Apply Metaheuristic ANGEL to Schedule Multiple Projects with Resource-Constrained and Total Tardy Cost

Shih-Chieh Chen*
Department of Information Management
Overseas Chinese University
Taichung, Taiwan, R.O.C.

Ching-Chiuan Lin
Department of Information Management
Overseas Chinese University
Taichung, Taiwan, R.O.C.

***Abstract*—In this paper the multiple projects resource-constrained project scheduling problem is considered. Several projects are to be scheduled simultaneously with sharing several kinds of limited resources in this problem. Each project contains non-preemptive and deterministic duration activities which compete limited resources under resources and precedence constraints. Moreover, there are the due date for each project and the tardy cost per day that cause the penalty when the project cannot be accomplished before its due date. The objective is to find the schedules of the considered projects to minimize the total tardy cost subject to the resource and precedence constraints. Since the resource-constrained project scheduling problem has been proven to be NP-Hard, we cannot find a deterministic algorithm to solve this problem efficiently and metaheuristics or evolutionary algorithms are developed for this problem instead. The problem considered in this paper is harder than the original problem because the due date and tardy cost of a project are considered in addition. The metaheuristic method called ANGEL was applied to this problem. ANGEL combines ant colony optimization (ACO), genetic algorithm (GA) and local search strategy. In ANGEL, ACO and GA run alternately and cooperatively. ANGEL performs very well in the multiple projects resource-constrained project scheduling problem as revealed by the experimental results.**

Keywords—multiple project scheduling; resource-constrained project scheduling; ANGEL; ant colony optimization; genetic algorithms; local search; metaheuristics

I. Introduction

The resource-constrained project scheduling problem (RCPSP) is an important problem both in practice and research. Many researchers work on the single-project case for several years and have very good results, but the research works for the multiple-projects case are only a few. The multiple projects RCPSP model is a more realistic model.

The work by Lova et. al. [1] indicated that 84% of the companies, in the Valencian Region-Spain which responded to their survey, work with multiple projects. This data is in line with the work by Payne [6] that indicated that up to 90% of all projects occurred in the multiple-project context. And the due date and tardy cost are also important realistic situations to be considered. These reasons motivate us to research and to find some good algorithms on this topic.

We summarize some research works for the multiple projects RCPSP. Fendley [8] used multiple projects with 3 and 5 projects and concluded that the priority rule MINSLK is the best for the measurements project slippage, resource utilization and in-process inventory. Kurtulus and Davis [2] showed six new priority rules to the multiple projects instances they designed. They showed that the priority rules MAXTWK and SASP are the best when the objective is to minimize the mean project delay. Kurtulus and Narula [3] showed that the priority rule Maximum Penalty is the best to minimize the sum of the project weight delay. Dumond and Marbert [5] designed five resource allocation heuristics and four strategies to assign due dates to the projects. They showed that the priority rule FCFS with the Schedule Finish Time Due Date rule is the best to minimize the mean completion time, the mean lateness, the standard deviation of lateness and the total tardiness. Lova et al. [1] developed a multi-criteria heuristic to schedule multiple projects with the one-time criteria (mean project delay or multiple project duration) and one-no-time criteria (project splitting, in-process inventory, resource leveling or idle resources).

RCPSP has been proven to be an NP-Hard problem. Many evolutionary algorithms and metaheuristics were proposed to solve RCPSP and the related extension problems. Tseng and Chen [9] proposed an algorithm ANGEL which combines ant colony optimization (ACO), genetic algorithm (GA) and local search strategy (LS) to solve the single project RCPSP. Chen and Lin [13] proposed a discrete particle swam optimization to solve RCPSP. The experimental results of these works showed that they are compatible to other state-of-the-art algorithms in the literature for solving instance sets in PSPLIB [11]. Tseng and Chen [10] also proposed a two-phase genetic local search algorithm to solve the single project RCPSP with multiple modes. Chen [12] proposed a two-phase genetic local search algorithm to solve the single project RCPSP with generalized precedence constraints. Rivera et al. [7] proposed an algorithm which combined the GRASP, Scatter Search and justification to solve RCPSP. These researches showed that combined heuristics and evolutionary algorithms, like ANGEL, can solve these RCPSP and related extension problems efficiently.

