A Population-based Approach to the Resource-Constrained Project Scheduling Problem*

Vicente Valls[a], Francisco Ballestín[a] , and Sacramento Quintanilla[b]

a Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain. Vicente.Valls@uv.es Francisco.Ballestin@uv.es.
b Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, Valencia, Spain. Maria.Quintanilla@uv.es.

Latest Revision: October, 2001

Abstract — In this paper, we present a population-based approach to the RCPSP. The procedure has two phases. The first phase handles the initial construction of a population of schedules and these are then evolved until high quality solutions are obtained. The evolution of the population is driven by the alternative application of an efficient improving procedure for locally improving the use of resources, and a mechanism for combining schedules that integrates scatter search and path relinking characteristics. The objective of the second phase is to explore in depth those vicinities near the high quality schedules. Computational experiments on the standard j120 set generated using ProGen, show that our algorithm produces higher quality solutions than state-of-the-art heuristics for the RCPSP in an average time of less than five seconds.

Keywords: population-based algorithms; scatter search; path relinking; hybrid heuristics; resource constrained project scheduling.

1. Introduction

The resource-constrained project-scheduling problem (RCPSP) may be stated as follows. A project consists of a set of n activities numbered 1 to n, where each activity has to be processed without interruption to complete the project. The dummy activities 1 and n represent the beginning and end of the project. The duration of an activity j is denoted by d_j where $d_1 = d_n = 0$. There are R renewable resource types. The availability of each resource type k in each time period is R_k units, $k = 1,..., K$. Each activity j requires r_{jk} units of resource k during each period of its duration where $r_{1k} = r_{nk} = 0$, $k = 1,..., K$. All parameters are assumed to be non-negative integer valued. There are precedence relations of the finish-start type with a zero parameter value (i.e., FS = 0) defined between the activities. In other words, activity i precedes activity j if j cannot start until i has been completed. The structure of a project can be represented by an activity-on-node network $G = (V, A)$, where V is the set of activities and A is the set of precedence relationships. S_j (P_j) is the set of successors (predecessors) of activity j. It is assumed that $1 \in P_j$, $j = 2,...,$ n, and $n \in S_j$, $j = 1,...,$ n-1. The objective of the RCPSP is to find a schedule S of the activities, i.e., a set of starting times $(s_1,s_2,...,s_n)$ where $s_1=0$ and the precedence and resource constraints are satisfied, such that the schedule duration $T(S) = s_n$ is minimised. Let T^* be the minimum schedule duration or minimum makespan.

As a job shop generalisation, the RCPSP is NP-hard in the strong sense (see Blazewicz et al., 1983). In recent years, the applications and difficulties of the RCPSP and its extensions have attracted increasing interest from researchers and practitioners. We refer to the surveys published by Icmeli et al. (1993), Özdamar and Ulusoy (1995), Kolisch and Padman (1997), Herroelen et al. (1998) and Brucker et al. (1999). For solving the RCPSP, the most competitive algorithms seem to be those of Demeulemeester and Herroelen (1992), Sprecher (1996), Brucker et al. (1998) and Mingozzi et al. (1998). However, only small-sized problem instances with up 30 activities can be solved exactly in a

* This research was partially supported by the CICYT under contract TAP99-1123 and by the Generalitat Valenciana under contract FP 198-CB-12-303.

satisfactory manner. Therefore, heuristic solution procedures remain as the only feasible method of handling practical resource-constrained project scheduling problems. Many heuristic approaches have been proposed for RCPSP (we refer to the aforementioned surveys). Kolisch and Hartmann (1999) have given an overview of heuristics focusing on X-pass methods (single pass methods, multi-pass methods, sampling procedures) and metaheuristics (simulated annealing, genetic algorithms, tabu search). Hartmann and Kolisch (2000) provide an investigation of the performance of those RCPSP heuristics. Their computational results indicate that the best metaheuristics outperform the best sampling approaches. As stated by the authors, this is mainly because sampling procedures generate each schedule anew without considering any information given by previously visited solutions – as metaheuristics typically do.

Li and Willis (1992) propose an iterative forward/backward scheduling technique for project scheduling under general resource constraints: renewable, non-renewable and doubly constrained resources with uneven availabilities over time. Basically, the procedure iteratively applies serial forward and backward scheduling until no further improvement in the project duration can be obtained. The activity finish (start) times of a forward (backward) schedule determines the activity priorities for the next backward (forward) schedule. Özdamar and Ulusoy (1996) take up again the basic idea of iterative forward/backward scheduling and construct a totally different algorithm. They employ a parallel schedule generation scheme in which the selection of the next activity to be scheduled is made according to a conflict base procedure.

Nonobe and Ibaraki (1999) presented a tabu search based heuristic algorithm for a generalisation of the RCPSP. It is able to handle multi-mode processing, variable availability of renewable and non-renewable resources, and complex objective functions. They tested their code for a number of benchmarks of the jobshop and the RCPSP, as well as some problems from real applications.

Hartmann (2000) extends his activity list GA (Hartmann, 1998) with several features: an extended representation that includes an additional gene that determines the decoding procedure, adapted crossover and mutation operators, and a new method for computing an initial population. This new self-adapting GA increases the quality solution although at some computational cost.

Merkle et al (2000) present an ant colony optimization approach for the RCPSP. They tested their algorithm on the standard set j120 generated using ProGen and available from the PSPLIB library (Kolisch and Sprecher, 1997). They compared their algorithm to various other heuristics which are included in the Hartmann and Kolisch computational study. Every heuristic was allowed to construct and evaluate at most 5000 schedules for each problem instance. The computational results of this experimental design show that their algorithm performed best on average. They also claim that their algorithm outperforms that of Nonobe and Ibaraki.

Möhring, R. H. et al (2000) propose a Lagrangian-based list heuristic to compute feasible solutions for resource-constraint projects. A computational study on the j60, j90, j120 instance sets from PSPLIB shows that the quality of the feasible solutions compared to results obtained with state-of-the-art local search algorithms.

Valls et al (2001) propose a metaheuristic algorithm, CARA, for solving the resource-constrained project-scheduling problem. The procedure is a non-standard implementation of fundamental concepts of tabu search without explicitly using memory structures. They consider two phases in the application of the algorithm. Phase 1 makes use of temporal information whereas Phase 2 is based on two new multi-pass sampling methods (the β biased random sampling method and the window sampling method) adequately defined to make controlled moves. The algorithm is a robust procedure that can be successfully applied to diversely characterised instance problems without the need for parameter adjustment. Computational experiments on the standard sets j30, j60, j90 and j120 for the RCPSP and generated using ProGen show that the quality of CARA is quite good and that computational times are short. They also show that CARA can compete with state-of-the-art heuristics for the RCPSP.

Alcaraz and Maroto (2001) generalise the original activity list GA of Hartmann (1998). In their GA, each representation includes an additional gene that determines whether the activity list is scheduled in a forward or a backward way. The initial population, crossover techniques, and mutation operator are modified to handle this new representation. Various possibilities for the different components of the genetic algorithm are explored and the best are chosen.

Tormos and Lova (2001) describe a hybrid multi-pass method that combines random sampling procedures with a backward-forward method. In each iteration of their process a schedule is calculated by means of a regret-biased sampling method, using the LFT priority rule and a schedule generation scheme. Afterwards, a backward and a forward pass are applied to the schedule. Both the parallel and serial schemes are used.

Population-based strategies, often referred to as evolutionary methods, manipulate a collection of solutions rather than a single solution at each stage. A prominent subclass of these methods is based on strategies for 'combining' solutions, as illustrated by genetic algorithms, scatter search and path relinking methods. Another prominent subclass consists of methods that are primarily driven by utilizing multiple heuristics to generate new population members (Glover and Laguna, 1997).

Path relinking generates new solutions by exploring trajectories that connect elite solutions – by starting from one of these solutions and generating a path in the neighbourhood space that leads towards other solutions. This is accomplished by selecting moves that introduce into the current solution attributes contained in the ending solutions.

The scatter search methodology consists of creating a population of solution vectors that 'evolves' according to the characteristics of the particular problem instance being solved. It forms linear combinations of subsets of vectors (reference vectors) in the current population and creates new solutions that inherit the useful information contained in the selected reference vectors. The linear combinations are chosen to capture information not contained separately in the original vectors. An early application of scatter search to a stochastic project-scheduling problem can be found in (Valls et al, 1999).

