Elsevier Editorial System(tm) for Computers & Operations Research
Manuscript Draft

Manuscript Number:

Title: A Project Scheduling approach to Service Centre Management

Article Type: Special Issue: PMS

Keywords: Service Centre; dynamic project scheduling; predictive-reactive scheduling; fine-tuning.

Corresponding Author: Mr. David Gómez-Cabrero, M.D.

Corresponding Author's Institution: IDIBAPS

First Author: David Gómez-Cabrero, M.D.

Order of Authors: David Gómez-Cabrero, M.D.; Vicente Valls, PhD

Abstract: In this paper we consider the problem of task scheduling and resource assignments in a Service Centre. This paper proposes the use of project scheduling technology in the design of a system management model and the corresponding solution methodology to deal, in real time, with the problem. We have modelled it as a dynamic, stochastic, online, multimode project scheduling problem with scarce resources and generalized precedence constraints of minimum and maximum types. We have carried out extensive computational studies to analyse the performance of the possible configurations of the reactive-predictive algorithms we propose. These computational studies include the design of an instance generator, the use of fine-tuning heuristic procedures and an extensive statistical and comparison analysis.

COVER LETTER:
A Project Scheduling approach to Service Centre Management

1. A justification which is approved by all authors for submitting to this Journal.

All authors approve to submit this paper to "Computers & Operations Research, Special Issue: PMS."

2. Abstract.

In this paper we consider the problem of task scheduling and resource assignments in a Service Centre. This paper proposes the use of project scheduling technology in the design of a system management model and the corresponding solution methodology to deal, in real time, with the problem. We have modelled it as a dynamic, stochastic, online, multimode project scheduling problem with scarce resources and generalized precedence constraints of minimum and maximum types. We have carried out extensive computational studies to analyse the performance of the possible configurations of the reactive-predictive algorithms we propose. These computational studies include the design of an instance generator, the use of fine-tuning heuristic procedures and an extensive statistical and comparison analysis.

3. The Classifications selected as part of the submission procedure e.g. General reference works (handbooks, dictionaries, bibliographies, etc.)

101.055 90B36 Scheduling theory, stochastic.
101.065 90B50 Management decision making, including multiple objectives.
102.160 90C59 Approximation methods and heuristics.

4. Authors and Vitae.

David Gómez-Cabrero[a], Vicente Valls[b]

a IDIBAPS, Institut d'Investigacions Biomèdiques August Pi i Sunyer. Villaroel 170, 08036 Barcelona. Spain. david.gomez@uv.es[1]

b Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, vicente.valls@uv.es

VITAE: Vicente Valls is Professor of Statistics and Operational Research at the University of Valencia. He holds a Ph.D degree in Mathematics from the University of Valencia. His research interests include project management and scheduling, machine scheduling, metaheuristics and production planning. In addition to his academic activities, Dr. Valls has served as a consultant to a variety of firms.

David Gómez-Cabrero is a PhD student in Departamento de Estadística e Investigación Operativa, Universitat of Valencia. He earned his two Graduate Degrees in Mathematics and Statistics from the University of Valencia. He is now working as research staff at IDIBAPS.

[1] Corresponding autor.

A Project Scheduling approach to Service Centre Management

David Gómez-Cabrero[a], Vicente Valls[b]

a IDIBAPS, Institut d'Investigacions Biomèdiques August Pi i Sunyer. Villaroel 170, 08036 Barcelona. Spain. david.gomez@uv.es*

b Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, vicente.valls@uv.es

Abstract - In this paper we consider the problem of task scheduling and resource assignments in a Service Centre. This paper proposes the use of project scheduling technology in the design of a system management model and the corresponding solution methodology to deal, in real time, with the problem. We have modelled it as a dynamic, stochastic, online, multimode project scheduling problem with scarce resources and generalized precedence constraints of minimum and maximum types. We have carried out extensive computational studies to analyse the performance of the possible configurations of the reactive-predictive algorithms we propose. These computational studies include the design of an instance generator, the use of fine-tuning heuristic procedures and an extensive statistical and comparison analysis.

Keywords: Service Centre; dynamic project scheduling; predictive-reactive scheduling; fine-tuning.

1. Introduction

This paper deals with a complex problem of task scheduling and resource assignments that come up in the daily management of company Service Centres (SC). SC usually deals with the requests reported by either external customers/citizens or internal users of an organization/company. Internal as well as external services must be provided with a given level of service. A call from a customer or a vendor requesting information about products or orders, or placing a claim or asking for technical support, is an event which requires specialised attention from the organisation.

Tools have been developed to automate and rationalise the daily activity of SC's. These tools can be termed Service Centre Management Tools (SCMT). In particular, once an event is identified by an SCMT, it is classified according to a pre-defined hierarchical structure, which contains all events that can occur within the system. The database stores the hierarchical structure of possible events and the information required for their resolution. It also stores all information relative to the specific resolution of past events. We are aware of proposed SCMT's that can automate most of the operations of an SC. However, there are two important areas that have not been dealt with, automatic task scheduling and automatic management of human resources. SCMTs usually include so-called scheduling tools to help managers to dynamically make allocation and scheduling decisions at certain decision points. With the information provided by these tools, managers apply their own set of rules to give responses to the requests of new services that have arisen. This is a case of pure reactive scheduling (Herroelen and Leus, 2005). This simple approach is aimed at a quick generation of a feasible plan. This approach might and frequently does lead to infeasible scenarios that are usually resolved by recurring to extra time or unfulfilling the agreed response times.

The SC workforce can be basically divided into three levels. Level 1 or Call Centre made up of telephonists who follow previously established procedures. Level 2 or Work Desk made up of multidisciplinary specialists able to resolve the various problems that were unresolved at level 1. Level 3 made up of Management.

The Level 1 workforce is homogeneous. Everyone is capable of doing the same tasks after receiving a basic training course, with the help of a knowledge database and established procedures. The Level 2 workforce is heterogeneous. It is generally scarce, highly qualified, and expensive. Each specialist is able to solve some, but not all events. Some members of this workforce tend to specialise in certain tasks and/or acquire greater experience with certain clients. These members should then be preferentially selected to resolve appropriate events.

* Corresponding autor.

This paper is focused on the second level of the SC (SC-Level2). Once an SC event enters the SC-Level2, it gives place to one or several tasks to be performed by the SC-Level2 workforce. In the case of several tasks maximal and minimal generalized precedence relationships (GPR) between the tasks that reflect technological constraints might exist. If this is the case we say that the event gives place to a multi-task. If not we simply say that the event gives place to several tasks. A standard average service time is associated with each task. Tasks and multi-tasks are associated with a client-company service level agreement (SLA) that establishes maximum response times, taken into account from the time the events that give place to them enter the SC, for the beginning and the end of tasks with penalties for delays.

When an event occurs at a given time it gives place to a task (multi-task). If the event occurs again, it gives place to another task (multi-task), which is a copy of the previous task (multi-task). Tasks (multi-tasks) originated by the occurrence of the same event have the same characteristics and we will say that they belong to the same type of task (multi-task).

The service level agreed for each client, project, and task (multi-task) is a fundamental aspect of the operation of an SC. The service level agreements are negotiated during the contractual phase, before the SC services are provided. In practice, the maximum response times are established for categories of events not for each task (multi-task) individually. A task (multi-task) inherits its maximum response times from the category to which the event that gives place to it belongs to.

The SC-Level2 workforce is made up of specialists, with specialisations in one or more areas of knowledge. A knowledge area describes a set of tasks a specialist with specialisation in this area can handle. For each area, the efficiency level of a specialist describes how well he or she can handle the tasks in that area and can be quantified by the average service time. Higher (lower) levels of efficiency indicate that the person is one of the most (least) suitable for the area and implies a reduction (increase) of the average service time. Pre-emption is not allowed. To process a task just one specialist specialised in the area to which the task belongs is required. To ease the notation, we will consider that all workers (specialists) are available at any time instant. We will also say that the workers that can handle the same types of tasks (multi-task) with the same efficiency levels belong to the same type of worker.

In general terms, the main objective for an SC-Level2 scheduling system is to obtain, in real time, a feasible plan of action (a task schedule and an assignment of workers to tasks) which satisfies the technological constraints, which does not use more specialists than are available in each time period and which satisfy the service level agreements. Plans are made or remade at certain points in time. During execution, however, continuously new events are incorporated into the system and resolved events are dropped from the system. The events arrive at uncertain times and their service times are also uncertain. Due to the full recording capability of SCMT's, we can realistically assume that some advance knowledge about the probability distributions of both types of times is available. In this type of rapidly changing environment, planning becomes a continuous online process. Therefore, the system can be qualified as dynamic, stochastic, online, resource constrained and multimode.

This paper proposes the use of project scheduling technology in the design of a system management model and the corresponding solution methodology to deal, in real time, with the SC-Level2 allocation and scheduling problem.

