Construction Resource Allocation and Leveling Using a Threshold Accepting–Based Hyperheuristic Algorithm

Georgios K. Koulinas[1] and Konstantinos P. Anagnostopoulos[2]

Abstract: In this study we propose a threshold accepting based hyperheuristic for solving in a single run both the resource-constrained project scheduling problem or resource allocation, and the resource leveling problem. Having their roots in the field of artificial intelligence, hyperheuristics operate in the "low-level" heuristics domain rather than in the solutions domain. The hyperheuristic has been implemented within a commercial project management software package. Low-level heuristics operate on the solution domain defined by the priority values that the software uses for resource allocation. A case example from the literature and computational experiments on randomly generated projects demonstrate that the hyperheuristic achieves good performance in a timely manner, improving the results provided by the software. **DOI: 10.1061/(ASCE)CO.1943-7862.0000492.**

CE Database subject headings: Resource Allocation; Optimization; Construction management; Scheduling.

Author keywords: Resource allocation; Resource leveling; Optimization; Construction management; Scheduling.

Introduction

Over the last fifty years, the critical path method (CPM) has been widely used as a project scheduling and control tool for supporting project managers to ensure project completion on time and on budget. In its basic form, CPM disregards resource management issues by assuming that resources required by activities are unlimited. However, project managers are frequently confronted with complex management situations due to limited availability of resources. The main concerns related to resource management were formulated long ago in the resource-constrained project scheduling problem (RCPSP) and the resource leveling problem (RLP).

In constrained-resource scheduling or resource allocation, limited resource availability during project execution is assumed. The aim is to minimize the project duration by rescheduling efficiently the project activities, since finding feasible solutions often extends the project completion time calculated by CPM with no resource limitations. On the other hand, resource leveling aims to reduce indirectly the total cost since it prevents from hiring and firing of resources on a short-term basis. The scheduling objective of resource leveling is to make the resource requirements as smooth as possible on a day-by-day basis, usually, with unlimited resource availability and without extending the project duration beyond the initial one that was calculated by the CPM (Demeulemeester and Herroelen 2002). Although in the typical form of resource leveling the resource availability is infinite, the problem can be considered with situations of limited resources leading to an extension of the initial project duration (Hiyassat 2000; Neumann and Zimmermann 1999). As Chan et al. (1996) note, though, in practice the distinction between resource allocation and resource leveling is not so clear, and a decision maker will consider a trade-off between the increasing resource availability and the prospect of shorter project durations.

In this paper we consider both resource leveling and allocation objectives integrated into a biobjective model. We propose a threshold accepting based hyperheuristic (TAH) to handle both resource allocation and leveling. Owing "a debt to work within the field of Artificial Intelligence on automated planning systems" (Burke et al. 2003a), hyperheuristics are a new approach to optimization aiming to manage solution methods rather the solutions themselves (Burke et al. 2003a; Chakhlevitch and Cowling 2008). The proposed algorithm runs within Microsoft Project (MS-Project), by acting on the priorities levels that MS-Project assigns on activities without any concerns for the heuristics used by the software.

Literature Review

Early attempts have been focused on inventing procedures to solve optimally the resource-constrained project scheduling and resource leveling problems. Regarding to RCPSP, Gavish and Pirkul (1991) used dynamic programming, and Demeulemeester and Herroelen (1992) and Brucker et al. (1998) developed branch-and-bound algorithms. Implicit enumeration (Ahuja 1976), integer programming (Easa 1989) and dynamic programming techniques (Bandelloni et al. 1994) implemented for the RLP.

However, both RCPSP and RLP have been proved to be NP-hard problems (Demeulemeester and Herroelen 2002; Neumann and Zimmermann 2000). Consequently, optimization methods seeking for exact solutions are suitable only for small-sized projects, due to the so called "combinatorial explosion" phenomenon. To get over this difficulty, many heuristic and metaheuristic procedures have been proposed in the literature to find an acceptable but not necessarily optimal solution.

Boctor (1990) and Kolisch (1996) investigated heuristic algorithms based on priority rules for the RCPSP. Concerning to the RLP, Burgess and Killebrew (1962) presented a priority rule based

[1]Ph.D. Candidate, Dept. of Production and Management Engineering, School of Engineering, Democritus Univ. of Thrace, Vas. Sofias 12, 67100 Xanthi, Greece (corresponding author). E-mail: gkoulina@pme.duth.gr

[2]Professor, Dept. of Production Engineering and Management, School of Engineering, Democritus Univ. of Thrace, Vas. Sofias 12, 67100 Xanthi, Greece. E-mail: kanagn@civil.duth.gr

Note. This manuscript was submitted on August 1, 2010; approved on September 22, 2011; published online on September 26, 2011. Discussion period open until December 1, 2012; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Construction Engineering and Management***, Vol. 138, No. 7, July 1, 2012. ©ASCE, ISSN 0733-9364/2012/7-854–863/$25.00.

J. Constr. Eng. Manage. 2012.138:854-863.

heuristic; Harris (1990) proposed the minimum moment and the so called PACK methods; Hiyassat (2000) devised a heuristic to reduce the needed calculations in the minimum moment approach; and Neumann and Zimmermann (2000) presented several heuristic procedures, and a variant of their exact branch-and-bound procedure.

Several metaheuristic approaches have also been proposed to reach near-optimum solutions in these problems such as genetic algorithms (Hegazy 1999; Chan et al. 1996; Hartmann 1998; Leu et al. 2000; Senouci and Eldin 2004), simulated annealing (Boctor 1996; Bouleimen and Lecocq 2003), neural network models (Savin et al. 1996), tabu search (Nonobe and Ibaraki 2001; Neumann and Zimmermann 2000; Pan et al. 2008), hybrid metaheuristics (Lova et al. 2009), particle swarm optimization (Zhang et al. 2006a, b), and ant systems (Christodoulou 2010) have also been proposed.

It is worth noticing that the RLP is a special case of the classic RCPSP problem, and various objective functions have been used in the past in order to modelize real-world resource leveling problems (Demeulemeester and Herroelen 2002). Similarly, various extensions of the basic RCPSP have been formulated, for instance, by introducing generalized precedence relations for the activities (RCPSP-GPR), and allowing for activity preemption (PRCPSP). Recent literature reviews for the RCPSP and its extensions can be found in (Hartmann and Briskorn 2010; Weglarz et al. 2011).

Formulation of the Problem

To formulate the problem, we use an acyclic activity-on-node (AoN) representation of project scheduling network. The network consists of n activities (nodes) and precedence relations (arcs) between activities. We denote d_i and f_i the duration and the finish time of activity i respectively, and assume that node $1(n)$ is the dummy activity of the project beginning (completion) with no ingoing (outgoing) arcs.

