



Research article

MEMINV: A hybrid efficient approximation method solving the multi skill-resource constrained project scheduling problem

Huu Dang Quoc

Department of Economic Information System and Electronic Commerce, Thuong Mai University, 79 Ho Tung Mau, Cau Giay, Ha Noi, Viet Nam.

Correspondence: Email: huudq@tmu.edu.vn.

Abstract: The Multi-Skill Resource-Constrained Project Scheduling Problem (MS-RCPSP) is an NP-Hard problem that involves scheduling activities while accounting for resource and technical constraints. This paper aims to present a novel hybrid algorithm called MEMINV, which combines the Memetic algorithm with the Inverse method to tackle the MS-RCPSP problem. The proposed algorithm utilizes the inverse method to identify local extremes and then relocates the population to explore new solution spaces for further evolution. The MEMINV algorithm is evaluated on the iMOPSE benchmark dataset, and the results demonstrate that it outperforms. The solution of the MS-RCPSP problem using the MEMINV algorithm is a schedule that can be used for intelligent production planning in various industrial production fields instead of manual planning.

Keywords: memetic algorithm; evolutionary computing; optimization; MS-RCPSP problem

1. Introduction

The MS-RCPSP [1–5] is a complex scheduling problem widely studied in operations research and project management. It is a type of project scheduling problem that considers the order in which tasks should be performed and the resources required to perform each task, such as labor, equipment and materials. In the MS-RCPSP, resources are considered limited, finite and reused, meaning there are restrictions on the number of renewable resources available at any given time. The goal of solving the MS-RCPSP is to minimize the project's completion time, cost or both (multi-objective) while ensuring that all resource constraints are satisfied. It is a widely studied problem with numerous real-

world applications, including construction, software development and manufacturing.

The MS-RCPSp problem has an important constraint: a resource can only perform the task if it possesses the right skill type and the skill level is greater than or equal to the required skill level. This constraint accurately describes reality. Moreover, resources have many skill types and skill levels, so it is necessary to plan for allocating resources to perform practical tasks to improve the efficiency of project implementation. MS-RCPSp is a problem of class NP-Hard, so the optimal solution cannot be found in polynomial time. However, metaheuristic methods can solve the problem to find approximate solutions in less time. In order to find a suitable solution for the MS-RCPSp problem, this paper proposes the MEMINV algorithm developed from the Memetic algorithm [6–8] integrated with the Inversion method. The solution to the MS-RCPSp problem when applying MEMINV is a schedule that allocates resources to perform project tasks.

The major contributions of this paper involve the following:

- Proposing the Inversion method to detect the local extreme of populations in the evolutionary process over the generations. The local extreme detection is based on examining the number of successive evolutionary generations of a population without changing the total project time.
- Proposing the Adaptive Localserach method to dynamically change the parameters during the algorithm's evolution, including mutation coefficient, crossover coefficient and the number of neighboring individuals used to perform the search.
- Proposing hybrid algorithm MEMINV based on combining the Memetic algorithm and the Inversion method to improve problem-solving efficiency.
- Conduct experiments on the iMOPSE[4,5] dataset to verify the proposed algorithm.

The remainder of this paper is organized as follows: Part 2 provides an overview of related works in the RCPSp and MS-RCPSp problems and studies existing research on evolutionary methods employed to address these problems. Additionally, this section introduces the Memetic algorithm, known for its effectiveness in solving NP-Hard class problems. Section 3 describes the mathematical formulation of the problem, including the mathematical constraints and their implications. Part 4 presents the MEMINV algorithm, a hybrid approach combining the Memetic algorithm with the Inversion method, aimed at enhancing solution efficiency.

2. Background and Literature review

The MS-RCPSp [2–5,9–11] is a sub-classification of the Resource-Constrained Project Scheduling Problem (RCPSp), which has been proven to be an NP-Hard class, meaning that their solutions cannot be found in polynomial time. Therefore, it is common to use approximate, evolutionary algorithms to find an acceptable solution quickly.

2.1. Approximate methods to solve the Resource-Constrained Project Scheduling Problem

Numerous scientists have extensively researched and published various methods to discover practical solutions for RCPSp and MS-RCPSp problems. These solutions are primarily based on evolutionary computation algorithms, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE) and Cuckoo Search (CS). Table 1 summarizes some outstanding works to solve those problems.

Table 1. Summary of early works.

No.	Based on Algorithm	Problem	References
1		RCPSP	13,14,15,16
2	GA	NPV-RCPSP	12
3		MS-RCPSP	17,18,19,20
4	PSO	RCPSP	21,23
5		MS-RCPSP	2,22
6	DE	RCPSP	24,25
7		MS-RCPSP	26
8	CS	MS-RCPSP	31,32

Many authors widely use the GA algorithm to solve the RCPSP family of problems.

In [12], the authors use the GA integrated with the Immune to solve the NPV-Based RCPSP problem. The algorithm is also applied with two additional variables to improve efficiency: local search and forward-backward improvement. Mateja Đumić et al. [13] using GA combined with many different methods to determine the feasible schedule of RCPSP. In [14], O. Shuvo et al. consider RCPSP by integrating chemical reaction optimization and genetic algorithm (CRO-GA), and the authors have redesigned the basic operators of CRO and GA and the priority-based selection operator to find out the schedules. The authors [15] introduce the quantum-inspired genetic algorithm (QIGA), which is an adjusted GA algorithm to solve the RCPSP problem. In this algorithm, the authors improve the initialization and update stages population using quantum gates and superposition. In paper [16], the authors employ a combination of GA (Genetic Algorithm) and a two-point crossover operator to tackle the resource-constrained project scheduling problem with transfer times (RCPSPPTT).

The GA algorithm is used to solve the MS-RCPSP. In [17], the authors proposed the genetic programming hyper-heuristic (GP-HH) algorithm, which has many improvements to improve solution quality, including repair-based decoding of a solution, using ten simple heuristic rules designed to construct a set of low-level heuristics, using GA with a high-level strategy and the design-of-experiment (DOE) method is employed to investigate the effect of parameters setting. The performance of GP-HH is evaluated on the iMOPSE dataset. The authors [18] use the breadth and depth methods hybrid with the GA algorithm. In [19], the authors proposed the MOGP-HH/D algorithm to find out the multi-objective of MS-RCPSP, which uses many technicals, including NSGA-II algorithm, local search... design-of-experiment (DOE) by Taguchi method. In their study [20], the authors used several methods to obtain a multi-objective solution for the problem. The first utilized a priority rule for population initialization, followed by applying the NSGA-II algorithm to facilitate the search process. Finally, they integrated a local search into the search process to enhance the quality of the results

Particle Swarm Optimization (PSO) is another commonly used metaheuristic for solving scheduling problems. The authors of the paper [21] proposed an algorithm that is improved from PSO to solve RCSPS. The new algorithm uses the "Valid Particle Generator" technique to detect feasible solutions and, at the same time, applies more adaptive methods to improve the quality of the solutions. In [2], the authors proposed a novel M-PSO algorithm to solve MS-RCPSP. The M-PSO is a hybrid

between the PSO algorithm and the Migration method that supports the population escapes from local extrema. D. Q. Huu et al. [22] combined PSO with the re-assignment technical to recalculate the resource assigned to execute the task after each generation. The proposed algorithms are conducted with the iMOPSE dataset. In [23], the authors used PSO for the resource-constrained project scheduling problem with varying resource levels (RCPSPVRL), an RCPSP extension. In the RCPSPVRL, the total project duration is divided into many periods, each requiring a different quantity of resources.