The remaining parts of the paper are organized as follows. In Section II, we provide a description of the problem. In Section III, we describe ouer ANGEL algorithm. In Section IV, the computational results are shown and in Section V the concluding remarks are given.

II. Problem Description

We consider a multiple project scheduling problem with n independent projects $P_1, \cdots, P_n$ where $d_1, \cdots, d_n$ and $c_1, \cdots, c_n$ are the due dates and tardy costs per day for the projects respectively. There are K kinds of common resources which are renewable that all projects share on them. For project P_i, $i = 1, 2, \cdots, n$, the set J_i consists of num_i non-dummy activities and each activity has deterministic duration and resource demand for execution. By the single-project approach, we add two dummy activities as the source and the sink to bind the projects together. Hence this multiple-project problem can be considered as a single project problem in which the activity set J has $num = \sum_{i=1}^{n} num_i$ non-dummy activities. All non-dummy activities in J are renumbered from 1 to num sequentially. Activity 1 to num_1 are the activities of project P_1, activity $num_1 + 1$ to $num_1 + num_2$ are the activities of project P_2 and so on. Activity 0 and activity $num + 1$ are the source and the sink, the dummy activities. So $J = \left(\bigcup_{i=1}^{n} J_i \right) \cup \{ 0, num + 1 \}$ is the set of activities. Fig. 1 shows a multiple projects instance with 2 independent projects and the corresponding combined single project is shown in Fig. 2, where d_j is the duration and r_{jk} is the resource demand for resource k of activity j.

Let $PSet_i$ be the set of all immediate predecessors (activities) of activity i. The precedence constraints are given that activity i cannot be executed before all activities belong to $PSet_i$ have finished. For resource k, the per-period-availability is given by a constant R_k. Each dummy activity has zero duration and does not require any resource.

In multiple projects RCPSP, let m_1, …, m_n be the makespans we scheduled for each project respectively, then the total tardy cost TC is defined in equation (1) as the sum of tardy costs of the projects which cannot be accomplished in their due dates. The objective of the problem considered in this paper is to minimize the total tardy cost (TC) subject to all the precedence and resource constraints.