The resource allocation and leveling problem can be defined as follows:

$$\min M_x = \sum_{t=1}^{f_n} u_t^2 \tag{1}$$

$$\min f_n \tag{2}$$

subject to

$$f_i \leq f_j - d_j \quad \text{for all precedence relations } (i,j) \tag{3}$$

$$f_1 = 0, \quad d_1 = 0, \quad d_n = 0 \tag{4}$$

$$f_n \leq f_n^P \tag{5}$$

$$\sum_{i \epsilon P_t} u_{it} \leq U \quad \text{for } t = 1, \ldots, f_n \tag{6}$$

The aim is to find a feasible schedule so that both the moment M_x around the horizontal axis of the resource usage histogram and the project duration f_n are being minimized (Eqs. (1) and (2)). Eq. (3) assures that precedence relations are not violated, and Eq. (4) that the project starts at time 0. A single resource R is assigned to each activity, whose usage is assumed to remain constant throughout the progress of the activity. It is also considered a resource availability U which is constant during the project execution. The sum u_{it} of resource usage of all ongoing activities P_t in any time period t cannot exceed U (Eq. (6)).

This is a biobjective model consisting of the two traditional objectives of resource leveling and resource allocation. Eq. (1) represents the minimization of the moment around the horizontal axis of the resource usage histogram. This resource leveling objective expresses the minimization of the sum of the squared deviations of the resource requirements around the average resource requirement for each time period, i.e., the ideal histogram is a rectangle and the moment of the resource usage histogram is minimized as it approaches to the ideal one (Burgess and Killebrew 1962; Harris 1990). Also, numerous different objective functions have been proposed in the literature to obtain an acceptable resource profile (Demeulemeester and Herroelen 2002; Neumann and Zimmermann 1999). Since the evaluation function is independent of the hyperheuristics, only minor changes to Eq. (1) are needed to use another performance measure.

The second objective is the minimization of project duration f_n (Eq. (2)), which is traditionally the objective of resource allocation. On account of the limited resource units that can be assigned to activities, the project completion time after resource allocation will most likely be greater than the duration calculated initially by CPM, without considering the resource usage limit. As a result, a maximal acceptable duration f_n^P for completing the project after leveling and allocation is specified (Eq. (5)).

The model described by Eqs. (1)–(6) has no a single optimal solution, rather it has a set of nondominated solutions—a solution is said to be nondominated if there is no other feasible solution better in both the moment M_x and the project duration f_n. Although many multiobjective algorithms have been proposed for finding the nondominated solutions, i.e., the so called Pareto front, we use the oldest and much simpler approach of the linear aggregating functions. This method consists in using a trade-off coefficient λ to combine the two objectives into a single value to be optimized. Thus the two objective functions are combined into the following function, so as to maximize the weighed sum of relative improvements:

$$\max z' = \lambda \times \frac{M_x^P - M_x}{M_x^P} + (1-\lambda) \times \frac{f_n^P - f_n}{f_n^P} = 1 - \left[\lambda \times \frac{M_x}{M_x^P} + (1-\lambda) \times \frac{f_n}{f_n^P}\right] = 1 - z \tag{7}$$

where M_x^P and f_n^P are the moment and the maximum project duration respectively resulting from the application of the standard leveling feature of the software. Given that $\max(1-z) = \min(z-1)$, the aggregated objective function becomes:

$$\min z = \lambda \times \frac{M_x}{M_x^P} + (1-\lambda) \times \frac{f_n}{f_n^P} \tag{8}$$

The scalar $0 \leq \lambda \leq 1$ reflects the preferences of the decision maker. When $\lambda = 0$, the project manager completely ignores the leveling of resources; conversely, when $\lambda = 1$, he/she only wants to minimize the fluctuations in resource usage.

Hyperheuristic Algorithms

Although the core idea of multilevel algorithms is an old one (Fischer and Thomson 1963), the term "hyperheuristic" has been recently proposed to describe a multilevel computational procedure in which an upper heuristic layer controls the application of some underlying heuristic methods rather the solutions themselves (Burke et al. 2003a). These low-level heuristics in turn depend upon the characteristics of the solution space under exploration. Thus, only the low-level heuristics are accessible to the

J. Constr. Eng. Manage. 2012.138:854-863.

hyperheuristic, which has no knowledge of both the solutions domain and the function of the low-level heuristics. The lower heuristic layer can consist of simple heuristic operators, metaheuristic algorithms or even other more simple hyperheuristics.

Hyperheuristics have been emerged as an alternative approach for solving NP-hard problems, by overcoming some drawbacks of the popular metaheuristic algorithms, especially their dependence on the problem that they try to solve. Hyperheuristics are metaheuristic algorithms in the sense that they can be applied to a variety of problems (Burke et al. 2003a). Unlike metaheuristics that guide the search in a space of solutions, though, a hyperheuristic operates in a space of heuristics choosing and applying one low-level heuristic from a given set at each decision point (Chakhlevitch and Cowling 2008). Despite the significant progress in developing metaheuristic algorithms for a wide variety of application areas so far, including project scheduling problems, such approaches are strongly dependent on incorporating the expertise of a professional in a given problem domain. Instead, a hyperheuristic usually aims at reducing the amount of domain knowledge in the search process. The resulting approach should be simple and easy-to implement, and it would be robust enough to effectively treat a range of problem instances from a variety of domains (Burke et al. 2003a).

Hyperheuristic algorithms have been used for treating scheduling problems in industrial and manufacturing environment, such as the well-known job shop scheduling problem (Fischer and Thomson 1963; Dorndorf and Pesch 1995; Storer et al. 1995) and the flow shop scheduling problem (Ouelhadj and Petrovic 2010), and in resource leveling (Anagnostopoulos and Koulinas 2010) and resource-constrained project scheduling problem (Anagnostopoulos and Koulinas 2011). In addition, hyperheuristics have been developed for a variety of applications such as large-scale university exam timetabling problems (Burke et al. 2003b) and determining shipper sizes for storage and transportation to reduce packaging waste (Dowsland et al. 2007). A review of hyperheuristic algorithms applications can be found in the study of Chakhlevitch and Cowling (2008).

Several metaheuristic-based strategies for designing hyperheuristics have been investigated in the literature: simulated annealing (Storer et al. 1995; Dowsland et al. 2007; Anagnostopoulos and Koulinas 2010), tabu search (Burke et al. 2003b), variable neighborhood search (Qu and Burke 2009), and GRASP (Anagnostopoulos and Koulinas 2011) based hyperheuristic. Also, Burke et al. (2005) proposed an ants system based algorithm, and Dorndorf and Pesch (1995), and Han and Kendall (2003), genetic algorithm based hyperheuristics.

Proposed Hyperheuristic

The proposed algorithm operates within MS-Project by altering the priorities assigned to activities. As the most commercial software packages do, MS-Project permits the assignment of priorities to project activities allowing the project manager to control how the activities are "leveled" in relation to one another. It is noted that in MS-Project terminology "leveling" means creating a feasible solution for resource allocation using the heuristics embedded in the software, and as side-effect a better resource leveling.

Priorities are defined either as numbers (0 to 1000) or as linguistic values ("lowest" to "highest"). An activity that has been assigned the "highest" (or 1000) priority is not leveled. Since the priority of each activity affects the project schedule, leveling with modified priorities has an influence to the resource profile. The developed hyperheuristic operates through the low-level heuristics on the priorities domain, and then via the MS-Project heuristics on the scheduling. Consequently, MS-Project scheduling processes are not transparent to the proposed algorithm.