The management model we propose in this paper can be described as follows: The system maintains a workable baseline schedule at any time. At a given time t, a deterministic and static algorithm (a *predictive procedure*) generates a *predictive schedule S*(*t*), taking into consideration the unfinished tasks already in the system and assuming the average service times as their deterministic durations. During execution, tasks may take longer or shorter than initially expected, causing deviations between the actually realized and planned completion times. Each time such a deviation occurs, a fast *reactive procedure* is applied to repair the predictive schedule while maintaining resource allocation. The problem is considered static during *react* units of time: tasks arriving between times *t* and *t*+*react* are ignored until time *t*+*react* when the predictive procedure is called again. The predictive procedure maintains the current resource allocation only for the tasks in process at time *t*+*react* and not necessarily for the tasks previously scheduled but not started yet. This concatenation of procedures is periodically applied at subsequent times with time intervals of length react (rescheduling times). The actual schedule that is obtained after these modifications is called the *realized schedule.*

It is interesting to note that the problem of testing whether there is a feasible schedule for a given instance of PS|temp|Cmax is NP-complete due to the existence of GPRs with minimal and maximal time-lags between the tasks [27]. For this reason, we have not considered the use of scheduling policies that would generate a schedule by deciding at certain decision points the starting time of a set of tasks not scheduled yet ([25] and [26]). The tasks of a multi-task are linked by GPRs therefore scheduling these tasks one by one would lead almost certainly to infeasibilities.

The rest of the paper is organised as follows: In Section 2 we will formally introduce the project scheduling approach. Section 3 briefly reviews the main literature on the project scheduling topics related to the models dealt with in this paper. In Section 4 we will describe the predictive-reactive procedure. Section 5 describes the instance generator and the selection of test instance sets. The computational results and configuration selection are shown in Section 6. The final section is reserved for conclusions and future work.

2. A project scheduling approach

The static and deterministic problem

At each re-scheduling time, the predictive procedure generates a predictive schedule by solving a static and deterministic project-scheduling problem that can be described as follows:

The project consists of a set T of tasks and a set MT of multi-tasks, where each task has to be processed without pre-emption to complete the project. A multi-task is composed of a set of tasks linked by *GPR*s. If T_{mt} denotes the set of tasks that belongs to a multi-task then $T_{mt} \subseteq T$. The dummy tasks 1 and n represent the beginning and end of the project.

The specialists (workers) are the resources required to process the tasks. Each task j has associated a standard average service time (duration), d_j, and should be processed by just one worker that can handle the task. For each type of task j, the efficiency level of a specialist i describes how well he or she can handle the tasks that belong to that type of task and can be quantified by the average duration. The efficiency level is represented as an integer value, e_{ji}. In this paper, we will consider six efficiency values: e_{ji} = -1, 1, 2, 3, 4 and 5. The value e_{ji} = -1 means that specialist i cannot handle the tasks of type j. The value e_{ji} = 1 (2) means that the average duration, d_{ji}, of a task of type j when processed by specialist i decreases an α % (β %) with respect to the standard average duration d_j. The value $e_{ji} = 4$ (5) means that the average duration d_{ji} increases a γ% (δ %) with respect to the standard average duration d_j. The value $e_{ji} = 3$ means that the average duration d_{ji} is equal to the standard average duration d_j. The number of different efficiency values and the associated increase/reduction percentages values will depend on the actual application. In this paper, we will consider α = δ = 25 and β = γ =13. We will also say that the workers that can handle the same types of tasks with the same efficiency levels belong to the same type of worker. TW will denote the set of types of workers.

The tasks in the system at any re-scheduling time t can be classified into two groups: the tasks already in the system but not started yet (T_{ns}) and the tasks in process (T_{ip}). The tasks in T_{ns} are included in T. If task $j \in T_{ip}$ it means that a worker i is processing j. Then, j defines a task j' in T for which both the starting time $s_{j'} = t$ and the assignment of worker i are fixed and which average duration d'_{ji} is calculated as follows: $d'_{ji} = s_j + d_{ji} - t$ if $t < s_j + d_{ji}$ and $d'_{ji} = 1$ if $t > s_j + d_{ji}$ where s_j is the starting time of task j. The arrival time of a task (multi-task) j into the system will be denoted as $arr(j)$.

The client-company service level agreement establishes for each task (multi-task) j a maximum starting time, ms_j, and a maximum finishing time, mf_j. Both dates can be exceeded, however some costs are incurred. We define sc_j (fc_j) as the cost associated to the delay of one unit in the starting (finishing) time of task (multi-task) j.

The dummy tasks 1 and n require no resources, do not have associated any maximum starting and finishing dates and their durations are null. We will say that tasks that have the same characteristics belong to the same type of task. TT will denote the set of types of tasks.

Durations, costs, and maximum dates are assumed to be non-negative integers. If the calculated average duration is not an integer then we round it up to the nearest integer.

The model considers generalised precedence relationships (GPRs) between the tasks, i.e. minimal and maximal time lags between tasks starting times. A minimal (maximal) time lag indicates that a task cannot start earlier (later) than certain time units after the start of another task. Time lags are considered to be positive, negative or zero integers. Only minimal relationships will be considered in the model without any loss of generality, negative/positive maximal time lags will be replaced by equivalent positive/negative minimal time lags of opposite direction. Therefore, a GPR between tasks i and j will always be expressed as: $s_j - s_i \geq l_{ij}$. The tasks and the GPRs can be represented by a weighted directed graph $G = (V, E)$, where $V = \{1, 2,..., n\}$ is the set of tasks, E is the set of *GPR*s and the weight of an arc $(i,j) \in E$ is l_{ij}. For the sake of simplicity we have not explicitly considered the other three types of GPRs: start-finish, finish-start and finish-finish. However, it is easy to see that their consideration would not have changed the contents of the paper as it would have unnecessarily complicated its writing. Notice,

also, that in this setting it is not true that the four different time lag types can be converted into one another.

A schedule S is represented by two vectors $S = (s , w)$ where the vector $s = (s_1, s_2, ..., s_n)$ of non negative integers, indicates for each task j its starting time s_j and $w = (w(1), w(2), ..., w(n))$ assigns to each task i a worker $w(i)$ which can handle the task, $w(1) = w(n)$ is a dummy worker. The finishing time of a task j is calculated as $f_j = s_j + d_{jw(j)}$. Given the inherent complexity of the problem we are dealing with, the goal of the scheduling system is to obtain a feasible solution satisfying all the previous constraints. Nevertheless, a feasible solution may not exist for certain instances of the problem and even the problem of knowing if it exists a feasible solution is a NP-complete problem. Therefore, the solution methodology should be prepared to handle infeasible solutions. Nevertheless, the procedures we have developed deal with resource-feasible schedules which do not use more specialists than are available in each time period. Therefore, we will consider two kinds of possible infeasibilities: GPR infeasibility and TARDCOST infeasibility.

Given a schedule S, the following function measures the first type of infeasibility: $GPR(S) = \sum_{(i,j)\in E} viol(i,j)$ where $viol(i,j) = \max\{0, s_i - s_j + l_{ij}\}$. Obviously, GPR($S$) = 0 iff the schedule S satisfies all GPRs. The second type of infeasibility is measured by the following function:

$$\text{TARDCOST}(S) = \sum_{j\in T/T_{mt}} \max\{0, s_j - arr(j) - ms_j\} * sc_j + \sum_{j\in T/T_{mt}} \max\{0, f_j - arr(j) - mf_j\} * fc_j$$
$$+ \sum_{j\in MT} \max\{0, s_j - arr(j) - ms_j\} * sc_j + \sum_{j\in MT} \max\{0, f_j - arr(j) - mf_j\} * fc_j$$

where: $s_j = \min\{s_i$: task i belongs to multitask $j\}$ and $f_j = \max\{f_i$: task i belongs to multitask $j\}$ for $j \in MT$. Obviously, TARDCOST(S) = 0 iff the schedule S satisfies all SLAs.

The objective of the static and deterministic scheduling problem is to find a resource-feasible schedule that minimizes the functions GPR(S) and TARDCOST(S) in lexicographical order. Therefore, we are dealing with a bi-objective scheduling problem where the evaluation of a sequence S is given by eval (S) = (GPR (S), TARDCOST (S)) and eval(S)<eval(S') if "GPR(S)<GPR(S')" or "GPR(S) = GPR(S') and TARDCOST(S) <TARDCOST (S')".

The dynamic and stochastic problem

At every point in time t, the predictive schedule $S(t)$ predicts how the scheduling system expects the tasks to be processed given the information available at that time. During execution, tasks may take longer or shorter than initially expected causing deviations between the actually realized and planned starting, and finishing, times. Also, continuously new events are incorporated into the system and resolved events are dropped from the system. When new information becomes available the predictive schedule is repaired (reactive scheduling between re-scheduling times) or partially recomputed (predictive scheduling at rescheduling times). Therefore, the predictive schedule evolves with time, since the beginning of the scheduling effort at time $t = 0$. The actual realization of the scheduling of a task j is only fully known at its completion. Since this time onwards the staring time and duration of task j are fixed in the predictive schedule. Gradually, the predictive schedule becomes the realized schedule. The realized schedule at time t, $S^*(t)$, is the sub-schedule of $S(t)$ relative to the completed tasks. The goal of the scheduling system is to obtain a feasible realized schedule at every time t. In this paper, we present a predictive-reactive procedure where elements can be implemented in different ways giving place to different versions of the procedure. The merits of these versions will be evaluated using simulation as will be apparent in Section 6.

3. Literature review

In this section we briefly review the main literature on the classical resource constraint project scheduling problem (RCPSP) and some of its generalizations that are related with the models this paper deals with.

3.1 RCPSP and some generalizations

Research efforts in RCPSP have been extensive and numerous generalizations of it have been described. An extensive study of the problem RCPSP can be found in [13]. One of the most important generalizations, named RCPSP/max, is to consider generalized precedence relations of minimum and maximum types (GPR), see [27]. In the project-scheduling field it has been considered the problem of delivery dates assigned to each task. One of the first papers was [36], in which resources are not considered and penalties for late or early finishing are given. In [37] the same problem with resource constraints is considered. [31] studies the RCPSP with due dates.

