World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:10, 2008

Solving an Extended Resource Leveling Problem with Multiobjective Evolutionary Algorithms

Javier Roca, Etienne Pugnaghi, and Gaëtan Libert

***Abstract*—**We introduce an extended resource leveling model that abstracts real life projects that consider specific work ranges for each resource. Contrary to traditional resource leveling problems this model considers scarce resources and multiple objectives: the minimization of the project makespan and the leveling of each resource usage over time. We formulate this model as a multiobjective optimization problem and we propose a multiobjective genetic algorithm-based solver to optimize it. This solver consists in a two-stage process: a main stage where we obtain non-dominated solutions for all the objectives, and a postprocessing stage where we seek to specifically improve the resource leveling of these solutions. We propose an intelligent encoding for the solver that allows including domain specific knowledge in the solving mechanism. The chosen encoding proves to be effective to solve leveling problems with scarce resources and multiple objectives. The outcome of the proposed solvers represent optimized trade-offs (alternatives) that can be later evaluated by a decision maker, this multi-solution approach represents an advantage over the traditional single solution approach. We compare the proposed solver with state-of-art resource leveling methods and we report competitive and performing results.

***Keywords*—**Intelligent problem encoding, multiobjective decision making, evolutionary computing, RCPSP, resource leveling.

I. Introduction

PROJECT Scheduling is a mechanism which translates the performance imperatives of a project plan into a sequence of activities to be executed in order to deliver the optimized performance indicator predicted by the higher-level plan. More precisely, project scheduling deals with the exact allocation of resources (e.g. people, machines, raw materials, etc.) to activities over time, i.e., finding a resource that will process the activity and finding the time of processing. The obtained schedule must respect the precedence, duration, capacity and incompatibility constraints. One common desirable property on a schedule is the minimization of the project duration (makespan). Other desirable property is the balanced use of the resources over time. Resource leveling is a process that permits to minimize the variation of the resource usage over time, and thus, reduce extraordinary demands or excessive fluctuations in the usage of resources, which may be costly in certain contexts. Project scheduling and leveling are among the top challenges in project management.

Many previous research works have analyzed independently the makespan minimization of a project and the minimization of the variability on the resource usage. This work introduces an extended resource leveling problem which considers a resource constrained project and seeks, simultaneously, the minimization of the project's makespan and the minimization of the variability of each resource usage. We model this scheduling problem as a multiobjective optimization problem and we propose to solve it in a two-stage process where we apply multiobjective evolutionary algorithms as the underlying solving mechanism.

This document is organized as follows: in section II we present a review of the resource constrained project scheduling problem (minimization of makespan), in section III we present a review of the resource leveling problem (resource smoothing), in section IV we introduce the extended resource leveling problem, in section IV we describe a solver for this model, in part V we present the results of the experiments with the proposed solver, and finally, in part VI we present our conclusions.

J. Roca is with Intelligent Software Company, Arquennes 7181 Belgium, and is a doctoral candidate at the Faculté Polytechnique de Mons, Mons 7000 Belgium (e-mail: javier.roca@ fpms.ac.be).

E. Pugnaghi is with Intelligent Software Company, Arquennes 7181 Belgium (e-mail: etienne.pugnaghi@planningforce.com).

G. Libert is with Computer Science Department, Faculté Polytechnique de Mons, Mons 7000 Belgium (e-mail: gaetan.libert@fpms.ac.be).

II. The Resource-Constrained Project Scheduling Problem

A. The RCPSP Model

Informally, the Resource-Constrained Project Scheduling Problem (RCPSP) seeks the answer to the following question: "*Given the limited availability of resources, what is the best way to schedule the activities in order to complete the project in the shortest possible time?*" Among the practical applications of this model we can mention the construction of buildings and the production planning. The RCPSP addresses the operational, short term, scheduling. The RCPSP may be formulated as follows: a project consists of a set of activities $A = \{1,...,n\}$ where each activity has one mode of execution and has to be processed without interruption (i.e. no preemption allowed). There exist two dummy activities 1 and n which represents the root (start) and the sink (end) of the projects respectively. The duration of an activity j is denoted by d_j, the root and the sink have a duration of d_1=0 and d_n=0 respectively. There are precedence relations between the activities in A, which are given by sets of immediate predecessors *Pred*$_j$ indicating that and activity *j* may not be started before all of its predecessors are completed (i.e. finish-to-start with zero-lag). Analogously, *Succ*$_j$ is the set of immediate successors of activity *j*. The precedence relations can be represented by an activity-on-node network which is assumed to be acyclic. There are *K* renewable resource types; each resource *k* has a constant per-period-availability R_k. With the exception of the root and sink activities, each activity *j* requires r_{jk} units of resource *k* during each period of its duration. All parameters are assumed to be non-negative integer valued.

The objective of the RCPSP is to find a schedule *S* of the activities on A, i.e. a set of starting times $\{s_1,....,s_n\}$ where s_1=0, the precedence and resource constraints are satisfied (*feasible schedule*), and the schedule duration (makespan) T(*S*)=s_n is minimized. The RCPSP has been widely studied over the past few decades. For a comprehensive survey on the RCPSP the reader is referred to the survey made in [1]. Fig. 1 illustrates a RCPSP instance represented as an *activity-on-node* network.

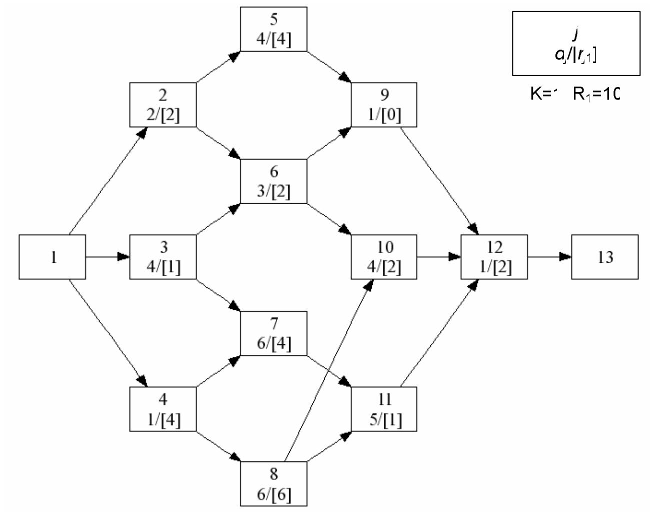


Fig. 1 A RCPSP Instance with 13 activities and 1 resource (adapted from [10])

B. Solving the RCPSP

As the RCPSP is a generalization of the *job shop* problem, it is a *NP-hard* problem in the strong sense [2]. Therefore, heuristic and metaheuristic solution procedures are indispensable when solving large problem instances as they usually appear in practical cases [3]. In order to apply an optimization algorithm to a problem, at first a suitable representation of solutions has to be chosen. For this reason, schedules are often represented by sequences of activities. From these sequences feasible starting times are derived by appropriate decoding procedures (so-called *schedule generation schemes*).