The hyperheuristic was coded using the embedded in the software programming language Visual Basic for Applications, and an MS-Project add-in has been developed to increase user friendliness. This software-depended algorithm provides the project managers with an effective and entirely automated tool that improves the resource allocation and the resource usage histogram resulting from the standard feature of the software.

Threshold Accepting Hyperheuristic

The threshold accepting hyperheuristic (TAH) is based on the threshold accepting metaheuristic algorithm (Dueck 1990). According to the core idea of threshold accepting metaheuristic, the algorithm tries to escape from local optima by accepting some solutions that increase the value of the objective function using a deterministic acceptance criterion. This metaheuristic fits well with the main characteristic of the core hyperheuristic idea for creating simple and easy-to-implement procedures.

The crucial factor of the algorithm is the definition of an appropriate threshold sequence τ_r which determines whether a new solution could be accepted or not. The thresholds are high at start to allow for diversification, and are decreased during the searching process. After a number of iterations, the threshold value is very small and TAH converts into a local search hyperheuristic to allow for intensification of the search. Since not much is known about how to choose this sequence in a way to improve the performance of the algorithm, the threshold sequence is often determined in a rather ad hoc approach, and linearly decreasing sequences appear to be preferred (Winker and Maringer 2007). In this study the thresholds are calculated according to the equation $\tau_{r+1} = \tau_r - b$, where b is a constant. The steps of the TAH algorithm are illustrated as follows:

1. Initialize the number of iterations per cycle Cycle_Iter and the sequence of thresholds τ_r, $r = 1, 2, \ldots r_{\max}$.
2. Generate the initial schedule S_c using the automatic leveling feature of the software and compute z_c.
3. Set $S_{\text{best}} = S_c$, $r = 0$ and Count_Iteration $= 1$ while stopping condition is not met do.
4. While Count_Iterations $\leq$ Cycle_Iter do Set $r = r + 1$.
5. Select a low-level heuristic l with probability $p(l)$, apply it to the priorities of current schedule S_c, and compute the new schedule S_n.
6. If ($z_n < z_{\text{best}}$) then set $S_{\text{best}} = S_n$ and $z_{\text{best}} = z_n$.
7. If ($z_n - z_c \leq \tau_r$) then accept the new solution, and set $z_c = z_n$ and $S_c = S_n$.
8. Set Count_Iterations = Count_Iterations + 1 end while.
9. Update all choice probabilities $p(l)$; Set Count_Iterations = 1 end while Output S_{best}.

The hyperheuristic starts with the initial schedule S_c generated using the automatic leveling feature of the software, chooses with a probability a low-level heuristic, and applies it once to the priorities domain (step 2). The resulting new solution S_n is accepted if it is better than S_c, or if it is "not much worse" than the current one, meaning that the algorithm moves to solutions with higher objective function value in order to avoid being trapped in local minimums (step 7). The choice probability $p(l)$ of each low-level heuristic remains constant during a fixed number of iterations (Cycle_Iter) that correspond to a cycle. For the first Cycle_Iter iterations all heuristics have equal choice probabilities. Afterward a bias which is based on an elementary learning mechanism is introduced in this choice, in a manner that the choice probability $p(l)$ for each heuristic l to be a function of its past performance. The probability $p(l)$ is updated after Cycle_Iter iterations, by calculating the

J. Constr. Eng. Manage. 2012.138:854-863.

cumulative improvement caused by heuristic l over the total number of its calls (step 9). These (normalized) probabilities will be used throughout the next cycle, i.e., the next Cycle_Iter iterations.

Low-Level Heuristics

TA hyperheuristic controls eleven low-level heuristics. Based on simple moves such as "swap" and "replace", these heuristics comply with the requirement for simplicity introduced by the hyperheuristic framework. The low-level heuristics operate on the activities priorities. Nine priority levels are used in this study, ranged from 100 to 900. It is possible to fix the priorities of some activities, and the algorithms to operate on the others.

The heuristics are partitioned into four groups according to the criterion used for choosing the activity whose priority will be modified. Group A contains heuristics based on random selection, and heuristics of group B choose activities according to their total slack (TS). The heuristics belonging to groups C and D find the time periods Δ and δ with the maximum $u_{\max}$ and the minimum $u_{\min}$ resource demand over the project horizon respectively, and modify the priority of an activity according to its relative contribution to these undesirable facts. The values for selecting activities are calculated by the equations:

$$\bar{u}_i = \frac{u_{i\Delta}}{u_{\max}} \quad \text{for } iP_\Delta \tag{9}$$

$$\bar{\bar{u}}_i = \frac{u_{i\delta}}{u_{\min}} \quad \text{for } iP_\delta \tag{10}$$

where $P_\Delta(P_\delta)$ is the set of all activities in progress in time period $\Delta(\delta)$. A description of the low-level working model follows:

Group A—Random based low-level heuristics

- L1: select at random two activities and swap their priorities.
- L2: select at random one activity and replace its priority with a new one randomly selected.

Group B—TS based low-level heuristics

- L3: assign the lowest priority value (100) to the activity with the largest TS.
- L4: assign the highest priority (900) to the activity with the smallest TS.
- L5: select the activity with the largest TS and replace its priority with a random one selected within the interval (100, 500). Select the activity with the smallest TS and replace its priority with a random one selected within the interval (500, 900). In case of a tie, select the activity at random.

Group C—Maximum resource demand based low-level heuristics

- L6: assign the "lowest" priority (100) to the activity with the largest $\bar{u}_i$ value.
- L7: assign the "highest" priority (900) to the activity with the smallest $\bar{u}_i$ value.
- L8: select the activity with the largest $\bar{u}_i$ and replace its priority with a random selected within the interval (100, 500). Select the activity with the smallest $\bar{u}_i$ and replace its priority with a random selected within the interval (500, 900). In case of a tie, select the activity at random.

Group D—Minimum resource demand based low-level heuristics

- L9: assign the "highest" priority (900) to the activity with the largest $\bar{\bar{u}}_i$ value.
- L10: assign the "lowest" priority (100) to the activity with the smallest $\bar{\bar{u}}_i$ value.
- L11: select the activity with the largest $\bar{\bar{u}}_i$ and replace its priority with a random selected within the interval (500, 900). Select the activity with the smallest $\bar{\bar{u}}_i$ and replace its priority with a random selected within the interval (100, 500). In case of a tie, select the activity at random.

The rationale behind the criterion based heuristics is to reduce the resource demand peaks over the project duration, and to increase the consumption in days with very small demand forming valleys in the resource profile graph.

Illustrative Example

The TA hyperheuristic has been applied to the example project solved in (Hegazy 1999) with 20 activities and six resources assigned. Table 1 summarizes the duration, predecessors and daily resource usage for every activity. The daily constraint (Eq. (6)) is fixed at 16 units for the resource "R4" which is considered as critical. The initial project duration according to CPM calculations is 32 days, the moment M_x is equal to 3375 and the resource usage on a day-by-day basis ranges from 2 to 21 units. The archive of this example, in MS-Project format, is available at http://www.civil.uwaterloo.ca/tarek/hegazyfre1.html.