Additionally, the DE algorithm discovers extensive applications among various research groups for solving scheduling problems. In [24], the authors proposed a two-stage multi-operator differential evolution (DE) algorithm to solve RCPSP. The algorithm processes each generation through two stages, including the exploration stage, and based on the diversity of the population and the quality of solutions, this approach dynamically places more importance on the most-suitable DE and then repeats the same process during the exploitation phase. In [25], improving DE (IDE) with the mutation technique is used to speed up the algorithm's convergence speed. The author evaluates the new algorithm and compares it with other algorithms, such as HGA and PSO, to demonstrate its effectiveness. The authors [26] consider a multi-objective MS-RCPSP problem with priority constraints to find the minimum deviation from the expected time to complete each project and allocate resources. The schedule alternative is found based on the improved DE algorithm.

Researchers have also proposed and developed some variations of the MS-RCPSP problem. In [27], the author extended MS-RCPSP to resource scheduling and cooperative multi-robot system problem task allocation. for SAR (RSTA-RSSAR). In RSTA-RSSAR, the skills of multi-robot systems are considered from both depth and breadth. In it, the processing time of activities in RSTA-RSSAR changes with skill ability resources provided. In order to solve the problem effectively, a genetic differential evolution algorithm (PS-GDEA) is proposed. Another variant is Real-RCPSP [28], which extends from MS-RCPSP by adding a constraint on task execution time that varies according to resource ability usage instead of a fixed duration. The authors also proposed an algorithm to solve the Real-RCPSP problem based on the Adaptive method combined with the DE algorithm called A-DEM.

Furthermore, numerous research groups have embraced the utilization of the CS algorithm as a valuable tool for addressing scheduling problems effectively. The authors use the Cuckoo Search (CS) [29,30] algorithm to solve the scheduling problem. The CS uses Lévy Flight random walk to evolve the population, using two search techniques: Local Search and Global Search. Authors of [31,32] use this metaheuristic to solve the MS-RCPSP.

Moreover, many other researchers have also proposed state-of-the-art algorithms that offer practical solutions for finding feasible schedules for the MS-RCPSP problem. The authors [33] recommend the multi-objective evolution strategy (MOES) framework to discover multi-objective solutions by focusing on generating and using mutation operators for good scheduling. In [34], the authors proposed a discrete oppositional multi-verse optimization (DOMVO) algorithm with many processing steps to get high effective such as using the black/white holes technique combined with the path relinking technique, the opposition-based learning (OBL), a repair-based decoding scheme, and the design-of-experiment (DOE) method.

Usually, to experiment with the proposed algorithms, the authors often use two primary datasets, PSLIB [35] and iMOPSE [4,5], which was suggested by P. B. Myszkowski.

2.2. Memetic algorithm

Memetic algorithms (MAs) [6–8] are optimization algorithms that incorporate both evolutionary and local search procedures to solve complex optimization problems. MAs have gained increasing popularity in recent years due to their ability to overcome the limitations of traditional evolutionary algorithms (EAs) by incorporating problem-specific knowledge to guide the search process.

In a memetic algorithm, individuals in the population are encoded as solutions to the optimization problem and evolve through selection, crossover (recombination) and mutation. The genetic algorithm generates new candidate solutions, while the local search method is used to refine the solutions generated by the genetic algorithm.

Using local search methods in memetic algorithms can lead to faster convergence and improved solution quality compared to traditional genetic algorithms. This is because local search methods can exploit the structure of the problem space to find better solutions quickly. Moreover, the genetic algorithm provides a mechanism for exploring the search space and avoids getting trapped in local optima.

The Memetic algorithm performs the following steps:

- Step 1. Initialization: Generate an initial population of candidate solutions, each represented as a chromosome.
- Step 2. Fitness evaluation: Evaluate the fitness of each chromosome in the population using a fitness function.
- Step 3. Selection: Select pre-chromosomes for reproduction based on their fitness values.
- Step 4. Crossover: Combine the selected pre-chromosomes to create offspring chromosomes through crossover (recombination) operation.
- Step 5. Mutation: Introduce random variations into the offspring chromosomes through mutation operation.
- Step 6. Local search: To improve their solutions, perform a local search on some of the offspring chromosomes.
- Step 7. Acceptance: Accept the offspring as the new population if they are better than the current population, otherwise keep the current population.
- Step 8. Termination: Terminate the algorithm if a satisfactory solution is found or a stopping criterion is met.
- Repeat steps 2 to 8 to create the next generation of solutions.

However, there are also some disadvantages to using MAs. One of the main drawbacks is that MAs can be computationally expensive, as they require both global and local search procedures. Another challenge is designing effective, problem-specific and efficient local search procedures can be complex. Furthermore, MAs can also suffer from premature convergence, where the algorithm becomes stuck in a local optimum and cannot escape finding the global optimum.

3. MS-RCPSP problem model

The MS-RCPSP has a wide range of real-world applications, such as in project management, scheduling of manufacturing processes and resource allocation in large-scale engineering projects. The ability to effectively solve the MS-RCPSP has significant implications for the efficiency and success

of these real-world applications.

To solve the MS-RCPSP, advanced optimization techniques are required to find the optimal schedule. Solving the MS-RCPSP is to minimize the makespan, or the total time required to complete the project while ensuring that all resource constraints are satisfied. The solution must also consider any dependencies between tasks, meaning that predecessor tasks must be completed before others begin.

3.1. Resource-Constrained Project Scheduling Problem (RCPSP) and MS-RCPSP

RCPSP (Resource-Constrained Project Scheduling Problem) [1,9–12] is a problem of finding a schedule to complete project tasks with limited renewable resources (usually smaller than the number of tasks). The RCPSP problem is to find the best schedule to perform the project tasks under resource constraints, and the tasks have a fixed order of execution. The objective function of the RCPSP problem is evaluated based on the execution time (makespan), the cost of implementation (cost) or a combination of both (multi-objective).

In RCPSP, multiple tasks are performed, and each task is described by start and end times with some significant constraints as follows:

- When a task starts, it cannot be stopped until it is finished.
- Tasks are related to each other in order of execution. That is, the predecessors' task needs to be completed before the start of the successor's task.
- Resources are finite and can be reused for other tasks. The number of resources allocated should not exceed the amount available. Although a task can use multiple resources, a resource can be used for only one task concurrently.

Multi-Skill (MS) RCPSP problem [2–5] is extended from the RCPSP. The MS-RCPSP added a new constraint to the resources: each resource has many skills and a certain level. Therefore, each task will require a specific skill type and skill level resource. The MS-RCPSP is a widely studied problem with numerous real-world applications, including construction, software development and manufacturing.

Example 1: A project with 10 tasks and 3 renewable resources.

- Tasks that require resources with the skill level required to perform are shown in Table 2.
- Resources with complementary skills and skill levels are shown in Table 3.

The notations:

- W_i : Task i , L_j : Resource j
- S_{ij} represents a skill type i with a skill level of j .

Table 2. The task requirement.

Task	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}
Required skill of the resource	$S_{2.1}$	$S_{1.2}$	$S_{1.3}$	$S_{3.1}$	$S_{2.2}$	$S_{1.3}$	$S_{2.2}$	$S_{3.2}$	$S_{3.1}$	$S_{1.2}$

Based on the resource requirements for execution and the capacity of the given resources, we can build a matrix describing the resource's ability to perform the task, as shown in Figure 1 below.

Table 3. The skills of resources.

Resource	L ₁	L ₂	L ₃	L ₄
Skills	S _{1.3}	S _{1.2}	S _{1.3}	S _{2.2}
	S _{2.1}	S _{3.2}	S _{3.1}	S _{3.1}

Task \ Resource	L ₁	L ₂	L ₃	L ₄
W ₁	✓	✗	✗	✓
W ₂	✓	✓	✓	✗
W ₃	✓	✗	✓	✗
W ₄	✗	✓	✓	✓
W ₅	✗	✗	✗	✓
W ₆	✓	✗	✓	✗
W ₇	✗	✗	✗	✓
W ₈	✗	✓	✗	✗
W ₉	✗	✓	✓	✓
W ₁₀	✓	✓	✓	✗
✓ : the resource can be executed the task				
✗ : the resource can not be executed the task				

Figure 1. Resource's ability to perform the tasks.