$$TC = \sum_{m_i > d_i} (m_i - d_i) \cdot c_i \tag{1}$$

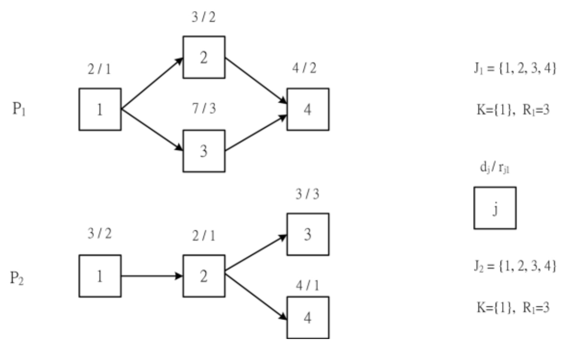

Fig. 1. A multiple projects instance

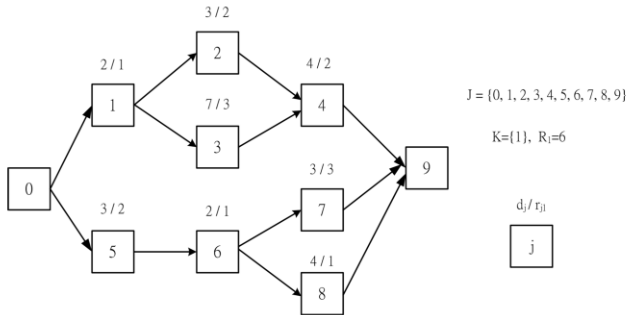

Fig. 2. The corresponding combined single project instance

In this paper, we use a precedence-feasible activity list to represent a solution. When a precedence-feasible activity list is given, we use the list scheduling method to create a schedule. To make a schedule, we apply either the forward scheduling or the backward scheduling on the activity list to set the execution starting time for each activity. The forward scheduling sets the execution starting time of an activity, from the front to the end of the list, as early as possible but satisfies the resource constraints. That is, if an activity lacks for some resources to start its execution, then its execution starting time will be delayed until some activity is finished and the resources are released to satisfy its resource demand. The backward scheduling sets the execution starting time of an activity, from the end to the front of the list, as late as its finish time is just right before the earliest execution starting time of all its successors. If an activity lacks for some resource for its execution, then the execution starting time will be set earlier just right before the starting time of some activity in order to satisfy its resource demand.

III. The ANGEL Metaheuristic for the Multiple Projects RCPSP

In this section, we present the strategies in ANGEL metaheuristic for the multiple projects RCPSP. We modify the algorithm of Tseng and Chen [12] because we solve this problem by single-project approach but coincide the characteristic of this problem. ANGEL consists of the ACO, the GA and the local search strategy. All parts of the metaheuristic ANGEL are described in detail in the following.

A. The Ant Colony Optimization (ACO)

In original ACO several ants share the common memory set called *pheromone* and each ant find its own path of solution independently. The schemes *local updating*, *global updating*, and *evaporation* change the common memory by the experience of each ant, experience from global best solution and decreasing during time past respectively. We apply ACO to generate a population of precedence-feasible activity lists. To construct an activity list by a specific ant *x*, we first put the dummy activity 0 into the first position of the list. Then, if activity v is put in position j , ant x has to choose another activity from the candidate set N_j and put the chosen activity to position $j+1$ of the list. The candidate set N_j consists of the activities whose predecessors have been put in the list. When activity νin position *j,* the probability that ant x chooses activity w to be in position $j + 1$ is defined in equation (2), where q_0 is an user-defined parameter, q is a random number drawn between 0 and 1, and τ_{vu} is the amount of pheromone been deposited on the ordered pair (ν, u). S is a random variable selected according to the probability distribution given in equation (3), where $\tau_{min} = min_{u \in N_j}\{\tau_{vu}\}$.

$$w = \begin{cases} \arg \max\limits_{u \in N_j}\{\tau_{vu}\} & , \quad \text{if } q \le q_0 \\ S & , \quad \text{otherwise} \end{cases} \tag{2}$$

$$p^x_{vw} = \begin{cases} (\tau_{vw} / \tau_{\min})^2 / \sum\limits_{u \in N_j} (\tau_{vu} / \tau_{\min})^2 & ,\text{if } w \in N_j \\ 0 & ,\text{otherwise} \end{cases} \tag{3}$$

The formulae of the local updating, the evaporation, and the global updating are described in equations (4)-(6) respectively.

$$\tau_{vw} \leftarrow \tau_{vw} + \Delta\tau \tag{4}$$

$$\tau \leftarrow (1-\rho)\cdot\tau \tag{5}$$

$$\tau_{vw} \leftarrow (1-\alpha)\cdot\tau_{vw} + \alpha\cdot\Delta\tau_{vw} \tag{6}$$

In (4), $\Delta\tau$ is a small increment when the local updating is performed on the ordered pair (ν, u). In (5), the evaporation means the amount of pheromone of all ordered pairs are decremented by a ratio ρ, where $0 < \rho < 1$. The increment $\Delta\tau_{vu}$ of the global updating in (6) is defined in (7), where $0 < \alpha < 1$ and TC_{gb} is the minimal total tardy cost of schedules ever found. Note that the global updating is only conducted on the list with the minimum total tardy cost in each ACO iteration.

$$\Delta\tau_{vw} = \begin{cases} 1/TC_{gb} & \begin{array}{l}\text{if activity } w \text{ is next to activity } v \\ \text{in the list with the minimum total} \\ \text{tardy cost found by ants in an iteration}\end{array} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

By the effort of the ACO, the ants find for us a population of activity lists, then evaluate the associated schedules by forward and backward scheduling method, and evaluate the makespans and the corresponding total tardy costs. From the schedules conducted by forward and backward scheduling, only the one with smaller total tardy cost will be reserved. Tie will be broken by random selection.