1) Schedule Generation Schemes

The schedule generation schemes (SGS) are the core of most of the heuristic and metaheuristic solution procedures for the RCPSP [3]. In a SGS the activities are iteratively scheduled (i.e. s_j is defined for each activity) and in each iteration an eligible activity is chosen according to some selection mechanism. The most common selection mechanisms are the ones proposed by *priority rules, activity lists* and *random key*s (reviewed in the following sections). The main types of SGS are the *Serial* SGS and the *Parallel* SGS, illustrated in Fig. 2 (a) and (b), respectively. Computational experiments of various authors have shown that for some instances the serial SGS produces better schedules, for other instances the parallel SGS is more suitable [4].

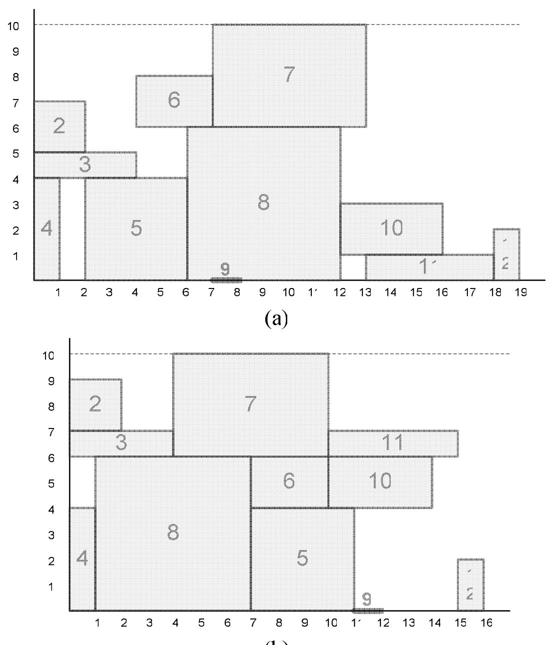


Fig. 2 Feasible schedules for the project in Fig. 1 generated with different SGS and the LPT rule

a) Serial SGS

The serial schedule generation scheme (S-SGS) is an *activity oriented* SGS where a schedule is generated in *n* stages. With each stage λ∈ {1,…, *n*} two disjoint activity sets are associated: the set of scheduled activities and the set E_λ of all eligible activities (i.e. all activities for which all predecessors are already scheduled). In each stage one eligible activity *j* ∈ E_λ is chosen (e.g. by a priority rule) and scheduled at its *earliest precedence-* and *resource-feasible time*. Afterwards, the resource profiles of the partial schedule and the set of eligible activities are updated. As the S-SGS schedules activities with the earliest start procedure (respecting the precedence and resource constraints) then in the obtained schedules no activity can be (locally or globally) shifted to the left (*active schedules*) [4].

b) Parallel SGS

The parallel SGS (P-SGS), is a *time oriented* SGS. In each step of the scheduling process a set of activities (which might be empty) from the eligible activity decision set is scheduled.

With each stage λ a time point t_λ and three disjoint activity sets are associated: the set of finished activities, the set A_λ of all active activities (i.e. activities which are already scheduled in the partial schedule, but finish after time t_λ), and the set E_λ of all eligible activities (i.e. all unscheduled activities *j* for which all predecessors are completed up to time t_λ and for which sufficient resources are available when *j* is started at time t_λ). In each stage a maximal resource-feasible subset of eligible activities in E_λ is chosen and scheduled at time t_λ. Afterwards, the resource profiles of the partial schedule and the sets of active and eligible activities are updated. The next decision point $t_{\lambda+1}$ is given by the minimum of the next value t_k^u where a resource profile changes and the minimal completion time of all active activities [4]

c) Backward Scheduling

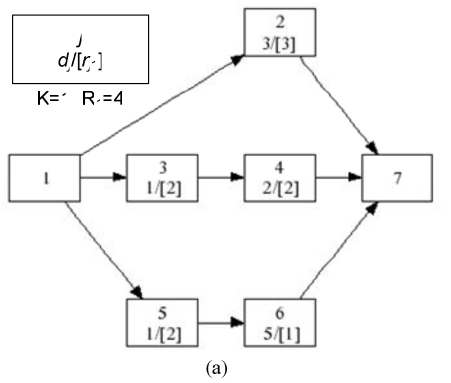
An important variant of the previously described SGSs are their *backward* counterparts, namely the backward S-SGS (BS-SGS) and the backward P-SGS (BP-SGS). While in the previously described schemes, the schedules are generated from *root-to-sink* (*forward scheduling*), in the corresponding backward schemes the schedules are constructed in the reverse direction, from *sink-to-root* (*backward scheduling*), while respecting the precedence and resource constraints. After obtaining a *backward schedule* it is globally shifted to the left in order to obtain an active schedule. For some instances the BP-SGS and the PS-SGS can produce better schedules than the forward counterparts, as it is illustrated in Fig. 3 where the first schedule is obtained with a S-SGS and second with a BS-SGS.

2) Priority Rule, Activity List, and Random Keys

A *priority rule* is defined as a mapping which assigns each activity *j* in the set of eligible activites E_λ a value *v*(*j*) and an objective stating whether the activity with the minimum or the maximum value is selected [3]. Thus, the priority rule is used to select an activity *j* from within a set of eligible activities based on the value of *v*(*j*). Typical examples are the longest processing time (LPT) and the shortest processing time (SPT) rules. Scheduling schemes and priority rules are usually combined in order to obtain different priority rule based heuristics [3].

In an activity list representation the solution is encoded as a *precedence-feasible* list of activities AL $=\{A_1,A_2,..A_n\}$. Each activity can appear in the list in any position after all its predecessors. The activity list can be included as the selection mechanism in a SGS, thus, an activity *j* is selected respecting the relative order of the activity list. An activity list could be easily converted as a priority rule, thus *v*(*j*)= index of *j* in AL (the activity chosen depends on the position of the activity in the list AL).

The random keys representation is similar to the activity list except that each element in the list RK is a random value that defines the priority of an activity RK=$\{rk_1, rk_2,.. rk_n\}$ where rk_1=0 and rk_n=0. Analogously, the random key representation can be included as the selection mechanism in a SGS, thus *v*(*j*) = rk_j. Usually activities 1 and *n* are not included in this representation as they are dummy activities, as illustrated in Fig. 3.



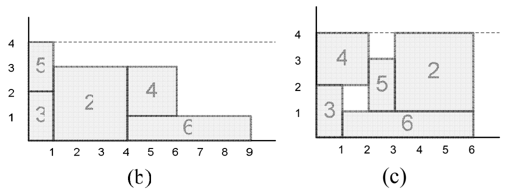


Fig. 3 (a) A RCPSP instance (adapted from [5]) solved with a SGS and RK={0.6,0.9,0.1,0.8,0.3}

III. The Resource Leveling Problem

A. The RLP Model

The resource leveling problem (RLP), also known as *resource smoothing,* is a special case of a project scheduling problem where the objective is to reduce extraordinary demands or excessive fluctuations in the usage of resources (e.g. use the required resources as even as possible over time), which may be *costly* in certain contexts.

The general RLP may be formulated as follows: Let $c_k \geq 0$ be a cost for resource *k* and denote by $r_k^S(t)$ the *resource usage* of resource *k* in period $t \in \{1,\ldots,T\}$ for a given schedule *S*, where $r_k^S(0)$=0 and the resources are assumed to be *unlimited*. The objective of the RLP is to minimize some *measure of variability* (MV) evaluated over the resource usage.

