA_N2		
		Average makespan	Computation time	Deviation[a]	Average makespan	Computation time	Deviation[a]
Crossover rate	0.4	131.05	1.91	0.003	129.60	2.93	−0.008
	0.6	130.65	1.89	0.000	128.60	3.35	−0.016
	0.8	130.75	1.77	0.001	129.25	2.65	−0.011
Mutation rate	0.05	132.60	1.64	0.015	130.35	2.90	−0.002
	0.1	131.75	1.76	0.008	130.05	2.92	−0.005
	0.2	130.65	1.89	0.000	128.60	3.35	−0.016
Number of generation	20	130.65	1.89	0.000	128.60	3.35	−0.016
	100	130.65	1.89	0.000	128.60	3.30	−0.016
	200	130.65	1.89	0.000	128.60	3.28	−0.016
Replacement percentage	10	132.25	1.59	0.012	129.90	3.03	−0.006
	20	130.65	1.89	0.000	128.60	3.35	−0.016
	50	130.00	2.23	−0.005	128.60	3.20	−0.016
Population size	30	130.65	1.89	0.000	128.60	3.35	−0.016
	100	129.15	2.57	−0.011	128.05	3.79	−0.020
	150	129.10	3.23	−0.012	127.70	4.26	−0.023
Generations that cannot achieve a better solution	9	130.65	1.89	0.000	128.60	3.35	−0.016
	15	130.35	2.16	−0.002	128.25	3.33	−0.018
	30	129.90	2.20	−0.006	128.50	3.27	−0.016
Local search level	0	135.70	0.38	0.039	135.70	0.38	0.039
	3	132.15	0.57	0.011	130.55	0.73	−0.001
	5	130.65	1.89	0.000	128.60	3.35	−0.016
Number of best local search	0	135.70	0.38	0.039	135.70	0.38	0.039
	3	130.65	1.89	0.000	128.60	3.35	−0.016
	5	130.55	7.76	−0.001	128.15	16.21	−0.019

[a] Deviation is calculated by (average makespan- standard setting of GA_N1)/standard setting of GA_N1

crossover rate to 0.4, and all the other parameters are kept the same, as defined in Sect. 4.1 under Parameterization. For each parameter settings, 20 instances have been generated and tested. Table 2 shows that the GA has a larger deviation from the optimal solution for projects with 120 tasks. We fix problem size to 120 tasks instances since it is more challenging.

Table 6 demonstrates the computational results for computing resources allocation in a cloud manufacturing system with different parameter settings of the GA. If we use the parameter settings defined in Sect. 4.1 under Parameterization, the average makespan obtained is 130.65 for the GA_N1. We called it standard setting of the GA_N1. We use it as a benchmark to evaluate the performance of each parameter setting by calculating (average makespan- standard setting of the GA_N1)/standard setting of the GA_N1 that is listed under the deviation column. In the deviation column, a positive value represents that the corresponding settings of GA has found solutions that are worse than the standard setting of GA_N1's solutions (130.65). A negative value represents that the corresponding settings of the GA has found solutions that are better than the standard setting of the GA_N1's solutions. It is obvious that in all cases GA_N2 performs equal to or better than GA_N1 when solution quality is concerned. However, the mangers might prefer GA_N1 since computation time is an important factor in cloud manufacturing for fast response to customer request. Except for the number of generation, the results show that all parameters have influences on the performance of the GA. The mutation rate, population size, local search level, and number of best local search have significant influences on the performance of the GA. Increasing mutation rate, population size, local search level, or number of best local search can lead to a smaller makespan with the effect of increasing computation time. When local search level (number of best local search) set to 0, it means that local search has been disabled. Both the GA_N1 and GA_N2 perform poorly when local search has not been implemented. Among all the settings, the GA_N2 performs the best when population size is set to 150 and the average computation time for it is 4.26 s. When both local search level and number of best local search set to 5, the computation time for GA_N2 is the longest.

Conclusions and future works

Fast optimal allocation of computing resources is one of the most important problems in cloud manufacturing. In this study, a GA-based algorithm has been proposed to solve the problem of computing resource allocation in a cloud manufacturing system. We focus on the computation time which is an important factor in cloud manufacturing for fast response to customer request. Experiments demonstrate the effectiveness of the proposed GA, and show GA's high performance for solving optimally allocated computing resources in the cloud manufacturing system. To further improve the performance of the GA, a full design of experiment should be done for different problem sizes.