B. The Genetic Algorithm (GA)

The GA we proposed is a permutation-based GA. The chromosomes in the population are the activity lists which consist of 90% constructed by the ACO and 10% randomly generated. The fitness of an activity list is the inverse value of the total tardy cost of the corresponding schedule. The crossover, mutation and selection operators are as follows.

We implement *two-point forward crossover* and *two-point backward crossover*, which are modified versions of the two-point forward-backward crossover proposed by Alcaraz and Maroto [4], in GA. The two-point forward crossover constructs the offspring from front to rear, and the two-point backward crossover constructs the offspring from rear to front. The crossover operators are defined as follows.

Two parent lists, called father and mother, produce two offspring, called son and daughter. We first randomly draw two crossover-points denoted by L_1 and L_2. To produce the son, when the two-point forward crossover is applied, the first positions of the son are directly taken from the first L_1 activities of the father. Then, in the father and the mother, the activities that have been taken are marked. The next $L_2 - L_1$ positions of the son are taken from the first $L_2 - L_1$ unmarked activities of the mother. In the father, these taken activities are marked. The rest positions of the son are taken from the rest unmarked activities of the father. All the activities taken from the mother or the father are in their relative order. The daughter is produced by interchanging the roles of the father and the mother. The two-point backward crossover works as a “reverse version” of the forward crossover that takes the activities and constructs the offspring from rear to front. To apply the crossover operators, the lists in the population are randomly divided into *pop*/2 pairs and a probability threshold, *pcro*, is specified. For each pair of lists, the two-point forward crossover is applied if the random number drawn is greater than the threshold. Otherwise, the two-point backward crossover is applied.

We design two mutation operators which try to pick out some activities and then randomly put them back as long as the precedence relations are satisfied. First, the activities in a list are classified to two classes A and B. Those activities in class A are picked out by a larger probability *pmut2* while those in class B by a smaller probability *pmut3*. When mapping a list to the corresponding schedule by the list scheduling method, an activity may not be started right after all its predecessors finished because lacking the resources it needs. We call this activity a *delayed activity*. If an activity in a list is a delayed activity while this activity is not a delayed activity in most of lists in the population, this activity belongs to class A in this list. Otherwise, activities not belonging to class A in a list belong to class B. When applying the mutation to a list, for each activity in the list, if the activity belongs to class A, it is picked out by probability *pmut2*, otherwise, it is picked out by probability *pmut3*. In a random order, those pick-out activities are then randomly put back to the list as long as the precedence relations are preserved.

We implement the *ranking selection* and the *2-tournament selection* in our GA. After the crossover and the mutation, there are 2**pop* lists in the population, *pop* parent lists and *pop* offspring lists. In the ranking selection, we select the first *pop* lists from the population that is ranking by the makespan to construct the new population. In the 2-touranment selection, two lists are selected randomly from the population and the one with smaller makespan will be put in the new population. This procedure will be repeated *pop* times to construct the new population.

C. The Local Search Strategy

The local search strategy in this study is the forward-backward local search (FBLS) proposed by Tseng and Chen [9]. This local search utilizes the standard representation of permutation to reduce the search space and both forward scheduling and backward methods to improve the solution quality that very few computational effort is needed. The FBLS tries to search better solutions for a given permutation by following steps: (i) evaluate the forward schedule of the list, sort the operation starting times of activities and make the standard representation permutation the list by the order of operation starting times; (ii) evaluate the backward schedule of the list, sort the operation starting times of activities and then make a new permutation the list by the order of operation starting times; (iii) evaluate the forward schedule of the list, sort the operation starting times of activities and then make a new permutation the list by the order of operation starting times; compare the makespans of the schedules evaluated from the previous three steps and replace the list by the permutation which has the smallest makespan at last. From the experimental results conducted by Tseng and Chen [9], this local search is a very fast and effective local search to improve the solution quality in RCPSP.