In the so-called *deviation problems* the deviations (overloads) of the resource usages from a given resource profile are minimized. Typical examples of measures of variability for these problems are the *resource leveling index* (RLI), defined in (1) and the *squared deviation* (SD), defined in (2).

$$\sum_{k=1}^{r} c_k \sum_{t=1}^{T} \left| r_k^S(t) - Y_k \right| \tag{1}$$

$$\sum_{k=1}^{r} c_k \sum_{t=1}^{T} \left(r_k^S(t) - Y_k \right)^2 \tag{2}$$

Where Y_k is a target value that may be replaced by the average resource usage defined as

$$Y_k = \frac{1}{T}\sum_{i=1}^{T} r_k^S(i)\,. \tag{3}$$

On the other hand, in the so-called *variation problems*, the resource usages should not vary much over time. Analogously to the deviation problems, this can be achieved by minimizing the measures of variability defined in (4) and (5).

$$\sum_{k=1}^{r} c_k \sum_{t=1}^{T} \left| r_k^S(t) - {}_k^S(t-1) \right| \tag{4}$$

$$\sum_{k=1}^{r} c_k \sum_{t=1}^{T} \left(r_k^S(t) - {}_k^S(t-1) \right)^2 \tag{5}$$

In the present work we will refer to the resource usage of a resource as the *resource usage profile* (RUP), defined as

RUP(r_k,S)={u_1,u_2,..u_T} (6)

where $u_i = r_k^S(i)$.

As an example, consider the RUP for the resource usage in Fig. 4 which is RUP(R1,S)={0,0,6,5,7,0,9,6,0,0}.

B. The RL-RCPSP Model

The resource leveling RCPSP (RL-RCPSP) is a special case of RLP where the activities and resources have the same properties as the RCPSP (i.e. one mode of execution, no preemption, renewable and *scarce* resources). The RL-RCPSP considers the minimization of some measure of variability *without* considering the makespan minimization.

C. Solving the RLP

Between the solving methods for the RLP reported in the literature with special objective functions we can mention:

Exact algorithms based upon enumeration, integer programming, or dynamic programming have been proposed by, for example in [6] and [7]. Heuristic methods based on priority-rules have been devised, for example in [8], [9], [10], and [11]. Only small problem instances with up to 20 activities have been solved (approximately) by those methods [12]. However, it has been remarked by Leu *et al* [13] that due to the variety of network structures and resources, no single heuristic method can always produce the best solution for all the resource leveling problems.

Different metaheuristic methods as the ones proposed by [13] , [14], [15] and [16].

IV. The Extended Resource Leveling Problem

A. The ERLP Model

Based on our observations of practical and real-world scenarios on the workforce and project management domains, we found a resource leveling problem for which we were not able to find a suitable existent formal model. We will refer to this model as the *Extended Resource Leveling Problem* (ERLP). In the ERLP each resource has an associated *work range* and the objectives are multiple (i.e. the *simultaneous* minimization of the project's makespan *and* the smoothing of the usage of each resource). The ERLP considers the resources as scarce. The ERLP differs from the RL-RCPSP on the optimization function (the RL-RCPSP only considers makespan) and differs from traditional RLP as it consider strictly scarce resources and multiple objectives. A work range is defined as the time interval that must considered for calculating the resource usage profile of a specific resource. It is important to remark that the work range refers to a time range evaluated over a schedule and *does not* represent a time window constraint. In the ERLP the work range can be Full, Dynamic, or Effective:

1) The *full range* refers to a time interval that considers the entire duration of a project, hence, the resource usage profile of a resource is calculated over the interval [*Start of Project, End of Project*]. This is the traditional range considered by the RLP.
2) The *dynamic range* considers the time interval starting on the first resource usage and ending on the last resource usage. The resource usage profile for this range is calculated over the interval [*Start time of first activity, End time of last activity*].
3) The *effective range* considers a set of possibly discontinuous time intervals where the resource has been used (i.e. resource usage is greater than 0). The resource usage profile for this range is calculated on this set of intervals.

Fig. 4 illustrates the work range types and their corresponding time interval(s). The main interest on considering the work ranges is based on the fact that the MV can vary sharply depending on the work range used for its calculation, mainly due to the bias induced by *idle times* (i.e. where the resource is not being used) not considered in some of the work ranges (that could affect the average of the resource usage) and, logically, due to the length of the work range. Table I illustrates the impact of considering different work ranges with RLI calculated over the profile in Fig 4.

The choice of the work range to associate to each resource is a decision to be taken by the project manager, for example, depending on the type of each resource's predefined work basis.

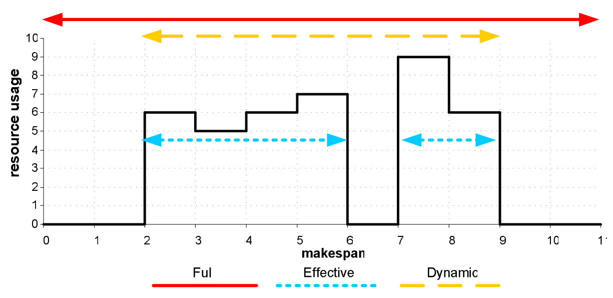


Fig. 4 Example of the resource usage profile of one resource and the possible work ranges

TABLE I
IMPACT OF DIFFERENT WORK RANGES ON THE RLI

Work Range	Interval	Y_k	RLI
Full	[0,11]	3.25	35.75
Dynamic	[2,9]	5.57	12.28
Effective	{[2,6],[7,9]}	6.50	6.00

The ERLP can be formulated as a RCPSP with the particularity that each resource *k* has an associated work range $wr_k \in WR=\{full, dynamic, effective\}$. The objectives of the ERLP are:

1) To *minimize the makespan* of the project, and
2) To *minimize some measure of variability* MV on each resource *k* that is evaluated over the resource usage profile (RUP defined in (6)) of *k* considering the associated wr_k as follows:

$$RUP(k,S,wr)=\begin{cases} RUP(k,S) & for\ wr=full \\ RUPD(k,S) & for\ wr=dynamic \\ RUPE(k,S) & for\ wr=effective \end{cases} \quad (7)$$

where

$$RUPD(r_k,S)=\{\ r_k^S(i):\ fu(r_k,S)\le i\le lu(r_k,S)\ \}, \quad (8)$$

fu(r_k*,S)* is the first period where r_k was used in *S*,
lu(r_k*,S)* is the last period where r_k was used in *S*, and

$$RUPE(r_k,S)=\{\ r_k^S(i):\ r_k^S(i)>0\} \quad (9)$$

thus, there exist *k* objectives to minimize defined as
MV(RUP(k,S,wr$_k$)) (10)

Given a measure of variability MV, the ERLP can be formulated as a *multiobjective problem* (MOP) to optimize *k*+1 objective functions simultaneously, where the decision variables set is defined as

$$x=[S]^T \quad (11)$$

, the vector function is defined as

$$f(x)=[makespan(S),MV(RUP(1,S,wr_1),\cdots,MV(RUP(k,S,wr_k)]^T, \quad (12)$$

and the objective is

$$\min(f(x)). \quad (13)$$

Fig. 5 illustrates an ERLP instance with 8 activities, 3 resources and with WR$_k$={full, dynamic, effective}. Fig. 6 shows a possible schedule for this instance. Table II details the work ranges and the RLI calculations for the obtained schedule.