3.2. Problem definitions

The MS-RCPSP problem can be conceptually formulated based on the notations in Table 4.

Table 4. The notations.

Symbol	Description
C_i	The set of tasks need to be completed before task i can be executed
S	The set of all resource's skills S^i : the subset of skills owned by the resource i , $S^i \subseteq S$;
S_i	The skill i ;
t_j	The duration of task j
L	The resources used to execute tasks of the project
L^k	The subset of the resources which can be performed task k ; $L^k \subseteq L$
L_i	The resource i
W	The tasks of the project need to do
W^k	The subset of task which can be executed by the resource k , $W^k \subseteq W$
W_i	The task i

r^i	The subset of the skill required by task i . A resource has the same skill and skill level equal to or greater than the requirement that can be performed.
B_k, E_k	The begin time and end time of the task k
$A_{u,v}^t$	The variable to identify the resource v is running task u at time t ; 1: yes, 0: no;
h_i	The skill level i ;
g_i	Type of skill i ;
m	Makespan of the schedule
P	The feasible solution
P_{all}	The set of all solution
$f(P)$:	The function to calculate the makespan of P solution
n	Task number
z	Resource number

The MS-RCPSP problem could be state as follow:

$$f(P) \rightarrow \min \quad (1)$$

Where:

$$f(P) = \max_{W_i \in W} \{E_i\} - \min_{W_k \in W} \{B_k\} \quad (2)$$

Subject to the following constraints:

$$\bullet \quad S^k \neq \emptyset \quad \forall L_k \in L \quad (3)$$

$$\bullet \quad t_{jk} \geq 0 \quad \forall W_j \in W, \quad \forall L_k \in L \quad (4)$$

$$\bullet \quad E_j \geq 0 \quad \forall W_j \in W \quad (5)$$

$$\bullet \quad E_i \leq E_j - t_j \quad \forall W_j \in W, j \neq i, W_i \in C_j \quad (6)$$

$$\bullet \quad \forall W_i \in W^k \exists S_q \in S^k : g_{S_q} = g_{r_i} \text{ and } h_{S_q} \geq h_{r_i} \quad (7)$$

$$\bullet \quad \forall L_k \in L, \forall q \in m : \sum_{i=1}^n A_{i,k}^q \leq 1 \quad (8)$$

$$\bullet \quad \forall W_j \in W \exists ! q \in [0, m], ! L_k \in L : A_{j,k}^q = 1; \text{ where } A_{j,k}^q \in \{0; 1\} \quad (9)$$

The above constraints have the following meanings:

- Constraint (3) guarantees that each resource has at least one skill
- Constraints (4, 5) say that the execution time of any task must be at least 0 (in fact, every real task, with a minimum execution time, is always greater than 0, the case is zero to illustrate two dummy tasks representing the start and end of the project)
- Constraint (6) implies that the predecessor task (task i) must end before the successor task (task j) starts. The time when task i ends is denoted E_i , and the time when successor task j starts is $E_j - t_j$ (finish time minus execution time).
- Constraint (7) ensures that for every task $i \in W_k$ (set of tasks that resource k can perform),

there is always a skill $S \in S_k$ (skill set of resource k) such that $g_S = g_{S_i}$: r 's skill type coincides with L_i 's skill type that task i requires. $h_{S_q} \geq h_r^i$: the skill level of the performing resource is higher or equal to the required skill level.

- Constraint (8) states that at each time (q), each resource can only perform at most one task. If $\sum_{i=1}^n A_{i,k}^q = 0$, then resource k is not assigned to any task. If $\sum_{i=1}^n A_{i,k}^q = 1$, then resource k is assigned to a single task.

- Constraint (9) ensures that each task is assigned to only one renewable resource and performed by only one resource.

In the MS-RCPSP problem, each task has additional skill requirements of the renewable resource needed to perform. Each resource is also divided into different skill types and skill levels. Figure 1 depicts an example of the resource skill constraints required to complete a task.

4. Proposed algorithm

This section describes the proposed evolutionary algorithm for the MS-RCPSP problem named MEMINV, a variation of the Memetic algorithm [6-8] enhanced with the Inversion method.

4.1. Individual presentation

In the MS-RCPSP problem, the population is calculated and evolved over several generations to improve the makespan of the project. Representing each individual in the population is a vector whose elements correspond to the total number of project tasks. The value at each vector element denotes the resource index assigned to perform the corresponding task.

Considering the project in example 1, an individual can be represented as a vector as shown in Figure 2 below.

<i>Task index</i>	1	2	3	4	5	6	7	8	9	10
<i>Resource index</i>	4	3	1	2	4	1	4	2	3	3




Figure 2. Individual vector.

4.2. Adaptive Local Search method

The goal of the adaptive method is to adjust the crossover parameter μCR through each generation to increase the algorithm's efficiency. The adjustment depends on the results of the evolutionary performance of the population. The specific steps are as follows:

- The crossover parameter of the i th individual, denoted CR_i , is calculated by the Cauchy random function of the crossover parameter CR . The CR is calculated based on the number of successful evolution individuals.

- When implementing the MEMINV algorithm, instead of using a fixed number of neighboring individuals at the local search step, the algorithm dynamically calculates and changes the number of

neighboring individuals based on the number of individuals that successfully evolve.

- Suppose the population has ten individuals, with the project duration given in Figure 3. It is necessary to find three neighboring individuals with P_5 , the algorithm runs through the following steps:
 - Step 1: Sort the population in ascending order of the project execution time of each individual (calculated using the objective function); the result is shown in Figure 4.
 - Step 2: Considering the sorted list of P_{all} to calculate the distance with the P_5 individual, it gets $d_5 = 13$.
 - Step 3: Considering P_{all} to get three individuals adjacent to P_5 . Neighboring individuals obtained include $P^5_S = \{P_2, P_4, P_6\}$.

P_{all}	Project time
P_1	123
P_2	130
P_3	175
P_4	127
P_5	137
P_6	150
P_7	201
P_8	185
P_9	162
P_{10}	173

Figure 3. The population to find neighbors.

P_{all} sorted	Project time	d_5
P_1	123	14
P_4	127	10
P_2	130	7
P_5	137	0
P_6	150	13
P_9	162	25
P_{10}	173	36
P_3	175	38
P_8	185	48
P_7	201	64

Figure 4. Finding the neighboring individuals by distance from d_5 .

The Adaptive local search is shown in the algorithm 1.

Algorithm 1. AdaptiveLocalSearch

Input: P_{all} : the population

current: the current individual to find the neighbors

μCR : crossover probability (*global variable*)

Output: P_{best}^i : the best individual gets from the neighboring individuals

Begin

1. $P_R = \{ \}$; $w_{min} = 3$; $w_{max} = 10$;
 2. $fitness \leftarrow f(P_{all})$
 3. $i = index(current)$ // the index of current individual in P_{all}
 4. $mp = fitness(i)$
 5. $N = sizeOf(P_{all})$
 6. $P_{all}^{sort} \leftarrow sort(P_{all}, fitness)$ // sort population by fitness
 7. $D_i = findDistance(i)$
 8. For $j=1$ to N
 9. If $abs(fitness(j) - mp) \leq D_i$
 10. $P_S = P_S + \{P_j\}$
 11. End if
 12. If $(sizeOf(P_S) > w)$ break;
 13. End for
 14. $P_{best}^i = fbest(P_S)$
 15. If $(P_{best}^i \neq localbest)$ $S_{CR} = S_{CR} + \{\mu CR\}$
 16. $CR_i = randci(\mu CR, 0.1)$
 17. $c = rand(0,1)$
 18. $\mu CR = (1 - c) \times \mu CR + c \times mean(S_{CR})$ //Adaptive factor
 19. $w = w_{max} - i/N \times (w_{max} - w_{min})$
 20. Return P_{best}^i
- End Function
-

where:

findDistance: function to find the maximum distance from position i (*index of the curent individual*)

to others to be able to get enough number of individuals neighboring of P_i .