The stopping condition was defined to be 2,000 iterations. The parameter b and the initial value for the threshold τ were set equal to 0.001 and 1.0, respectively. Fixed after trial experimentation, these settings assure that the probability of accepting a non improving solution is high during the first 50% of iterations to favor diversification in search. Thereafter the threshold value is very small, so the probability of accepting a non improving solution virtually converge to zero and the algorithm switches to local search hyperheuristic to allow for intensification.

Table 2 presents the results reached by the TAH and the genetic algorithm (GA) developed by Hegazy (1999). The results are organized according to the objective function used. At the bottom of each column are illustrated the project duration, the moment M_x of resource R4, the value of the objective function z and the resource usage range. The second column contains the results

Table 1. Example Data

Activity	Duration	Predecessors	Resource requirements per day					
			R1	R2	R3	R4	R5	R6
A	6		5	2	2	2	7	4
B	3		3	5	2	3	9	6
C	4	A	2	4	4	2	3	1
D	6		5	4	3	5	5	4
E	7	A, B	3	5	2	3	8	0
F	5	C	4	1	4	9	2	5
G	2	D	4	1	4	3	9	8
H	2	A, B	5	5	4	0	9	1
I	2	G, H	3	2	4	3	4	2
J	6	F	1	5	4	6	7	3
K	1	C, E	3	3	2	4	5	1
L	2	E, G, H	3	2	2	8	3	4
M	4	I, K	2	2	2	2	4	8
N	2	F, L	1	4	4	3	4	1
O	3	L	5	5	4	6	2	3
P	5	J, M, N	3	2	3	4	7	8
Q	8	O	4	5	4	2	3	4
R	2	D, O	5	3	3	3	7	8
S	6	P, R	2	4	6	2	3	4
T	2	Q	1	6	2	7	5	2
Daily resource availability			7	10	10	16	18	13

J. Constr. Eng. Manage. 2012.138:854-863.

achieved by the automatic leveling feature of MS-Project, which reduces the M_x to 2449, the daily fluctuation range of R4 to 12 units, and extends project duration to 49 days. This makespan is considered as the maximal acceptable project duration f_n^P(Eq. (5)). This is the starting point for the hyperheuristic.

The next two columns show the set of priorities and the achieved results using the minimization of project duration as objective function ($\lambda = 0$, Eq. (8)). As shown, the GA reduces the duration to 44 days and the usage range to 10 units, while the TAH algorithm finds a better solution since it improves the duration to 43 days. Note that Pan et al. (2008) found the same duration of 43 days by resolving the same example with their tabu search algorithm and mentioned that this is optimal in terms of makespan minimization. This smaller duration increases M_x to 2475 and the usage range to 11 units, which are acceptable because in this case the predefined objective is the project duration minimization.

The same example has been also solved using as objective function the z value with $\lambda = 50\%$, i.e., the minimization of both the project duration and the moment M_x of R4 equally weighted (Eq. (8)). Hegazy (1999) reports that the GA achieves a z value of 92.17% ($M_x = 2265,\ f_n = 45$). TAH found better solution than GA, since it reaches a z value of 91.43%, and smaller resource usage range (8 units) as well. This solution has a duration of 48 days which is smaller than the f_n^P constraint (Eq. (5)). TAH required 47 s to meet the stopping condition of 2,000 iterations.

From this example case we see that the hyperheuristic improves the automatic leveling feature of MS-Project and enhances the software's capability to allocate the resource efficiently. Moreover TAH reaches better results than the GA. These conclusions clearly cannot be generalized based exclusively on a sole example project.

Table 2. Results for the Example

		Priorities			
		Proj. Dur. 100%		Proj. Dur. 50%	
		Mx of R4 0%		Mx of R4 50%	
Activity	Software	GA	TAH	GA	TAH
A	100	900	900	700	800
B	100	500	100	300	800
C	100	800	200	500	800
D	100	800	800	700	800
E	100	800	900	100	600
F	100	800	400	600	800
G	100	500	100	100	100
H	100	100	100	600	700
I	100	800	100	600	800
J	100	500	100	600	100
K	100	500	100	800	800
L	100	500	100	600	800
M	100	800	400	700	700
N	100	200	300	500	700
O	100	600	600	700	800
P	100	800	500	500	700
Q	100	100	200	100	700
R	100	700	800	700	800
S	100	500	100	400	700
T	100	100	600	800	600
Duration (days)	49	44	43	45	48
M_x	2449	2381	2425	2265	2079
	100	89.80	87.76	92.17	91.43
Resource range	12	10	11	10	8

Advanced Analysis

Algorithms and Dataset

In this section, we study the computational behavior of the TAH as a function of a number of parameters, and at the same time we compare it with two other hyperheuristics: a simulated annealing hyperheuristic (SAH) proposed in (Anagnostopoulos and Koulinas 2010) for solving solely the resource leveling problem, and a multistart random descent hyperheuristic (MRDH) for scheduling a sales summit (Cowling et al. 2001). It should be emphasized that our aim is to better understand the computational behavior of TAH by investigating the performance of these single-point (as opposed to population based) algorithms on a problem for which no hyperheuristic has been proposed in the literature. For a fair comparison, both the MRDH and SAH manage the set of low-level heuristics proposed in the present paper.

The SA hyperheuristic uses the same bias process and stopping condition with TAH. The main difference between the two algorithms is the use by SAH of a probabilistic acceptance criterion instead of the deterministic one used in TAH. The probability of accepting a worse solution depends on the factor of temperature T which is initially high to allow for diversification and decreases according to the cooling schedule $T_i = a \times T_{i-1}$ to allow for intensification. The parameter a represents the rate of temperature reduction (cooling rate).

The MRDH is an adaptation to the problem under study of a hyperheuristic proposed by Cowling et al. (2001). Additionally, the MRDH embodies a recursively applied multistart process in the priorities domain to escape from local optima. The low-level heuristics are applied in a descent fashion, i.e., each selected heuristic is reapplied until no improvement to the current solution achieved. MRDH starts by randomly constructing priorities for the activities. Then it chooses at random a low-level heuristic and applies it once to the solution space. The resulting new solution is accepted if it is better than the current one, and the chosen low-level heuristic is reapplied as long as no further improvement is observed. Otherwise, the new solution is rejected and the hyperheuristic selects another heuristic. This process continues until all the available low-level heuristics have been applied at least once to the solution space. Subsequently, a new random solution is constructed and the hyperheuristic restarts. It is worth noticing that this algorithm does not require parameters to be tuned, apart from the stopping condition.

Fig. 1 illustrates the userforms of our MS-Project add-in used for defining the settings of each algorithm. Initially, the user defines, individually or by groups, the low-level heuristics to be used by the algorithm. In this study all available heuristics are selected. Both the stopping condition—the maximum number of iterations or the allowable total run time for each hyperheuristic—and the number of priority levels are also determined. Finally, the user selects one of the proposed algorithms, and if TAH (SAH) is selected values for the initial threshold τ and the parameter b (the temperature T and the cooling rate a) are defined. Both algorithms require also setting the value of Cycle_Iter parameter. MRDH needs no additional parameters definition. All algorithms have run on a PC Pentium IV 3.0 GHz with 512 MB RAM, running under Microsoft Windows XP. The stopping condition, after preliminary testing, was set to 2,000 iterations.

