A Scatter Search Hybrid Algorithm for Resource Availability Cost Problem

Hexia Meng, Bing Wang, Yabing Nie, Xuedong Xia and Xianxia Zhang

Abstract This paper discusses the resource availability cost problem (RACP) with the objective of minimizing the total cost of the unlimited renewable resources by a prespecified project deadline. A tabued scatter search (TSS) algorithm is developed to solve the RACP. The deadline constraint is handled in coding. A tabu search module is embedded in the framework of scatter search. A computational experiment was conducted and the computational results show that the proposed TSS hybrid algorithm is effective and advantageous for the RACP.

Keywords RACP · Scatter search · Tabu search

1 Introduction

The resource-constrained project scheduling problem (RCPSP) is one of the main branches of project scheduling, which is known to be NP-hard [1]. It involves minimizing the completion time or time cost of the project subject to precedence relations and the limited resources. The RCPSP has been studied extensively and several effective exact [2] and heuristic [3,4] algorithms have been proposed. Among these algorithms, tabu search has been proved to be an efficient algorithm to solve the RCPSP [5,6].

This paper discusses the resource availability cost problem (RACP), which aims to minimize the total cost of the unlimited renewable resources required to complete the project by a prespecified project deadline. It was introduced by Möhring [7] as an NP-hard problem. The RACP is a variant of the RCPSP [8], but it differs from the RCPSP in the sense that the time for completing the project is limited and the resources are unlimited at a non-decreasing discrete cost function. Because RACP and RCPSP are closely related and the RCPSP has been well

H. Meng · B. Wang(✉) · Y. Nie · X. Xia · X. Zhang
School of Mechatronic Engineering and Automation, Shanghai University,
Shanghai 200072, China
e-mail: susanbwang@shu.edu.cn

J.H. Kim and Z.W. Geem (eds.), *Harmony Search Algorithm,*
Advances in Intelligent Systems and Computing 382,
DOI: 10.1007/978-3-662-47926-1_5

studied, it is natural to think of solving the RACP by iteratively solving a serial of RCPSPs. We call it the indirect way. The literature on solution methods for the RACP is relatively scarce.

Möhring [7] proposed an exact procedure based on graph theoretical algorithms, Demeulemeester [9] proposed an exact cutting plane procedure, and Rangaswamy [10] proposed a branch-and-bound for the RACP. Rodrigues and Yamashita [11] developed a hybrid method in which an initial feasible solution is found heuristically, and proposed new bounds for the branching scheme. Drexl and Kimms [12] proposed two lower bounds for the RACP using Lagrangean relaxation and column generation techniques. Shadrokh and Kianfar [13] presented a genetic algorithm to solve the RACP, where tardiness is permitted with defined penalty. Yamashita et al. [14] used scatter search to solve the RACP, and computational results show that scatter search is capable of providing high-quality solutions of the RACP in reasonable computational time. Further, Yamashita et al. [15] adopted scatter search to solve the RACP with uncertain activity durations and created robust optimization models.

However, Qi et al. [16] pointed that the process of solving the RACP in indirect way is very complicated and inefficient. He presented two methods to improve the efficiency of solving the RACP, one of which is directly solving the RACP instead of transforming it to the RCPSP. We shall refer to this way as the direct way. In the RACP, the completion time and total cost of the project are determined by the start time of each activity. Therefore, determining the start time of each activity is the key to the direct way to solve the RACP. Qi et al. [16] predigested the process of solving the RACP by using the start times to code the schedule. Then he proposed a pseudo particle swarm optimization (PPSO) to make the process of looking for the best solution efficiently. Ranjbar et al. [17] developed two metaheuristics, path relinking and genetic algorithm to tackle the RACP. The problem is represented as a precedence feasible priority list and converted to a real schedule directly using an available schedule generation scheme.

The direct way can simplify the process of solving the RACP by avoiding transforming the RACP to the RCPSP. In this paper, we study the solution method for the RACP based on the direct way. Firstly, we use the start time of each activity to code the schedule and handle the deadline constraint by coding. Then, a tabued scatter search (TSS) algorithm combined with scatter search and tabu search is developed to solve the RACP.

The remainder of this paper is organized as follows. The RACP is described in Section 2. Section 3 describes the proposed TSS algorithm and its components for the RACP. Section 4 shows the computational results on the benchmark problem instances. The conclusions of this study are given in Section 5.