4.3. Catching the local extreme

The traditional memetic algorithm operates by evolving a population over several generations through several steps, including selection, crossover, mutation, local search and recombination to create a new population. However, while evolving, the algorithms may fall into the local extreme and iterate infinitely without leaving, so finding a better result is impossible. Therefore, the Inversion technique moves the population to another solution space far from the local extreme. Integrating the Inversion technique into the memetic algorithm leads to improved results.

In this step, the algorithm uses a marked variable c_{fail} to count the times the makespan is not changed over generations; if c_{fail} was more significant than a specified threshold (c_{max}), then the population has fallen into the local extreme. Equation (10) represents the calculation of c_{fail} .

$$c_{fail} = \begin{cases} c_{fail} + 1 & \text{if } new_{makespan} = old_{makespan} \\ 0 & \text{if } new_{makespan} \neq old_{makespan} \end{cases} \quad (10)$$

4.4. Inversion method

Throughout the algorithm's evolution, upon detection of a local extreme, the Inversion method is invoked to facilitate the population's elimination of the local extreme. This is achieved through the following steps:

Step 1: Repeating with each individual

Step 2: Considering each task i of the individual, get out the L^i of resources to execute the current task, reordering the resources index in L^i .

Step 3: Replacing the current resource with the resource which mirrors in L^i . The formula (11) shows this action.

$$L^i = \{L_1, L_2, \dots, L_m\}, m \leq n$$

$$\forall W_i \in W, j \in [1, m] : L_j \leftarrow L_{m-j} \quad (11)$$

Figure 5 presents the Inversion technical by replacing the execution resource. Primarily, the task being performed by resource L_2 will be changed to L_{m-1} , and resource L_m does the task will be allocated to L_1 .

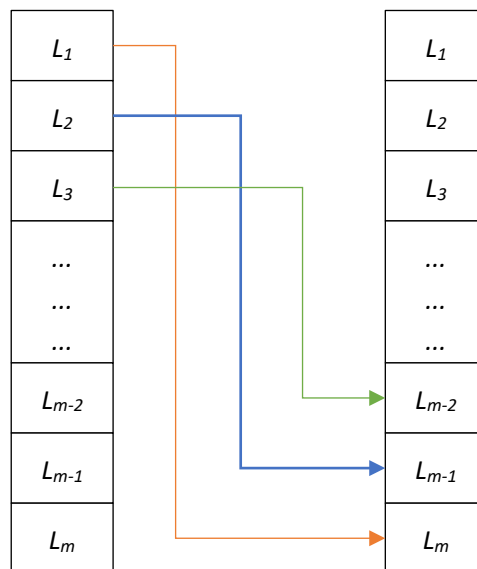


Figure 5. Replace the resource that performs the task.

After applying the Inversion method, the population will be moved to a new location, allowing the algorithm MEMINV to escape the local extreme, continue the evolutionary computation and expand the solution space.

The Inversion method is present in Algorithm 2 below.

Algorithm 2. InversionInput: P_{all} – the population needs to moveOutput: P_{new} – the result population

Begin

```

1.  {
2.   $P_{new} = \{ \}$ 
3.   $i = 1$ 
4.   $p\_size = size(P)$ 
5.  while(  $i < p\_size$  ) {
6.       $P_i \leftarrow P_{all}[i]$ ;
7.       $j\_task = 1$ 
8.       $n\_task = size(P_i)$ 
9.      for {  $j\_task = 1; j\_task < n\_task; j\_task++$  )
10.     {
11.          $L^i \leftarrow$  the resources matching with  $L(j\_task)$  requirement
12.          $L^i \leftarrow Reorder(L^i)$ 
13.          $i\_current \leftarrow$  current resource index to run task  $i$ 
14.          $i\_current = size(L^i) - idx + 1$ 
15.          $L_{j\_task} = L^i[i\_current]$ 
16.     } //  $j\_task$ 
17.      $P_{new} = P_{new} + \{P_i\}$ 
18. }
19. Return  $P_{new}$ 
20. }
```

4.5. MEMINV algorithm

The MEMINV algorithm is a hybrid algorithm from the memetic algorithm and the Inversion method. The steps of the algorithm are detailed in Figure 6.

In the Figure 6, ALC is Active Local Search method used to dynamically find the individual in the algorithm's evolution. The improvement of MEMINV is demonstrated in the ALC and "Inversion steps", with the ALC, the algorithm will conduct an individual based on dynamically changing the number of neighboring and the crossover coefficient in the evolutionary process to get the local best and "Inversion steps" will check for local extrema within the population and redirects it to another region if it has fallen into one. This step is executed by applying the Inversion method.