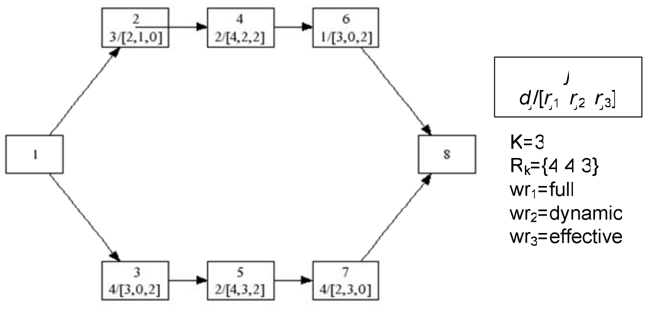


Fig. 5 An ERLP instance with 8 activities and 3 resources

B. Solving the ERLP

Based on the structure of the ERLP it may be classified as a discrete multiobjective combinatorial problem (e.g. find a feasible sequence of activities that optimize the model objectives). In order to solve the ERLP we consider a constructive metaheuristic to *build* and *optimize* the schedule simultaneously. The mechanics of this metaheuristic allows optimizing the MOP formulation of the ERLP.

Our proposed solver is based on a multiobjective evolutionary algorithm which incorporates specific ERLP knowledge in its encoding. In the following paragraphs we detail the basic elements of our metaheuristic approach. We propose to solve the ERLP in two stages:

1) A main (coarse-grained optimization) stage where we use a metaheuristic solver (i.e. schedule generator) based on a multiobjective evolutionary algorithm considering an intelligent encoding to obtain a population of optimized schedules,
2) An *postprocessing* stage (fine-grained optimization) which consists in the leveling improvement of the obtained population.

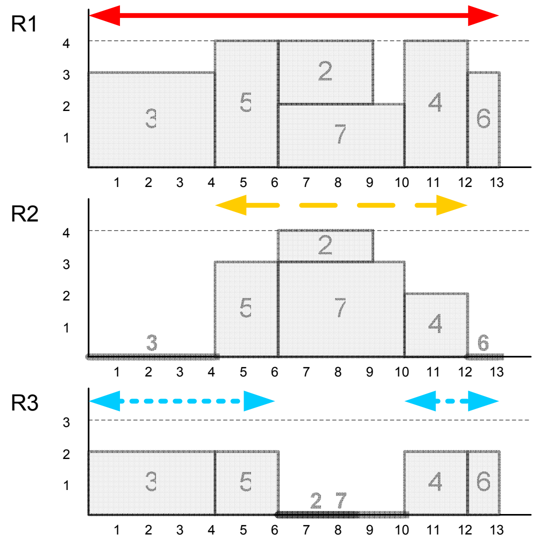


Fig. 6 An example schedule for the ERLP problem in Fig. 5

TABLE II
WORK RANGES DETAILS

Resource	Work Range	Interval	Y_k	RLI
R$_1$	Full	[0,13]	3.46	7.54
R$_2$	Dynamic	[4,12]	3.63	9
R$_3$	Effective	{[0,6],[10,13]}	2	0

1) Pareto Optimization and Evolutionary Algorithms

In multiobjective problems often some of the criteria are in conflict, i.e. an improvement in one of them can only be achieved at the expense of worsening another. Moreover, some of the criteria may be incommensurable. The incommensurability of criteria adds to the difficulty of the problem because the aggregation or comparison of different objectives is not straightforward (this is the case of the makespan and the resource leveling objectives in the ERLP as they are incommensurable). Moreover, in most multiobjective optimization problems there is no single-best solution or global optima and it is very difficult to establish preferences among the criteria before the search process is carried out. One way to overcome potential conflicts between –possibly incommensurable- objectives and uncertainty in the criteria preferences is the application of Pareto optimization techniques. The *Pareto optimization* is based on the *dominance* relation, described as follows [17]: Suppose we have two distinct vectors $V=(v_1,v_2,\dots,v_k)$ and $U=(u_1,u_2,\dots u_k)$ containing the objective values of two solutions for a k-objective minimization problem, then:

- V *strictly dominates* U if $v_i < u_i$, for $i = 1,2,..,k$.
- V *loosely dominates* U if $v_i \leq u_i$, for $i = 1,2,..,k$ and $v_i < u_i$, for at least one *i*.
- V and U are incomparable if neither V (strictly or loosely) dominates U nor U (strictly or loosely) dominates V.

A solution *x* is said to be non-dominated with respect to a set of solutions *S* if there is no other solution in *S* that dominates *x*. The Pareto-optimal front is the set of all non-dominated solutions in the whole solution space [18]. When there is no knowledge of the localization of the Pareto-optimal set, the set found should be referred to as the obtained non-dominated set or the Pareto front approximation. In the ERLP, the final outcome is to find the Pareto front approximation considering the objectives indicated in (12). This Pareto front should provide the decision maker (DM) with near to optimal trade-offs between project duration and resource leveling.

An evolutionary algorithm (EA) is a metaheuristic that uses some mechanisms inspired by biological evolution, as reproduction, mutation, recombination, and selection. EA seem particularly suitable to solve MOP because they can deal simultaneously with a set of possible solutions (i.e. the population). The *multiobjective evolutionary algorithms* (MOEA) are EAs that are specifically structured to solve MOP problems. The outcome of a MOEA is considered to be a set of mutually *non-dominated* solutions (the Pareto front approximation). The most well-known algorithms for multi objective optimization are based on *Genetic Algorithms* (usually referred as *Multiobjective Genetic Algorithms* or MOGA). We consider a MOGA as our solver mechanism due to their ability to handle multiobjective combinatorial problems.

C. Evolutionary Solver

1) First Stage

We propose to use a MOGA as our main solving mechanism (first stage). For our tests we consider the NSGA-II (*Non-Dominated Sorting Genetic Algorithm II)* as it is frequently reported in the literature. NSGA-II is a well-established MOGA using an *elitist* approach. Its fitness assignment scheme consists in sorting the population in different fronts using the non-domination order relation. To form subsequent generations, the algorithm combines the current population and its offspring generated with the standard bimodal crossover and polynomial operators. Finally, the best individuals in terms of non-dominance and diversity are chosen. For a complete description of this algorithm the reader is referred to [19].

a) Chromosome Encoding

In order to apply the MOGA approach we need to define a suitable chromosome encoding (problem representation). One of the most competitive evolutionary based algorithms for the RCPSP (i.e. minimization of the makespan) is the *intelligent encoding* (IE) introduced in [20]. The IE concludes that one of the key factors on the effectiveness of its approach is the *incorporation of specific knowledge* of the problem on the solving mechanism. In the IE the authors propose a chromosome encoding that employs the S-SGS and the P-SGS, and the combination of forward and backward scheduling. The joint use of these characteristics results in an intelligent encoding which exploits the problem specific knowledge in an efficient way (generation of the schedule). As one of the objectives of the ERLP is to minimize the makespan we consider a similar approach for our chromosome encoding.