In this paper, we present a population-based approach to the RCPSP. The procedure incorporates different strategies for generating and improving a population of schedules. The method has two phases. The first phase constructs an initial population of schedules and evolves them until high quality solutions are obtained. Random and rapid construction procedures are used to create a high quality and diverse initial population. The subsequent evolution is driven by the alternative application of an improving procedure and a schedule combination mechanism that blends characteristics of scatter search and path relinking. The improving procedure that is applied to each population schedule is an iterative procedure for improving the local use of resources. It incorporates an oscillatory mechanism that alternatively searches two different regions of the schedule space (strategic oscillation). The combination mechanism obtains new schedules that blend characteristics from the best population solutions. To achieve this, it produces schedules inside the spanned convex region.

The objective of the second phase is to closely explore regions near high quality sequences. Exploring such a region means, firstly, generating a population by taking a random sample from a region near the sequence, and secondly, applying to the population the procedures used in the first phase – but with a variation. The on-going search is interrupted when a better sequence is obtained and a fresh search starts from the point of the newer sequence. Phase two begins from the best solution obtained in phase one.

The suggested approach is the result of combining various procedures which have been individually designed for a specific purpose – and so can be used separately. The rest of the paper is organised as follows. Section 2 describes a procedure for improving a given sequence: the Homogeneous Interval Algorithm, or HIA. Section 3 presents a procedure for generating sequences within the 'convex hull'

of a given set of schedules: the Convex Search Algorithm, or CSA. The combination of these two algorithms with a procedure for constructing an initial population of schedules is the first phase of our approach (Section 4). The second phase and the complete algorithm are described in Section 5. The proposed algorithm can be easily modified to obtain versions producing higher quality, or more rapid, solutions. Some of these possibilities are described in Section 6. Computational tests are included in Section 7. Finally, a summary and some concluding remarks appear in Section 8.

2.- The Homogeneous Interval Algorithm: an improving procedure.

All of the possible sequences in a project use the same quantity of resource units $TR = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{i,k}$ but do not necessarily require the same number of time units. Given a schedule S, the average resource utilisation can be defined as TR /T(S). Therefore, the average resource utilisation increases as the makespan decreases. In this way, minimalising the length of a project is the equivalent to maximising the average resource utilisation. The number of time periods of low utilisation (compared to the average achieved by optimal solutions) generally decreases as the makespan diminishes. Therefore, the Homogeneous Interval Algorithm (HIA) tries to diminish the duration of an active schedule by improving the local use of resources. To achieve this, it searches the sequence and pauses at each homogeneous interval where it tries to increase the use of resources at that interval. The sequence can be searched left to right or right to left depending on whether, respectively, Forward_HIA or Backward_HIA is applied. respectively. HIA is the result of alternatively applying Forward_HIA and Backward_HIA until no further improvements can be obtained.

Strictly speaking, HIA functions in the space of the activity list (AL) representations. An iterative improvement process begins from an AL representation and in order to pass from one AL representation to the next, it advances the position of a subset of activities, and consequently pushes back the position of other activities.

The activity list representation

An *activity list* (Kolisch and Hartmann,1999) is any precedence feasible permutation $\lambda = (j_1,j_2,...,j_n)$ of the activities. If $i = j_p$ we can say that the activity i is the position p. Given a schedule S, any activity list $\lambda = (j_1,j_2,...,j_n)$ that satisfies: $s_i < s_j$ implies $p < q$ when p (q) is the position of i (j) in λ is called an *activity list (AL) representation* of S. The serial schedule generation scheme (SGS) can be used as a decoding procedure to obtain an active schedule $S(\lambda)$ from an activity list $\lambda = (j_1,j_2,...,j_n)$ by selecting at each stage p the activity j_p. A schedule S may be represented by several activity lists that only differ in the positions assigned to those activities with the same start-times. However, if *S* is active and $\lambda(S)$ is any activity list representation of *S* then $S(\lambda(S)) = S$. If the activity label is used to order the activities with the same start times then the representations is unique.

An activity list is not necessarily an AL representation of an active schedule. Nevertheless, if λ' is an activity list $\lambda = \lambda (S(\lambda'))$ is an AL representation of the schedule $S(\lambda')$ so that $S(\lambda) = S(\lambda')$. This transformation, maintained in case of a tie the order of λ', we call redefine λ'.

In the rest of the paper we will assume, except when redefining an activity list, that the rule for tie-breaking is to choose the activity with the smallest activity label and unique AL representation of S we will call $\lambda(S)$. We also assume that all considered schedules are active and use S, and $\lambda(S)$ interchangeably.

As a way of illustrating the preceding ideas, consider the following example in Figure 1.

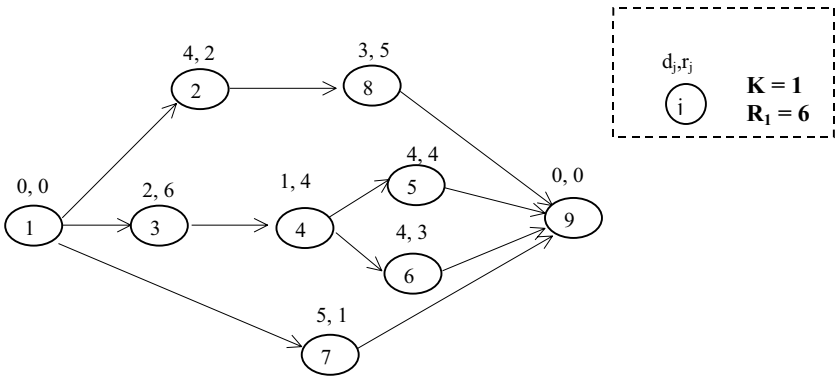

Figure 1. Activity network for illustrative example.

A feasible schedule S of the above problem is shown in Figure 2.

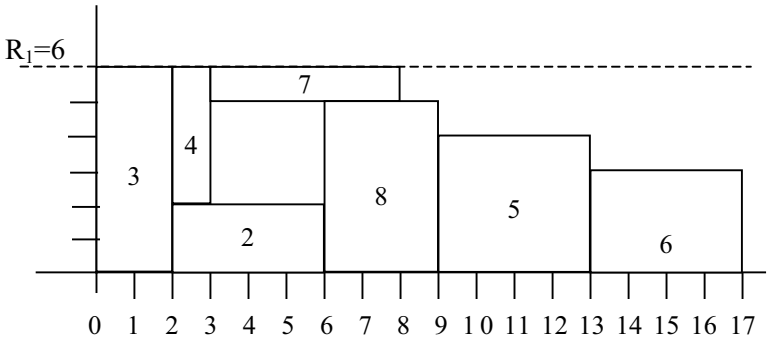

Figure 2. Feasible schedule S for the illustrative example.

The vector λ = (1,3,2,4,7,8,5,6,9) is an activity list representation of the schedule S. The vector λ′ = (1,3,2,4,8,7,5,6,9) is not an activity list representation of the schedule S as it is not compatible with the start times, p. e. $s_7 < s_8$ and activity 8 is before activity 7 in λ′. Indeed, it is not an AL representation of any active schedule. Nevertheless**,** if we redefine λ′ we obtain λ. According to the definition the only AL representation of S is λ = (1,3,2,4,7,8,5,6,9).

The average resource utilisation of S is TR/17 = 69/17 = 4.058. In the interval [3,6], three resource units are used and this can be considered a low resource utilisation. If the position of activity 6 is appropriately advanced in λ = (1,3,2,4,7,8,5,6,9), then we obtain activity list λ' = (1,3,2,4,6,7,8,5,9) which, when decodified, gives the schedule S' from Figure 3 – whose makespan is three units shorter than S. The average resource utilisation of S' is TR/14 = 69/14 = 4.928.

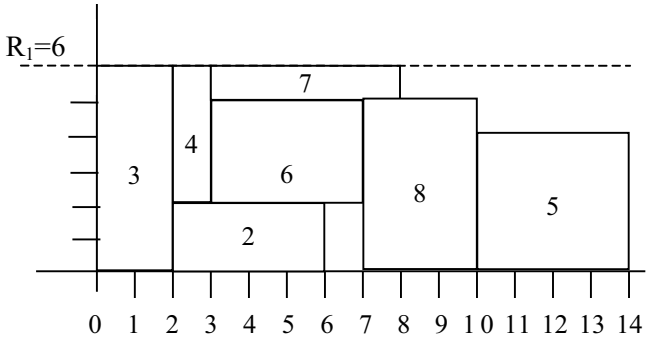

Figure 3. Feasible schedule S' = S(λ').

Homogeneous Intervals

Given an active sequence S, a homogeneous interval for S is a time interval $I = [I_s,I_f]$, with $I_s < I_f$, that has two properties:

(1) All activities i that are sequenced in at least one unit of I are sequenced in all I, meaning: $\{i \in V; [I_s,I_f] \cap]s_i,f_i[\neq \varnothing\} = \{i \in V; [I_s,I_f] \subseteq [s_i , f_i]\}$.