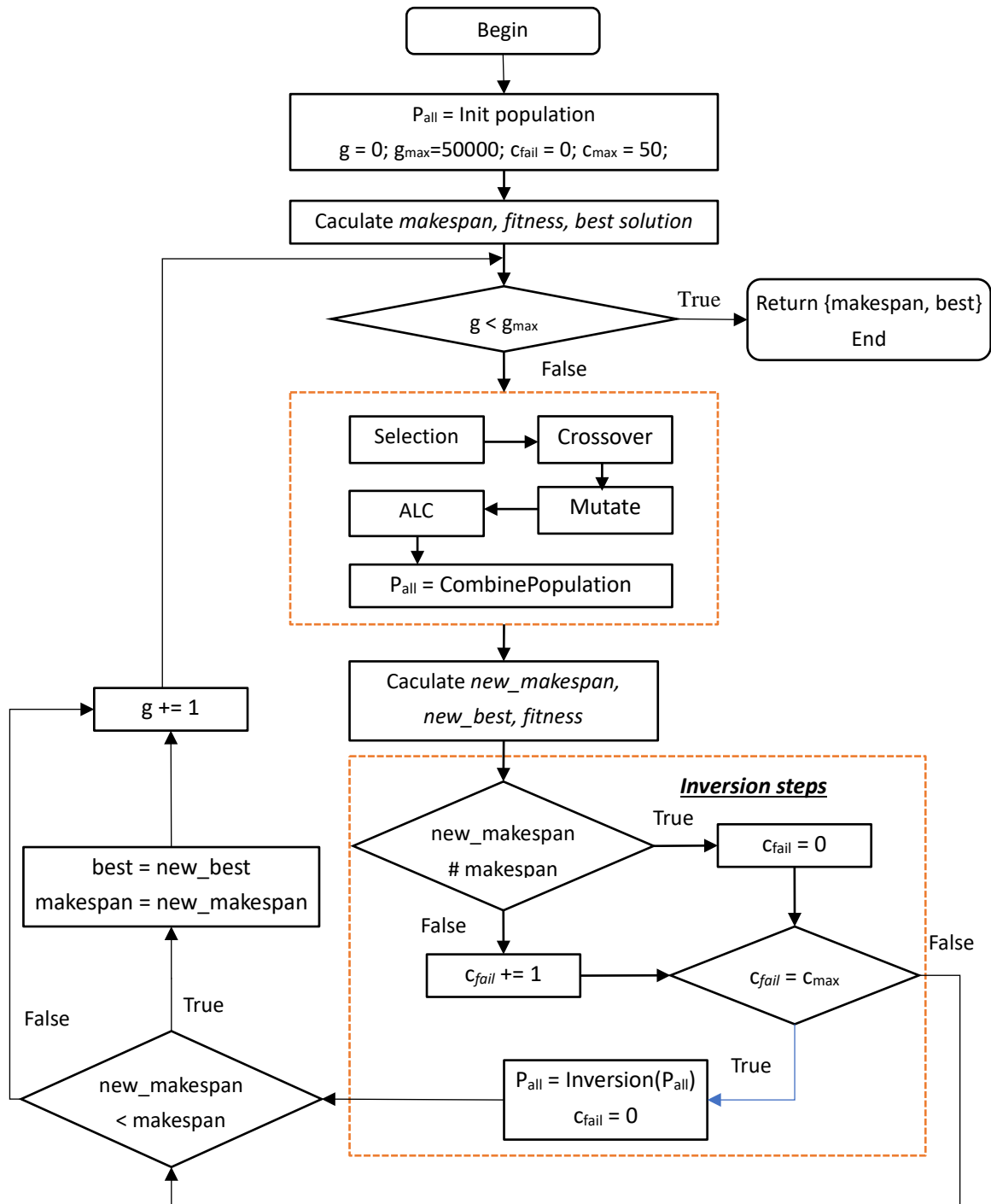


Figure 6. The MEMINV algorithm diagram.

The MEMINV algorithm is present in Algorithm 3.

Algorithm 3. MEMINV

Input: dataset, g_{max} : number of generation,

S_{CR} : global variable, set of crossover coefficients of successful evolution individuals

μ_{CR} : Crossover Probability (global variable)

μ_{MP} : Mutation Probability (global variable)

w: number of neighbors for local search (global variable)

Output: best individual and makespan

```

1. Begin
2. {
3.   {Read dataset}.
4.    $g = 0, c_{fail} = 0, c_{max} = 50, S_{CR} = \{\}; \mu_{CR} = 0.5; w = 3;$ 
5.    $P_{all} = \{\text{population initialization}\}$ 
6.   {makespan, best, fitness} = {Calculating the fitness of population
   and the best individual}
7.   while( $g < g_{max}$ )
8.     {
9.       predecessors = Selection(population); // Select predecessor
   chromosomes for reproduction based on their fitness values
10.      offspring = Crossover(predecessors,  $\mu_{CR}$ ); // Perform crossover
   on the population
11.      offspring = Mutate(offspring,  $\mu_{MP}$ ); // Perform mutation on the
   offspring
12.      offspring = AdaptiveLocalSearch( $P_{all}$ ,offspring); // Perform
   local search on some of the offspring
13.       $P_{all} = \text{CombinePopulation}(P_{all}, \text{offspring});$  // Combine the pre-
   population and offspring to create the new population
14.      {new_makespan, new_best, fitness} = {Calculating the fitness
   of population and the best individual from  $P_{all}$ }
15.      if new_makespan != makespan then
16.         $c_{fail} = 0$ 
17.      else
18.         $c_{fail} += 1$ 
19.      end if
20.      if ( $c_{fail} = c_{max}$ )
21.         $P_{all} = \text{Inversion}(P_{all})$ 
22.         $c_{fail} = 0$ 
23.      end if
24.      if (makespan < new_makespan)
25.        makespan = new_makespan;
26.        best = new_best;
27.         $S_{CR} += 1;$ 
28.      end if
29.       $g = g+1$ 
30.    } // end while
31.    return {best, makespan }
32.  }
```

In Algorithm 2, lines 15–19 aim to detect local extremes, while commands in lines 20–23 move the population by calling the Inversion function. After moving the population, the algorithm continues

on the new solution area. In order to facilitate the parameter adjustment during the evolutionary process, the algorithm utilizes a global variable called S_{CR} . This variable keeps track of the number of individuals that have successfully evolved over multiple generations.

5. Experimental results and analysis

In order to evaluate the performance of the proposed MEMINV algorithm, we experimented on the iMOPSE [4,5] dataset presented in Table 5.

Table 5. The iMOPSE dataset

	Code	Name	Tasks	Resources	Precedence Constraints	Skills
1	iM01	100_5_22_15	100	5	22	15
2	iM02	100_5_46_15	100	5	46	15
3	iM03	100_5_48_9	100	5	48	9
4	iM04	100_5_64_15	100	5	64	15
5	iM05	100_5_64_9	100	5	64	9
6	iM06	100_10_26_15	100	10	26	15
7	iM07	100_10_47_9	100	10	47	9
8	iM08	100_10_48_15	100	10	48	15
9	iM09	100_10_64_9	100	10	64	9
10	iM10	100_10_65_15	100	10	65	15
11	iM11	100_20_22_15	100	20	22	15
12	iM12	100_20_46_15	100	20	46	15
13	iM13	100_20_47_9	100	20	47	9
14	iM14	100_20_65_15	100	20	65	15
15	iM15	100_20_65_9	100	20	65	9
16	iM16	200_10_128_15	200	10	128	15
17	iM17	200_10_50_15	200	10	50	15
18	iM18	200_10_50_9	200	10	50	9
19	iM19	200_10_84_9	200	10	84	9
20	iM20	200_10_85_15	200	10	85	15
21	iM21	200_20_145_15	200	20	145	15
22	iM22	200_20_54_15	200	20	54	15
23	iM23	200_20_55_9	200	20	55	9
24	iM24	200_20_97_15	200	20	97	15
25	iM25	200_20_97_9	200	20	97	9
26	iM26	200_40_133_15	200	40	133	15
27	iM27	200_40_45_15	200	40	45	15
28	iM28	200_40_45_9	200	40	45	9
29	iM29	200_40_90_9	200	40	90	9
30	iM30	200_40_91_15	200	40	91	15

5.1. Experimental parameters

Experiments were conducted with the following parameters and data:

- Benchmark instances: iMOPSE dataset (iM01 to iM30), each instance is a project with full input parameters such as the number of tasks, the number of resources used to execute the project, the number of precedence constraints and the number of resource skills.
- The initial number of individuals (i.e., solution, individual, schedule): 100
- The number of generations: 50,000
- The number of times the experiment was carried out on each element of the dataset: 30
- Experiment tools: Visual Studio 2019, C#, Net Framework 4.5
- PC configuration: CPU Core i5 2.2 GHz, 6GB RAM, Windows 10

5.2. Performance Analysis

Experimental results with the iMOPSE dataset are presented in Table 5. The results are aggregated, compared and evaluated based on three factors:

- BEST - best makespan values - this is the shortest time to complete the project
- AVG - average execution time of 50 experiments with each instance dataset
- STD values - the standard deviation value between experiments on the same data instance

The schedule found by the proposed MEMINV algorithm is compared with the GA algorithm developed by Myszkowski [36], as shown in Table 6.