J. Constr. Eng. Manage. 2012.138:854-863.

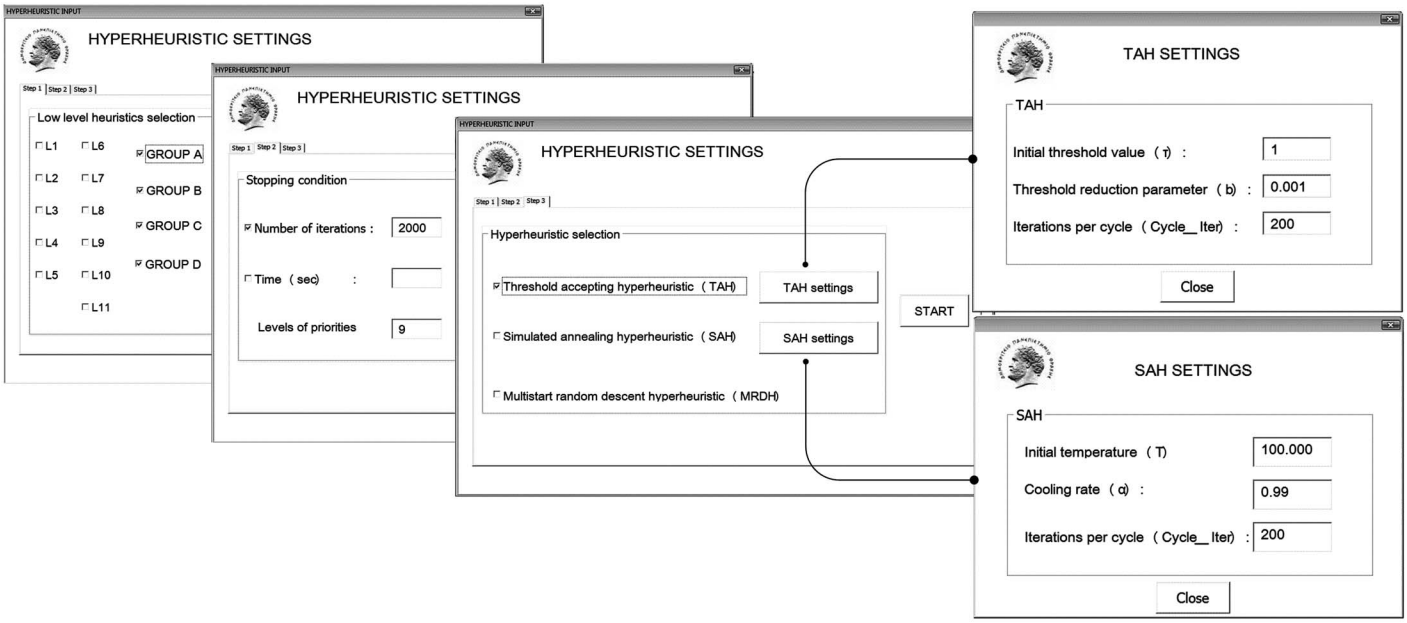

Fig. 1. User defined settings for the hyperheuristics

We study the computational behavior of the hyperheuristics as a function of three factors: the number of activities n, the order strength OS (or network density), and the resource availability Res. The experimental analysis aims to determine if the algorithmic performance is affected by any of these factors, and if different levels of factors cause distinguishable better algorithmic performance. Although well-known benchmarks used in the literature for the RCPSP, no such set exists for the problem described by Eqs. (1)–(6), so it is necessary to generate our own benchmark set.

The algorithms have been applied on a set of random generated projects created with the network generator RanGen1 (http://www.projectmanagement.ugent.be/rangen.php). RanGen1 generates randomly AoN networks using as input parameters the number of activities n and the order strength OS.

The first two factors are the input parameters of the network generator. The parameter n, taken equal to 20, 30, 40, 50 and 60 activities, determines the size of the project. The density of the network OS is defined as the total number of precedence relations p, including the transitive ones, divided by the theoretical maximum number of such precedence relations (Demeulemeester and Herroelen 2002):

$$\text{OS} = \frac{2p}{n(n-1)} \tag{11}$$

As most project activity networks are rather sparse, networks with OS value of 0.1, 0.15 and 0.20 are studied. Apart from these network morphological factors, the possible effect of the resource availability to the algorithmic performance is examined. The resource availability Res is set equal to the 2/3, 1/2 and 1/3 of the maximum daily resource demand for each generated project.

Since five levels for n, three levels for OS and three levels for Res are considered, $5 \times 3 \times 3 = 45$ different combinations are studied. In order to improve the reliability of the analysis, three networks for each combination have been generated and three replicates of each instance have been solved, that is $45 \times 3 \times 3 = 405$ instances. Each hyperheuristic is evaluated according to the objective function z with $\lambda = 0.5$.

Results

Table 3 summarizes the results of the hyperheuristics application to the random networks. Due to the large number of instances, only average z values for each parameter level are presented. As shown, MRDH remains the most efficient algorithm for three of the five levels of parameter n, apart from networks with 30 and 40 activities, wherein TAH is slightly superior. Practically, MRDH and TAH have the same performance for the different levels of the network size parameter. This conclusion is also valid for the levels of the density parameter OS. The MRDH hyperheuristic achieves the larger improvement to the objective function for every OS level, and TAH follows closely. The SAH achieves the worst results for each level of this parameter. Regarding to the factor Res, there are differences in the performance of algorithms for the various levels. For Res set to 1/3 of the maximum, TAH is more effective with SAH following, while MRDH has slightly worse performance than the latter. For Res equal to 1/2, TAH is still the best algorithm but MRDH achieves better results than SAH. Finally, for Res set to 2/3, SAH remains in the last place, TAH is the second more

Table 3. Average z Values (%) for the Hyperheuristics

n	TAH	SAH	MRDH
20	97.71	98.31	97.36
30	97.47	98.33	97.48
40	97.31	98.13	97.34
50	97.39	98.43	97.35
60	97.41	98.47	97.39
OS	TAH	SAH	MRDH
0.10	96.93	98.03	96.80
0.15	97.47	98.25	97.45
0.20	97.97	98.73	97.88
Res	TAH	SAH	MRDH
1/3	98.47	98.64	98.68
1/2	97.21	97.89	97.49
2/3	96.69	98.47	95.96

J. Constr. Eng. Manage. 2012.138:854-863.

efficient algorithm, and MRDH achieves the largest improvement with average z value of 95.96%.