S/P	**B/F**	**RK_2**	**...**	**RK_{n-1}**	**SL_1**	**...**	**SL_k**

Fig. 7 Intelligent Chromosome Encoding for the ERLP

We propose to use the encoding structure of Fig. 7 which is based on the IE encoding. In this encoding, we include specific knowledge of the *solving mechanism* as we consider the S/P and B/F genes along with the random key RK_j that permit to build a feasible schedule (i.e. no need to repair the obtained schedule). The RK_j genes represent the random key values for each activity *j*, the random keys representation is chosen due to its flexibility and easiness to be adapted to the standard genetic operators, as described afterwards.

As we use a SGS (either serial or parallel), this implies that the activities will be scheduled by considering their *earliest start time* (EST) in a *resource-feasible* period. Regarding the resource leveling optimization, the consequences of using a SGS are twofold:

1) The SGS will allocate the resources up to the maximum resource availability and will try to use resources as *much as possible.* Due to this allocation policy the SGS cannot guarantee a balanced resource allocation as it could create usage peaks and valleys in different scheduled activities, as illustrated in Fig. 8 (a) and (b) where the schedule with a best resource usage balance (Fig. 8 (c)) cannot be obtained with the original maximum availability R_1=5 by using a S-SGS.
2) The EST policy limits the exploration of the problem search space as the SGS *does not* consider starting times other than the EST (e.g. it does not consider a possible slack range). In consequence, a SGS cannot guarantee an optimal solution for the resource leveling. The main reason to not use the slack range is due that the traditional slack-based techniques (e.g. as the *critical path method* (CPM), where the resources are assumed as unlimited) cannot be easily adapted and applied on scarce resources (as the slack range cannot be calculated *before* building the schedule). Fig. 9 illustrates this limitation, notice that the second schedule on this figure cannot be generated by a SGS

(as the schedule in Fig.9 (b) has a lower RLI than Fig .9 (a)).

To overcome the first limitation we propose to include the genes SL_i in our encoding, which indicate the new maximum availability for each resource to be considered by the SGS (represented as a percentage that will be evaluated over the original maximum level). In this way the SGS can explore different alternatives to build schedules with a better resource usage balance. One important remark is that a reduction on the resource availability could, potentially, increase the makespan. This represents a trade-off between makespan and resource usage smoothing that can be naturally handled by the underlying MOGA. In Fig. 8 (c) we remark that by reducing R_1=5 to R_1=3 or R_1=4 an optimal resource leveling can be achieved with an S-SGS.

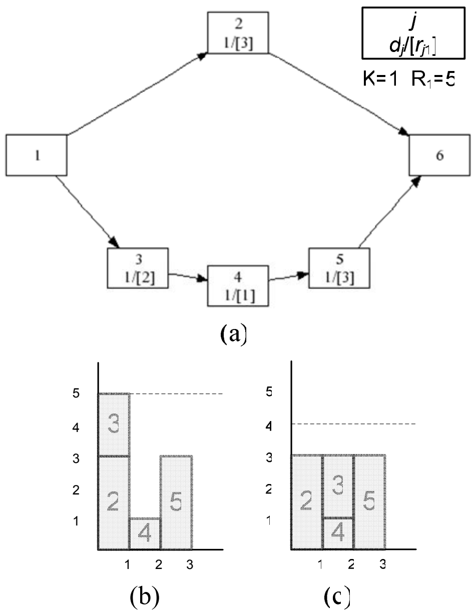


Fig. 8 Example of reducing the maximum availability of a resource

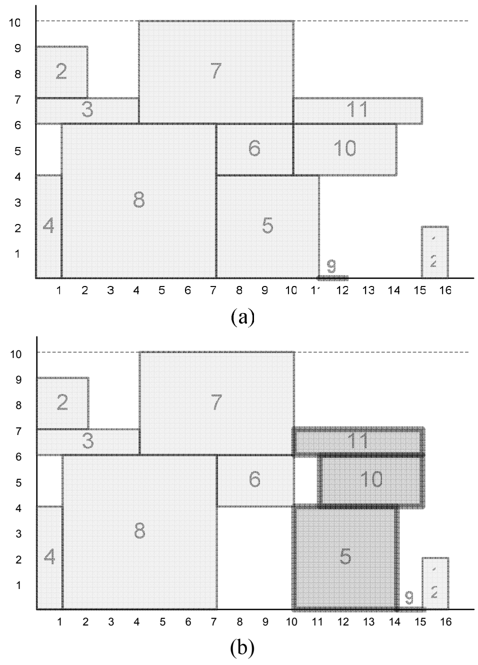


Fig. 9 Example of shifting activities on a schedule

The second limitation is more complicated to solve. One logical approach is to modify the SGS to schedule activities within the EST and the *latest start time* (LST) of an activity. However, as the resources are strictly *constrained*, the LST cannot be calculated before the schedule is actually built. Due to this limitation we propose a *leveling improvement* mechanism based on activity *shifting* that will be applied *after* obtaining a schedule (*post-processing*). This mechanism is described afterwards. Consequently, our solving mechanism will consider two stages:

1) The generation of a non-dominated population (coarse-grained), and
2) The leveling improvement of the obtained population without decreasing the population quality (fine-grained).

Fig. 10 shows an example chromosome for the proposed encoding. This chromosome will decode as a S-SGS (forward scheduling), with RK= {0.5, 0.4, 0.3, 0.7}, and will consider R_1=4 (80% of original R_1=5). The obtained schedule is the same as the one in Fig. 8(c).

S	**F**	**0.5**	**0.4**	**0.3**	**0.7**	**80%**

Fig. 10 Chromosome encoding example

b) Genetic operators

We consider the following genetic operators for the proposed MOGA:

1) *Crossover*: Given two chromosomes we obtain two new chromosomes (offspring) by using a *standard two-point crossover*. As we use random keys we don't need a structured crossover operator as the one proposed by the IE.

2) *Mutation*: If a gene S/P or B/F is selected for mutation, then the value is switched (i.e. from Serial(Parallel) to Parallel(Serial) for gene S/P and from Backward(Forward) to Forward(Backward) for gene B/F). If a gene RK$_j$ is selected for mutation then standard *swap mutation* is applied with other RK$_i$ gene. In the case of the mutation of genes SL$_k$ a new random value is assigned under the consideration of respecting the maximal resource requirement of resource *k*.
3) *Selection*: standard *binary tournament selection.*
4) *Initial Population:* Each gene value is initialized with a standard random generator. Alternatively, a *bias random sampling method* could be applied (e.g. the β*-BRSM* proposed in [21]).

The fitness function of this MOGA considers the objectives defined in (12).

2) Second Stage

As we indicated in the previous section, the use of a SGS has two main drawbacks regarding the resource leveling objectives that cannot guarantee to find an optimal resource balancing. The first limitation was solved by including SL$_k$ genes in the encoding. In this section we propose a method to overcome the second limitation and *improve* the solutions obtained with a SGS *w.r.t.* the resource leveling objectives. Leu *et. al.* [13] proposes a method based on *activity shiftings* to overcome the EST limitation by letting the algorithm to choose the start time of the activities by considering the range [EST,LST]. However, this method is based on CPM slack calculations which are not compatible with *scarce resources*.