Another modality is the Multimode RCPSP (MRCPSP) where each task can be processed using different combinations of resources; different processing modes may result in different task durations. Papers related to this problem are [10], [18], [19] and [20]. We model the static SC-Level2 problem as a MRCPSP considering GPRs and delivery dates, as we will see in the next subsections.
There are models that consider dynamic and stochastic variations of the project-scheduling problem. Artigues and Roubellat, [3], studied the problem of, given a feasible schedule for MRCPSP with due and delivery dates, inserting a new task minimizing the maximum delay. The Dynamic RCPSP is solved (if possible) in [14] by using "conflict-based repair" techniques. The problem of dynamic sequencing is widely studied in the context of real-time systems; Manimaran, [23] and [24], show the design of architectures and algorithms to sequence the tasks efficiently in real time.
There are several approaches to the project scheduling problem where actual activity durations may be different from expected. One approach considers that durations are known before scheduling although they can be modified by perturbations during the realization of the schedule. One of the most popular procedures within this approach is the one proposed by Goldratt [15] but it has been widely discussed (see [22]). Van den Vorder, [33] and [34], proposes alternative methods to Goldratt and considers the trade-off between the project duration and stability.
Another modality considers time durations defined as probability distributions, it is called stochastic RCPSP (SRCPSP). Scheduling policies is a methodology widely used in SRCPSP; a scheduling policy Π proposes "actions" at each decision time point. With "action" it is understood the assignment of a starting time for a given activity. Decision time points are those time points at which activities finish and the initial time point. A full characterization of scheduling policies can be found in [26] and [27]. We do not consider that scheduling policies are a viable option for the SC-Level2 problem due to the existence of GPRs.
Research in dynamic environments that consider the possibility of perturbations in task durations is scarce. Alvarez and Diaz, [2], examined the case of task sequencing in a factory where the lengths of durations can vary, new tasks can arrive, and there are breakdowns. An analysis of combinatorial stochastic online problems where special emphasis is given to the dynamic stochastic scheduling problem can be found in [35]. None of the two papers considers generalized precedence relations. Two recommended references about scheduling with uncertainty are [21] and [8].

3.2 Service Centre Problem
The static and deterministic problem (see section 2) has been formulated as a multiobjective project scheduling problem. The books by Bagchi [4] and T'Kindt and Billaut [30] may be considered as reference books for researchers in the field of multi-criteria scheduling problems. Another interesting work is [6] which considers the problem of generating schedules for a multi-skilled workforce; Viana and Sousa [38] applied multi-objective versions of the simulated annealing and tabu search in RCPSP. The insertion of new tasks within a given sequence is a recent problem in project scheduling, see [3]. However, given that the fitness function in the SC problem is different, we opted to use traditional (and low computational) methods of insertion, the insertion can be improved by means of local search procedures afterwards. The static problem considers expected durations. The static SC problem can be found in the literature, without multitasking, in [32].

4. Predictive-reactive algorithms

In this section we present the main algorithmic elements of the scheduling system we propose. At every point in time t, a schedule $S(t)$ is maintained. At every re-scheduling time t, a predictive procedure is applied to $S(t)$. The predictive procedure performs two main actions. First, an *Insertion algorithm* inserts the tasks arrived to the system since the last re-scheduling time into $S(t)$. Second, *Local Search algorithms* are applied to the enlarged $S(t)$ with the aim of eliminating, or reducing infeasibilities, giving place to the predictive schedule $S(t)$. During execution, tasks may take longer or shorter than initially expected causing deviations between the actually realized and planned starting, and finishing, times. Each time this happens, a *Reactive algorithm* repairs the schedule $S(t)$. The algorithms applied in the predictive phase take into consideration the *workload* assigned to the workers by the current $S(t)$ to make decisions.

4.1 Workloads

Given a re-scheduling time t, the current $S(t)$, the set T of tasks in the system at time t, a worker i and a worker type tw, we define:

$$workload(i) = \sum_{j \in ASSIGN(i)} \left\lfloor f_j - \max\left(t, s_j\right) \right\rfloor \text{ where } ASSIGN(i) = \left\{ j \in T \middle| \, w(j) = i \right\}, \text{ and}$$

$$workload(tw) = \frac{\sum_{i \in tw} workload(i)}{(f_T - t) * |tw|}$$

where $f_T = \max\{f_j : j \in T\}$ and $|tw|$ denotes the number of workers of type tw available. Notice that $0 \leq workload(tw) \leq 1$.

4.2 Insertion algorithm

At every re-scheduling time t, the Insertion algorithm extends the current $S(t)$ by scheduling the tasks arrived to the system since the last re-scheduling time. The Insertion algorithm schedules the tasks which have arrived but not yet scheduled, in increasing order of maximum starting time. It works as follows:

Let j denotes the next task to be scheduled and *W(j)* the set of workers that can handle task j. For a given worker i we define $ltask(i)$ as the task $j' \in ASSIGN(i)$ such that $f_{j'} = \max\{f_h : h \in ASSIGN(i)\}$. The Insertion algorithm selects a worker $i \in W(j)$ and schedules task j starting at $t_i = \max\{t, f_{ltask(i)}\}$. Different selection rules provide different versions of the algorithm. For example, select the worker i such that $f_{ltask(i)}$ is minimum (version *IEST*) or select the worker i such that $t_i + d_{j,i}$ is minimum (version *IEFT*). In both cases, the minimum worker workload is used as a time breaking rule.

IEST and *IEFT* are very fast versions of the Insertion algorithm that do not take into account the GPRs present in the multi-tasks, a fact which could lead to great infeasibilities which could be difficult to repair. It seems, therefore, convenient to consider more elaborate versions of the Insertion algorithm, which are more time consuming but still fast and that schedule the tasks of a multitask as a whole. We have used the hybrid genetic algorithm, *GA*, described in [32] to generate a schedule for a given multitask considered as a project in itself. Times t_i are considered as availability dates in the workers calendars. Therefore, we have considered two other versions of the Insertion algorithm, *IESTGA* and *IEFTGA*, which schedule the tasks in a multi-task using *GA* and schedule the tasks using *IEST* and *IEFT*, respectively.

4.3 Generating S(0)

To generate the initial schedule $S(0)$ we can use any of the versions of the Insertion algorithm to schedule the tasks already in the system at time $t = 0$ in increasing order of the maximum finishing time. This gives four methods to generate the initial schedule: *InEST*, *InFST*, *InESTGA* and *InEFTGA*.

4.4 Local Search algorithms

The four Local Search (*LS*) algorithms we have developed aim at reducing the infeasibility of $S(t)$ at every re-scheduling time t. A pair of these LS algorithms aim at decreasing the value of GPR($S(t)$) and the other pair, the value of TARDCOST($S(t)$). In each pair, an algorithm is based on the Interchange operator and the other, on the Insertion operator. Given a worker i, $LIST(i)$ will denote the set $ASSIGN(i)$ with the tasks ordered in increasing order of the starting time. The Interchange operator consists of selecting two tasks assigned to two different workers and exchanging their positions in the ordered lists of tasks assigned to the workers. The Insertion operator selects a task assigned to a worker and inserts it into the ordered list of tasks assigned to another worker. It is not difficult to see why we have called them *IntGPR*, *InsGPR*, *IntTARD* and *InsTARD*, respectively.

The four *LS* algorithms share a similar but not identical structure and make use of the upper limits $maxtasks = maxtasks' = perc*|T|$ and $maxworkers = perc*|workers|$, where $|workers|$ denotes the number of workers available and the rational value $perc \in [0,1]$ is an input parameter.

4.4.1 Algorithm IntGPR

Given a task j, we define the contribution of task j to GPR($S(t)$) as $\mathrm{GPR(j)} = \sum_{(i,j)\in E} \mathrm{viol(i,j)} + \sum_{(j,i)\in E} \mathrm{viol(j,i)}$. The algorithm *IntGPR* seeks to decrease $GPR(S(t))$ by re-scheduling tasks j for which $GPR(j) > 0$. In this context, re-scheduling a task j means to find an appropriate task j' with $w(j') \neq w(j)$, interchange the workers assigned to them and re-define the starting times of tasks j and j' appropriately. Next, we present an outline of the algorithm.

Algorithm IntGPR

1. Make $nw = nt = R = 0$
2. Generate the list $TASKLIST = \{j \in T: \text{GPR}(j) > 0\}$ in decreasing order of GPR(j)
3. While ($nt < maxtasks$ and $TASKLIST \neq \emptyset$):
    3.1 Select the first task j in $TASKLIST$
    3.2 Make $nt = nt + 1$
    3.3 Obtain $R = INTERCHANGE(j)$
    3.4 Eliminate task j from $TASKLIST$
4. If ($R = 0$) then STOP
5. Go to 1

INTERCHANGE(j)

1. Determine the searching interval $[a, b]$
2. If ($a = b$) then go to 5
3. Generate the list $WORKERLIST = \{i: i \in W(j) \text{ and } i \neq w(j)\}$ in increasing order of $workload(i)$
4. While ($nw < maxworkers$ and $WORKERLIST \neq \emptyset$):
    4.1 Select the first worker i in $WORKERLIST$
    4.2 Make $nw = nw + 1$
    4.3 Generate the set $SET = \{j' \in ASSIGN(i): s_{j'} \in [a, b]\}$
    4.4 Order SET either in increasing order of starting time if $b = s_j - 1$ or in decreasing order of starting time if $a = s_j + 1$
    4.5 If ($j \in T_{mt}$) then $SET = SET \setminus \{j': j \text{ and } j' \text{ belongs to the same multitask}\}$
    4.6 While ($SET \neq \emptyset$):
        4.6.1 Select the first task $j' \in SET$
        4.6.2 Construct $S'(t)$ from $S(t)$ by interchanging tasks j and j'
        4.6.3 If ($GPR(S'(t)) < GPR(S(t))$) then $R = 1$, $S(t) = S'(t)$ and go to 5
        4.6.4 Make $SET = SET \setminus \{j'\}$
5. Return R

To fully describe algorithm *IntGPR* we need to explain how the searching interval is constructed and how tasks j and j' are interchanged. Given a task j the searching interval for j is a time interval $[a, b]$ such that if we only change the starting time of task j in such a way that the new value s'_j belongs to this interval then GPR(j) decreases with certainty. The searching interval for j is constructed in the following way.