Due to the complex characteristics, determining how to find a feasible solution that can satisfy all users' demands with the minimum resources utilization in a short time is the most important matter in resource management in cloud manufacturing system. Hence, future works can study on designing more efficient algorithms that minimize makespan and resources utilization simultaneously.

References

Abrishami, S., & Naghibzadeh, M. (2012). Deadline-constrained workflow scheduling in software as a service cloud. *Scientia Iranica*, *19*(3), 680–689.

Agarwal, A., Colak, S., & Erenguc, S. (2011). A neurogenetic approach for the resource-constrained project scheduling problem. *Computer and Operations Research*, *38*(1), 44–50.

Alcaraz, A., & Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, *102*, 83–109.

Alcaraz, J., Maroto, C., & Ruiz. R. (2004). Improving the performance of genetic algorithms for the RCPS problem. In *Proceedings of the 9th International Workshop on Project Management and Scheduling* (pp. 40–43).

Baar, T., Brucker, P., & Knust, S. (1998). *Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem* (pp. 1–18). US: Springer.

Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, *44*, 262–275.

Bardsiri, A. K., & Hashemi, S. M. (2012). A review of workflow scheduling in cloud computing environment. *International Journal of Computer Science and Management Research*, *1*(3), 348–351.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, *149*(2), 268–281.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, *112*(1), 3–41.

Chen, R. M., Wu, C. L., Wang, C. M., & Lo, S. T. (2010). Using novel particle swarm optimization scheme to solve resource-constrained scheduling problem in PSPLIB. *Expert Systems with Applications*, *37*(3), 1899–1910.

Chen, W., Shi, Y. J., Teng, H. F., Lan, X. P., & Hu, L. C. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, *180*(5), 1031–1039.

Cheng, Y., Tao, F., Liu, Y., Zhao, D., Zhang, L., & Xu, L. (2013). Energy-aware resource service scheduling based on utility evaluation in cloud manufacturing system. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *227*(12), 1901–1915.

Coelho, J., & Tavares, L. (2003). Comparative analysis of metaheuristics for the resource constrained project scheduling problem. Technical Report, Department of Civil Engineering, Instituto Superior Tecnico.

Debels, D., & Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, *55*(3), 457–469.

Eshelman, L. J., & Schaffer, J. D. (1992). Real-coded genetic algorithms and interval schemata. In L. Darrel Whitley (Ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann Publishers.

Fang, C., & Wang, L. (2012). An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. *Computers and Operations Research*, *39*, 890–901.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading: Addison Wesley.

Gonçalves, J. F., Resende, M. G., & Mendes, J. J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, *17*(5), 467–486.

Grèze, L., Pellerin, R., Leclaire, P., & Perrier, N. (2014). CIGI2011: A heuristic method for resource-constrained project scheduling with activity overlapping. *Journal of Intelligent Manufacturing*, *25*(4), 797–811.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, *45*(6), 733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, *49*(5), 433–448.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *207*(1), 1–14.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MA: University of Michigan.

Jia, Q., & Seo, Y. (2013a). An improved particle swarm optimization for the resource-constrained project scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *67*(9–12), 2627–2638.

Jia, Q., & Seo, Y. (2013b). Solving resource-constrained project scheduling problems: conceptual validation of FLP formulation and efficient permutation-based ABC computation. *Computers and Operations Research*, *40*(8), 2037–2050.

Ji, X., & Yao, K. (2014). Uncertain project scheduling problem with resource constraints. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0980-x.

Kaur, N., Aulakh, T. S., & Cheema, R. S. (2011). Comparison of workflow scheduling algorithms in cloud computing. *International Journal of Advanced Computer Science and Applications*, *2*(10), 81–86.

Khanzadi, M., Soufipour, R., & Rostami, M. (2011). A new improved genetic algorithm approach and a competitive heuristic method for large-scale multiple resource-constrained project-scheduling problems. *International Journal of Industrial Engineering Computations*, *2*(4), 737–748.

Kim, B., Youn, C. H., Park, Y. S., Lee, Y., & Choi, W. (2012). An adaptive workflow scheduling scheme based on an estimated data processing rate for next generation sequencing in cloud computing. *JIPS*, *8*(4), 555–566.

Kochetov, Y., & Stolyar, A. (2003). Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*, Russia.

Kolisch, R., & Drexl, A. (1996). Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, *43*(1), 23–40.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, *174*, 23–37.