D. The ANGEL

In the process of ANGEL, we apply the ACO first to generate activity lists, followed by applying the forward and the backward scheduling to each of them and reserve the better one. These lists along with several randomly generated lists are used as the initial population of GA. The local search is applied to the new lists to search better solutions. In GA, if the best schedule ever found is not improved for *GenStuck* generations, we apply the mutation operator and the 2-tournament selection to the population. And then start GA again. After applying the mutation operator *LoopStuck* times, it seems that the population is highly homogeneous, and then the ACO is applied again, construct new population, and begins another run of ANGEL. The procedure of ANGEL is shown in Fig. 3.

IV. Computational Results

We create eight sets of multiple projects instances, as shown in Table I, by combining single project instances from the PSPLIB. Each of the instance set J30 and J60 contains 480 single project instances and each instance contains 30 and 60 non-dummy activities. We combine the instances of J30 and J60 randomly to be the multiple projects problem instances.

```
Procedure ANGEL
  While ( !termination_angel )
    ACO, local search to the list, and generate the population
    While ( !termination_ga)
      Crossover
      Evaluate schedule, local search and local update pheromone
      Ranking selection
      If ( the best solution is improved ) global update pheromone
      If ( the best solution is not improved for a long time)
        Crossover
        Mutation
        Evaluate schedule and local search
        2-tournament selection
    End
    Global update pheromone
  End
```

Fig. 3. Procedure of ANGEL

TABLE I. Multiple Projects Instance Sets

Instance set	Instances	Projects in each instance
30_2	240	2 projects with 30 activities each
30_4	120	4 projects with 30 activities each
30_8	60	8 projects with 30 activities each
60_2	240	2 projects with 60 activities each
60_4	120	4 projects with 60 activities each
60_8	60	8 projects with 60 activities each
30_60_2_2	240	4 projects, 2 with 30 and 2 with 60 activities
30_60_4_4	120	8 projects, 4 with 30 and 4 with 60 activities

To show the effect of our method, we first define the upper bound for the instances. Suppose a multiple projects which consists of projects $P_1, \cdots, P_n$ where $d_1, \cdots, d_n$ and $c_1, \cdots, c_n$ are the due dates and tardy costs per day for the projects respectively. Let u_i be the best makespan when project P_i is scheduled as a single project RCPSP, then the upper bound for this instance is evaluated by equation (8).

$$UB = \sum_{i=1}^{n} (u_i - d_i) \cdot c_i \tag{8}$$

The statistical property of UB of each multiple project instance set is shown in Table II. As for the lower bound, it is obvious that zero is a trivial lower bound for each instance. We also define the improvement ratio IR in equation (9).

$$IR = \begin{cases} \text{(UB-TC)/UB} \times 100\% & \text{if UB} \neq 0 \\ 100\% & \text{if UB} = 0 \ \& \ \text{TC} = 0 \\ -100\% & \text{if UB} = 0 \ \& \ \text{TC} \neq 0 \end{cases} \qquad (9)$$

TABLE II. STATISTICAL PROPERTY OF UB IN EACH INSTANCE SET

Instance set	Max	Min	Ave.	S. Dev.
30_2	1334	0	175.59	263.38
30_4	1913	0	482.69	442.56
30_8	2047	52	801.32	485.14
60_2	3040	0	446.60	707.19
60_4	4076	0	906.26	1069.02
60_8	6204	0	1790.90	1478.86
30_60_2_2	3346	0	622.20	741.90
30_60_4_4	4308	4	1388.95	1102.76