2 The Resource Availability Cost Problem

The RACP can be stated as follows. A project consists of $n+2$ activities subject to finish-start precedence relations $(i, j) \in H$, where activities 0 and $n+1$ are dummy activities that indicate the start and finish of the project respectively.

For each activity i, it is started at time st_i and requires r_{ik} units of renewable resource type $k\,(k=1,...,m)$ in every time unit of its deterministic and non-preemptive duration d_i. Let D represent the project deadline, which is prespecified and the project must be completed before the deadline. Let A_t denote the set of activities in progress during the time interval $(t-1,t]$ and a_k represent the availability of resource type k.The RACP can be conceptually modeled as follows:

$$\min \sum_k C_k(a_k) \tag{1}$$

$$\text{s.t.} \quad st_i + d_i \le st_j \quad \forall (i,j) \in H \tag{2}$$

$$st_0 = 0 \tag{3}$$

$$st_{n+1} + d_{n+1} \le D \tag{4}$$

$$\sum_{i \in A_t} r_{ik} \le a_k \quad k=1,...,m,\ t=1,...,D \tag{5}$$

The objective function (1) is to minimize the total resource cost of the project, wherein $C_k(a_k)$ denotes a discrete non-decreasing cost function associated with the availability a_k of resource type k. Constraint (2) takes the precedence relations among the activities into account, where activity i immediately precedes activity j. Constraint (3) denotes that the project should be started at time zero and constraint (4) indicates that the project must be completed before the deadline. Constraint (5) shows that the renewable resource constraints are satisfied.

From the model, it can be seen that both the resource availability values $a_k\,(k=1,...,m)$ and the start times $st_i\,(i=1,...,n)$ of the activities are variables, and the decision variables of the RACP are $a_k\,(k=1,...,m)$. Indeed, the value of a_k depends on the start time of each activity, i.e. $a_k = \max_{t=1,...,D} \sum_{i \in A_t} r_{ik}$. Different start times of activities may result in different values of a_k. This is also the main basis of the direct way to solve the RACP. The objective of the RACP is to find a precedence feasible schedule, such that the project can be completed before the deadline and the total resource cost is minimized.

3 TSS for RACP

In this section, a tabued scatter search (TSS) algorithm combined with scatter search and tabu search is developed to solve the RACP based on the direct way. We use the start time of each activity to code the schedule, so that the RACP can be solved directly. The specific coding and the proposed TSS algorithm will be introduced in the following.

3.1 Coding

To solve the RACP directly, we adopt the direct coding method. The start time of each activity is used to code the schedule. Let $\mathbf{s}$ denote a solution of the RACP, then $\mathbf{s} = (st_1, ..., st_n)$. In $\mathbf{s} = (st_1, ..., st_n)$, each component $st_i (i = 1, ..., n)$ is the start time of activity $i (i = 1, ..., n)$. For the RACP, if the start time of each activity $st_i (i = 1, ..., n)$ in $\mathbf{s}$ satisfies the precedence relations and the completion time of the project does not exceed the deadline D, then $\mathbf{s}$ is a feasible solution of the RACP.

Step 1. Calculate the earliest start time est_i of each activity from $i = 1$ to n;
Step 2. Let $lst_{n+1} = D$;
Step 3. Calculate the latest start time lst_i of each activity from $i = n$ to 1;

Fig. 1 Procedure to obtain $[est_i, lst_i]$

For a given solution $\mathbf{s}$, the completion time of the project can be determined directly. To satisfy the deadline constraint of the project, we handle the deadline constraint in coding in this paper. Let est_i and lst_i denote the earliest and latest start time of activity i respectively, then $est_i \le st_i \le lst_i$. Interval $[est_i, lst_i]$ is called the effective value interval of st_i by us. It contains all possible values of st_i, and the value of st_i can only be selected from $[est_i, lst_i]$ when generating new solutions. The procedure to obtain $[est_i, lst_i]$ is shown in Fig. 1. The earliest start time est_i of each activity from $i = 1$ to n is calculated according to the critical path method (CPM) [18] in step 1. The latest start time of activity $n+1$ is initialized with the deadline D, i.e. $lst_{n+1} = D$ in step 2. Finally, the latest start time lst_i of each activity from $i = n$ to 1 is calculated according to CPM in step 3.