On what follows we restrict our attention to the GPRs between j and another task and vice versa. These GPRs can be classified as violated (they contribute to GPR(j)) or non-violated (their contribution to GPR(j) is null).

For a given non violated GPR there is a time instant t' of maximal (minimal) value such that if the starting time of only task j is changed to a new value $s'_j \leq t'$ ($s'_j \geq t'$) then the non-violated GPR remains non-violated. Let $b1$ ($a1$) be the minimum (maximum) of these values over all non-violated GPRs. Notice that $b1 \geq s_j$ and $a1 \leq s_j$.

For certain violated GPRs there is a time instant t' of minimal (maximal) value such that if the starting time of only task j is changed to a new value $s'_j \geq t'$ ($s_j \leq t'$) then the violated GPR becomes non-violated. Let $b2$ ($a2$) be the minimum (maximum) of these values over those violated GPRs. We can compute the number $ninc$ ($ndec$) of violated GPRs that would not be violated any longer if s_j would be increased (decreased) as much as necessary. Notice that $b2 \geq s_j$ and $a2 \leq s_j$ and that $ninc + ndec$ is the number of violated GPRs.

Then, we define:

- $a = s_j + 1$ and $b = \min\{b1, b2\}$ if $ninc > ndec$
- $a = \max\{a1, a2, t\}$ and $b = s_j - 1$ if $ninc < ndec$
- $a = b = s_j$ if $ninc = ndec$

Notice that $ninc > ndec$ ($ninc < ndec$) implies that we only consider the possibility of delaying (advancing) task j.

Let us focus now on how to construct the schedule $S'(t)$ generated by interchanging tasks j and j'. Given a task k we define $beforek$ ($afterk$) as the task placed before (after) k in the list $LIST(w(k))$ if task k is not the first (last) task in the list. To construct $S'(t)$ the worker assignments and the starting times are maintained fixed for all tasks other than j and j'. To determine the new schedule for j and j' we proceed as follows: We investigate if a minimal (maximal) time instant t' in the interval $[\max(a, f_{beforej'}), \min(b, s_{afterj'})]$ such that $t' + d_{jw(j')} \leq s_{afterj'}$ in the case $ninc < ndec$ ($ninc > ndec$) exists. We also investigate if there exists a minimal time instant t'' in the interval $[f_{beforej}, s_{afterj}]$ such that $t'' + d_{j'w(j)} \leq s_{afterj}$. If any of

these time instants do not exist, the interchange is not produced and $S'(t) = S(t)$. In other case, we define $s'(j) = t'$, $w'(j) = w(j')$, $s'(j') = t''$ and $w'(j') = w(j)$. For the case in which j is the first (last) task in the list we define $f_{beforej} = t$ ($s_{afterj} = \infty$). Similarly for task j'.

4.4.2 Algorithm InsGPR

The algorithm *InsGPR* seeks to decrease $GPR(S(t))$ by re-scheduling tasks j for which $GPR(j) > 0$. In this context, re-scheduling a task j means to find an appropriate worker $i \neq w(j)$, assign task j to worker i and re-define the starting time of task j. The general outline of algorithm *InsGPR* is the same as that of algorithm *IntGPR* except that line 3.3 should now read: Obtain $R = INSERT(j)$. The outline of the procedure $INSERT(j)$ is the following:

INSERT(*j*)

1. Determine the searching interval $[a, b]$
2. If $(a = b)$ then go to 5
3. Generate $S'(t) = RightJustify(S(t),a)$
4. Generate the list $WORKERLIST = \{i: i \in W(j) \text{ and } i \neq w(j)\}$ in increasing order of $workload(i)$
5. While ($nw < maxworkers$ and $WORKERLIST \neq \emptyset$):
   - 5.1 Select the first worker i in $WORKERLIST$
   - 5.2 Make $nw = nw + 1$
   - 5.3 Construct $S''(t)$ from $S'(t)$ by inserting task j in i
   - 5.4 If $(GPR''(j) < GPR'(j))$ then $R = 1$, $S'(t) = S''(t)$ and go to 6
   - 5.5 Make $WORKERLIST = WORKERLIST \setminus \{i\}$
6. Generate $S(t) = LeftJustify(S'(t))$
7. Return R

where $GPR''(t)$ ($GPR'(t)$) denotes the value of $GPR(j)$ with respect to $S''(t)$ ($S'(t)$).

To fully describe algorithm $INSERT(j)$ we need to explain how the functions $RightJustify(S(t),a)$ and $LeftJustify(S'(t))$ operate and how task j is inserted in i.

The schedule $S'(t) = RightJustify(S(t),a)$ is generated from $S(t)$ by delaying as much as possible all tasks whose finishing times are greater or equal than a in decreasing order of their starting times while their contributions to GPR(S(t)) are not increased. The schedule $S(t) = LeftJustify(S'(t))$ is generated from $S'(t)$ by advancing as much as possible all tasks in increasing order of their starting times while their contributions to $GPR(S(t))$ are not increased.

Let us explain now the procedure to construct the schedule $S''(t)$ from $S'(t)$ by inserting task j in i. The schedule $S''(t)$ can differ from $S'(t)$ only in the worker assigned to j and the starting time of j. The procedure aims at scheduling task j between two consecutive tasks in $LIST(i)$ in such a way that $s''(t) \in [a, b]$. More specifically, we consider two cases: A) $ninc < ndec$ and B) $ninc > ndec$.

Case A. Let k be the first task assigned to i that starts at time a or later, if such a task exists. If k does not exist two cases are considered. If $d_{ji} \leq b - a$, then $s''(j) = a$ and $w''(j) = i$. If $d_{ji} > b - a$, we define $S''(t) = S'(t)$.

If k does exist two cases are considered. If $d_{ji} \leq s'_k - a$, then we define $s''(j) = a$ and $w''(j) = i$. If $d_{ji} > s'_k - a$, we re-define a as $a = f_k$ and repeat the procedure until $a > b$. At that instant, we define $S''(t) = S'(t)$.

Case B. The procedure in this case is similar to that of case A but now the interval $[a, b]$ is scanned from b to a.

4.4.3 Algorithm IntTARD

The algorithm *IntTARD* seeks to decrease $TARDCOST(S(t))$ without increasing $GPR(S(t))$ by performing interchanges of selected tasks. The general outline of algorithm *IntTARD* is the same as that of algorithm *IntGPR* except that now the set *TASKLIST* is generated differently. Now, $TASKLIST = \{j \in T: late(j) = s_j - ms_j > 0\}$ and two different orderings are considered: in decreasing value of $late(j)$ or in decreasing ms_j. These two orderings give place to two different versions of the algorithm: *IntTARD_late* and *IntTARD_ms*. The outline of the procedure $INTERCHANGE(j)$ is now the following:

INTERCHANGE(j)

1. Make $nt' = 0$
2. Generate the list *SET* formed by all tasks ordered in increasing order of $late(j)$ or ms_j as appropriated.
3. While ($nt' < maxtasks'$ and $SET \neq \emptyset$):
   3.1 Select the first task j' in *SET* such that $w(j') \neq w(j)$
   3.2 If ($j \in T_{mt}$) then $SET = SET \setminus \{j': j$ and j' belongs to the same multitask$\}$
   3.3 Make $nt' = nt' + 1$
   3.4 Construct $S'(t)$ from $S(t)$ by interchanging tasks j and j'
   3.5 If ($GPR(S'(t)) \leq GPR(S(t))$ and TARDCOST($S'(t)$) < TARDCOST($S(t)$) then $R = 1$, $S(t) = S'(t)$ and go to 4
   3.6 Make $SET = SET \setminus \{j'\}$
4. Return R

To fully describe the current algorithm *INTERCHANGE*(j) we need to explain how tasks j and j' are interchanged. To construct $S'(t)$ the worker assignments and the starting times remain fixed for all tasks other than j and j'. To determine the new schedule of j and j' we proceed as follows. We investigate if there exists a minimal time instant t' in the interval $[f_{beforej'}, s_{afterj'}]$ such that $t' + d_{jw(j')} \leq s_{afterj'}$. We also investigate if there exists a minimal time instant t'' in the interval $[f_{beforej}, s_{afterj}]$ such that $t'' + d_{j'w(j)} \leq s_{afterj}$. If any of these time instants do not exist, the interchange is not produced and $S'(t) = S(t)$. In other case, we define $s'(j) = t'$, $w'(j) = w(i)$, $s'(j') = t''$ and $w'(j') = w(j)$. For the case in which j is the first (last) task in the list we define $f_{beforej} = t$ ($s_{afterj} = \infty$). Similarly for task j'.