(2) I is maximal with respect to the inclusion, meaning that there is no time interval J that has the property 1 and strictly contains I.

The property (1) means no activity in $]I_s,I_f[$ has begun or finished. Property (2) implies that some activity has begun or finished in I_s and I_f. Additionally, as S is an active sequence this means that no activity can begin without another activity finishing, and so (2) implies that in I_s and I_f , an activity finishes (in the first homogeneous interval that begins in 0, the activity that finishes is 1).

These considerations allow us to efficiently obtain the homogenous intervals: if $0 = t_0 < t_1 <...< t_q = T(S)$ are the instants when some activity in S finishes, then the set of homogenous intervals is represented by $\{[t_{j-1},t_j], j=1,...,q\}$.

A direct consequence of (1) is that the utilisation of each type of resource is the same in all the units of interval I, and so each homogenous interval has a constant load (according to the terminology used in other sequencing problems, Valls et al (1998)). The reciprocal is not true, because two different combinations of activities can use the same amount of resources.

Returning to schedule S in Figure 2, whose single activity list representation is $\lambda = (1,3,2,4,7,8,5,6,9)$. The homogenous intervals are those whose ends are two consecutive instants from the following list: {0,2,3,6,8,9,13,17}.

Improving the resource utilisation in an homogeneous interval

To obtain a sequence from S that makes better use of the resources and is shorter, we follow the strategy of searching the homogenous intervals in increasing order of I_s and thus trying to improve the use of resources in each. When the procedure arrives at a homogenous interval I, the procedure looks for the best way to use the resources by changing the sequenced activities. As the earlier intervals have been processed we assume that the activities beginning before I_s are already fixed. Nevertheless, the set of activities beginning at I_s can be substituted for another set B of activities that could begin I_s and improve resource utilisation. This substitution is achieved by building from $\lambda = \lambda(S)$ a new activity list so that the position of the activities considered fixed does not change and the position of the activities of B is advanced. Before continuing we need to state some definitions.

S is active sequence, λ an activity list representation of S, and I an homogenous interval for S. We will term **Fixed(I)** that set of activities considered already fixed, i.e., $Fixed(I) = \{i \in V / s_i < I_s\}$. **FirstPosition(I)** $= \max\{k / \lambda(k) \in Fixed(I)\} +1$ is the first position that could be changed. **Started(I)** $= \{i \in V / s_i = I_s\}$ the set of activities beginning in I. **Candidate(I)** the set of unfixed activities that can be sequenced in I_s without breaking the precedence relations, i.e., $Candidate(I)=\{i \in V / s_i \geq I_s$ and $f_j \leq I_s \forall j \in P_i\}$. **AvRes(I,k)** $= R_k - \sum_{i \in Fixed(I) / f_i > I_s} r_{ik}$ is the amount of resource k available in I_s after sequencing the activities in Fixed(I).

A grouped measurement of the proportion of resources used by an activity j is the Resource Utilisation Ratio:

$$RUR(j) = \frac{1}{K}\sum_{k=1}^{K}\frac{r_{j,k}}{R_k}$$

This measure can be naturally extended to a set B of activities: $RUR(B) = \sum_{i \in B} RUR(i)$.

Let us say that $B \subseteq Candidate(I)$ is an eligible set of activities if:

1. $\sum_{i \in B} r_{ik} \leq AvRes(I,k)\ \forall k$
2. $RUR(B) > RUR(Started(I))$

Returning to schedule S in Figure 2, let homogenous interval I = [3,6]. Therefore: Fixed(I) = {1,2,3,4}, First_Position = 5, Started(I) = {7}, Candidate(I) = {5,6} and RUR(Started(I))=2/6 + 1/6=1/2. The eligible sets and their resource utilisation ratios are: {5}, {6}, {6,7} and 2/3, 1/2, 2/3, respectively.

Given an homogenous interval I for the sequence λ and the eligible set B, the operator **SET_SHIFT(B,I,λ)** returns an activity list λ^B that is obtained from λ advancing the positions of the B activities in increasing order of their position in λ in such a way that the new position of the first B activity is FirstPosition(I) and all the B activities occupy consecutive positions.

Returning to schedule S in Figure 2, if we select the eligible set B = {6,7}, the result of applying SET_SHIFT(B,I,λ) is λ'' = (1,3,2,4,7,6,8,5,9) which produces sequence S' of Figure 3 when decodified.

Notice that in the sequence $S^B = S(\lambda^B)$, all the B activities begin at, or before, I_s and end after I_s. So, all the B activities are sequenced in the interval $[I_s , I_s +1]$. Also λ^B and λ have the first FirstPosition-1 components in common; the relative position of the B activities is the same in λ^B as in λ, and the relative position of the rest of the activities is also the same in λ^B and λ. Therefore, the modification that the operator SET_SHIFT(B,I,λ) exercises over S is in general less when I_s is greater.

Given a homogenous interval I of a schedule λ, the **Resource_Utilisation_Improving(I,λ)** function generates all eligible sets. It selects B* as an eligible set that maximises RUR(B) – randomly breaks the ties – and obtains λ^{B^*} = SET_SHIFT(B*,I,λ). λ^{B^*} is returned when redefined. If no eligible sets exist then λ is returned.

Forward_HIA

Once the function that improves the utilisation of resources in an homogenous interval is defined we can go on to describe the Forward_HIA algorithm. This is a procedure for strict improvement. From the best schedule at hand, λ, begins a search for a shorter schedule. To do this, it obtains the homogenous intervals and searches among them in growing order of their start instants. Each time an interval I is selected, the Resource_Utilisation_Improving(I,λ) procedure is applied, and so obtaining an activity list λ'. If the makespan of λ' is less than λ, then λ' becomes the new best schedule and the procedure begins again from λ'. If the makespan of λ' is greater than λ, then it is considered that the current search direction is unpromising despite the fact that the utilisation of resources in interval I has been increased. For this reason, the current interval is not modified and the next homogenous interval is studied instead. If T(λ') = T then the search continues to explore from λ' and the next homogeneous interval. In this way, we can find a shorter schedule by improving the use of resources in the following

intervals – taking into account that until the current interval, λ' is making better use of the resources than λ.
The procedure ends when no homogenous intervals are left unexplored. The outline of Forward_HIA(λ) is as follows:

Figure 4 FORWARD_HIA (λ) outline

1. t = 0 ; T = T(λ).
2. **while** (t < T):
  2.1. Sea I = [I_s,I_f] the first homogenous interval of λ that fulfils $I_s \geq t$.
  2.2. λ' = Resource_Utilisation_Improving(I,λ)
  2.3. **if** (T(λ') < T):
    Make λ = λ'; t = 0 ; T=T(λ')
  else if (T(λ') = T):
    Make λ = λ', t = min(f'_i / $s'_i \leq I_s < f'_i$)
  else:
    Make t = I_f
3. **return** λ

where s'_i (f'_i) is the beginning (ending) instant of activity i in the sequence S' = S(λ') and where t is the time counter that marks the homogenous interval to be studied.

Backward_HIA

Forward_HIA(λ) is a procedure that attempts to shorten a given schedule λ by improving the local use of resources. This is achieved by searching from left to right, from 0 to T(S), and stopping at each activity-end where the activities for sequencing are re-examined from that time instant – bearing in mind the criteria for the use of resources. Once this procedure is completed, it seems natural to apply the same techniques but working from right to left. Before explaining the new procedure Backward_HIA(λ) some definitions have to be introduced**.**

An *active* (or *left active*) schedule can be defined as a schedule where no activity can be started earlier without delaying some other activity or violating the constraints. A formal definition can be found in Sprecher et al. (1995). Analogously, a *right active schedule* is a schedule where no activity can be finished later without advancing some other activity, or violating the constraints, or increasing the makespan. Given a schedule S, to *justify* an activity $j \neq n$ (1) to the right (left) consists of obtaining a schedule S' such that $s'_i = s_i$, $i \neq j$, $s'_j \geq s_j$ ($s'_j \leq s_j$) and s'_j is as large (small) as possible. Given a schedule S, the right (left) justification of the activities j in decreasing (increasing) order of f_j (s_j) provides a right active (left active) schedule S^R (S^L). S^R (S^L) is not unique, and depends on the tie-breaking rule used. If $s^R_1 > 0$ ($s^L_n < T$), then S^R (S^L) is shorter than S. It is not difficult to see that $T(S^R) \leq T(S)$ ($T(S^L) \leq T(S)$).