By setting effective value interval $[est_i, lst_i]$ for st_i, the deadline constraint of the project is transformed to the restrictions of the start times of the activities. However, for $\mathbf{s} = (st_1, ..., st_n)$, $st_i \in [est_i, lst_i](i = 1, ..., n)$ cannot guarantee that $\mathbf{s}$ is feasible due to that the precedence relations may not be satisfied. For the solution which is infeasible, check up the start time of each activity successively and update the start times which do not satisfy the precedence relations. For a solution $\mathbf{s}$ which satisfies the precedence relations, a theorem is given as follows.

Theorem 1. If solution $\mathbf{s}$ satisfies the precedence relations, then the completion time *makespan* of the project must not exceed the deadline D, i.e. $makespan \le D$.

Proof. Because $\mathbf{s}$ satisfies the precedence relations, then $makespan = st_{n+1}$. As $st_{n+1} \le lst_{n+1}$ and $lst_{n+1} = D$, then $st_{n+1} \le D$, i.e. $makespan \le D$.

Theorem 1 gives the relationship between the precedence relations and the deadline. If **s** satisfies the precedence relations, it must satisfy the deadline constraint and thus it is a feasible solution of the RACP. The completion time and total cost of the project can be determined directly without transforming the RACP to RCPSP.

3.2 TSS Algorithm

Scatter search (SS) is an evolutionary method based on heuristic proposed by Glover [19]. Glover [20] further identified a template for SS. Martí et al. [21] provided the fundamental concepts and principles of SS. According to the template [20], SS contains five systematic methods, i.e. diversification generation method, improvement method, reference set update method, subset generation method and solution combination method. The implementation of SS is based on these methods. SS considers both intensification and diversification in search process, thus it has strong global search capability. SS has been successfully applied to solve the RACP [14,15].

Tabu search (TS) was initially proposed by Glover [22] and has been applied to many combinatorial optimization problems. It can avoid falling into local optimum by introducing tabu list and has proved to have strong search ability. However, TS depends on the initial solution strongly and the search process is serial. In project scheduling, the application of TS focuses on the RCPSP.

To solve the RACP more efficiently, we develop a tabued scatter search (TSS) algorithm combined with SS with TS in this paper. A tabu search module (denoted as M-TS) is embedded in the framework of SS. The framework of TSS algorithm is presented in Fig. 2.

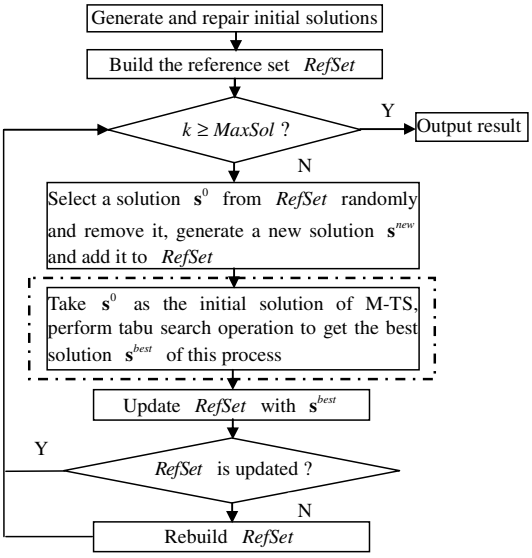

Fig. 2 Framework of TSS algorithm

As shown in Fig. 2, TSS starts with generating and repairing initial solutions. Then select b solutions from the initial solutions to build the reference set $RefSet$, including b_1 high-quality solutions and b_2 diverse solutions. If the stopping condition is not satisfied, i.e. the number of evaluated solutions k does not exceed the maximum number of solution evaluations $MaxSol$, select a solution $\mathbf{s}^0$ from $RefSet$ randomly and remove it. To ensure the size of $RefSet$, generate a new solution $\mathbf{s}^{new}$ and add it to $RefSet$. After that, M-TS is carried out with the initial solution $\mathbf{s}^0$ and the best solution of this process $\mathbf{s}^{best}$ is got. Then update $RefSet$ with $\mathbf{s}^{best}$. If $RefSet$ is not updated, rebuild it. Finally, output the optimization result.

From Fig. 2, we can see that TSS takes SS as the main framework and replaces subset generation and solution combination with M-TS. Compared with SS [14], TSS differs in initial solutions, M-TS and reference set update.