For example, the 17th instance of the multiple projects instance set 30_2 consists of the 4th and 81st instances from the single project instance set J30. The best makespans of each project in single project RCPSP are 62 and 83. The due dates are 55 and 55, and the tardy costs are 28 and 19 per day for each single project respectively. Then

$$UB = (62-55) \times 28 + (83-55) \times 19 = 728$$

for the multiple project instance. The makespans for this multiple project instance when they are scheduled simultaneously are 55 and 62, then the total tardy cost

$$TC = (55-55) \times 28 + (62-55) \times 19 = 133$$

and the improvement ratio

$$IR = \frac{728-133}{728} \times 100\% = 81.73\%.$$

In our computational experiments, each instance set is tested 3 times and based on the average *IR*, the best case, the worst case and the average case are presented. Each instance is searched with 1000 or 5000 schedules evaluated. Table 4-10 show the computational results of IR and the percentage of the instances with zero total tardy cost ($TC = 0$) for all instance sets except the instance set 30_8. In instance set 30_8, the total tardy costs are zero for all instances within 1000 schedules.

TABLE III. COMPUTATION RESULTS OF INSTANCE SET 30_2

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	0	95.40	11.78	76.25
	Worst	100	0	95.36	11.96	76.25
	Average	100	0	95.38	11.90	76.67
5000	Best	100	0	96.61	10.29	80.42
	Worst	100	0	96.43	10.54	79.17
	Average	100	0	96.49	10.46	80.00

TABLE IV. COMPUTATION RESULTS OF INSTANCE SET 30_4

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	79.93	99.25	2.98	89.17
	Worst	100	79.17	99.20	2.94	90.00
	Average	100	79.13	99.23	2.99	89.44
5000	Best	100	86.13	99.68	1.70	92.50
	Worst	100	85.36	99.64	1.75	90.83
	Average	100	86.68	99.67	1.65	91.67

TABLE V. COMPUTATION RESULTS OF INSTANCE SET 60_2

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	-100	88.80	28.25	74.58
	Worst	100	-100	88.04	30.98	73.33
	Average	100	-100	88.29	30.07	73.89
5000	Best	100	2.84	93.71	15.52	78.33
	Worst	100	0	93.43	16.43	78.33
	Average	100	0.95	93.52	16.12	78.33

TABLE VI. COMPUTATION RESULTS OF INSTANCE SET 60_4

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	65.68	98.07	6.05	83.33
	Worst	100	57.73	97.73	7.51	85.00
	Average	100	59.92	97.91	6.70	84.44
5000	Best	100	81.04	99.13	2.95	86.67
	Worst	100	76.32	98.97	3.83	87.50
	Average	100	78.57	99.03	3.46	87.23

TABLE VII. COMPUTATION RESULTS OF INSTANCE SET 60_8

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	86.17	99.51	1.96	88.33
	Worst	100	80.81	99.42	2.59	88.33
	Average	100	84.27	99.46	2.24	88.33
5000	Best	100	98.66	99.97	0.19	95.00
	Worst	100	97.51	99.95	0.33	95.00
	Average	100	98.10	99.96	0.25	95.55

TABLE VIII. COMPUTATION RESULTS OF INSTANCE SET 30_60_2_2

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	59.99	98.93	4.32	87.92
	Worst	100	66.43	98.82	4.46	87.08
	Average	100	63.72	98.87	4.36	87.36
5000	Best	100	82.75	99.61	1.98	93.33
	Worst	100	79.13	99.53	2.19	92.92
	Average	100	80.79	99.58	2.09	93.05

TABLE IX. COMPUTATION RESULTS OF INSTANCE SET 30_60_4_4

Schedules	Case	IR (%)				$TC = 0$ (%)
		Max	Min	Ave.	S. Dev.	
1000	Best	100	94.40	99.82	0.80	93.33
	Worst	100	96.31	99.82	0.68	91.67
	Average	100	95.46	99.82	0.75	92.50
5000	Best	100	99.18	99.99	0.11	98.33
	Worst	100	99.59	99.99	0.05	97.50
	Average	100	99.29	99.99	0.09	98.06