4.4.4 Algorithm InsTARD

The algorithm *InsTARD* seeks to decrease TARDCOST($S(t)$) without increasing GPR($S(t)$) by re-scheduling tasks j for which $delay(j) > 0$. In this context, re-scheduling a task j means to find an appropriate worker $i \neq w(j)$, assign task j to worker i and re-define the starting time of task j. The general outline of algorithm *InsTARD* is the same as that of algorithm *IntGPR* except that now the set *TASKLIST* is generated as in *IntTARD*. Accordingly, we consider two versions of the algorithm: *InsTARD_late* and *InsTARD_ms*. The outline of the procedure *INSERT*(j) is now the following:

INSERT(j)

1. Determine the inserting limit a
2. If ($a = s_j$) then go to 7
3. Generate $S'(t) = RightJustify(S(t),a)$
4. Generate the list *WORKERLIST* = $\{i: i \in W(j)$ and $i \neq w(j)\}$ in increasing order of workload(i)
5. While ($nw < maxworkers$ and $WORKERLIST \neq \emptyset$):
   5.1 Select the first worker i in *WORKERLIST*
   5.2 Make $nw = nw + 1$
   5.3 Construct $S''(t)$ from $S'(t)$ by inserting task j in i
   5.4 If (GPR''(j) < GPR'(j) and TARDCOST($S'(t)$) < TARDCOST($S(t)$) then $R = 1$, $S'(t) = S''(t)$ and go to 6
   5.5 Make $WORKERLIST = WORKERLIST \setminus \{i\}$
6. Generate $S(t) = LeftJustify(S'(t))$
7. Return R

To fully describe algorithm *INSERT*(j) we need to explain how to determine the inserting limit a and how task j is inserted in i. The inserting limit a is defined as the minimum time at which task j can start without increasing the value of GPR(j). To insert task j in i we define $b = s_j - d_{ji}$ and use the procedure of Case A in algorithm *InsGPR*.

4.5 Reactive procedures

Given that the scheduling system has to operate in real time and that the reactive procedure is usually applied very often, we have designed a low computational cost reactive procedure. It works as follows:

Each time a task j finishes at a time t^* different from the time f_j predicted by the predictive schedule a reactive algorithm is applied to repair the schedule $S(t^*)$. If task j finishes before predicted then the procedure advances the starting time of the next task in $LIST(w(j))$ as much as possible without

increasing GPR($S(t)$) and without exceeding t^*. If task j finishes after predicted but the starting time of the next task in $LIST(w(j))$ is greater or equal than f_j then the previous procedure is also applied.

If task j finishes after predicted and the starting time of the next task in $LIST(w(j))$ is less than f_j then we propose two alternative reactive procedures that maintain the worker assignments and that we call $R1$ and $R2$, respectively. Procedure $R1$ simply delays the tasks that follow task j in $LIST(w(j))$ with the objective of eliminating overlaps between tasks. $R1$ does not take into account possible variations of GPR($S(t)$)) or of TARDCOST($S(t)$).

On the contrary, $R2$ is a recursive procedure that repairs $S(t^*)$ by delaying some tasks and taking into account the GPRs. Before delaying any task, the contribution of each generalized precedence relationship to GPR($S(t^*)$) is computed and called the initial contribution of this GPR. The procedure $R2$ considers first task *afterj*. If task *afterj* has already been started then the procedure stops. If not, the starting time of *afterj* is delayed up to time t^*. If a task is delayed then it is immediately explored. To explore a task means to consider in turn all the GPRs with origin in this task and end in a non started task. If the current contribution of the considered GPR is greater than its initial contribution then the start of the final task of this GPR is delayed until the initial contribution is restored. This procedure is recursively applied to all delayed tasks. Once all successors of a delayed task k have been explored the procedure examines the task afterk. If $s_{afterk} < f_k$ then task afterk is delayed up to time f_k. $R2$ stops when all delayed tasks have been explored. The procedure $R2$ is finite because each time a final task of a GPR is delayed the contribution of this GPR decreases and also because the number of tasks is finite.

5. Instance Generator and test instance sets

In the previous section we have presented an algorithmic structure whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm. The instantiation of such an algorithmic structure requires to choose among a set of different possible components and to assign specific values to all free parameters. We will refer to such an instantiation as a *configuration*.

To find a good configuration researches typically configure their algorithms in an iterative process on the basis of a sufficiently large number of runs of different configurations that are felt as promising on a sufficiently large set of instances. Usually, such a process is heavily based on personal experience. However, to the best of our knowledge the SC problem has not been previously considered and, consequently, no benchmark instances are available in the literature. Therefore, in order to be able of analysing the performance of the algorithms proposed in this paper we need first to generate a set of test instances.

In order to allow a systematic evaluation of the performance of the algorithms, characteristics of the SC problem have to be identified. The characteristics can then serve as the input parameters for the systematic generation of instances. The variation of the levels of these problem parameters in a full factorial design study allows producing a set of well-balanced instances. In this section we describe an instance generator for the SC problem and propose several benchmark instance sets generated by using the proposed generator that will be used as test instances in the computational experiments dealt with in the next section. The methodology applied for finding a good configuration through statistically guided experimental evaluations and the results obtained in these experiments constitute the content of section 5.

5.1 Instance generator

In this sub-section we give a brief summary of the selected characteristics of SC instances, that is, the input parameters of the instance generator and present the parameter settings used for generating the benchmark instances. A detailed description of the parameters and their realization can be found in [17].

We group the input parameters into three classes: First, variable parameters associated to the SC problem characteristics that we consider most relevant, which levels are systematically varied; second, the secondary parameters whose actual levels have to be randomly selected in the ranges determined by the selected variable parameters levels, and, third, the fixed parameters which levels have to be selected in ranges fixed for all instances.

We next provide a short description of the variable parameters followed by the set of their possible values. The other parameters will be introduced when the description of the instance generation process requires it.

num_area _types = number of knowledge areas, {3, 5, 10}.
num_task _types = number of task types per knowledge area, {1, 5, 10}.
num_mtask _types = number of multitask types, {0, 1}.
max_mtask_size = maximum number of tasks in a multitask, {5, 10, 15}.

num_worker_types = number of worker types, {3, 5, 10}.
dur_arrival = determines the ranges in which the average activity durations and the inter-arrival times between tasks can be chosen, {1, 2, 3}.
cross_training = indicates the degree of worker cross training {0.25, 0.50, 0.75}.
restrictiveness = denotes the parameter "Thesen's estimator for the restrictiveness [29]", *RT*, used in ProGen/max, {0.25, 0.50, 0.75}.
redundancy = denotes the parameter "degree of redundancy", ρ, used in ProGen/max, {0.25, 0.50, 0.75}.
CST = denotes the parameter "cycle structure tightness", *CST*, used in ProGen/max, {0.25, 0.50, 0.75}.

We have generated two sets of test instances, *SET_MT* and *SET_NMT*. *SET_MT* contains one type of multitasks (*num_mtask _types* = 1) whereas *SET_NMT* does not contain any multitask (*num_mtask _types* = 0). Utilizing a full factorial design of the variable parameters other than *num_mtask _types* with 5 replications per cell we have generated a total of $3^9*5 = 19683*5=98415$ benchmark instances for set *SET_MT*. Utilizing a full factorial design of the variable parameters not related to the generation of multitasks with 5 replications per cell we have generated a total of $3^5*5 = 243*5=1215$ benchmark instances for set *SET_NMT*.

Given a setting of parameters, the generator proceeds in several steps for generating an SC instance. In what follows the expression '*name*:= *rand*[*a*, *b*]' means that the value of parameter *name* is generated as a random integer number in the interval [*a*, *b*].

Step 1. The number of task types included in a given knowledge area is generated as a random integer number in the interval [1, *num_task_types*].

Step 2. *Determination of a task type.*

Given a task type *i* the generator simulates that several tasks which are 'copies' of the task type arrive into the system at different time instants. It is assumed that the probability distribution of (type *i*) task arrivals is a Poisson distribution for which the mean number of task arrivals per unit of time is $1/interarr_i$. It is also assumed that the processing time of a (type *i*) task performed by a worker *j* follows an exponential distribution with mean d_{ji}, calculated from the standard average duration d_i and the efficiency level e_{ji}.

If *dur_arrival* = 1, then d_i := *rand*[10, 15] and $interarr_i$:= *rand*[1, 5]

If *dur_arrival* = 2, then d_i := *rand*[5, 10] and $interarr_i$:= *rand*[5, 10]

If *dur_arrival* = 3, then d_i := *rand*[1, 5] and $interarr_i$:= *rand*[10, 15]

In all cases: sc_i := rand[1, 5], fc_i := rand[1, 5], mf_i := rand[$d_i+d_i*1.6$, $d_i+ d_i*1.6^2$], ms_i := rand[$d_i /2$, $mf_i - d_i$].

Step 3. *Determination of a multitask type*

Given a multitask *i* the number of tasks in the multitask, *num_tasks*, is such that *num_tasks* := rand[5, *max_mtask_size*]. Each task is randomly assigned to a type of task and inherits all the parameters associated to it.