3.2.1 Initial Solutions

Generate $PSize$ initial solutions with the diversification generation method [14]. For the start time st_i of activity i, divide interval $[est_i, lst_i]$ in g sub-intervals. Let $M[i][h], i = 1,...,n, h = 1,...,g$ denote a frequency matrix. Each sub-interval is selected with probability inversely proportional to its frequency in M, and then a value for st_i is randomly generated in this interval. The procedure to generate diverse solutions can refer to [14]. If the new generated solution is infeasible, repair it. The quality of a solution is evaluated by objective function (1) and the diversity of a solution is evaluated by the minimum distance from the solution to the reference set [14].

3.2.2 M-TS

In TSS, M-TS is introduced to replace the subset generation and solution combination of SS. It takes $\mathbf{s}^0$ from $RefSet$ as initial solution and the best solution $\mathbf{s}^{best}$ obtained by M-TS is used to update $RefSet$. The flowchart of M-TS is shown in Fig. 3.

(1) Initial solution of M-TS

TS depends on the initial solution strongly. To improve the search quality and efficiency of M-TS, the initial solution $\mathbf{s}^0$ of M-TS is selected randomly from $RefSet$. Because $RefSet$ contains both high-quality and diverse solutions, taking the solution in $RefSet$ as the initial solution of M-TS can ensure the intensification and diversification of TSS algorithm.

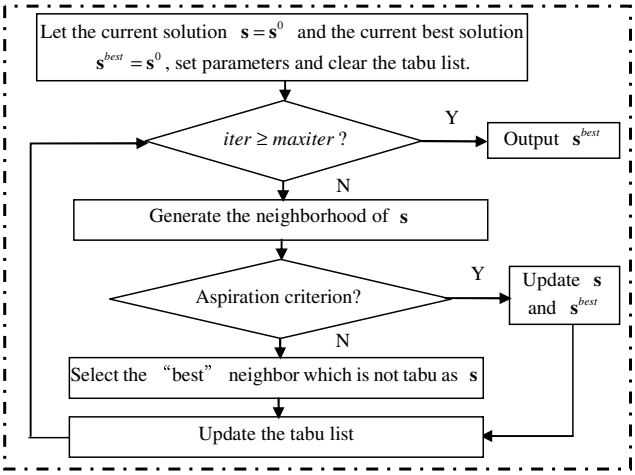

Fig. 3 Flowchart of M-TS

To avoid the search process trapping into circulation, remove the selected solution $\mathbf{s}^0$ from *RefSet*. Then generate a new feasible solution $\mathbf{s}^{new}$ according to diversification generation method. If $\mathbf{s}^{new}$ is better than the worst solution in *RefSet* ether in quality or diversity, add it to *RefSet*. Otherwise, generate a new solution again until a new solution is added to *RefSet*.

(2) Neighborhood structure

Neighborhood structure is an important element for TS. In this paper, a kind of neighborhood structure is constructed based on the coding. For a current solution $\mathbf{s}=(st_1,...,st_n)$, the procedure of constructing neighborhood is presented in Fig. 4.

Step 1. Select *NSize* activities randomly from the current solution $\mathbf{s}$;
Step 2. for each selected activity i ᐸ
perform the move operation on $\mathbf{s}$ and get a new neighbor solution $\mathbf{s}^c$;
if $\mathbf{s}^c$ is infeasible then
repair $\mathbf{s}^c$;
end if
add the neighbor solution $\mathbf{s}^c$ into the neighborhood $N(\mathbf{s})$.
end for

Fig. 4 Procedure of constructing neighborhood

As shown in Fig. 4, a neighbor solution is got by performing move operation on the selected activities of the current solution, and neighborhood $N(\mathbf{s})$ includes

$NSize$ neighbor solutions. The move operation means that generate a random value $random$ from $[est_i, lst_i]$ which is not equal to $st_i(\mathbf{s})$ and then let $st_i(\mathbf{s}) = random$. If the neighbor solution is infeasible, repair it and then add it into $N(\mathbf{s})$. The value of $NSize$ will be determined experimentally.

(3) Tabu element and move selection
Because the neighborhood is constructed based on move operations on activities, the tabu elements are the selected activities. When a neighbor solution is selected as the new current solution, the corresponding moved activity is put into the tabu list.

The move selection rule is to select the move that is non-tabu that has the lowest cost or satisfies the aspiration criterion. When all of the possible moves are tabu and none of them can satisfy the aspiration criterion, select any solution randomly as the current solution.

In each iteration of TSS, M-TS stops when the number of continuous search steps k exceeds the maximum number of search steps $maxiter$. The best solution $\mathbf{s}^{best}$ is obtained and further used to update $RefSet$. The value of $maxiter$ and the tabu list length L will be determined in experiments.