The algorithmic performance for the several levels of the analysis factors is displayed in Fig. 2–4. Fig. 2 shows clearly the supremacy of TAH and MRDH over SAH. The size of the network, though, does not affect the performance of the algorithms significantly, since very small differences in the performance of algorithms are observed. This conclusion is also confirmed from the analysis of variance (ANOVA) that has been performed on the results of the 405 instances solved by each algorithm. The core idea of ANOVA is to assume that all the nonrandom variations in experimental observations are due to differences in mean performance values at various levels of the experimental factors (Montgomery 2001). In this analysis, ANOVA aims at demonstrating if different factor levels cause a significantly different algorithmic performance. Table 4 contains the results of ANOVA as derived by using the SPSS Statistics 17.0 software package. Concerning the factor n, the p-value for all hyperheuristics is greater than the significance

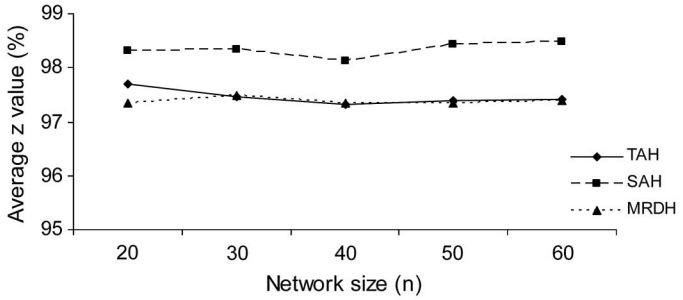

Fig. 2. Hyperheuristic performance as function of the number of activities

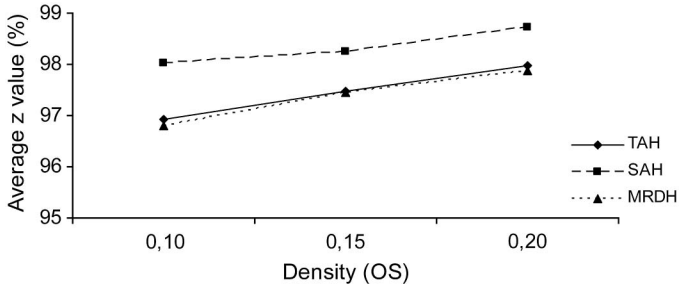

Fig. 3. Hyperheuristic performance as function of the network density

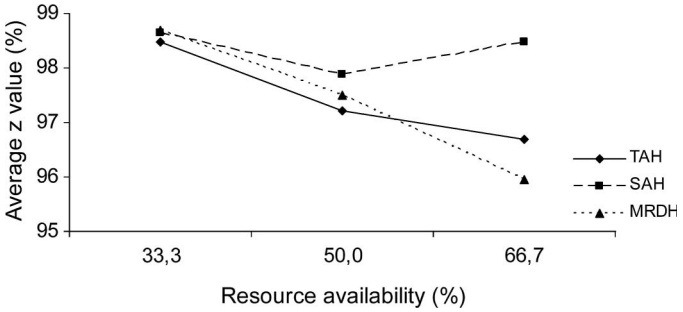

Fig. 4. Hyperheuristic performance as function of the resource availability

level 5% (the confidence intervals are 95%), and thus its effect on the performance of the algorithms is considered as negligible.

Fig. 3 illustrates the performance for different levels of OS. As shown, both TAH and MRDH appear to have clearly better results than SAH, the MRDH has a small advantage over TAH, and the performance of all algorithms deteriorates as OS increases. Since the p-values of all algorithms for parameter OS are very close to zero (Table 4), it could be said that the algorithmic performance depends on the network density. In addition, we argue that the resource availability affects the algorithmic performance, as Fig. 4 illustrates, since increasing availability helps TAH and MRDH to find better solutions, and all three algorithms have almost similar behavior for low availability. As shown in Table 4, the p-values are smaller than 5%, verifying the effect of Res factor in algorithmic performance. The interactions of $n \times$ OS and $n \times$ Res mean that the effect of one parameter (n) depends on the effect caused by the other two factors. Also, an interaction between OS $\times$ Res and a three-way interaction (n $\times$ OS $\times$ Res) are observed. The main conclusion derived from these interactions is that more complex and resource-constrained projects prevent algorithms from achieving large improvement.

There exist instances for which the hyperheuristics failed to improve the solution obtained by the software. This appears because MS-Project finds a good enough solution for these networks such that it could not be found a better one by a hyperheuristic. More specifically, the TAH failed to improve 16 (3.95%), the SAH 42 (10.37%) and the MRDH 55 (13.58%) of the 405 instances. A further investigation of the experimental results is shown in Table 5, in which an entry (i, j) contains the number of instances that the algorithm i has improved but not j. Regarding to SAH, we observed 33 instances which failed while TAH succeeded and 21 that MRDH succeeded, and 20 instances that both TAH and MRDH improved the software's solution but not SAH. The results of Table 5 confirm that the TAH is the most reliable among the three algorithms. On the other hand, MRDH is very possible to be trapped quickly in local minimums, and thus the multistart process should be applied more frequently to increase its performance.

Another issue that has been investigated is the number of iterations that every algorithm required on average for finding the best found solution for all the instances that achieved improvement. More specifically, the TAH achieved on average its best solution in 851 iterations, the SAH in 159, and MRDH in 1014 iterations. Additionally, in order to show how early or late in the search each algorithm succeeds, we have grouped the number of the iterations that every hyperheuristic reached its best found solution into ten classes (Table 6). For TA hyperheuristic, it is observed that 200 iterations were enough to find a local optimum for about 25% of the instances, and the rest classes contain almost equal number of elements. As expected, TAH performs better when its threshold is large, and its efficiency reduces with decreasing threshold.

Concerning the two other algorithms, SA hyperheuristic reaches 92% of the better than the software found solutions within the first 200 iterations. So SAH practically works only with very high temperature values; and if this parameter value decreases, it is frequently trapped into local minimums. This characteristic is considered to be responsible for its poor performance regarding to other hyperheuristics, and for more failures in treating instances (42 against 16) than TAH. As for MRDH, the ten classes have almost the same number of instances meaning that this algorithm can practically achieve its best value everywhere in the search, apparently because the multistart process helps it to escape from local minimums. Finally, it is noted that no best solution was found by any algorithm in the last iteration. These findings prove that, for

J. Constr. Eng. Manage. 2012.138:854-863.

Table 4. ANOVA Table for the Advanced Analysis

Source	Dependent Variable	Type III Sum of Squares	df	Mean Square	F	Sig.
n	z_TAH	0.001	4	0.000	0.946	0.437
	z_SAH	0.001	4	0.000	0.993	0.411
	z_MRDH	0.000	4	3.070E-5	0.112	0.978
OS	z_TAH	0.007	2	0.004	18.468	0.000
	z_SAH	0.003	2	0.002	11.411	0.000
	z_MRDH	0.008	2	0.004	14.548	0.000
Res	z_TAH	0.023	2	0.011	57.620	0.000
	z_SAH	0.004	2	0.002	13.907	0.000
	z_MRDH	0.050	2	0.025	91.660	0.000
$n \times$ OS	z_TAH	0.005	8	0.001	2.925	0.004
	z_SAH	0.005	8	0.001	4.134	0.000
	z_MRDH	0.006	8	0.001	2.868	0.004
$n \times$ Res	z_TAH	0.004	8	0.001	2.720	0.006
	z_SAH	0.007	8	0.001	5.584	0.000
	z_MRDH	0.005	8	0.001	2.231	0.025
OS $\times$ Res	z_TAH	0.005	4	0.001	6.228	0.000
	z_SAH	0.004	4	0.001	5.967	0.000
	z_MRDH	0.003	4	0.001	3.085	0.016
$n \times$ OS $\times$ Res	z_TAH	0.010	16	0.001	3.173	0.000
	z_SAH	0.011	16	0.001	4.612	0.000
	z_MRDH	0.013	16	0.001	2.991	0.000