As the second stage on our solver we propose a simple metaheuristic technique based also on a MOGA that we denominate *MORLI* (multiobjective *resource leveling improvement*). This technique takes an input schedule *S* and generates a population of non dominated schedules that –possibly- dominates *S* in terms of the minimization on the resource variability. The goals of the MORLI are twofold:

1) *Maintain* the makespan of *S*.
2) *Improve* the resource leveling of a given schedule *S*

To accomplish the first goal we define that the MOGA will penalize schedules with a makespan different to the makespan of *S*. To accomplish the second objective, MORLI performs a series of *right shifts* (also referred as *local shifting* by Demeulemeester and Herroelen in [22]) on specific activities over the input schedule *S*. These activity shifts refers to the sequencing of activities on a *slack range* by considering a given schedule. This slack range (SLK) for each activity *j* is calculated over *S* as

$$SLK_j = \left| s'_j - s_j \right|. \tag{14}$$

Where s$_j$ is the start time of activity *j* in the schedule S, and s'$_j$ is the start time of activity *j* in schedule *S*' which is obtained with a BS-SGS (without being globally shifted to the left) and an activity list AL that corresponds to the list of the activities of *S* in the order they were chosen in each iteration of the SGS. Table III illustrates this calculation for the schedule shown in Fig. 9(a) while and Fig. 11 shows the corresponding S' generated by the BS-SGS and evaluated over this schedule.

TABLE III
EXAMPLE OF SLACK RANGE CALCULATION

j	s$_j$	s'$_j$	*SLK$_j$*
2	0	2	2
3	0	0	0
4	0	1	1
5	7	10	3
6	7	8	1
7	4	4	0
8	1	2	1
9	11	14	3
10	10	11	1
11	10	10	0
12	15	15	0

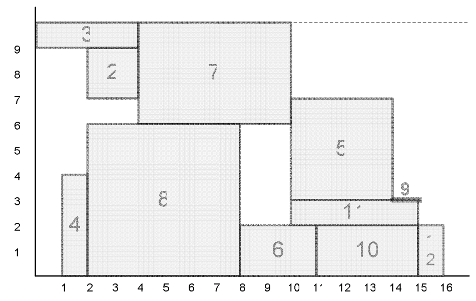


Fig. 11 Schedule generated with a BS-SGS without global left shifting

a) Chromosome Encoding

We propose to use the encoding structure in Fig. 12, where each gene AS$_j$ corresponds to the *shifting percentage* of activity *j*. This percentage is evaluated over the associated SLK$_j$, therefore *shifted starting time* (SST) of activity *j* is calculated as

$$SST_j = SLK_j \times AS_j \tag{15}$$

The SST must be considered by the SGS in order to schedule activity *j*, thus, the SGS will find a precedence and resource feasible starting time for activity *j* starting from the new earliest starting time (NST) calculated as:

$$NST_j = EST_j + SST_j \tag{16}$$

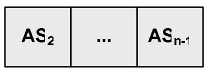
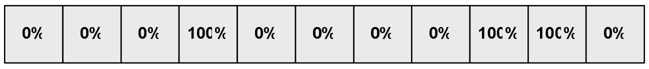


Fig. 12 Encoding for the leveling improvement

In order to illustrate this encoding, Fig. 13 shows a MORLI chromosome that is decoded as the schedule shown in Fig. 9(b), the new starting times for each activity *j* are shown in Table IV.

TABLE IV
RESULTING DECODING OF THE MORLI
EXAMPLE CHROMOSOME

j	SLK_j	AS_j	SST_j	NST_J
2	2	0%	0	0
3	0	0%	0	0
4	1	0%	0	0
5	3	100%	3	10
6	1	0%	0	7
7	0	0%	0	4
8	1	0%	0	1
9	3	0%	0	14
10	1	100%	1	11
11	0	100%	0	10
12	0	0%	0	15



0%	0%	0%	100%	0%	0%	0%	0%	100%	100%	0%

Fig. 13 Example MORLI chromosome encoding

b) Genetic Operators

We consider the following genetic operators for the proposed GA:

1) *Crossover*: Given two chromosomes we obtain two new chromosomes (offspring) by using a *standard one-point crossover*.
2) *Mutation*: If a gene AS_j is selected for mutation, then new random value is assigned between 0 and 100.
3) *Selection*: standard *binary tournament selection.*
4) *Initial Population:* Each gene value is initialized with a standard random generator.

The fitness function of the MORLI is defined as multiobjective:

1) Minimize the absolute makespan difference of the input schedule and the obtained schedule,
2) Minimize the MV for each resource

V. EXAMPLE ANALYSIS

In order to validate the effectiveness of the proposed solving method, we proceed to compare our results with state-of-art resource leveling algorithms.

A. Experiments Setup

TABLE V
COMPUTATIONAL SETUP FOR THE EXPERIMENTS

Element	Description
CPU	1.66Ghz
Language	JAVA
Environment	Eclipse 3.3. JDK JRE1.6.0_05-b13
NSGA-II	Implemented using the library JMETAL [23]
ERLP Model and Project Scheduler	Implemented on a JAVA port of *PSPSolver* [24]

TABLE VI
MOGA PARAMETERS

Parameter	Value
Population	$1.5 \times n$
Crossover rate	0.7
Mutation rate	0.2

For our experiments we considered the computational setup detailed in Table V and the MOGA parameters detailed in Table VI. The MOGA parameters were obtained after many experiments and trials and are applied in both stages of the proposed solver. We fix to 400 the maximum schedule evaluations for the first stage and to 100 for the postprocessing stage (e.g. a maximum of 500 evaluations).

B. Particle Swarm Optimization Algorithm

The first algorithm we analyze is the EPSO algorithm proposed by Li *et al.* in [15] based in *particle swarm optimization* (PSO), in this paper an example analysis is proposed for the network detailed in Fig. 14. As this problem is not an ERLP (it considers *unlimited* resources) we first proceed to adapt it. The adaptation consists in defining availability values to each resource, in this case we use the values Rk={32,32,19}, which corresponds to the maximum resource usages of a *non-optimized* schedule indicated in the same paper (alternatively, this can be calculated as the maximum levels requested after building a schedule considering a priority rule as the *shortest-processing-time*), and we consider the *full* range for all the resources. One particularity of this instance is that the objective to minimize is the *variance* (σ^2) of the sum of the usage of resources in each period of time.

The PSO proposed by Li *et al.* was able to find the schedule *S*={0,0,0,3,0,7,3,5,5,9,8,10,12,12,14}. The RUP for this schedule is detailed in Table VII and the corresponding variance of the resource usage is 63.41.