3.2.3 Reference Set Update

The reference set plays an important role in SS. The reference set update method is used to build and maintain a $RefSet$. In SS [14], $RefSet$ is updated by the solutions produced by subset generation and solution combination. In each iteration, several new solutions are used to update $RefSet$. In TSS, $RefSet$ is updated by the $\mathbf{s}^{best}$ obtained by M-TS. As M-TS is serial, only one solution is obtained to update $RefSet$ in each iteration, which is quite different with that in SS [14]. M-TS takes the solution in $RefSet$ as the initial solution and constructs an effective neighborhood for the current solution, and the best solution of the search process is saved as $\mathbf{s}^{best}$. Then $\mathbf{s}^{best}$ is further used to update $RefSet$, which improves the quality of the solution used to update $RefSet$ and reduces the number of updates. The performance of TSS will be tested in the following experiment.

4 Optimization Results and Discussions

This section tests the effectiveness of the TSS algorithm proposed in this paper. The computational experiments consist of two parts. In the first part, we tuned the TSS algorithm. In the second part, the effectiveness of TSS for the RACP was tested by comparing TSS with SS [14]. All algorithms will be performed for 10 runs for each instance and the best solution among 10 runs is taken as the result. All procedures were coded in C++ language under the Microsoft Visual Studio 2012 programming environment. All computational experiments were performed on a PC Pentium G630 2.7 GHz CPU and 2.0 GB RAM.

4.1 Generation of Problem Instances

The RACP and RCPSP share common data, thus it is easy to adapt existing instances of the RCPSP to the RACP. Progen [23] is an instance generator for RCPSP, which was used to generate the project scheduling problem library (PSPLIB) involving 30, 60, 90 and 120 activities and four resource types. In this paper, the tested instances are directly obtained from the PSPLIB and there are three important parameters as follows:

- Resource factor (RF): reflects the density of the different resource types needed by an activity and takes the values 0.25, 0.5, 0.75 and 1.0.
- Network complexity (NC): reflects the average number of immediate successors of an activity and takes the values 1.5, 1.8 and 2.1.
- Deadline factor (DF): reflects the deadline of the project such that $D = DF \cdot \max_{i=1,...,n} eft_i$. In this paper, DF is fixed at 1.2 according to [14].

Each combination of n, m, RF, NC, DF gives one instance, resulting in a total $4\times1\times4\times3\times1=48$ instances. For TSS, the parameters are set as below:

The number of initial solutions $PSize = 50$, the size of the reference set $b=10$, the number of high-quality solutions $b_1 = 7$, the number of diverse solutions $b_2 = 3$ and $g=4$. Stopping condition $MaxSol = \beta \cdot m \cdot n^2$, where β is a coefficient used to adjust the value of $MaxSol$. When n and m are determined, the value of $MaxSol$ is determined by β. β is a positive integer and its value will be tuned in the experiment. For each instance, the costs $C_k (k=1,...,4)$ are drawn from a uniform distribution $U[1,10]$ randomly. To make comparison easier, $C_k (k=1,...,4)$ are fixed at 2,8,5,1 in this paper. For the M-TS of TSS, we set the tabu list length L experimentally for each instance as follows:

$$L = [5\times1.2^{n/30}] \tag{6}$$

The coefficient 5 was obtained by tuning and the value of L is decided by n. The neighborhood size $NSize$ is set to 3,4,5,6 for $n=30,60,90,120$ respectively and the maximum number of search steps $maxiter$ is set to 20 experimentally.

4.2 Determining the Termination Condition

This section is designed to determine the termination condition $MaxSol$ by testing the performance of TSS under different $MaxSol$. For the 12 instances with 30 activities, $MaxSol = \beta \cdot 4 \cdot 30^2$. β is set to 5, 10, 15 and 25, the corresponding values of $MaxSol$ and the obtained best results under different values of $MaxSol$ are presented in Table 1. In Table 1, *Cost* represents the objective value of the solution, and *CPU* represents the CPU time consumed by TSS. The symbol "*" marks the best obtained *Cost* which cannot be improved with the increment of $MaxSol$.