Kolisch, R. (1995). *Project scheduling under resource constraints: Efficient heuristics for several problem classes*. Heidelberg: Springer Press.

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, *14*(3), 179–192.

Kolisch, R. (1996b). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, *90*(2), 320–333.

Kolisch, R., & Sprecher, A. (1997). PSPLIB—A project scheduling problem library: OR Software—ORSEP operations research software exchange program. *European Journal of Operational Research*, *96*(1), 205–216.

Lee, G., Chun, B. G., & Katz, R. H. (2011). Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of HotCloud* (pp. 1–5).

Laili, Y., Tao, F., Zhang, L., Cheng, Y., Luo, Y. L., & Sarker, B. R. (2013). A ranking chaos algorithm for dual scheduling of cloud service and computing resource in private cloud. *Computers in Industry*, *64*, 448–463.

Laili, Y., Tao, F., Zhang, L. & Ren, L. (2011), The optimal allocation model of computing resources in cloud manufacturing system. In *IEEE Seventh International Conference on Natural Computation*.

Laili, Y. J., Tao, F., Zhang, L., & Sarker, B. R. (2012). A study of optimal allocation of computing resources in cloud manufacturing systems. *International Journal of Advanced Manufacturing Technology*, *63*, 671–690.

Leon, V. J., & Ramamoorthy, B. (1995). Strength and adaptability of problem space based neighborhoods for resource-constrained scheduling. *OR Spectrum*, *17*(2–3), 173–182.

Lodha, P. R., & Wadhe, M. A. P. (2013). Study of different types of workflow scheduling algorithm in cloud computing. *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, *2*(4), 421.

Mehra, M., Jayalal, M. L., Arul, A. J., Rajeswari, S., Kuriakose, K., & Satya Murty, S. A. V. (2014). Study on different crossover mechanisms of genetic algorithm for test interval optimization for nuclear power plants. *International Journal of Intelligent Systems and Applications*, *6*(1), 20–28.

Mendes, J. J., Goncalves, J. F., & Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers and Operations Research*, *36*(1), 92–109.

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, *6*(4), 333–346.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, *42*(6), 797–813.

Rahman, M., Hassan, R., Ranjan, R., & Buyya, R. (2013). Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency and Computation: Practice and Experience*, *25*(13), 1816–1842.

Ranjbar, M. (2008). Solving the resource constrained project scheduling problem using filter-and-fan approach. *Applied Mathematics and Computation*, *201*, 313–318.

Shi, Y. J., Qu, F. Z., Chen, W., & Li, B. (2010). An artificial bee colony with random key for resource-constrained project scheduling. In *Life System Modeling and Intelligent Computing* (pp. 148–157). Springer, Berlin Heidelberg.

Shue, L. Y., & Zamani, R. (1999). An intelligent search method for project scheduling problems. *Journal of Intelligent Manufacturing*, *10*(3–4), 279–288.

Tao, F., Laili, Y., Xu, L., & Zhang, L. (2012). FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system. *IEEE Transactions on Industrial Informatics*, *9*(4), 2023–2033.

Valls, V., Ballestin, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *185*(2), 495–508.

Varalakshmi, P., Ramaswamy, A., Balasubramanian, A., & Vijaykumar, P. (2011). An optimal workflow based scheduling and resource allocation in cloud. In *Advances in Computing and Communications* (pp. 411–420). Springer, Berlin Heidelberg.

Wall B. W. (1996). A Genetic Algorithm for Resource Constrained Scheduling, PhD Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, USA.

Wu, D., Greer, M. J., Rosen, D. W., & Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing System*, *32*(4), 564–579.

Wu, L., & Yang, C. (2010). A solution of manufacturing resources sharing in cloud computing environment. *Cooperative Design, Visualization and Engineering* (pp. 247–252). Berlin Heidelberg: Springer.

Yassa, S., Chelouah, R., Kadima, H., & Granado, B. (2013). Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*.

Zhang, Q., Zhu, Q., & Boutaba, R. (2011). Dynamic resource allocation for spot markets in cloud computing environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on IEEE* (pp. 178–185).

Zamani, R. (2013). A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. *European Journal of Operational Research*, *229*(2), 552–559.

Ziarati, K., Akbari, R., & Zeighami, V. (2011). On the performance of bee algorithms for resource-constrained project scheduling problem. *Applied Soft Computing*, *11*(4), 3720–3733.