Backward_HIA (λ) begins with the best schedule S obtained by Forward_HIA (λ). Firstly, S is justified to the right and S^R is obtained, then Backward_HIA (λ) performs the same operations as Forward_HIA (λ) – but symmetrically. That is to say, the new procedure searches S^R from right to left, from T(S) until 0, and pauses at the beginning of each activity. In this way, the predecessor activities are considered as successor and vice versa. The decodifier sequences the activities in inverse order as indicated in the activity list and the activities are sequenced as late as possible, etc. Backward_HIA(λ) returns the best schedule obtained when left justified.

Note that this oscillatory mechanism is applicable to any improving function that works with active sequences, given that Backward_HIA is equivalent to Forward_HIA when applied to the reverse network.

HIA

HIA is the result of alternating procedures Forward_HIA (λ) and Backward_HIA(λ) until no further improvement can be produced.

Figure 5 HIA(λ) outline

1. λ' = FORWARD_HIA (λ)
2. λ = BACKWARD_HIA (λ')
  if(T(λ) = T(λ')):
    return λ'. **Stop**.
  else
    λ' = FORWARD_HIA (λ)
  if (T (λ') < T(λ)) **go to** 2
3. return λ

The oscillatory mechanism described in HIA(λ) provides an effective interplay between intensification and diversification. The search is alternatively driven to two different regions, the region where the utilisation of resources tends to be high at the beginning of the sequence; and the region where the resources tend to be better utilised at the end of the sequence – whereas the intensifier mechanism is the same in both regions.

3.- The Convex Search Algorithm: a method for combining schedules

In this section we present the Convex Search Algorithm. This procedure generates schedules inside the region spanned by a given set of schedules. The algorithm is based on the topological order representation of schedules (Valls et al, 1999) which is a special case of the priority value representation of Lee and Kim (1996) and Cho and Kim (1997).

The sum as operator for combining priority values

The serial schedule generation scheme (SGS) can be used as a decoding procedure to obtain an active schedule S(γ) from a vector γ of priority values by selecting at each stage the eligible activity j with the lowest priority γ_j. (We assume that the lower its priority value the more important is the activity and that a tie-breaking rule has been adopted). Therefore, the order the activities are sequenced in is determined by the relative order of the priorities; and not by their absolute values. In other words, the quality of S(γ) is determined by the relative importance of the activities indicated by the assigned priorities.

If γ and γ' are both vectors of priority values compatible with the precedence relations then $1/2\gamma + 1/2\gamma'$ is also a vector of priority values compatible with the precedence relations since if i is an activity that precedes j and fulfils $\gamma_i < \gamma_j$ and $\gamma'_i < \gamma'_j$ then $1/2(\gamma_i + \gamma'_i) < 1/2(\gamma_j + \gamma'_j)$. The vector $\gamma^1 = 1/2\gamma + 1/2\gamma'$ is the halfway point of the segment that joins γ and γ'. In the same way, the vector $\gamma^2 = 1/2\gamma + 1/2\gamma^1$ ($\gamma^3 = 1/2\gamma^1 + 1/2\gamma'$) is the half-way point of the sector that joins γ and γ^1 (γ^1 and γ'). Following iteratively this process, in the kth iteration, we obtain 2^k-1 priority vectors in the segment that joins γ and γ'.

If activity i is more important than activity j according to both γ and γ' (that is to say, $\gamma_i < \gamma_j$ and $\gamma'_i < \gamma'_j$), then activity i will also be more important than activity j according to γ^1 ($\gamma^1_i < \gamma^1_j$). So if γ and γ' coincide in the relative importance given to the two activities, then γ^1 maintains the joint relative importance. If activity i is more important than activity j, according to γ, ($\gamma_i < \gamma_j$), activity i is less

important than activity j, according to γ', ($\gamma'_i > \gamma'_j$), and $D_{ij} = \gamma_j - \gamma_i < D'_{ij} = \gamma'_i - \gamma'_j$, then γ^1 maintains the relative importance of γ' [$\gamma_j - \gamma_i < \gamma'_i - \gamma'_j \rightarrow 1/2(\gamma_j + \gamma'_j) < 1/2(\gamma_i + \gamma'_i) \rightarrow \gamma^1_j < \gamma^1_i$]. If, on the other hand, $D_{ij} > D'_{ij}$, then γ^1 maintains the relative importance of γ. But, $\gamma_j - \gamma_i > \gamma'_i - \gamma'_j \rightarrow 1/2(\gamma_j + \gamma'_j) > 1/2(\gamma_i + \gamma'_i) \rightarrow D^1_{ij} = 1/2(\gamma_j + \gamma'_j) - 1/2(\gamma_i + \gamma'_i) = 1/2[(\gamma_j - \gamma_i) - (\gamma'_i - \gamma'_j)] \leq 1/2D_{ij}$. So, if k is sufficiently large, the procedure will generate a priority vector where the relative importance of the pair i and j coincides with its relative importance according to γ'.

If γ^p, p=1,..., 2^k-1, now indicates the priority vectors generated in the segment that joins γ and γ' numbered in ascending order of their euclidean distance to γ ($\gamma^p = (1 - p/2^k)\,\gamma + p/2^k\,\gamma'$) so the relative importance of the activities according to γ^p appears progressively less like γ and more like γ' as p increases. As we will later see, these properties can be applied to the schedule series $S(\gamma^p)$ with respect to S(γ) and S(γ').

Combining schedules

The procedure described above can be used for the schedules S and S' to generate new schedules that combine the relative orders of the activities sequenced according to S and S'. To do this, it is first necessary to find two priority value vectors γ and γ' where S = S(γ) and S' = S(γ'); secondly, the vectors γ^p, p=1,..., 2^k-1 are calculated, and finally, the vectors are decodified and so obtaining the schedules $S(\gamma^p)$, p=1,..., 2^k-1.

Now this procedure is not completely certain as an infinite number of priority value vectors exist that represent the same schedule S, and the components of these vectors can be arbitrarily large or small and the difference between one priority and the next priority could also be arbitrarily large or small. Even if $\gamma_j < \gamma_i$ is fulfilled, i will always be sequenced before j if the activity i precedes that of j. These inconveniences disappear if, instead of using priority value vectors, we use topological order schedule representations.

The topological order representation

A *topological order* of the activities of a project is an order that is compatible with the precedence relations (i.e., vector $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$ of integer numbers between 1 and n, and that satisfies: $(i,j) \in A$ implies $\gamma_i < \gamma_j$ and that $\gamma_i \neq \gamma_j$, $i \neq j$). Given a schedule S, any vector $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$ of integer numbers between 1 and n that satisfies: $s_i < s_j$ implies $\gamma_i < \gamma_j$ and that $\gamma_i \neq \gamma_j$, $i \neq j$ – is called a *topological order (TO) representation* of S. The topological order representation is a special case of the priority value representation where the priority values are distinct integer positive numbers, lesser or equal to n, and compatible with the activity start times. When using the serial schedule generation scheme (SGS) as a decoding procedure, the orders play the role of priority values. A schedule S may be represented by several TO vectors that only differ in the orders assigned to those activities with the same start-times. However, if *S* is active and *γ(S)* is a TO representation of *S* then *S(γ(S)) = S*. However, if it is used to tie-break the activity label then the representation is unique. A topological order is not necessarily a TO representation of an active schedule. Nevertheless, if γ' is a topological representation then γ = γ(S(γ')) is a TO representation of the schedule S(γ') so that S(γ) = S(γ'). This transformation, supported in case of a draw in the order of γ', we will call redefine γ'.

Returning to figure 2, the vector γ = (1,4,2,3,7,8,5,6,9) is a topological order representation of the schedule S. The vector γ' = (1,4,2,3,7,8,6,5,9) is not a TO representation of the schedule S as it is not compatible with the start times, p. e. $s_7 < s_8$ and $\gamma'_7 > \gamma'_8$. In fact, it is not a TO representation of any active schedule. Nevertheless, if we redefine γ' we obtain γ.

In the rest of the paper, except when we are re-defining a TO vector, we will assume that the rule for tie-breaking is to choose the activity with the smallest activity label and we will term as *γ(S)* the only TO representation of S. Under this rule, the only TO representation of S is γ = (1,3,2,4,7,8,5,6,9). And,

if $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$ and $\lambda = (j_1,j_2,\ldots,j_n)$ are the TO and AL representations of a given S, respectively, then $j_k = i$ iff $\gamma_i = k$ is fulfilled. In the rest of the paper, we use S, *γ(S)* and *λ(S)* interchangeably.