Table 5. Matrix of Pairwise Algorithmic Comparisons

	Failures		
Algorithm of comparison	TAH	SAH	MRDH
TAH	—	33	40
SAH	7	—	34
MRDH	1	21	—
Both the other	0	20	27

Table 6. Number of Instances as a Function of Iterations Needed by Each Algorithm for Finding a Local Optimum

	Number of instances		
Iterations	TAH	SAH	MRDH
1–200	98	335	33
201–400	36	4	37
401–600	33	1	35
601–800	30	4	37
801–1000	25	2	29
1001–1200	29	3	41
1201–1400	32	2	31
1401–1600	35	4	26
1601–1800	32	3	35
1801–2000	39	5	46

this experiment, the 2000 iterations are enough for the algorithms to find good solutions.

Table 7 contains the average running time needed by each algorithm to complete the stopping condition. As expected, the running time strongly depends on the size of the network n. TAH proved to be the fastest algorithm for all parameter levels and MRDH the slowest. It seems that the recursively applied multistart process causes delay to MRDH, while the deterministic way in which TAH accepts a worst solution appears to be less time consuming. Regarding to the effect of OS on the running time, the small differences confirm that the average running time is independent of the network density. Finally, the factor Res affects the running time of the algorithms, since larger availability helps each algorithm to allocate faster and easier the constrained resource. As is the case with parameter n, TAH needs less time than any other algorithm, SAH follows and MRDH remains the slowest for the studied levels of resource availability factor.

Table 7. Time Efficiency of the Hyperheuristics (sec)

n	TAH	SAH	MRDH
20	32.07	47.07	53.52
30	56.32	78.58	92.79
40	80.83	108.63	133.03
50	100.96	132.48	167.49
60	132.96	171.91	217.82
OS	TAH	SAH	MRDH
0.10	78.22	103.67	130.64
0.15	82.77	111.13	136.45
0.20	80.90	108.40	131.70
Res	TAH	SAH	MRDH
1/3	102.99	134.96	174.36
1/2	79.82	107.80	128.37
2/3	59.07	80.45	96.06

To sum up this analysis, the three hyperheuristics construct solutions of competitive quality against the software. Hyperheuristics TAH and MRDH are practically equivalent and both better than SAH regarding the obtained solutions; TAH is better than MRDH regarding the time efficiency; and most reliable since it has improved the most instances. Thus we can conclude that the TAH is the better algorithm. Finally, it is noted that the deterministic

J. Constr. Eng. Manage. 2012.138:854-863.

criterion used by the TAH for accepting a worse solution was be proved more efficient and less time consuming than the stochastic one used by the SAH.

Conclusion

Hyperheuristics are an emerging approach to optimization that allows professionals to rapidly produce solutions of acceptable quality in real-world problems. In this paper we presented a threshold accepting hyperheuristic for solving in a single run both the resource leveling and allocation problems. The hyperheuristic controls eleven low-level heuristics, and was implemented within a widely used project management software package.

The proposed hyperheuristic has been programmed as an upper algorithmic layer that controls a set of heuristics based on simple moves such as "replace" and "swap". The low-level heuristics are applied to the priorities of the project activities, and the hyperheuristic has no access to the solution domain but only selects the heuristic that would be applied to the priorities. The TA hyperheuristic, and the two other hyperheuristics included in this paper for comparison purposes, i.e., a simulated annealing hyperheuristic (SAH) and a random descend hyperheuristic with a multistart process embedded (MRDH), are based on single-point metaheuristic approaches in the sense that the corresponding hyperheuristic applies one low-level heuristic at each iteration.

The proposed threshold accepting hyperheuristic has been proved a promising procedure for solving both resource allocation and leveling problems. Applied to an example project from the construction project management literature (Hegazy 1999), the proposed approach found better solutions than the GA metaheuristic that initially solved the case example. Moreover, TAH has been tested on a set of randomly generated projects, against the SAH and MRDH. TAH and MRDH are practically equivalent and both better than SAH regarding the obtained solutions; TAH proved to be the less time consuming algorithm and the most reliable procedure in the sense that it has improved the most instances.

The proposed hyperheuristic shows the flexibility of the hyperheuristic framework, i.e., the development of a hyperheuristic is independent of the utilized low-level heuristics, and thus it is easy-to add or remove low-level heuristics without any concern of how the hyperheuristic operates. This flexibility facilitates the reapplicability of developed procedures, as this study has made apparent, and makes easy the development of decision support systems for project scheduling.

Although both the presented case study and the experimental analysis on randomly generated activity networks demonstrate that the algorithm reaches satisfactory results, modifications to the proposed approach could improve its performance. For example, a more efficient learning mechanism, sophisticated low-level heuristics such as crossover operators, and a software-independent procedure to generate feasible schedules more efficiently could improve both the runtime and the quality of obtained solutions. In any case this study shows that hyperheuristics is of a great research interest for handling project scheduling problems.

References

Ahuja, H. N. (1976). *Construction Performance Control by Networks*, Wiley, NY.

Anagnostopoulos, K. P., and Koulinas, G. K. (2010). "A simulated annealing hyperheuristic for construction resource levelling." *Constr. Manage. Econ*, 28(2), 163–175.

Anagnostopoulos, K., and Koulinas, G. (2011). "Resource-constrained critical path scheduling by a GRASP based hyperheuristic." *J. Comput. Civ. Eng.*, xxx(none), xxx–xxx.10.1061/(ASCE)CP.1943-5487.0000116 (Mar. 12, 2011).

Bandelloni, M., Tucci, M., and Rinaldi, R. (1994). "Optimal resource leveling using non-serial dynamic programming." *Eur. J. Oper. Res.*, 78(2), 162–177.

Boctor, F. F. (1990). "Some efficient multi-heuristic procedures for resource-constrained project scheduling." *Eur. J. Oper. Res.*, 49(1), 3–13.

Boctor, F. F. (1996). "Resource-constrained project scheduling by simulated annealing." *Int. J. Prod. Res.*, 34(8), 2335–2351.

Bouleimen, K., and Lecocq, H. (2003). "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version." *Eur. J. Oper. Res.*, 149(2), 268–281.

Brucker, P., Knust, S., Schoo, A., and Thiele, O. (1998). "A branch-and-bound algorithm for the resource constrained project scheduling problem." *Eur. J. Oper. Res.*, 107(2), 272–288.

Burgess, A. R., and Killebrew, J. B. (1962). "Variation in activity level on a cyclic arrow diagram." *Ind. Eng.*, 13(2), 76–83.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003a). "Hyperheuristics: An emerging direction in modern search technology." *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger, eds., Kluwer Academic Publishers, Dordrecht, 457–474.