We generate the network structure of the multitask type by applying the Algorithm 2 (Direct method) of Progen\max in [28]. The number of sources and sinks are random integer numbers in the interval [1, 0.2**num_tasks*]. The rest of the parameter settings are the following: ξ = 0.5; slack factor (*SF*) is set to 0.5; the number of cycles is *rand*[max{1, 0.25 *num_tasks* },min{5, 0.5 *num_tasks* }]; MTL^{min} = 0.2; MTL^{max} = 0.5; n_c^{min} = 0.2 *num_tasks*; and n_c^{max} = 0.5 *num_tasks*.

Also: sc_i := rand[1, 5], fc_i := rand[1, 5], mf_i := rand[*LPL*+*LPL**1.6, *LPL*+ *LPL**1.6^2], ms_i := rand[*LPL/2*, mf_i – *LPL*], where *LPL* denotes the length of a longest path in the multitask network.

Step 4. Assignment of worker types to knowledge areas

Let us suppose that *num_worker_types* = m_1 and *num_area _types* = m_2. We consider three cases:

1) $m_1 = m_2$.

We randomly assign the worker types to the knowledge areas.

2) $m_1 < m_2$.

We randomly assign the m_1 worker types to m_1 knowledge areas. If m_2-$m_1 \le m_1$, then we randomly assign the m_2-m_1 non assigned areas to m_2-m_1 worker types. If m_2-$m_1 \ge m_1$, then we randomly assign the m_1 worker types to m_1 non assigned areas. We repeat the procedure until the set of non assigned areas is empty.

3) $m_1 > m_2$.

We proceed as in case 2 by interchanging the roles of the worker types and knowledge areas.

At this point in time, we have selected a set of worker-type/knowledge-area assignments in which each worker type is assigned to at least a knowledge area and each knowledge area has at least a worker

type assigned to it. Now we try to enlarge this set according to the level of the parameter *cross_training*. We consider all non selected worker-type/knowledge-area assignments one by one. Given a such assignment we calculate $r := [0, 1]$. If $r <$ *cross_training*, then the assignment is added to the set. If not, it is discarded. For each assignment of a worker type i to a knowledge area j we calculate $e_{ji} := rand[1, 5]$.

Step 5. Arrival of tasks and multitasks to the system

We have to specify the tasks and multitasks that arrive to the system. For each (multitask) task of type i we follow the procedure described in Figure 1 to generate the (multitasks) tasks of type i that arrive to the system.

We have considered a time horizon of 200 units. We have also considered that the first type *i* (multitask) task arrives to the system at time $-mf_i + d_i$ to avoid the situation in which there are no tasks in the system during the first periods of the planning horizon. T_i denotes the set of (multitasks) tasks of type *i* that have arrived to the system.

5.2 Test instance sets.

The cardinality of set SET_MT is so huge (98415) that makes computationally impracticable to use it as a basis for choosing the best configuration of the procedure proposed in this paper. Therefore, we have devised a strategy to reduce the number of test instances, which we think it reflects a compromise between computational effort on one hand and diversity and problem hardiness on the other hand.

A configuration is defined by specifying: the insertion algorithm, the method to generate the initial solution, the reactive procedure, the local search procedure, the value of *react* and the value of *perc*. Among all possible configurations we have selected the configuration SR = (IEST, InEST, R2, IntTARD&InsTARD, 5, 0.30) and applied the algorithm defined by it to all instances in SET_MT using a personal computer Pentium Dual Core 3.20Ghz. We have also considered a subset of the variable parameters: PARAM = {*num_area_types*, *num_task_types, max_mtask_size*, *num_worker_types*, *dur_arrival, cross_training*}. Then, we have defined the following four test sets:

GPR_A (TARD_A). For each combination of the possible levels of the parameters in PARAM we consider the subset of instances from SET_MT characterized by this combination. The instance in this subset for which the value of GPR(S_{final}) (TARDCOST(S_{final})) obtained with SR is maximum is selected to be in GPR_A (TARD_A). It makes a total of 729 instances in GPR_A (TARD_A).

GPR_B (TARD_B). For each combination of the possible levels of the parameters in PARAM we consider the subset of instances from SET_MT characterized by this combination and by a SR computation time of less than 10 seconds. The instance in this subset for which the value of GPR(S_{final}) (TARDCOST(S_{final})) obtained with SR is maximum is selected to be in GPR_B (TARD_B). As none of those subsets are empty there are a total of 729 instances in GPR_B (TARD_B).

6. Computational Results.

In this section we show the computational analysis developed with the aim of choosing the optimal predictive-reactive configuration among the set of all possible configurations CONF. Before starting computational studies to compare configurations, statistical analyses were necessary in order to identify the correct way to develop computational experiments with a minimum cost.

First we studied which types of statistical tests were more appropriate, parametric or non-parametric tests. We randomly selected 10 instances from the set GPR_B. We performed 1000 runs of SR on each instance obtaining 10 sets of GPR(S_{final}) values and 10 sets of TARDCOST(S_{final}) values. The Kolmogorov-Smirnov test, [9], has not rejected the hypothesis that the GPR(S_{final}) values are normally distributed, however it has rejected this hypothesis for the TARDCOST(S_{final}) values. Non-parametric tests are therefore a better option.

Secondly we checked how many runs per configuration/instance were needed. The strategy of performing only one run per configuration is supported by the following experiment. We performed 10 runs of SR on the instance set GPR_B obtaining 10 sets of GPR(S_{final}) values. The non-parametric Friedman test, [7], did not detect differences between runs.

After those preliminary experiments we carried out the configuration selection in two phases. At Phase 1 we defined some experiments to reduce |CONF|. In Phase 2 we used a heuristic method known as FRace to remove those less promising configurations. In Phases 1 and 2, GPR_A, GPR_B, TARD_A and TARD_B were used. Finally we compared the results of the final selected configuration with those obtained with SR in the entire set of instances SET_MT and SET_NMT.

6.1 PHASE 1: First Selection.

Using configuration SR in GPR_A, GPR_B. TARD_A and TARD_B, we conducted experiments to discard less promising configurations:

Exp1: Varying *react* values in {1,..., 10} computational results show that for GPR(S_{final}), the extreme values {1,2,8,9,10} result in worse computational results. Therefore we restricted the settings values *react* to {3,...,7}. Considering the computational cost, small values of *react* generate a total greater computational cost, but a lower average cost every time local search and insertion algorithms are used.

Exp2: Comparing the two reactive policies, computational results show that R2 is significantly better than R1. We discard R1.

Exp3: When considering Insertion Algorithms, it is needed to consider genetic algorithms to insert multi-tasks in the system. Consequently we discard IEFT and IEST.

Exp4: With regard to the algorithms to generate initial solution, algorithms not using Genetic Algorithms to insert multi-tasks are discarded again.

6.2 PHASE 2: FRace.

The process to determine the best configuration is one of the most important parts in the implementation of metaheuristic algorithms, but it is also the part with the highest computational cost. To reduce the computational cost, several methods have been developed such as heuristic CALIBRA, [1], FRace, [5], and A-B-Domain, [16]. In this work, and following the results from the preliminary statistical analysis we should use non-parametric tests to compare between configurations. Therefore, we have decided to apply the FRace methodology for choosing between the set of configurations CONF.

5.2.1 FRace

The FRace is a heuristic method that allows us to discard configurations as soon as there is statistical evidence against them. The process starts with an initial set of configurations, CONF and a set of instances, INST. In the first step all configurations solve a set of randomly selected instances INST_S$\subset$INST. Se define INSTd=$\varnothing$.

FRace uses a block design (see [11]) where the values obtained for each instance in INST_S$\cup$INSTd are considered independent. Each block corresponds to the computational results for each instance on each configuration. In each block i (related to each instance) quantities eval(S_{final}) are sorted from lowest to highest; the position in the list is the rank, and this rank is assigned to the configuration related. In case of equally evaluated configurations they are positioned consecutively in the list at random. These configurations are assigned the same rank: the average of their positions. If the test rejects the null hypothesis, it is considered that there are differences between configurations. In this case *confb* denotes the configuration with the best rank value and pairwise comparisons are computed for every individual $a \in$ CONF against *confb*, if a comparison is significant then *a* is discarded. At the end of the iteration INSTd=INSTd$\cup$INST_S, INST=INST\INST_S. The process is repeated until only one configuration remains or until INST=$\varnothing$.

Given the stochastic and heuristic nature of FRace we have considered it necessary to design a procedure that provides greater confidence in the discarding decision. We have designed a FRace-based process named Discard-B-A that works in two steps as follows:

STEP1: The first step considers GPR_B as the set of instances, and CONF as the set of configurations. At each FRace iteration, 25 instances are randomly selected from GPR_B. A maximum number of 300 instances are used. This process is run 5 times. We found that the original pairwise test to discriminate between two configurations was too restrictive. Therefore we defined the following pairwise comparison test, Racepor (see Figure 2). The idea is to check if A is better *pperc* % of times than B, where A is *confb*. The use of *pperc* allows us to vary the degree of discrimination between configurations. For the experiments we used *pperc* = 0.6.

Discard-B-A runs FRace over GPR_B five times. CONFfi denotes the set of configurations not discarded for the experiment *i*, where *i* = 1,...,5. We compute CONFf = CONFf1$\cap\ldots\cap$CONFf5.

STEP 2: The set of configurations is CONFf and the set of instances is GPR_A. At each FRace iteration, 25 instances are randomly selected from GPR_A. Due to the increase in the computational cost, a maximum of 200 instances are used.

FRace is run 5 times thus obtaining 5 final sets whose intersection defines the final set of the step, named CONFF.