The Convex Search Algorithm

Given two TO representations γ , γ' and a integer $k \geq 1$, the vector $\gamma^p = (1 - p/2^k)\ \gamma + p/2^k\ \gamma'$, $p = 1,..., 2^k-1$, is a vector of priorities compatible with the precedence relations in the segment that joins γ with γ'. Although γ^p is not a TO vector, as it is not a permutation of {1,...,n}, it can be easily transformed into a TO vector that gives the same schedule as γ^p when decodified. SEGMENT(k,γ,γ') is a function that returns the 2^k-1 priority vectors of γ^p after having been transformed into TO vectors.

The Convex Search Algorithm (CSA) uses these ideas to explore the 'convex hull' of a set of schedules SET = $\{S^1, S^2,..., S^q\}$. To do this, it first obtains the TO representations of the schedules in SET, then applies the function SEGMENT to each of the TO's, and finally returns the generated TO's after decodifying them.

The Convex Search Algorithm can be written as:

Figure 6 CSA(k,SET) outline

1. Sean γ^1 , γ^2, …,γ^q the TO representations of the schedules in SET
2. POP = ∅
3. **For** [i =1,q-1]
   3.1. **For** [j = i+1,q]
   3.2. POP = POP ∪ SEGMENT(k,γ^i,γ^j)
4. **Return** NEWSET = {S(γ), γ∈ POP}

Analysis of SEGMENT

Given two TO representations γ and γ', the function SEGMENT(k,γ,γ') generates 2^k-1 priority vectors γ^p, $p = 1,..., 2^k-1$, that can be found in the geometric segment joining the vectors γ and γ' of R^n. And, by extension, any two consecutive points in the succession γ, $\{\gamma^p, p=1,..., 2^k-1\}$, γ' are always at the same euclidean distance. The question to be answered is: will something similar happen to the schedules S(γ), S(γ^p), $p = 1,..., 2^k-1$, S(γ')?

To analyse the relation of the schedules S(γ^p), $p = 1,..., 2^k-1$, between themselves and with S(γ) and S(γ') it is necessary to specify a distance between schedules.

Given an active schedule S, $s_i(S)$ will be the instant that the activity i begins in S, and after(i,S) the number of activities that begin after i in S; i.e. after $(i,S) = |\{j / s_j(S) > s_i(S)\}|$. Given two sequences S and S' we can define the distance between S and S' as:

$$d(S,S') = (1/n)\sum_{i=1}^{n} |\ \text{after}(i,S) - \text{after}(i,S')\ |.$$

It is simple to show that d(S,S') fulfils the properties of a distance function (Engelking, 1989) in the active schedule space. In particular, d(S,S') = 0 if and only if S = S'.

We made the following experiment. Our test problems have been extracted from the set j120 constructed by the project generator ProGen (Kolisch et al., 1995) available in the PSPLIB project scheduling problem library (http://www.bwl.uni-kiel.de/Prod/psplib/index.html). The set j120 consists of 600 projects with four resource types and 120 activities. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network

complexity (1.5, 1.8, 2.1), resource factor (0.25, 0.5, 0.75, 1.00) and resource strength (0.1, 0.2, 0.3, 0.4, 0.5). This means 3x4x5 = 60 combinations of the possible values of the parameters. For each combination of parameter values, the set j120 has 10 instances randomly generated. For our experiment, we have selected the first instance of each of the 60 groups of 10 instances generated with the same combination of parameter values.

For each instance we have randomly generated 20 pairs of active schedules – producing a total of 1200 pairs. For each pair of schedules S, S' we obtained their TO representations γ, γ' and we applied SEGMENT(4,γ,γ') obtaining in this way 15 active schedules S^1, S^2,..., S^{15}. We then adjusted by squared minimums a straight line, Y = b + mX, the scatter plot {p, d(S^p, S)}, p =1,...,15 obtaining a correlation coefficient r. By repeating the calculations for {p, d(S^p, S')}, p =1,...,15 we obtained the straight line Y = b' + m'X with a correlation coefficient r'.

Figure 7 (figure 8) shows the scatter plot of {(d(S,S'), r), $\forall$ S,S'}({(d(S,S'), r'), $\forall$ S,S'}). We can see that, in general, r (r') assumes values near to 1 (-1), in fact 94'92 % (95.58 %) of the correlation coefficients are greater (lesser) than 0'95 (-0'95) and 99'17% (99.00 %) of the correlation coefficients are greater (lesser) than 0'9 (-0'9). Additionally, we can see that the number of times r (r') is below (above) a certain value, let us say 0'95 (-0'95), diminishes as the distance between S and S' grows. In general, the regression lines explain much of the variation of d(S^p, S) and d(S^p, S') and we can say that the distance d(S^p, S) (d(S^p, S')) increases (diminishes) linearly as p increases. As p increases, the succession of sequences S(γ^p) distances itself from S(γ) as it nears S(γ').

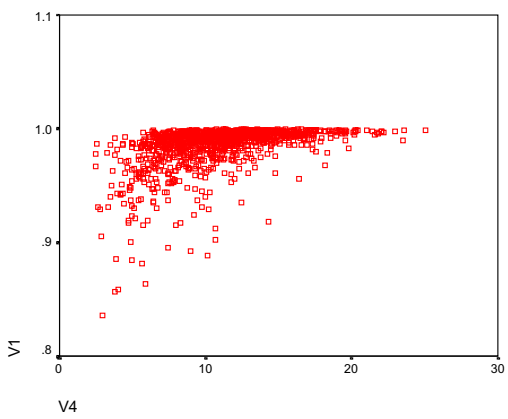

Figure 7

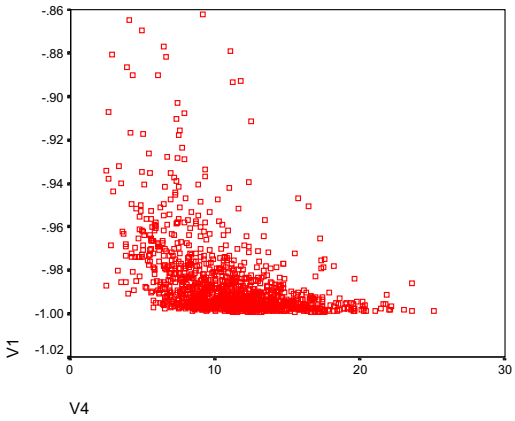

Figure 8

We can now see how close S^1 is to S and S^{15} is to S'. Figure 9 shows the scatter plot of {d(S,S'), d*(S^1, S)}, $\forall$ S,S', where S^1 is the first schedule generated by SEGMENT(4,S,S') and where the distance S^1 from S has been standardised in such a way that for all the pairs S,S', the standardised distance of S from S' is always 16, that is to say, d*(S^1, S) = d(S^1, S)*16/ d(S, S'). The data shows that in 91.83 % of the cases the standardised distance between S^1 and S is between 0 and 3, that is to say, that in 91.83 % of the cases the distance between S^1 and S is less than 3/16 of the distance separating S from S'. Figure 10 shows the scatter plot of {d(S,S'), d*(S^{15}, S')}, $\forall$ S,S'. In this instance, in 91.17 % of cases the distance between S^{15} and S' is less than 3/16 of the distance separating S and S'. In general, the succession S^p, p = 1,...,15, begins from a schedule relatively close to S and ends in a schedule relatively close to S'.

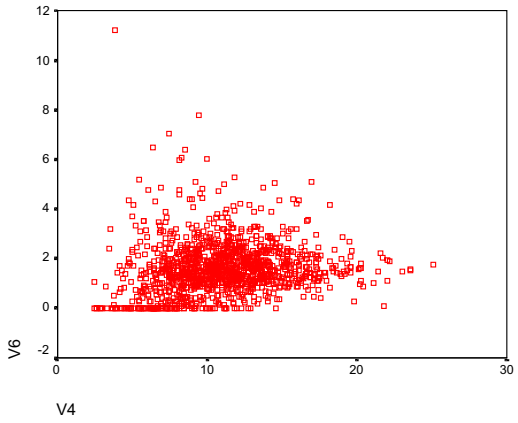

Figure 9

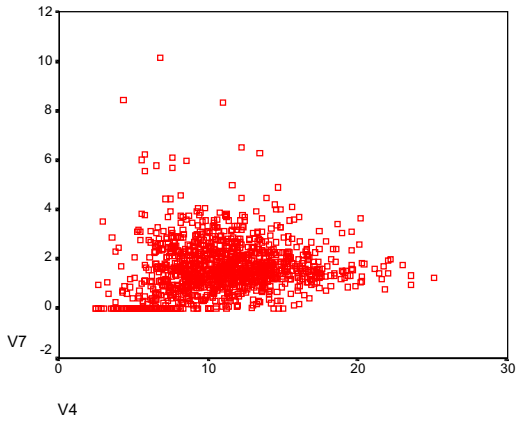

Figure 10

4.- Phase 1

HIA and CSA can be integrated in an iterative procedure that, starting from an initial population of schedules, looks to initially improve each of the schedules (HIA) and, later, search the convex hull of the selection of the best schedules of the improved population (CSA). The best solutions generated by CSA form the initial population for the next iteration.