From Table 1, we can see that TSS performed quite differently in 12 instances. Different instances got the obtained best solution under different values of $MaxSol$. For example, Instance 2 and 6 got their best solutions under $MaxSol = 5 \cdot 4 \cdot 30^2$, and Instance 5 got the best solution under $MaxSol = 15 \cdot 4 \cdot 30^2$. However, there are 7 instances (Instance 1, 3, 4, 9, 10, 11, 12) in 12 instances got their best solutions under $MaxSol = 10 \cdot 4 \cdot 30^2$ with small time cost. For instances with 60, 90 and 120 activities, TSS can also get better solutions in reasonable time under $MaxSol = 10 \cdot m \cdot n^2$. Therefore, $MaxSol = 10 \cdot m \cdot n^2$ can achieve a compromise between solution quality and CPU time, and it would be adopted in the TSS and SS in the following.

Table 1 The performance of TSS under different *MaxSol*

Instances	$5 \cdot 4 \cdot 30^2$		$10 \cdot 4 \cdot 30^2$		$15 \cdot 4 \cdot 30^2$		$25 \cdot 4 \cdot 30^2$	
	Cost	*CPU*	*Cost*	*CPU*	*Cost*	*CPU*	*Cost*	*CPU*
1	156	0.9	149*	1.7				
2	227*	1.0						
3	297	1.0	294*	1.9				
4	466	0.7	453*	1.4				
5	194	1.0	194	1.7	189*	2.6		
6	231*	1.1						
7	238	1.1	236	2.1	236	3.1	226*	5.0
8	321	1.0	308	2.0	308	2.9	303*	4.7
9	132	1.1	130*	2.2				
10	269	1.0	266*	1.8				
11	294	1.0	283*	2.7				
12	325	1.0	314*	1.8				

4.3 Comparing TSS with SS

To investigate the effectiveness of TSS, we compared it with SS [14]. For SS, the parameters $PSize$, b, b_1, b_2, $MaxSol$ and g take the same values with that in TSS. Fig. 5 and Fig. 6 respectively show the comparisons on solution quality and computational efficiency between TSS and SS.

In Fig. 5, *Ave. Cost* denotes the average cost of the 12 instances with the same activity number n. For $n = 30, 60, 90, 120$, the *Ave. Cost* obtained by TSS is smaller than that obtained by SS, demonstrating that TSS outperforms SS on solution quality. In Fig. 6, *Ave. CPU* denotes the average CPU time of the 12 instances with the same activity number n. For $n = 30, 60, 90, 120$, the *Ave. CPU* consumed by TSS is larger than that consumed by SS, which demonstrates that TSS is worse than SS on computational efficiency. From the above analysis, it can be seen that the proposed TSS algorithm is able to obtain better solutions by spending relatively large CPU time. Thus the TSS algorithm proposed in this paper is an effective algorithm to solve the RACP.

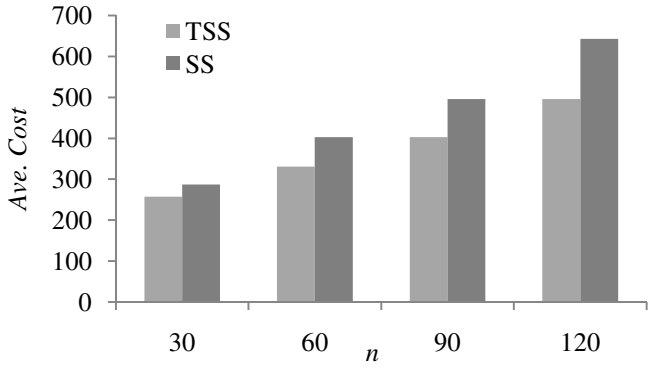

Fig. 5 Comparison on solution quality

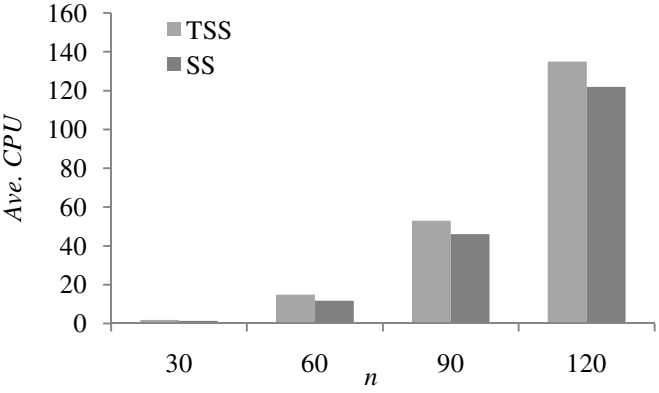

Fig. 6 Comparison on computational efficiency

5 Conclusions

In this paper, we have presented the TSS algorithm combined with SS and TS for solving the RACP. To solve the RACP directly, we use the start times of activities to code the schedule. The deadline constraint is handled in coding by setting an effective value interval for the start time of each activity. In computational experiments, TSS is compared with SS and the results show that TSS can obtain better solutions by spending relatively larger CPU time than SS, and it is an efficient method for the RACP.