5.2.2 The use of Discard-B-A

The set of configurations CONF includes all possible configurations of valid combinations of the following sets: (IEFTGA, IESTGA) (insertion algorithms) (InEFTGA, InESTGA) (initial solution), (-,

use of IntGPR) (-, use of InsGPR), (-, use of IntTARD), (-, use of InsTARD) (*react* = 3,5,7) (*perc* = 30,60,90) and (*late*, *ms*) selection function in TARDCOST() local search methods. |CONF|=1008 because (*late*, *ms*) option is only relevant if TARDCOST() local search methods are used.
At STEP 1 of Discard-B-A(GPR_A,GPR_B), 753 configurations were discarded. At the end of STEP 2, |CONFF|=24. The analysis of this set allowed us to draw the following conclusions: (1) all configurations in CONFF use *perc* = 0.9, IntTARD, InsTARD and InsGPR; (2) (*late*) is the best option for TARDCOST() local search algorithms; (3) *react* has been chosen as the smallest possible value 3. From |CONFF| we selected configurations A1 and A2 because they obtained the best median and mean values (over all the instances used on all FRace experiments), respectively.
Discard-B-A was used also over (TARD_B, TARD_A); from this experiment A3 and A4 configurations were selected (using similar selection).

Comparison between A1, A2, A3 y A4 and SR.
Table I shows the description of the four selected configurations and the base configuration SR. *fts* denotes the function task selection in the TARDCOST() local search algorithms, where NA denotes that those algorithms are not used.

Tables II, III and IV, compare the five configurations with respect to GPR(), TARDCOST() and *cpu_pred_med*, respectively; *cpu_pred_med* denotes the average computational cost of insertions and local search procedures every *react* units of time.
Results regarding GPR (S_{final}) confirm that configurations A1 and A2 obtain the best results for all sets of instances except TARD_B as far as the average is concerned. Besides, configurations A3 and A4 obtain the best results in TARD_B. These results suggest that it is desirable to increase the value of *pperc* in Discard-B-A(TARD_B,TARD_A) Phase 1, in order to not discard promising configurations.
If we consider the total average, configurations A1 and A2 get the best results, though A2 is slightly higher, although rank tests showed no significant differences.
Results regarding TARDCOST (S_{final}) show that A1 and A2 get the best results in all sets of instances; we can observe clearly the effect of local search algorithms TARDCOST(). A1 is slightly better, but again, rank tests show no significant differences.
If we analyze the computational cost on the variable cpu_pred_med we see that all costs are similar although A3 and A4 require slightly smaller computational cost.
All configurations improve SR configuration, this was expected because they incorporate more local search algorithms, and also because the configuration SR was discarded in all FRace experiments.
Due to the necessity of choosing one configuration we selected A2 because it gets slightly better results on GPR(S_{final}), which is the main objective function, and it also has the lowest value of cpu_med_rev.

Final Experiment.
The purpose of this final experiment is to validate the results of A2 against SR over the sets of instances SET_T and SET_NMT.
For each one of the evaluation functions we show the results for each parameter generator. The results are shown in the tables V, VI and VII, where *dec*=(100 * (eval (SR)-eval (A2)) / eval (SR)); eval can be GPR() or TARDCOST().
That A2 has improved the results of SR is not remarkable because the algorithms considered in A2 perform a more thorough search. But it is interesting to highlight the parameters for which there has been a greater improvement.
First we analyse the results over SET_T. Regarding GPR () we can see that for small instances (considering *num_area _types* and *num_task _types*) A2 obtains a greater improvement. If we consider the number of tasks in a multitask, A2 improves performance by 60% in bigger instances, and the improvement is significantly more discreet (23% approx.) if the number of tasks is small. For the other parameters the performance is improved around 50%, and in general A2 improves most in those instances where restrictions are more severe. Regarding TARDCOST () the results are similar but the differences between the percentages are not so pronounced, we consider that it's because TARDCOST () is not so closely related to the temporal relations.
In tables V and VI the cases where the results show opposite directions of change (increasing or decreasing) for SR and A2 have been highlighted in grey. If we consider the quartiles, A2 and SR obtain similar results for quartile 25, but the difference is very pronounced for quartile 75; we believe this shows that A2 is able to reduce the GPR violations in instances of greater complexity.
The improvement is not as pronounced as it was in GPR_A, GPR_B, TARD_A and TARD_B, but that was expected because these four groups were considered the “hardest instances”. It is also interesting to note that the computational cost is smaller in A2.
The data obtained in SET_NMT confirm the above results. The improvements are in a margin between 4% and 20%; much lower because those problems do not include precedence constraints. The parameter of interdisciplinarity itself is affected. The more interdisciplinarity there was the less improvement we found which again supports the previous thesis that the more restricted the instances the greater the improvement.

7. Conclusions and future work.

In this paper we have considered the problem of task scheduling and resource assignments in a Service Centre. To solve the problem we have modelled it as a dynamic online stochastic multimode project scheduling problem with scarce resources and generalized precedence constraints.
To solve the problem we have designed a predictive-reactive approach that works at all times with a reference sequence. Reactive methods are used to update the sequence when the duration of a task is larger than expected. Also at any *react* time the newly arrived tasks are inserted and local search algorithms are used to reduce GPR and TARDCOST infeasibilities.
We have considered generalized precedence relationships in dynamic environments for which we have defined the concept of multitask. Because we do not know of any similar problems in the literature we have designed an instance generator.
We have refined the method considering all possible configurations and choosing between them by applying a heuristic method named FRace, which we have specially modified for our problem.
We have found that the predictive-reactive approach can work in dynamic environments where the durations of tasks are unknown. We have also created tools to analyse the occupancy levels of the system.
Our future aim would be to consider the problem of generating robust sequences against possible sources of uncertainty. It is also necessary to study the response capacity of the system against possible changes in the resources initially available, altered distributions of arrivals and/or durations,...

Acknowledgement

This research was partially supported by the Ministerio de Educación y Ciencia under contract DPI2007-63100.

References

1. Adenso-Díaz B, Laguna M. Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search. Operations Research 2006; 54(1): 99–114.
2. Álvarez E, Díaz F. An application of a real-time scheduling system for turbulent manufacturing enviroments. Robotics and Computer-Integrated Manufacturing (2004); 20: 485-494.
3. Artigues C, Roubellat F. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. European Journal of Operational Research (2000); 127: 294-316.
4. Bagchi, T. Multiobjective Scheduling by Genetic Algorithms. Kluwer (1999).
5. Birattari M., Stützle T., Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. Langdon, W. B. et al. (Eds.), GECCO 2002- Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann Publishers, San Francisco (2002) 11-18. San Francisco:.
6. Cai X, Li KN. A genetic algorithm for scheduling staff of mixed skills under multi-criteria. European Journal of Operational Research, 2000; 125: 359-369.
7. Conover WJ. Practical Nonparametric Statistics. John Wiley & Sons, New York, NY, USA, 3rd edition 1999.
8. Davenport AJ, Beck JC. A Survey of Techniques for Scheduling with Uncertainty. unpublished 2000.
9. DeGroot MH. Ch. 9 in Probability and Statistics. 3rd ed. Reading, MA: Addison-Wesley 1991.
10. De Reyck B, Herroelen W. The multi-mode resource project scheduling problem with generalized precedence relations. European Journal of Operational Research 1999; 119: 538-556.
11. Dean A, Voss D. Design and Analysis of Experiments. Springer Verlag, New York, NY, USA 1999.
12. Demeulemeester EL, Herroelen WS. New Benchmark Results for the Resource-Constrained Project Scheduling Problem Management Science 1997; 43: 1485–1492.
13. Demeulemeester E, Herroelen W. Project scheduling. A research handbook. International Series in Operations Research and Management Science vol.49, Kluwer Academic Publishers, 2002.
14. Elkhyari A, Guéret C, Jussien N. Conflict-based repair techniques for solving dynamic scheduling problems. In Proceedings CP´02, 2002; LNCS 2470: 702-707.
15. Goldratt E. Critical Chain. The North River Press, 1997.
16. Gómez-Cabrero D, Armero C, Ranashinge DN. The Travelling Salesman Problem: A self-adapting PSO-ACS algorithm. ICIIS 2007, International Conference on Industrial and Information Systems 2007; 479-484.
17. Gómez-Cabrero D, Valls V. Secuenciación en Proyectos Online y Dinámicos con Recursos Limitados. Modelado de un Centro de Servicios. Technical Report, University of Valencia; Departamento de Estadística e Investigación Operativa, Spain 2008.
18. Hartmann S. Project Scheduling with Multiple Modes: A Genetic Algorithm. Annals of Operations Research 2001; 102: 111-135.
19. Heilmann R. Resource-constrained project scheduling: a heuristic for the multi-mode case. OR Spektrum 2001; 23: 335-357.