Initial population

The initial population of solutions is generated with the goal of introducing quality and diversity. The procedure INITIAL_SET_1(*N1*) randomly generates ten activity list vectors. Two schedules are generated from each activity list vector λ by applying both serial and parallel SGS. We also use nine well-known priority rules. The rules are defined and described in Valls et al. (2001). Each priority rule originates one or two schedules depending on whether both, or only one, SGS can be applied. For each generated schedule S, an activity list representation with evaluation T(S) is obtained which becomes a candidate for inclusion in the initial population. The initial population consists of the *N1* candidates with the best evaluation.

Phase 1 outline

Having described the main components of Phase 1 in the previous sections, we now provide its outline.

Figure 11. Phase_1 outline

1. POP = INITIAL_SET_1(20)
2. POP' = HIA(POP)
3. **For** [i=1,2]
  3.1. POP5 = {the 5 best schedules in POP'}
  3.2. POP = CSA(4,POP5)
  3.3. POP40 = {the 40 best schedules in POP}
  3.4. POP' = HIA(POP40)
4. **Return** $\gamma(S_{best})$ where S_{best} is the best schedule obtained

where HIA(POP) = { HIA(λ), λ ∈ POP}.

Phase 1 can be interpreted in terms of scatter search methodology. The initial set of trial points is generated by steps 1 and 2. Then step 3.1 selects a subset of reference points. The CSA procedure

(step 3.2) is responsible for combining all the schedule reference pairs – as is usual in scatter search. To do this, and unlike the usual scatter search procedure, a trajectory of schedules is generated between each pair of reference points. This trajectory follows the path relinking strategy of incorporating each time more attributes of the guiding schedule in the initiating schedule. Steps 3.3 and 3.4 generate a new set of trial points from which the set of reference points for the next iteration will be selected. It is worth highlighting the application of the HIA improvement procedure to a selection of the best schedules obtained in the combination stage.

5.- Phase 2 and the complete algorithm

When this procedure is finished (end of phase 1), the best solution obtained so far is, hopefully, high quality. Experience seems to indicate (Valls et al, 2001) that good candidate schedules are usually to be found 'fairly close' to other good schedules. Multi-pass sampling methods can be adapted to explore, in a controlled way, regions of differing depths. Specifically, the *β biased random sampling method* (Valls et al, 2001) can be repeatedly applied to the best schedule obtained so far in order to generate a new population of relatively high quality. Generally, these schedules are inferior to the current best schedule but are promising starting points for the application of the HIA improvement procedure. Phase 2 begins at the best sequence obtained in Phase 1 and generates a new population of nearby schedules and initiates a process of improvement starting from each of the elements of population. Each time a new best solution is obtained the process begins again.

Phase 2 of the algorithm

Phase 1 starts from a population of relatively low quality solutions and drives the search to a local optimum γ_0, and hopefully, a high quality TO vector. As Phase 2 starts the search within a high quality region, it seems appropriate to make controlled moves to avoid regions of much lesser quality. Multi-pass sampling methods can be adapted to define such controlled moves.

Initial population

Given γ, a new TO vector γ' can be obtained by the application of what we call a *β biased random sampling method, β -BRSM* ($0 \leq \beta \leq 1$). This method makes use of the serial SGS. The following strategy is employed to select in each iteration the next activity to be scheduled. A number $p \in (0,1)$ is randomly generated. If $p < \beta$, then j is the eligible activity with lowest order. Otherwise, j is one of the other eligible activities selected by biased random sampling, employing the orders as priority values in order to obtain the selection probabilities. The parameter β has the following interpretation. On average, $n\beta$ is the number of iterations in which γ' and γ would select the same activity. So, if we define $\beta = 1\text{-}k/n$, then $k = n(1-\beta)$ is on average the number of iterations in which the decisions according to γ' and γ would be different. Therefore, the parameter β controls how different the new TO vector is from the original.

The procedure INITIAL_SET_2(γ_0, *N2*) generates four hundred TO vectors with $\beta = 1 - 20/n$ and two hundred TO vectors with $\beta = 1 - 10/n$ by applying the β biased random sampling method to γ_0 and then selecting the best *N2* for the initial population POP for Phase 2.

Phase 2 outline

The outline of Phase 2 is as follows.

Figure 12 Phase_2(*γ*) outline

1. *POP* = INITIAL_SET_2(*γ,20*)
2. Apply HIA(γ) to each $\gamma \in POP$ in increasing order of makespan
3. **Return** γ_{best} where γ_{best} is the best TO vector obtained

It is important to note that in Phase 2, each time an improved schedule is generated the whole procedure restarts from it. In this case, the new population is formed by the best N2 TO vectors selected among the N2 TO vectors generated by β*-BRSM* and the TO vectors which have not had HIA applied in the previous iteration.

A connection can be observed between Phase 2 and GRASP (Feo and Resende, 1989). Each application of Phase 2 of a given γ can be interpreted as the execution of multiple GRASP iterations. The β *-BRSM* constructive procedure is iterative, greedy, and adaptive. In each iteration, the next activity to be scheduled is probabilistically determined. The deterministic greedy function consists of selecting the eligible activity with least order according to γ. The selection probabilities vary in each iteration depending on the activities already sequenced. Unlike GRASP, the HIA improvement phase is only applied to a selection of the best schedules generated. Nevertheless, Phase 2 goes further than GRASP. When a better solution γ' is found, the whole procedure restarts – but now applying it to γ'. In this sense, we can say that Phase 2 traces a route in the region of the local optimum for HIA in a similar way to the procedures known as perturbation approaches (see for example, Martin et al. 1992).

The outline of the complete algorithm is as follows:

Figure 13 complete algorithm outline

1. γ_0 = PHASE_1
2. γ_1 = PHASE_2(γ_0)
3. **return** S(γ_1)

6.- Different implementations of the same algorithmic scheme

The described algorithm can be considered the combination of four basic procedures: a procedure for generating an initial population of schedules; an improvement procedure based on the utilisation of resources; a procedure for combining characteristics of schedules based on the exploration of a region spanned by the schedules; and a procedure that explores the neighbourhood of a high quality solution using the β biased random sampling method.

These four basic procedures can be implemented in different ways and create algorithms that offer varying speed and quality. The above described algorithm, hereafter termed the standard algorithm, is a specific algorithm that, we believe, represents a reasonable compromise between the quality of solution offered and the CPU time used to obtain solutions. Nevertheless, other versions can be implemented that emphasise one or other aspects. Below, we describe some possible versions which we will name by adding suffixes to the word standard.

Standard_20

This algorithm differs from the standard algorithm in the number of schedules selected from those generated by CSA. We obtain the outline of Phase 1 of the new algorithm by changing POP40 = {the 40 best schedules in POP} for POP20 = {the 20 best schedules in POP} and HIA(POP40) for HIA(POP20) in Step 3 of Figure 11.

Standard_10

This algorithm differs from the standard in that all the populations have size 10.

Standard_look-ahead

The Resource_Utilisation_Improving (I,λ) function is a greedy procedure; it selects at each interval the eligible B* set that locally maximises the utilisation of resources. Nevertheless, it is possible that this selection makes good resource utilisation impossible in the following homogenous intervals. For this reason, it would seem appropriate that the function that selects the best eligible B* set be a look-ahead procedure that takes into account the utilisation of resources in I_s and in the later periods – although decreasingly weighted as time advances. The standard_look-ahead algorithm is obtained from the standard algorithm by defining the Resource_Utilisation_Improving (I,λ) function in the following way:

Figure 14 Resource_Utilisation_Improving (I,λ) outline

1. If eligible sets do not exist **return** λ. If not,
2. Let M* = max $\left(15, \max\{ |B \backslash \text{Started}(I)| \ / \ B \text{ is eligible } \} + |\text{Started}(I)|\right)$
3. Calculate μ(B) for all eligible B:
  3.1. μ(B) = 0
  3.2. λ^B = SET_SHIFT(B,I,λ)
  3.3. $S^B = S(\lambda^B)$
  3.4. Para p = [FirstPosition, FirstPosition + M*-1]
    3.4.1. Sea i = $\lambda^B{}_p$
    3.4.2. Para j=[1,d_i]
      3.4.2.1. $\mu(B) = \mu(B) + RUR(i) / (\max(s^B{}_i - I_s , 0) + j + 1)$
4. Choose B* eligible / μ(B*) = max{μ(B) / B is elegible}
5. **Return** λ^{B^*} once redefined.