We can see from all the tables that the average *IR* ratio and the percentage of instances with zero total tardy cost increase, and the standard deviation of *IR* ratio decreases as the number of schedules evaluated increases. These results means the total tardy cost of multiple projects will be improved effectively by ANGEL if more searching is conducted. We can also observe that if more projects are to be scheduled simultaneously, there are greater chances that projects be accomplished in their due dates. This result also fits the realistic situation and suggests that in a company, all projects that share the common resources should be scheduled simultaneously.

V. CONCLUDING REMARKS

ANGEL had been applied to solve the single project RCPSP [9] and the single project RCPSP with multiple modes [10] and obtained good results. In this paper we consider the problem that multiple projects sharing common resources are to be scheduled simultaneously subject to the precedence and resource constraints. The objective is to minimize the total tardy cost of the projects. The computational results show that ANGEL is effective on this problem. It also reveals that projects sharing common resources should be scheduled simultaneously rather than scheduled one by one.

From the computational results and other researchers' works we can find that the combined algorithms or metaheuristics performed well in solving discrete combinatorial optimization problems. The further researches of us are to test the combination of different evolutionary algorithms, like GRASP or Particle Swarm Optimization, to find better algorithms to different optimization problems

REFERENCES

[1] A. Lova, C. Maroto and P. Tormos, "A multicritertia heuristic method to improve resource allocation in multiproject scheduling," Euro. J. Oper. Res., vol. 127, pp.408-427, 2000

[2] I. S. Kurtulus and E. W. Daivs, "Multi-project scheduling: categorization of heuristic rules performance," Mana. Sci., vol. 28, pp161-172, 1982.

[3] I. S. Kurtulus and S.C. Narula, "Multi-project scheduling: analysis of project performance,: IIE Trans., vol. 17, pp.58-66, 1985.

[4] J. Alcaraz and C. Maroto, "A robust genetic algorithm for the resource allocation in project scheduling," Ann. Oper. Res. Vol. 102, pp.83-109, 2001.

[5] J. Dumond and V. A. Marbert, "Evaluating project scheduling and due date assignment procedures: An experimental analysis," Mana. Sci., vol. 34, pp.101-118, 1998.

[6] J. H. Payne, "Management of multiple simultaneous projects: A state-of-the-art review," Inter. J. Proj. Mana., vol. 13, pp-163-168, 1995.

[7] J. Rivera, L. Velásquez, F. Serna and G. Zapata, "A Hybrid Heuristic Algorithm for Solving the Resource Constrained Project Scheduling Problem," Revi. EIA, vol. 10, pp.87-100, 2013.

[8] L. G. Fendley, "Towards the development of a complet multiproject scheduling system," J. of Indu. Engi. Pp.505-515, 1968.

[9] L. Y. Tseng and S. C. Chen, "A hybrid metaheuristic for the resource-constrained project scheduling problem," Euro. J. Oper. Res., vol. 175, pp.707-721, 2006.

[10] L. Y. Tseng and S. C. Chen, "Two-Phase Genetic Local Search Algorithm for the Multimode Resource-Constrained Project SchedulingG. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[11] R. Kolisch and A. Sprecher, "PSPLIB – a project scheduling problem library," Euro. J. Oper. Res., vol.96, pp205-216, 1996.

[12] S. C. Chen, "A Genetic Local Search Algorithm for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Constraints," The 6th IEEE Inter. Conf. Ubi-Media Comp., Aizu-Wakamatsu, Japan, November, 2013.

[13] S. C. Chen, "A Discrete Particle Swam Optimization for Scheduling Projects with Resource Constrained," The 2nd Inter. Conf. Comp., Meas., Cont. and Sens. Net., Tunghai University, Taiwan, May, 2014.