Burke, E., Kendall, G., and Soubeiga, E. (2003b). "A tabu search hyperheuristic for timetabling and rostering." *J. Heuristics*, 9(6), 451–470.

Burke, E. K., Kendall, G., Landa-Silva, D., O'Brien, R., and Soubeiga, E. (2005). "An ant algorithm hyperheuristic for the project presentation scheduling problem." *Proc., 2005 IEEE Congr. on Evol. Comput. (CEC 2005)*, IEEE Computer Society Press, Edinburgh, Scotland, 2263–2270.

Chakhlevitch, K., and Cowling, P. (2008). "Hyperheuristics: Recent developments." *Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence*, C. Cotta, M. Sevaux and K. Sorensen, eds., vol. 136, Springer, Berlin, 3–29.

Chan, W., Chua, D. K. H., and Kannan, G. (1996). "Construction resource scheduling with genetic algorithms." *J. Constr. Eng. Manage.*, 122(2), 125–132.

Christodoulou, S. (2010). "Scheduling resource-constrained projects with ant colony optimization artificial agents." *J. Comput. Civ. Eng.*, 24(1), 45–55.

Cowling, P., Kendall, G., and Soubeiga, E. (2001). "A hyperheuristic approach to scheduling a sales summit." *Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science*, E. Burke and W. Erben, eds., vol. 2079, Springer, Heidelberg, 176–190.

Demeulemeester, E., and Herroelen, W. (1992). "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem." *Manage. Sci.*, 38(12), 1803–1818.

Demeulemeester, E. L., and Herroelen, W. S. (2002). *Project Scheduling—A Research Handbook*, Kluwer Academic Publishers, Boston.

Dorndorf, U., and Pesch, E. (1995). "Evolution based learning in a job shop scheduling environment." *Comput. Oper. Res.*, 22(1), 25–40.

Dowsland, K. A., Soubeiga, E., and Burke, E. (2007). "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation." *Eur. J. Oper. Res.*, 179(3), 759–774.

Dueck, G. (1990). "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing." *J. Comput. Phys.*, 90(1), 161–175.

Easa, S. M. (1989). "Resource leveling in construction by optimization." *J. Constr. Eng. Manage.*, 115(2), 302–316.

Fisher, H., and Thompson, G. L. (1963). "Probabilistic learning combinations of local job shop scheduling rules." *Industrial Scheduling*, J. F. Muth and G. L. Thompson, eds, Prentice Hall, Englewood Cliffs, 225–251.

Gavish, B., and Pirkul, H. (1991). "Algorithms for multi-resource generalized assignment problem." *Manage. Sci. J.*, 37(6), 695–713.

Han, L., and Kendall, G. (2003). "An investigation of a tabu assisted hyperheuristic genetic algorithm." *Proc., 2003 IEEE Congr. on Evol. Comput. (CEC 2003)*, IEEE Computer Society Press, Canberra, Australia, 2230–2237.

J. Constr. Eng. Manage. 2012.138:854-863.

Harris, R. B. (1990). "Packing method for resource leveling (PACK)." *J. Constr. Eng. Manage.*, 116(2), 331–350.
Hartmann, S. (1998). "A competitive genetic algorithm for resource-constrained project scheduling." *Nav. Res. Logist. Q.*, 45(7), 733–750.
Hartmann, S., and Briskorn, D. (2010). "A survey of variants and extensions of the resource constrained project scheduling problem." *Eur. J. Oper. Res.*, 207(1), 1–14.
Hegazy, T. (1999). "Optimization of resource allocation and leveling using genetic algorithms." *J. Constr. Eng. Manage.*, 125(3), 167–175.
Hiyassat, M. A. S. (2000). "Modification of minimum moment approach in resource leveling." *J. Constr. Eng. Manage.*, 126(4), 278–284.
Kolisch, R. (1996). "Efficient priority rules for the resource-constrained project scheduling problem." *J. Oper. Manage.*, 14(3), 179–192.
Leu, S. S., Yang, C. H., and Huang, J. C. (2000). "Resource leveling in construction by genetic algorithm-based optimization and its decision support system application." *Autom. Constr.*, 10(1), 27–41.
Lova, A., Tormos, P., Cervantes, M., and Barber, F. (2009). "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes." *Int. J. Prod. Econ.*, 117(2), 302–316.
Montgomery, D. C. (2001). *Design and Analysis of Experiments*, 5th Ed., Wiley, NY.
Neumann, K., and Zimmermann, J. (1999). "Resource leveling for projects with schedule-dependent time windows." *Eur. J. Oper. Res.*, 117(3), 591–605.
Neumann, K., and Zimmermann, J. (2000). "Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints." *Eur. J. Oper. Res.*, 127(2), 425–443.
Nonobe, K., and Ibaraki, T. (2001). "Formulation and tabu search algorithm for the resource constrained project scheduling problem." *Essays and Surveys in Metaheuristics*, C. C. Ribeiro and P. Hansen, eds., Kluwer Academic Publishers, Dordrecht, 557–588.
Ouelhadj, D., and Petrovic, S. (2010). "A cooperative hyperheuristic search framework." *J. Heuristics*, 16(6), 835–857.
Pan, N. H., Hsaio, P. W., and Chen, K. Y. (2008). "A study of project scheduling optimization using Tabu Search algorithm." *Eng. Appl. Artif. Intell.*, 21(7), 1101–1112.
Qu, R., and Burke, E. K. (2009). "Hybridizations within a graph-based hyperheuristic framework for univ. timetabling problems." *J. Oper. Res. Soc.*, 60(9), 1273–1285.
Savin, D., Alkass, S., and Fazio, P. (1996). "Construction resource leveling using neural networks." *Can. J. Civ. Eng.*, 23(4), 917–925.
Senouci, A. B., and Eldin, N. N. (2004). "Use of genetic algorithms in resource scheduling of construction projects." *J. Constr. Eng. Manage.*, 130(6), 869–877.
Storer, R. H., Wu, S. D., and Vaccari, R. (1995). "Problem and heuristic search space strategies for job shop scheduling." *ORSA J. Comput.*, 7(4), 453–467.
Weglarz, J., Jozefowska, J., Mika, M., and Waligora, G. (2011). "Project scheduling with finite or infinite number of activity processing modes—A survey." *Eur. J. Oper. Res.*, 208(3), 177–205.
Winker, P., and Maringer, D. (2007). "The threshold accepting optimization algorithm in economics and statistics." *Optimization, Econometric and Financial Analysis*, E. J. Kontoghiorghes and C. Gatu, eds., Springer, Berlin, 107–125.
Zhang, H., Li, H., and Tam, C. M. (2006a). "Permutation-based particle swarm optimization for resource-constrained project scheduling." *J. Comput. Civ. Eng.*, 20(2), 141–149.
Zhang, H., Li, H., and Tam, C. M. (2006b). "Particle swarm optimization for preemptive scheduling under break and resource-constraints." *J. Constr. Eng. Manage.*, 132(3), 259–267.

J. Constr. Eng. Manage. 2012.138:854-863.