Note that μ(B) estimates the potential quality of the selected B to be shifted. μ(B) is the weighted sum, with decreasing weights, of the utilisation of resources of the first M* activities that are sequenced according to S^B in I_s or later. From the computational point of view, it is interesting to note that to calculate μ(B), B eligible, it is not necessary to always sequence all the activities. As the initial FirstPosition -1 activities of all the λ^B are the same, it is enough to schedule them just once. Later, to calculate each μ(B) it is enough to schedule M* activities each time. Note that scheduling all of them each time would be prohibitively costly.

Standard_look-ahead_parallel

The following strategy can be used to increase the power of Phase 1 of the algorithm standard_look-ahead. Initially, 20 schedules are generated with the procedure INITIAL_SET_1, and to each of these an HIA procedure was applied. Let S^1, S^2,..., S^{10} be the list of the 10 best schedules obtained in decreasing makespan order. Let odd_POP = { S^1, S^3,..., S^9} and even_POP = {S^2, S^4,…, S^{10}}. Step 3 of phase 1 is applied to each of these populations and so obtaining odd_POP* and even_POP* respectively. Finally, Step 3 is again applied to the result of the rejoining of the two populations. In this way, the beginning of the two populations evolves in parallel, separately combining the existing characteristics of each. Hopefully, the populations odd_POP* and even_POP* will contain high quality solutions that will have evolved towards distinct characteristics. In this way, the application of Step 3 to the combination of the two populations allows us to explore a previously unexplored region.

This strategy is an adaptation of the operational characteristic of our algorithm of the so-called island model developed by Kohlmorgen et al. (1999) inside the GA framework. Instead of connecting the islands by migration, we have united the two populations and combined them.

7.- Computational experiments

In this section we present the results of the computational studies concerning the algorithms introduced in previous sections. The experiments have been performed on a personal computer AMD at 400 MHZ. The algorithms have been coded in C.

Test problems

For test instances, we have used the standard sets j30, j60, j90 and j120 for the RCPSP. These were generated using ProGen (Kolisch et al., 1995). The sets j30, j60 and j90 consist of 480 projects with four resource types and 30, 60 and 90 non-dummy activities, respectively. The set j120 consists of 600 projects with four resource types and 120 activities. There are 2040 instances in total. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network complexity, resource factor and resource strength. Details of these problem instances are given in Kolisch et al., (1995) and Kolisch and Sprecher, (1997). They are available in the Project Scheduling Problem Library (PSPLIB) along with their optimum or best-known values. At September 10, 2001; the percentage of instances of j30, j60, j90 and j120 for which an optimal solution is known is 100%, 74.17%, 73.12% and 34.5%, respectively. These figures are an indication that for the current algorithms, and as expected, the set j120 is the most difficult to resolve and that there is possibly space for further improvement.

Parameter setting

To fix the values of the parameters of the algorithm we have performed some preliminary experiments with a little subset of problems of instance set j120. Some of these values were established when describing the algorithm, and the only value left as parameter in the CSA function $k = 4$. These parameters could be improved by customising them in each instance set. However, we decided, for the sake of clarity to maintain for all instance sets those values fixed in the preliminary study.

Computational results

Table 1 summarises the results of our first set of experiments. The first column indicates the instance set referred to in the results shown in each row. The second column, labelled *∑algorithm*, consists of the sum of the values obtained by our algorithm, and the third column, *∑PSPLIB*, shows the sum of the best values in PSPLIB as of September 10, 2001. The fourth (fifth) column, *av_dev (max_dev),* consists of the average (maximal) percentage deviations from the best solutions in PSPLIB. The number of instances for which our algorithm obtains the best known values in PSPLIB is reported in the sixth column (*best_sol*). In the same column, the cardinality of the instance set referred to appears between brackets. It is worth noting that these known values include those found by all versions of our algorithm. The average (maximal) computation time in seconds is shown in column seven (eight), labelled *av_CPU* (*max_CPU*). Finally, the average percentage deviation from the critical path makespan is reported in the ninth column, labelled *CPM_dev.*

	Σalgorithm	ΣPSPLIB	av_dev	max_dev	Best_sol	av_CPU	Max_CPU	CPM_dev
j120	75009	74013	1.00	5.48	223(600)	14.52	60.80	32.18
j90	45967	45741	0.29	5.13	372(480)	2.53	17.57	10.44
j60	38512	38368	0.25	5.26	385(480)	1.14	7.03	10.98
j30	28361	28316	0.13	3.44	443(480)	0.38	1.54	13.64

Table 1 Computational results for j30, j60, j90 and j120.

Predictably, *av_dev* and *av_CPU* increased with the size of the problem. Nevertheless, this increase is not linear. The second set of experiments analyses the performance of the different versions of the algorithm. In order to be brief, table 2 only shows the results for the instance set j120. The results are similar for the other sets.

	Σalgorithm	av_dev	Best_sol	av_CPU	CPM_dev
Standard_10	75384	1.44	206	4.96	32.84
Standard_20	75180	1.20	210	9.75	32.48
Standard	75009	1.00	223	14.52	32.18
Standard_look-ahead	74843	0.84	230	28.32	31.89
Standard_look-ahead_ Parallel	74671	0.64	271	59.43	31.58

Table 2 Performance of the five versions of the algorithm in the set j120.

The results of Table 2 show that, by maintaining the same algorithmic structure, it is easy to implement various versions of the algorithm with various trade-offs between solution quality and CPU time. The algorithms of the table are ordered from quickest to slowest and, simultaneously, from worst to best quality.

Comparison with other heuristics

In this subsection, we compare our algorithm to state-of-the-art algorithms. The algorithms of Hartmann (1998) and Bouleimen and Lecocq (1998) performed best among the 16 heuristics tested in the study of Hartmann and Kolisch (2000). The Hartmann algorithm (Hartmann 1) offered better results than Bouleimen and Lecocq for the instance set j120 and similar results for the instance sets j30 and j60. Later, an improved version of the genetic algorithm of Hartmann (Hartmann 2) was proposed by the author (Hartmann,2000). The tabu search algorithm of Nonobe and Ibaraki (1999) was not included in the this computational study. Merkle et al (2000) proposed an ant colony optimisation approach for the RCPSP. Möhring et al. (2000) propose Lagrangian-based list heuristics to compute feasible solutions for resource-constraint projects. Valls et al (2001) propose a metaheuristic algorithm, CARA, for solving the resource-constrained project scheduling problem. The procedure is a non-standard implementation of the fundamental concepts of tabu search without explicitly using memory structures embedded in a population-based framework. Tormos and Lova (2001) present a combination of a random sampling procedure and a backward-forward method. Alcaraz and Maroto (2001) propose a genetic algorithm in which the activity lists can be scheduled in a forward or a backward manner. Table 3 shows data obtained by the various authors and their computational tests on the j120 instance set.

Authors	Type of algorithm	CPM_dev	av_CPU
Alcaraz and Maroto	Robust GA	36.57	20
Hartmann 1	Activity list GA	36.74	13.15
Hartmann 2	GA extended	35.35	14.05
Merkle et al.	Ant Colony Optimization	36.65	25
Möhring et al.	Lagrangian based list	36.2	65
Nonobe and Ibaraki	Tabu Search	34.99	645.33
Tormos and Lova	Hybrid multi-pass	35.62	29.85
Valls et al. – CARA	Extended Tabu Search	34.53	17.00

Table 3 Comparison of state-of-the-art heuristic algorithms.

The times in Table 3 refer to seconds in different computers: Alcaraz and Maroto (PC at 166 MHz), Hartmann 1 and Hartmann 2 (Pentium-based computer at 133 MHz), Möhring et al. (Sun Ultra 2 workstation, 200 MHz, 1Gbyte memory), Merkle et al.(Pentium III, 500 MHz), Nonobe and Ibaraki (Sun Ultra 2 workstation, 300 MHz, 1Gbyte memory), Tormos and Lova (PC at 200 MHz) and Valls et al. – CARA (personal computer AMD at 400 MHZ).

We can see that all the versions of the algorithm described in this paper (Table 2) yield better quality solutions than any of the heuristic algorithms referred to in Table 3.