The experimental results show that the proposed algorithm MEMINV is more efficient than the GA algorithm on the following parameters:

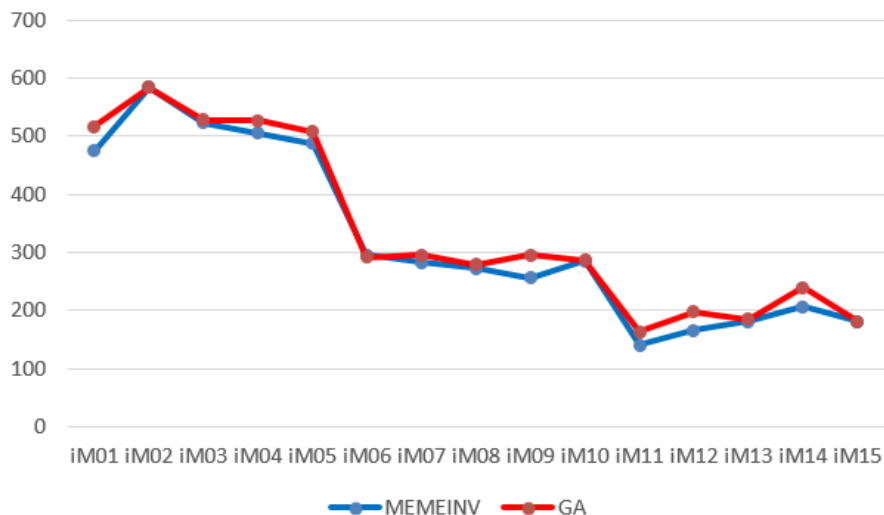
- In term of BEST values, the MEMINV algorithm outperforms GA in 26 out of 30 projects, achieving improvements ranging from 0 to 22.33%. However, in 4 projects, MEMINV performs worse than GA, including the projects with 100 tasks and one project with 200 tasks. Figure 7.a and 7.b illustrate the BEST values comparisons for 15 projects with 100 tasks and 15 remains with 200 tasks, respectively.

- Regarding AVG values, MEMINV yields better results than GA in 25 out of 30 projects, achieving improvements ranging from 0 to 23.31%. However, in 4 projects, MEMINV performs worse than GA, including four projects with 100 tasks and one project with 200 tasks. Figure 8.a and 8.b detail the AVG values comparisons for 15 projects with 100 tasks and 15 others with 200 tasks, respectively.

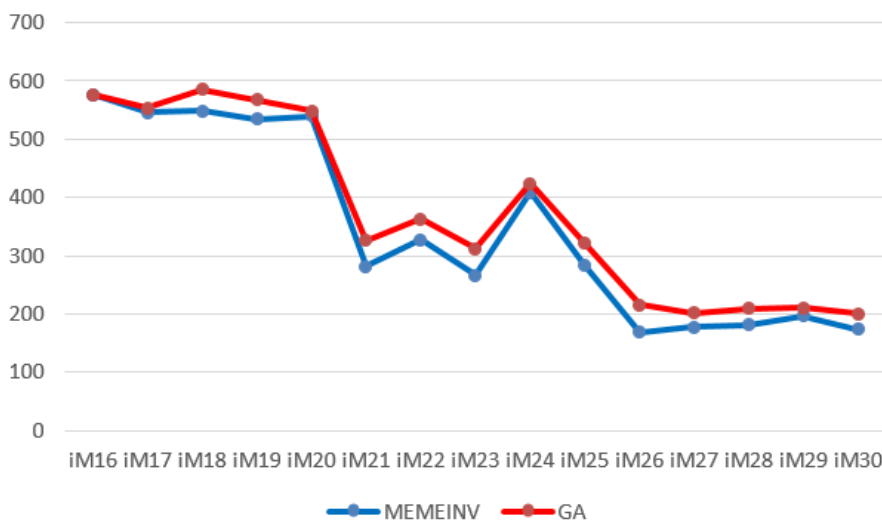
- Moreover, MEMINV exhibits greater stability than GA with a total standard deviation of 133.9, compared to GA's 177.2 on 30 datasets of iMOPSE. Figure 9.a and 9.b present the STD values comparisons for 15 projects with 100 tasks and 15 projects with 200 tasks, respectively. Overall, these results suggest that MEMINV is a more stable and efficient algorithm for scheduling projects with a large number of tasks, making it well-suited for real-world industrial applications.

Table 6. The results from the iMOPSE dataset.

Code	Name	GA			MEMINV			
		BEST	AVG	STD	BEST	AVG	STD	
<i>The projects have 100 tasks</i>								
iM01	100_5_22_15	517	524	5.3	475	478	2.8	
iM02	100_5_46_15	584	587	4.7	584	596	4.5	
iM03	100_5_48_9	528	535	9.7	523	526	10.7	
iM04	100_5_64_15	527	530	2.5	505	506	1.8	
iM05	100_5_64_9	508	521	9.9	487	498	4.9	
iM06	100_10_26_15	292	292	1.7	295	301	0.3	
iM07	100_10_47_9	296	296	2.3	283	285	2.0	
iM08	100_10_48_15	279	282	2.9	272	281	3.7	
iM09	100_10_64_9	296	305	6.6	256	260	0.9	
iM10	100_10_65_15	286	290	5.0	285	296	3.6	
iM11	100_20_22_15	163	169	5.8	140	141	2.1	
iM12	100_20_46_15	197	207	6.9	166	172	8.3	
iM13	100_20_47_9	185	186	0.5	181	183	0.6	
iM14	100_20_65_15	240	240	0.5	207	212	5.5	
iM15	100_20_65_9	181	187	4.5	181	182	1.3	
<i>The projects have 200 tasks</i>								
iM16	200_10_128_15	577	583	4.9	576	585	2.3	
iM17	200_10_50_15	553	577	17.5	546	572	17.5	
iM18	200_10_50_9	585	589	5.0	549	554	4.8	
iM19	200_10_84_9	567	583	11.4	535	542	6.8	
iM20	200_10_85_15	549	555	4.9	540	543	3.7	
iM21	200_20_145_15	326	329	1.9	282	285	1.7	
iM22	200_20_54_15	363	385	20.8	328	345	14.5	
iM23	200_20_55_9	312	318	4.2	267	273	5.0	
iM24	200_20_97_15	424	438	9.7	409	422	11.2	
iM25	200_20_97_9	321	326	6.2	283	285	1.1	
iM26	200_40_133_15	215	222	6.2	169	171	5.3	
iM27	200_40_45_15	201	210	6.3	178	179	0.1	
iM28	200_40_45_9	209	213	2.9	181	183	1.9	
iM29	200_40_90_9	211	215	3.1	196	200	1.8	
iM30	200_40_91_15	200	205	3.4	174	181	3.2	
Sum					177.2			133.9

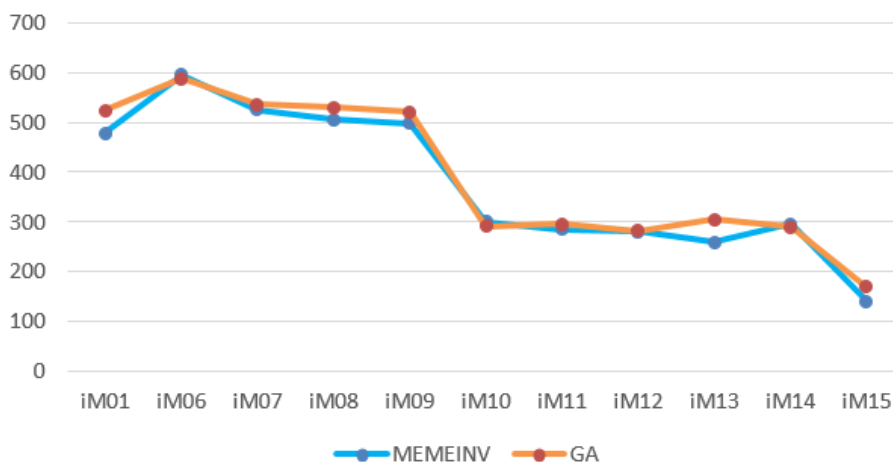


7.a. The BEST values of iM01 to iM15

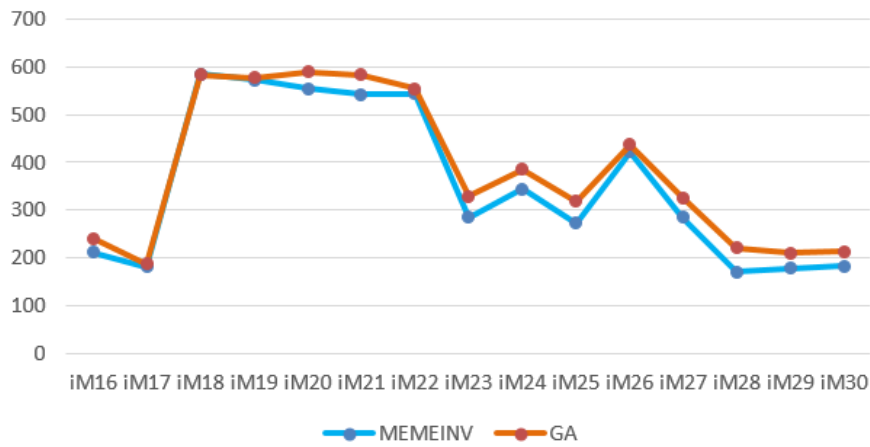


7.b. The BEST values of iM16 to iM30

Figure 7. Comparison of the BEST parameter between MEMEINV and GA.