For further research, TSS is recommended to solve other project scheduling problems. The RACP could be extended to the multi-mode RACP and the RACP with uncertainty, which is more practical.

Acknowledgement This work is partly supported by National Natural Science Foundation of China # 61273182.

References

1. Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G.: Scheduling subject to resource constraints: Classification and complexity. Discrete Appl. Math. **5**, 13–24 (1983)
2. Herroelen, W., De Reyck, B., Demeulemeester, E.: Resource-constrained project scheduling: A survey of recent developments. Comput. Oper. Res. **25**, 279–302 (1998)
3. Kolisch, R., Hartmann, S.: Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz, J. (ed.) Project Scheduling: Recent Models, Algorithms, and Applications, pp. 147–178. Kluwer Academic Publishers (1998)
4. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. Eur. J. Oper. Res **174**, 23–37 (2006)
5. Al-Fawzan, M.A., Haouari, M.: A bi-objective model for robust resource-constrained project scheduling. Int. J. Prod. Econ. **96**, 175–187 (2005)
6. Lambrechts, O., Demeulemeester, E., Herroelen, W.: A tabu search procedure for developing robust predictive project schedules. Int. J. Prod. Econ. **111**, 493–508 (2008)
7. Möhring, R.H.: Minimizing costs of resource requirements in project networks subject to a fix completion time. Oper. Res. **32**, 89–120 (1984)
8. Van Peteghem, V., Vanhoucke, M.: An artificial immune system algorithm for the resource availability cost problem. Flexible Int. J. Flexible Manuf. Syst. **25**, 122–144 (2013)
9. Demeulemeester, E.: Minimizing resource availability costs in time-limited project networks. Manage. Sci. **41**, 1590–1598 (1995)
10. Rangaswamy, B.: Multiple Resource Planning and Allocation in Resource-Constrained Project Networks. University of Colorado, Colorado (1998)
11. Rodrigues, S.B., Yamashita, D.S.: An exact algorithm for minimizing resource availability costs in project scheduling. Eur. J. Oper. Res. **206**, 562–568 (2010)
12. Drexl, A., Kimms, A.: Optimization guided lower and upper bounds for the resource investment problem. J. Oper. Res. Soc. **52**, 340–351 (2001)
13. Shadrokh, S., Kianfar, F.: A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. Eur. J. Oper. Res. **181**, 86–101 (2007)
14. Yamashita, D.S., Armentano, V.A., Laguna, M.: Scatter search for project scheduling with resource availability cost. Eur. J. Oper. Res. **169**, 623–637 (2006)
15. Yamashita, D.S., Armentano, V.A., Laguna, M.: Robust optimization models for project scheduling with resource availability cost. J. Sched. **12**, 67–76 (2007)
16. Qi, J.J., Guo, B., Lei, H.T., Zhang, T.: Solving resource availability cost problem in project scheduling by pseudo particle swarm optimization. J. Syst. Eng. Electron. **25**, 69–76 (2014)
17. Ranjbar, M., Kianfar, F., Shadrokh, S.: Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. Appl. Math. Comput. **196**, 879–888 (2008)
18. Shaffer, L.R., Ritter, J.B., Meyer, W.L.: The critical-path method. McGraw-Hill, New York (1965)

19. Glover, F.: Heuristics for integer programming using surrogate constraints. Decision Sci. **8**, 156–166 (1977)
20. Glover, F.: A template for scatter search and path relinking. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) AE 1997. LNCS, vol. 1363, pp. 1–51. Springer, Heidelberg (1998)
21. Martí, R., Laguna, M., Glover, F.: Principles of Scatter Search. Eur. J. Oper. Res. **169**, 359–372 (2006)
22. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**, 533–549 (1986)
23. Kolisch, R., Sprecher, A., Drexl, A.: Characterization and generation of a general class of resource-constrained project scheduling problems. Manage. Sci. **41**, 1693–1703 (1995)