TABLE VII
RUP FOR THE PSO SOLUTION

Resource	Time						
	1	2	3	4	5	6	7
R_1	24	24	24	28	28	16	16
R_2	17	17	17	12	12	19	19
R_3	6	6	6	9	9	13	13
Sum	47	47	47	49	49	48	48
Resource	Time						
	8	9	10	11	12	13	14
R_1	20	22	22	20	20	12	12
R_2	12	12	10	11	11	11	11
R_3	11	9	6	6	6	3	3
Sum	43	43	38	37	37	26	26

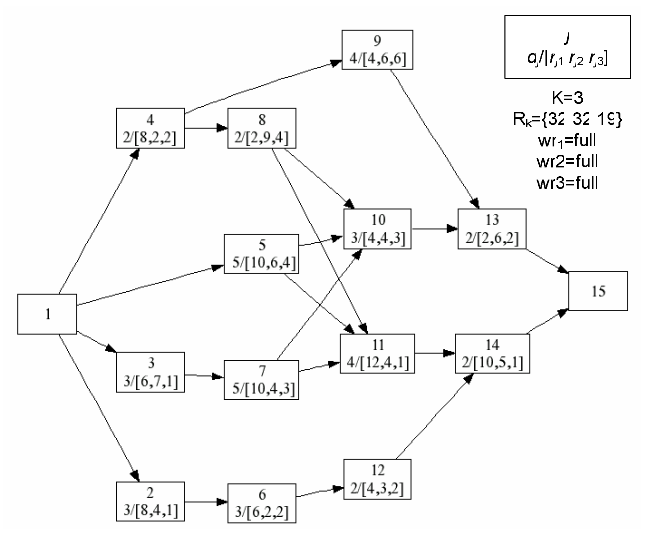


Fig. 14 An instance converted to ERLP (adapted from Li *et al* [15])

In a first experiment we consider only two objectives for the ERLP: the makespan and the minimization of the variance of *the sum* of the resource usage over time. After the first stage (i.e. 400 schedule evaluations) the solver reported the non-dominated population detailed in Table VIII. As can be observed, the outcome of the MOGA is a set of trade-offs between makespan and variance. The average time of execution of this stage is 0.64s.

TABLE VIII
POPULATION OBTAINED ON THE FIRST EXPERIMENT WITHOUT POSTPROCESSING

Chromosome	σ^2	*Mk
A={P,B,9,1,7,2,8,0,3,7,5,4,3,2,5,8,5,97%,59%,93%}	121,26	14
B={P,B,8,2,3,6,1,3,9,7,5,4,3,2,5,2,9,64%,98%,72%}	45,38	17
C={P,B,8,2,3,6,1,3,9,4,2,3,6,6,8,4,3,73%,94%,87%}	96,39	16
D={S,B,8,2,3,6,1,3,9,4,2,2,9,6,8,4,3,87%,94%,73%}	120,43	15

*Mk = makespan

TABLE IX
POPULATION OBTAINED ON THE FIRST EXPERIMENT AFTER POSTPROCESSING

Chromosome	σ^2	*Mk
A'={100,0,0,100,0,100,0,0,0,100,100,0,100}	61.41	14
B'={100,100,0,100,100,100,100,100,100,100,0,0,0}	34.13	17
C'={0,100,0,100,0,0,100,100,0,100,0,0,0}	53.73	16
D'={0,100,0,100,0,0,100,100,0,100,0,0,0}	58.71	15

*Mk = makespan

TABLE X
RUP OF THE BEST SCHEDULE OBTAINED WITH THE PROPOSED SOLVER

Resource	Time													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R1	24	24	24	20	28	24	18	18	20	24	20	20	12	12
R2	17	17	17	10	12	8	15	15	13	17	14	14	11	11
R3	6	6	6	7	9	7	9	9	9	12	10	10	3	3
Sum	47	47	47	37	49	39	42	42	42	53	44	44	26	26

TABLE XI
COMPARISON OF RESULTS BETWEEN THE PSO AND THE PROPOSED SOLVER FOR THE FIRST EXPERIMENT

Method	Best variance σ^2	Time to solve
PSO	63.41	720 s
MOGA ERLP (without postprocessing)	121.26	0.64 s
MOGA ERLP (with postprocessing)	61.41	1.84 s

In order to compare our solution to the one obtained with the PSO we chose the solution with the lower makespan (e.g. chromosome A), the corresponding schedule is *S*={0,0,0,3,0,5,3,5,7,8,8,8,11,12,14}. After applying the postprocessing stage (with an average processing time of 1.2s) these solutions where optimized as shown in Table VIII. After improving the schedule of chromosome A the solver obtains the schedule *S'*={0,0,0,4,0,5,3,6,8,9,8,8,12,12,14}, the RUP for this schedule is detailed in Table X. Table XI presents the results obtained with the PSO and our proposed MOGA considering the schedule with the minimum makespan. We can conclude from these results that our MOGA outperforms the PSO approach reported by Li *et al* for this example. As an important remark, the PSO reported by Li *et al.* outperforms three other RLP methods using this example (a basic PSO, a dynamic programming approach, and a genetic algorithm approach).

In a second experiment we consider the standard objectives of the ERLP. In this particular example the solver seeks to optimize the makespan and the variance (σ^2) of each resource. Table XII shows a comparison of the results found. An important remark is that two of the found solutions by the solver *dominate* the solution found by the PSO. For this specific example, we conclude that our solver can optimize simultaneously each objective and present competitive trade-offs to the DM.

TABLE XII
COMPARISON OF RESULTS BETWEEN THE PSO AND THE PROPOSED SOVLER FOR THE SECOND EXPERIMENT

Method	σ^2			Makespan	Time to solve
	R1	R2	R_3		
PSO	26.11	11.02	10.26	14	720 s
MOGA ERLP	26.11	6.71	7.19	14	
(with	21.19	7.17	8.42	14	2.64 s
postprocessing)	23.64	9.48	11.19	14	

C. Petrinet Approach

Raja and Kunaman [16] propose a Petrinet-based approach to solve the single and multi resource leveling. This method considers the resources as unlimited and the objective is to minimize the measure $\sum R_i^{\,2}$. In their paper, the authors propose a case study as detailed in Fig. 15. In a third experiment we adapt this problem to the ERLP setting the resource availability to R_I={12} (e.g. calculated by a using a priority rule) and the range to dynamic (as activity 2 requires 0 resources during 2 time units). This result is obtained in 0.28s (including the postprocessing) and the MOGA is able to find the best result proposed by Raja and Kunaman between the set of non-dominated solutions, as detailed in Table XIII.

TABLE XIII
POPULATION OBTAINED FOR THE THIRD EXPERIMENT

Schedule	$\sum R_i^{\,2}$	Makespan
S={0,0,2,8,2,13,13,11,17,20}	885.0	20
S={0,05,13,2,16,16,8,20,23}	741.0	23
S={0,0,8,16,2,19,19,11,23,26}	651.0	26
S={0,0,15,2,5,26,11,25,30,33}	507.0	33

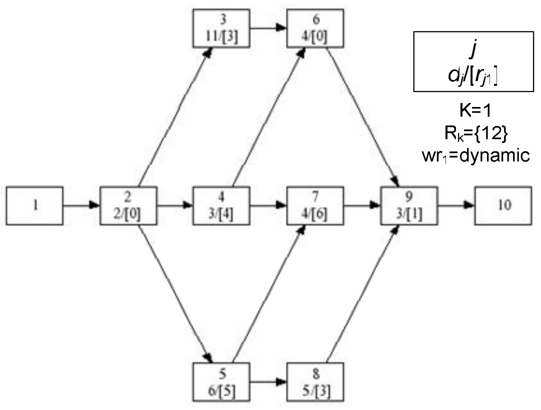


Fig.15 An instance converted to ERLP (adapted from Raja and Kunaman[16])

D. *Optimal Multiple Resource Leveling*

In our fourth experiment we analyze the method proposed by Younis and Saad in [7]. In this method the authors propose a mathematical formulation for the optimal resource leveling of multiple resources and present the study case illustrated in Fig. 16. We adapt this problem to the RLP by setting R_k={11,8,18}, the MV is the RLI and all the ranges as Full. Our solver was able to find the optimal values reported by Younis and Saad (e.g. when Rk={10,7,18}) in 0.21s, as detailed in Table XIV.