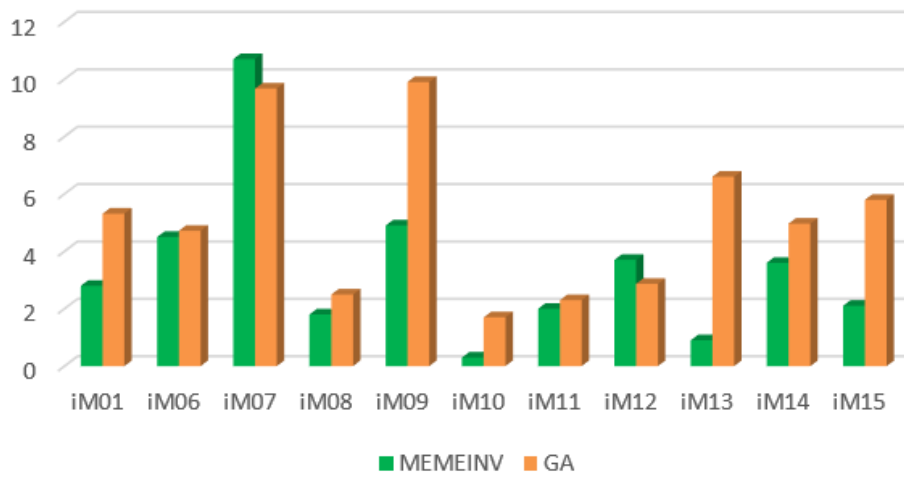


a. The AVG values of iM01 to iM15

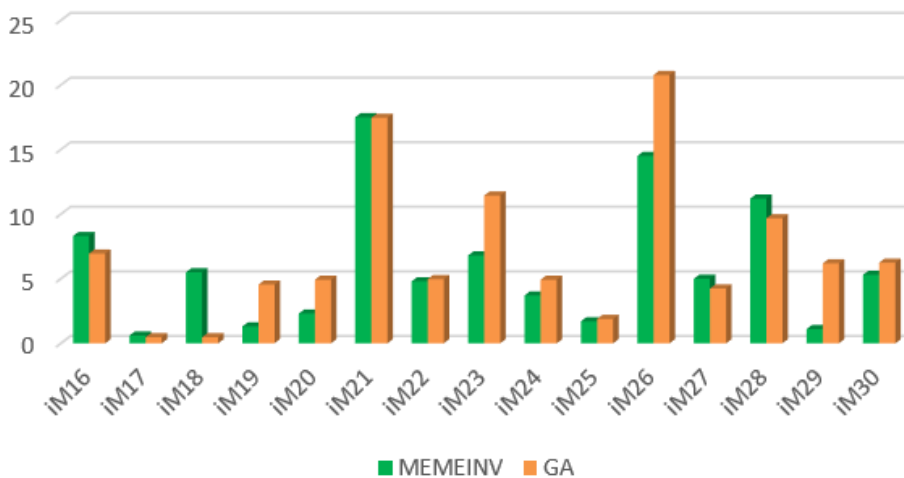


b. The AVG values of iM16 to iM30

Figure 8. Comparison of the AVG parameter between MEMINV and GA.



a. The STD values of iM01 to iM15



b. The STD values of iM16 to iM30

Figure 9. Comparison of the STD parameter between MEMINV and GA.

Experimental results demonstrate that the MEMINV algorithm is less effective for projects with fewer tasks, but it achieves higher efficiency for larger projects. This suggests that MEMINV is well-suited for scheduling real-world projects consisting of a series of jobs with a significant number of tasks. For instance, MEMINV can be applied in various fields, such as equipment manufacturing and industrial sewing lines.

6. Conclusion

In this paper, we have presented a mathematical formulation of the MS-RCPSP problem and proposed a new algorithm, MEMINV, that combines the strengths of the Memetic algorithm and the Inversion method. Our experiments on the IMOPSE dataset demonstrate that MEMINV outperforms previous algorithms regarding BEST and AVG values, achieving up to a 22.33% improvement on BEST values and a 23.31% improvement on AVG values. The total STD value suggests that the proposed algorithm exhibits more excellent stability, meaning that it produces less difference between experiment times.

The results suggest that MEMINV is a promising approach for solving complex scheduling problems with a large number of tasks, which is critical for various real-world industrial production scenarios, such as industrial machine lines or some other manufacturing sectors. Specifically, MEMINV can enable enterprises to create more efficient production schedules, leading to more automated and intelligent production systems that can replace manual labor.

In future work, we plan to investigate other approximation methods, such as random moves based on Gauss, Cauchy, etc., to enhance further the quality of the schedules produced by MEMINV. Overall, our research provides valuable insights into the MS-RCPSP problem and offers a practical solution that can improve the efficiency of production scheduling in various industries.

Funding

This research is funded by Thuongmai University, Hanoi, Vietnam.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Conflict of interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

1. R. Klein, Scheduling of Resource-Constrained Projects, *Springer Science & Business Media.*, **10** (2012).
2. D. Q. Huu, N. T. Loc, N. D. Cuong, An effective hybrid algorithm based on particle swarm optimization with migration method for solving the multiskill resource-constrained project