We note that the results for the second and fourth algorithms in Table 3 were obtained when limiting the number of schedules generated to 5000 whereas the fifth algorithm generated 3675 schedules on average. They are capable of computing better solutions if more iterations are allowed. In Table 4, we show additional information extracted from the aforementioned papers. All data refers to the j120 standard set. The data in the second row was obtained by the authors using the code supplied by Dr. Hartmann and described in Hartmann (1998). To obtain more details of the experiments refer to Valls et al. (2001). The data in the third and fourth rows was obtained by Merkle et al. (2000) when increasing the limit of generated sequences to 50,000 and 400,000 respectively, and applying a 2-opt strategy to the best solution found at every 20^{th} generation. The data in the fifth row was obtained by Möhring et al. (2000) by incorporating a local improvement heuristic to their algorithm.

Authors	Schedules and/or strategy	CPM_dev	av_CPU
Hartmann 1	59000	35.54	48.86 seconds
Merkle et al.	50000 + 2-opt strategy	33.68	25 minutes
Merkle et al.	400000 + 2-opt strategy	32.97	Not available
Möhring et al.	+ local improv	35.3	A marginally higher CPU time

Table 4 Further results on j120

As expected, the data in Table 4 indicates that the quality of the solutions given by the Hartman (1) and Merkle et al. increases as the number of sequences generated increases – and therefore the amount of computer time used. In the same way, the Möhring et al. improves with the addition of a local improvement heuristic. Nevertheless, these improvements do not meet the solution quality given by our algorithm – including the fastest version of it (Table 2). Taking into account the diversity of computers used in the experiments, we can confirm that our algorithm clearly outperforms – at least for the j120 test instances – the other state-of-the-art algorithms considered in this paper.

8.- Summary and concluding remarks

We have presented a population-based approach to the RCPSP. The procedure incorporates various strategies for generating and evolving a population of schedules. It is the result of combining four innovative basic procedures – each of which was designed for different functions and so can be independently used in the design of heuristic algorithms for the RCPSP. The four procedures include: a procedure for generating an initial population of schedules; an improvement procedure; a procedure for combining schedule characteristics; and a procedure for searching the region of a solution.

The procedure is organised in two phases. The first phase constructs the initial population of schedules and the subsequent evolution until a high quality solution is obtained. The second phase thoroughly searches regions near high quality sequences.

During the first phase, random and quick construction procedures that offer quality and diversity are used in the construction of the initial population. The subsequent evolution of the population is driven by the alternative application of two algorithms, HIA and CSA. HIA is an iterative procedure for locally improving the use of resources. It incorporates an oscillatory mechanism that alternatively searches two different regions of the schedule space (strategic oscillation). CSA is a procedure for generating schedules inside the convex region spanned by a given set of schedules. CSA includes features of scatter search and path relinking.

The second phase begins with the repeated application of a β biased random sampling method to the best schedule obtained in the first phase. In this way, a new population of relatively high quality schedules is obtained near the schedule. The parameter β controls how near the new schedules are from the original. The application of HIA to this new population generally gives improved schedules.

It is possible to easily generate different versions of the algorithm that represent different trade-offs between solution quality and computing time while maintaining the same algorithmic structure, the same design principles, and the same basic procedures. The computational trials undertaken with the instance set j120 from the PSPLIB library show that all the versions of our developed algorithm produce higher quality solutions than state-of-the-art heuristics for the RCPSP. This includes the fastest algorithm – which takes less than 5 CPU seconds on average.

We believe that the good results obtained are the result of the hybridization of ideas from diverse metaheuristic cultures and that their structuring in an innovative algorithmic scheme whose components have different, and clearly defined, objectives. This modular characteristic of our procedure allows us to generate, with relative ease, new algorithms by simply changing one of the modules – using, for example, a genetic algorithm to combine characteristics from the schedules.

References

Alcaraz, J. and Maroto, C., 2001. A Robust Genetic Algorithm for Resource Allocation in Project Scheduling, Annals of Operations Research, 102, pp.82-109.

Blazewicz, J., Lenstra, J. K. and Rinooy Kan, A. H. G, 1983. Scheduling subject to resource constraints: Classification and Complexity, Discrete Applied Mathematics, 5, 11-24.

Bouleimen, K. and Lecocq, H., 1998. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem, Technical Report, Service de Robotique et Automatisation, Université de Liège.

Brucker, P., Drexl, A., Möhring, R., Neumann K. and Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods, European Journal of Operational Research 112, 3-41.

Brucker, P., Knust, S., Schoo, A. and Thiele, O., 1998. A branch & bound algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research, 107, 272-288.

Cho, J. H. and Kim, Y. D., 1997. A simulated annealing algorithm for resource constrained project scheduling problems. Journal of Operational Research Society, 48, 736-744.

Demeulemeester, E. and Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Management Science, 38, 1803-1818.

Engelking, Ryszard. General topology. Second edition. Sigma Series in Pure Mathematics, 6. Heldermann Verlag, Berlin, 1989.

Feo, T. and Resende, M. G. C. (1989) A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operations Research Letters, Vol. 8, pp. 67-71.

Glover, F. And Laguna, M., 1997. Tabu Search, Kluwer Academic Publishers.

Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling, Naval Research Logistics, 45, 733-750.

Hartmann, S., 2000. A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints. Technical report.

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, European Journal of Operational Research, 127(2), 394, 407.

Herroelen, W., De Reyck, B. and Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research, 25 (4) 279-302.

Icmeli, O., Erenguc, S.S. and Zappe, C.J., 1993. Project scheduling problems: A survey. International Journal of Operations & Production Management, 13 (11) 80-91.

Kohlmorgen, U., Schmeck, H., and Haase, K, 1999. Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research., 90, 203-219.

Kolisch, R. and Hartmann, S., 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In Weglarz, J. (Ed.), Project Scheduling. Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, pp. 147-178.

Kolisch, R. and Padman, R., 1997. An integrated survey of project scheduling, Technical report 463, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel.

Kolisch, R, Sprecher, A. and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, 41:1693-1703.

Kolisch, R. and Sprecher,A. (1997): PSPLIB - A project scheduling library, European Journal of Operational Research, Vol. 96(1), pp. 205--216. (downloadable from http://www.bwl.uni-kiel.de/Prod/psplib/index.html).

Lee, J. K. and Kim, Y. D., 1996. Search heuristics for resource constrained project scheduling, Journal of the Operational Research Society, 47, 678-689.

Li, R. K. –Y. and Willis, J., 1992. An iterative scheduling technique for resource-constrained project scheduling, European Journal of Operational Research, 56, 370-379.

Martin, O., Otto, S.W. and Felten, E.W. (1992) Large-Step Markov Chains for TSP Incorporating Local Search Heuristics. Operations Research Letters, Vol. 11, No. 4, pp. 219-224.

Merkle, D., Middendorf, M. and Schmeck, H. 2000. Ant Colony Optimization for Resource-Constrained Project Scheduling. In Proceedings of the Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, 2000. To appear.

Mingozzi, A., Maniezzo, V., Ricciardelli, S. and Bianco, L., 1998. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation, Management Science 44, 714-729.

Möhring, R. H., Schulz, A. S., Stork, F. and Uetz, M., 2000. Solving Project Scheduling Problems by Minimum Cut Computations. Technical Report 680-2000. Fachbereich Mathematik. Technishe Universität Berlin.

Nonobe, K. and Ibaraki, T., 1999. Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem (RCPSP), Technical report 99010.

Özdamar, L. and Ulusoy, G., 1995. A survey on the resource-constrained project scheduling problem, AIIE Transactions, 27, 574-586.

Özdamar, L. and Ulusoy, G., 1996. An iterative local constraint based analysis for solving the resource constrained project scheduling problem, Journal of Operations Management, 14, 193-208.

Sprecher, A., 1996. Solving the RCPSP efficiently at modest memory requirements, Technical report 425, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel.

Sprecher, A., Kolisch, R. and Drexl, A., 1995. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research, 80, 94-102.

Tormos, P. and Lova, A., 2001. A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling, Annals of Operations Research, 102, pp.65-81.

Valls, V., Pérez, A., and Quintanilla, S, 1998. Pre-processing techniques for resource allocation in the heterogeneous case, European Journal of Operational Research, Vol. 107, pp. 470--491.

Valls, V., Laguna. M., Lino, P., Pérez, A., and Quintanilla, S, 1999. Project Scheduling with Stochastic Activity Interruptions, in Weglarz, J. (Ed.), Project Scheduling. Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, pp. 333-354.

Valls, V., Quintanilla, S, and Ballestín, F. 2001. Resource-constrained Project Scheduling: A Critical Activity Reordering Heuristic, European Journal of Operational Research. To appear.