TABLE XIV
COMPARISON OF RESULTS BETWEEN THE PSO AND THE PROPOSED SOLVER FOR THE FIRST EXPERIMENT

Method	RLI			*Mk	Time to Solve
	R1	R2	R3		
Optimal Resource Leveling	5.0	9.0	35.0	5	Not reported
MOGA ERLP (without postprocessing)	5.0 21.0	9.0 17.0	35.0 19.0	5 7	0.047s
MOGA ERLP (with postprocessing)	4.8 13.0	20.0 21.0	33.8 23.0	5 7	0.172s

*Mk = makespan

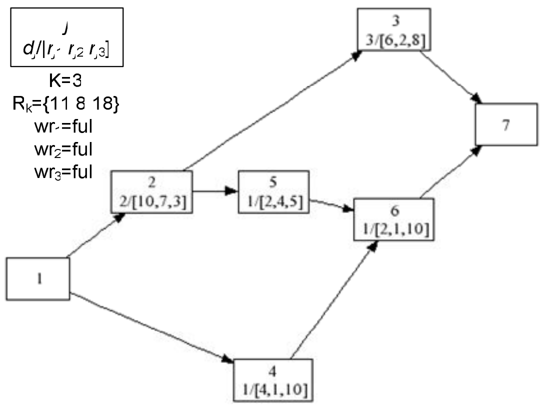


Fig. 16 An instance converted to ERLP (adapted fromYounis and Saad [7])

E. *ERLP Benchmark*

In order to share and compare our results with the research community we publish a benchmark set for the ERLP. This test set is freely available for download from http://rocamatics.wordpress.com. In this online resource we also publish the comments and the results obtained by other methods. Our benchmark consists in an adaptation from the RCPSP benchmark sets from PSPLIB [25]: J30, J60, J90, and J120. We modify these datasets to include different work ranges for each resource and we consider the RLI as the main MV. In this benchmark we report:

1. The ERLP parameters (work ranges) for each instance
2. The best non-dominated solutions for each instance (pareto fronts)
3. The best known individual MV for each resource in each instance

VI. CONCLUSION

We introduce an extended resource leveling model that accurately abstracts projects with a specific work range for each resource and consider the simultaneous optimization of the makespan and the resource leveling of each resource usage. We propose a metaheuristic-based solver for this model. The computational results show that the proposed solver is a fast and effective algorithm. It is our opinion that the success of the proposed solver is mainly due to the incorporation of specific knowledge of the problem via the intelligent encoding. The proposed encoding helps the solver to overcome potential SGS limitations. Other important feature of the proposed solver is its ability to propose alternative (non dominated) solutions to the DM as it is based on the MOEA approach. Future research could include a detailed performance analysis of the solver and its extension to the multi-mode version of the ERLP, and also the study of the application of the proposed encoding with other paradigms (tabu search, simulated annealing, ant systems, etc.)

REFERENCES

[1] R. Kolisch., S. Hartmann, *Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update*, European Journal of Operational Research, 2005.

[2] J. Blazewicz, W. Cellary, R. Slowinsky, J. Weglarz . *Scheduling under Resource Constraints: Deterministic Models*, Annals of Operations Research, 1987.

[3] R. Kolisch, S. Hartmann. *Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis* in *Project scheduling: Recent models, algorithms and applications*, Kluwer, 1999.

[4] P. Brucker, S. Knust, *Complex Scheduling*, Springer, 2005.

[5] V. Valls, F. Ballestín, S. Quintanilla, *Justification Technique Generalizations*, in *Perspectives in Modern Project Scheduling*, Springer, 2006.

[6] H.N. Ahuja, *Construction Performance Control by Networks*, Wiley, New York, 1976.

[7] M.A. Younis, B. Saad, *Optimal resource leveling of multi-resource projects*, Computers and Industrial Engineering, 31, 1996.

[8] A.R. Burgess, J.B. Killebrew, *Variation in activity level on a cyclical arrow diagram*, Journal of Industrial Engineering 13, 1962.

[9] R.B. Harris, *Precedence and Arrow Networking Techniques for Construction*, Wiley, New York, 1978.

[10] R.B. Harris, *Packing method for resource leveling (pack)*, Journal of Construction Engineering and Management 116, 1990.

[11] L. Kim, K. Kim, N. Jee, Y. Yoon, *Enhanced Resource Leveling technique for Project Scheduling*, Journal of Asian Architecture and Building Engineering, 466, 2005

[12] P. Brucker, A. Drexl, W. Mohring, K. Neumann, E. Pesh, *Resource-constrained project scheduling: Notation, classification, models, and methods*, European Journal of Operational Research 112, 3-41, 1999.

[13] Leu, C. Yang, J. Huang, *Resource leveling in construction by genetic algorithm-based optimization and its decision support system application*, Automation in construction, 2000.

[14] Y. Sheng-Li, M. Hong, L. Ri, *GA-Based Resource Leveling Optimization for Construction Project*, in *International Conference on Machine Learning and Cybernetics*, 2006

[15] X. Li, L. Zhang, J. Qi, S. Zhang, *An extended particle swarm optimization algorithm based on coarse-grained and fine-grained criteria an dits application*, Journal of Central South University of Technology, 15, 2008.

[16] K. Raja, S. Kumanan, *Resource Leveling Using Petrinet and Memetic Approach*, American Journal of Applied Sciences, 4, 2007.

[17] P. Dasgupta. P. Chakrabarti, S. Desarkar, *Multiobjective Heuristic Search: An introduction to Intelligent Search Methods for Multicriteria Optimization*, Computational Intelligence, Vieweg, 1999.

[18] C. Coello, D. Van Veldhuizen, G. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers, 2002.

[19] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II,* IEEE Transactions on Evolutionary Computation, 6, 2002.

[20] J. Alcaraz, C. Maroto, *A hybrid genetic algorithm based on intelligent encoding for project scheduling*, in *perspectives in modern project scheduling*, Springer, 2007.

[21] V. Valls, F. Ballestin,S. Quintanilla S, *A hybrid genetic algorithm for the resource-costrained scheduling problem*, European Journal of Operational Research, 2007.

[22] E. Demeulemeester, W. Herroelen, *Project scheduling. A research handbook*, Kluwer Academic Publishers, 2002.

[23] J. Durillo, A Nebro, F. Luna, B. Dorronsoro , E. Alba, *jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics*, Tech Report DNL06, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, 2006.

[24] J. Roca, F. Bossuyt, G. Libert, *PSPSolver: An Open Source Library for the RCPSP*, *Proceedings of the 26th Workshop of the UK Planning and Scheduling Special Interest Group*, 2007.

[25] R. Kolish, A. Sprechher, *PSPLIB A Project Scheduling Problem Library*, European Journal of Operational Research, 96, 1997.