- scheduling problem, *Appl. Comput. Intell. Soft Comput.*, **2022** (2022), Article ID 6230145. <https://doi.org/10.1155/2022/6230145>
3. D. Q. Huu, N. T. Loc, N. D. Cuong, P. T. Toan, New effective differential evolution algorithm for the multi-skill resource constrained project scheduling problem, in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI 2020)*., Published by IEEE, Nagoya, Japan, June 26–29, (2020). <https://doi.org/10.1109/ICCCI49374.2020.9145982>
 4. P. B. Myszkowski, M. Laszczyk, Investigation of benchmark dataset for many-objective multi-skill resource constrained project scheduling problem, *Appl. Soft Comput.*, **127** (2022), 109253. <https://doi.org/10.1016/j.asoc.2022.109253>
 5. P. B. Myszkowski, M. Laszczyk, I. Nikulin, M. Skowro, iMOPSE: A library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem, *Soft Comput. J.*, **23** (2019). <https://doi.org/10.1007/s00500-017-2997-5>
 6. A. J. Wilson, D. R. Pallavi, M. Ramachandran, S. Chinnasamy, S. Sowmiya, A review on memetic algorithms and its developments, *Electr. Autom. Eng.*, **1** (2022), 7–12. <https://doi.org/10.46632/eae/1/1/2>
 7. S. Afsar, J. J. Palacios, J. Puente, C. R. Vela, I. González-Rodríguez, Multi-objective enhanced memetic algorithm for green job shop scheduling with uncertain times, *Swarm Evolut. Comput.*, **68** (2022), 101016. <https://doi.org/10.1016/j.swevo.2021.101016>
 8. W. Seo, M. Park, D. W. Kim, J. Lee, Effective memetic algorithm for multilabel feature selection using hybridization-based communication, *Expert Syst. Appl.*, **201** (2022), 117064. <https://doi.org/10.1016/j.eswa.2022.117064>
 9. J. Piotr, E. Ratajczak-Ropel, A-team solving multi-skill resource-constrained project scheduling problem, *Proced. Computer Sci.*, **207** (2022), 3300–3309. <https://doi.org/10.1016/j.procs.2022.09.388>
 10. M. Laszczyk, P. B. Myszkowski, Improved selection in evolutionary multi-objective optimization of multi-skill resource-constrained project scheduling problem, *Inform. Sci.*, **481** (2019), 412–431. <https://doi.org/10.1016/j.ins.2019.01.002>
 11. J. Lin, L. Zhu, K. Gao, A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, *Expert Syst. Appl.*, **140** (2020), 112915. <https://doi.org/10.1016/j.eswa.2019.112915>
 12. M. Asadujjaman, H. F. Rahman, R. K. Chakraborty, M. J. Ryan, An Immune Genetic Algorithm for Solving NPV-Based Resource Constrained Project Scheduling Problem, *IEEE Access*, **9** (2021), 26177–26195. <https://doi.org/10.1109/ACCESS.2021.3057366>
 13. M. Đumić, D. Jakobović, Ensembles of priority rules for resource constrained project scheduling problem, *Appl. Soft Comput.*, **110** (2021), 107606. <https://doi.org/10.1016/j.asoc.2021.107606>
 14. O. Shuvo, S. Golder, M. R. Islam, A hybrid metaheuristic method for solving resource constrained project scheduling problem, *Evolut. Intell.*, **16** (2023), 519–537. <https://doi.org/10.1007/s12065-021-00675-x>
 15. H. M. H. Saad, R. K. Chakraborty, S. Elsayed, M. J. Ryan, Quantum-Inspired Genetic Algorithm for Resource-Constrained Project-Scheduling, *IEEE Access*, **9** (2021), 38488–38502. <https://doi.org/10.1109/ACCESS.2021.3062790>
 16. R. L. Lilia Kadri, F. F. Boctor, An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case, *European J. Operat. Res.*, **265** (2018), 454–462. <https://doi.org/10.1016/j.ejor.2017.07.027>

17. J. Lin, L. Zhu, K. Gao, A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, *Expert Syst. Appl.*, **140** (2020), 112915. <https://doi.org/10.1016/j.eswa.2019.112915>
18. J. Snauwaert, M. Vanhoucke, A new algorithm for resource-constrained project scheduling with breadth and depth of skills, *European J. Operat. Res.*, **292** (2021), 43–59. <https://doi.org/10.1016/j.ejor.2020.10.032>
19. L. Zhu, J. Lin, Y. Y. Li, Z. J. Wang, A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, *Knowledge-Based Syst.*, **225** (2021), 107099. <https://doi.org/10.1016/j.knosys.2021.107099>
20. T. Zhou, Q. Long, K. M. Y. Law, C. Wu, Multi-objective stochastic project scheduling with alternative execution methods: An improved quantum-behaved particle swarm optimization approach, *Expert Syst. Appl.*, **203** (2022), 117029. <https://doi.org/10.1016/j.eswa.2022.117029>
21. C. Stiti, O. B. Driss, A new approach for the multi-site resource-constrained project scheduling problem, *Proceed. Computer Sci.*, **164** (2019), 478–484. <https://doi.org/10.1016/j.procs.2019.12.209>
22. D. Q. Huu, N. T. Loc, N. D. Cuong, The R-PSO algorithm solving multi-skill resource-constrained project scheduling problem, *J. Milit. Sci. Technol.*, **5** (2021), 71–82. <https://doi.org/10.54939/1859-1043.j.mst.CSCE5.2021.71-82>
23. J. Joy, S. Rajeev, V. Narayanan, Particle swarm optimization for resource constrained-project scheduling problem with varying resource levels, *Proceed. Technol.*, **25** (2016), 948–954. <https://doi.org/10.1016/j.protcy.2016.08.185>
24. K. M. Sallam, R. K. Chakraborty, M. J. Ryan, A two-stage multi-operator differential evolution algorithm for solving Resource Constrained Project Scheduling problems, *Future Gener. Computer Syst.*, **108** (2020), 432–444. <https://doi.org/10.1016/j.future.2020.02.074>
25. L. Wu, Y. Wang, S. Zhou, Improved differential evolution algorithm for resource-constrained project scheduling problem, *J. Syst. Eng. Electron.*, **21** (2010), 798–805. <https://ieeexplore.ieee.org/abstract/document/6075518>
26. H. Kazemipoor, R. Tavakkoli-Moghaddam, P. Shahnazari-Shahrezaei, A. Azaron, A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems, *Int. J. Adv. Manuf. Technol.*, **64** (2013), 1099–1111. <https://doi.org/10.1007/s00170-012-4045-z>
27. J. Sun, Z. Peng, J. Cai, Problem specific genetic differential evolution algorithm for multi-skill resource-constrained project scheduling of collaborative multi-robot systems for search and rescue, in *2021 40th Chinese Control Conference (CCC)*, Shanghai, China, (2021), pp. 1808–1813. <https://doi.org/10.23919/CCC52363.2021.9549589>
28. N. T. Loc, Q. D. Pham, A-DEM: The adaptive approximate approach for the real scheduling problem, in: *Intelligence of Things: Technologies and Applications* (eds N. T. Nguyen, N. N. Dao, Q. D. Pham and H. A. Le), *ICIT 2022 Lecture Notes on Data Engineering and Communications Technologies.*, **148** (2022), Springer, Cham. https://doi.org/10.1007/978-3-031-15063-0_10
29. X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*, *Luniver Press*, ISBN-13: 978-1-905986-28-6, (2010).
30. X. S. Yang, S. Deb, Cuckoo search via Lévy flights, *Proc. World Congress Nat. Biol. Inspired Computing (NaBIC 2009)*, USA, (2009), pp. 210–214. <https://doi.org/10.1109/NABIC.2009.5393690>

31. D. Q. Huu, N. T. Loc, N. D. Cuong, P. T. Toan, New cuckoo search algorithm for the resource constrained project scheduling problem, in *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, Ho Chi Minh City, Vietnam, (2020), pp. 1–3. <https://doi.org/10.1109/RIVF48685.2020.9140728>
32. H. Maghsoudlou, B. Afshar-Nadjafi, S. T. A. Niaki, Multi-skilled project scheduling with level-dependent rework risk, three multi-objective mechanisms based on cuckoo search, *Appl. Soft Comput.*, **54** (2017), 46–61. <https://doi.org/10.1016/j.asoc.2017.01.024>
33. Y. Tian, T. Xiong, Z. Liu, Y. Mei, L. Wan, Multi-objective multi-skill resource-constrained project scheduling problem with skill switches: Model and evolutionary approaches, *Comput. Industr. Eng.*, **167**, (2022),107897. <https://doi.org/10.1016/j.cie.2021.107897>
34. L. Zhu, J. Lin, Z. J. Wang, A discrete oppositional multi-verse optimization algorithm for multi-skill resource constrained project scheduling problem, *Appl. Soft Comput.*, **85** (2019),105805. <https://doi.org/10.1016/j.asoc.2019.105805>
35. R. Kolisch, A. Sprecher, PSPLIB-a project scheduling problem library: or software-ORSEP operations research software exchange program, *European J. Oper. Res.*, **96** (1997), 205–216. [https://doi.org/10.1016/S0377-2217\(96\)00170-1](https://doi.org/10.1016/S0377-2217(96)00170-1)
36. GARunner tool. Available from: http://imopse.ii.pwr.wroc.pl/rcpsp_spsp_library.html



AIMS Press

©2023 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>)