20. Heilmann R. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. European Journal of Operational Research, 2003;144:348-365.
21. Herroelen W, Leus R. Project scheduling under uncertainty, survey and research potentials. European Journal of Operational Research 2005; 165: 289-306.
22. Herroelen, W., Leus R. and Demeulemeester E. Critical chain project scheduling: Do not oversimplify. Project Management Journal 2002; December Issue: 48-60
23. Manimaran G. Resource Management with Dynamic Scheduling in Parallel and Distributed Real-Time Systems. PhD Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras 1998.
24. Manimaran GC, Siva Ram Murthy. A new study for fault-tolerant real-time dynamic scheduling algorithms. Journal of Systems Architecture 1998; 45: 1-13.
25. Mohring RH, Radermacher FJ, Weiss G. Stochastic scheduling problems I: general strategies. ZOR – Zeitschrift für Operations Research, 1984; 28: 193-260.
26. Mohring RH, Radermacher FJ, Weiss G. Stochastic scheduling problems II: set strategies. ZOR – Zeitschrift für Operations Research, 1985; 29: 65-104.
27. Neumann K, Schwindt C, Zimmermann J. Project Scheduling with Time Windows and Scarce Resources. Second Edition Springer, 2003.
28. Schwindt C. Generation of Resource Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report WIOR-489. Institute for Economic Theory and Operations Research.University of Karlsruhe, 1996.
29. Thesen A. Measures of the restrictiveness of project networks. Networks 1977; 7: 193-208.
30. T´Kindt V, Billaut J-C. Multicriteria Scheduling: Theory, Models and Algorithms. Springer-Verlag, 2002.
31. Valls V, Ballestín F, Quintanilla S. Due Dates and RCPSP. Chapter 4 in the book Perspectives in Modern Scheduling, Weglarz J and J. Jozefowska J eds., Springer 2006.
32. Valls V, Pérez A, Quintanilla S. Skilled Workforce Scheduling in Service Centres. European Journal of Operational Research 2009; 193(3): 791-804.
33. Van de Vonder S, Demeulemeester E, Herroelen W, Leus R. The trade-off between stability and makespan in resource-constrained project scheduling. International Journal of Production Research, 2005b; 44(2): 215-236.
34. Van de Vonder S, Demeulemeester E, Leus R, Herroelen W. Proactive-reactive project scheduling trade-offs and procedures. Chapter 2 in the book Perspectives in Modern Scheduling, edited by J. Weglarz and J. Jozefowska, Springer 2006.
35. Van Hentenryck P, Bent R. Online Stochastic Combinatorial Optimization. MIT Press 2006.
36. Vanhoucke, M., Demeulemeester, E. and Herroelen, W. Exact procedure for the unconstrained weighted earliness-tardiness project scheduling problem. Research Report, Department of Applied Economics, Katholieke Universiteit , Leuven, Belgium 1999.
37. Vanhoucke, M. Optimal due date assignment in project scheduling. Vlerick Leuven Gent Management School Working Paper Series 2002-19, Vlerick Leuven Gent Management School, 2000.
38. Viana, A. and Sousa, J.P. Using metaheuristics in multiobjective resource constrained project scheduling. European Journal of Operational Research 2000; 120: 359-374.

VITAE: Vicente Valls is Professor of Statistics and Operational Research at the University of Valencia. He holds a Ph.D degree in Mathematics from the University of Valencia. His research interests include project management and scheduling, machine scheduling, metaheuristics and production planning. In addition to his academic activities, Dr. Valls has served as a consultant to a variety of firms.

David Gómez-Cabrero is a PhD student in Departamento de Estadística e Investigación Operativa, Universitat of Valencia. He earned his two Graduate Degrees in Mathematics and Statistics from the University of Valencia. He is now working as research staff at IDIBAPS.

Figure 1
Click here to download high resolution image

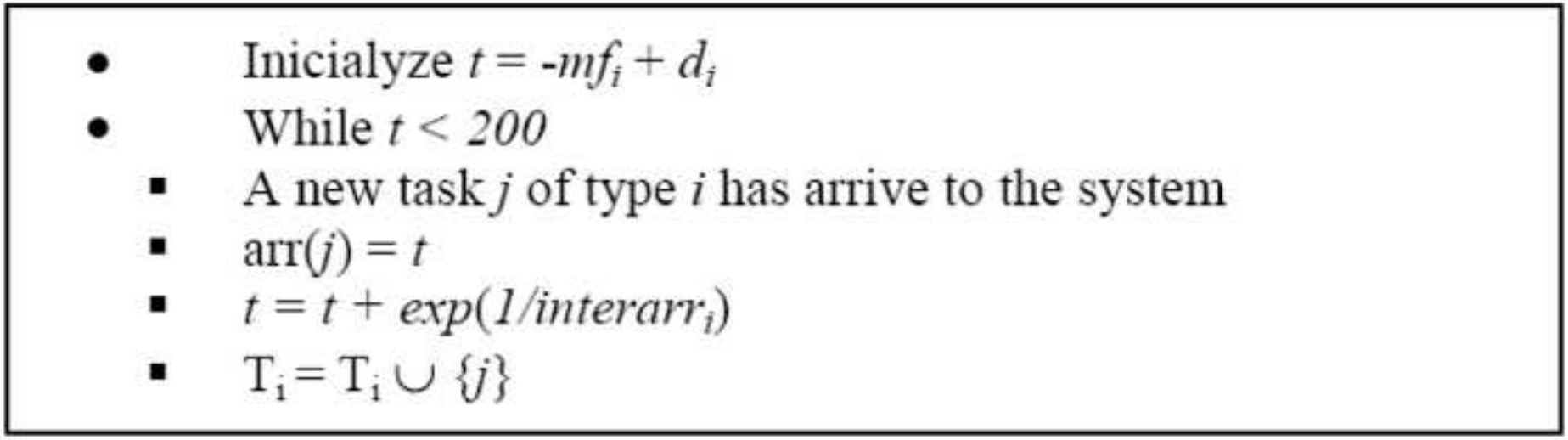

Figure 1. Generating tasks of type *i*

Figure 2
Click here to download high resolution image

Figure 2. Pairwise comparison Test "Racepor".

It is basically a test to compare ratios. Given two combinations A and B, and a set of instances, INSTt, we consider |INSTt| Bernoulli experiments, where 1 is obtained if eval(S_A)> eval(S_B) and 0 otherwise. We consider the following Statistical hypothesis testing

H_0: $p=pperc$ (Null hypothesis)

H_1: $p>pperc$

Distribution of $\hat{p}$:

$$\hat{p} \approx N(p_{perc}, \sqrt{\frac{p_{perc}(1-p_{perc})}{n}})$$

Table I

Table I. Selected Configurations and SR.

Conf	INIT_SOL	INSERT	IntGPR	InsGPR_	IntTARD	InsTARD	*react*	*perc*	*fts*
A1	*InEFTGA*	*IEFTGA*	0	1	1	1	3	90	late
A2	*InEFTGA*	*IESTGA*	1	1	1	1	3	90	late
A3	*InEFTGA*	*IESTGA*	1	1	0	0	5	30	NA
A4	*InEFTGA*	*IEFTGA*	1	1	0	0	3	30	NA
SR	EST	EST	0	0	1	1	5	30	late

Table II

Table II. *GPR*(S_{final})

	GPR_A		GPR_B		TARD_A		TARD_B		Total
	average	mean	average	mean	average	mean	average	mean	average
A1	563.67	423	358.72	329	420.26	283	246.39	214	397.26
A2	**560.47**	**421**	**354.54**	331	**418.43**	298	247.01	217	**395.11**
A3	1520.73	669	465.70	**320**	836.21	253	195.27	144	754.48
A4	1352.70	684	438.62	325	767.89	**250**	**194.61**	147	688.46
SR	2873.30	1324	913.99	696	1556.16	514	424.44	889	1441.97

Table III

Table III. TARDCOST(S_{final})

	GPR_A		GPR_B		TARD_A		TARD_B		Total
	average	mean	average	mean	average	mean	average	mean	average
A1	**25380.1**	**13802**	13297.53	9341	**43907.6**	**22947**	25163.21	17986	**26937.1**
A2	25672.8	14330	**13127.24**	**8809**	44273.0	23216	**24916.00**	**17495**	26997.2
A3	464431.8	281075	218555.5	175301	607573.6	363525	308993.20	273043	399888.5
A4	466596.4	272000	214079.1	179758	605220.7	369897	301055.30	262666	396737.9
SR	71690.7	31904	40266.74	19798	143729.6	97023	92129.58	65292	86954.2

Table IV

Tabla IV. *cpu_pred_med*.

	GPR_A	GPR_B	TARD_A	TARD_B	Media
A1	1.53	0.05	1.53	0.05	0.79
A2	**1.36**	0.05	**1.36**	0.05	**0.71**
A3	1.78	**0.04**	1.47	**0.03**	0.83
A4	2.75	0.06	2.23	0.04	1.27
SR	1.83	**0.04**	1.10	**0.03**	0.75

Table V

Table V. A2 vs SR, considering GPR(S_{final}) over SET_T.

		SR				A2				dec
		Average	**Desv.**	**25**	**75**	**Average**	**Desv.**	**25**	**75**	
num_area _types	3	420,19	1687,83	95	395	**182,00**	127,85	97,60	228,00	56,69%
	5	364,04	1209,28	85	373	**191,92**	137,45	99,60	244,40	47,28%
	10	357,5	1109,63	80	386	**203,85**	156,27	98,00	262,20	42,98%
num_task _types	1	418,4	1524,16	99	396	**176,28**	118,90	96,00	219,60	57,87%
	5	386,04	1541,31	90	392	**191,85**	136,35	99,20	242,80	50,30%
	10	341,44	968,65	74	369	**209,63**	163,12	100,20	272,20	38,60%
max_mtask_size	5	148,62	137,47	54	202	**114,47**	62,12	62,00	154,40	22,98%
	10	298,79	561,66	99	379	**177,78**	101,76	99,80	234,40	40,50%
	15	721,29	2292,28	158	736	**285,52**	175,28	158,80	367,40	60,42%
num_worker_types	3	345,9	1193,61	86	362	**186,43**	141,92	94,40	229,00	46,10%
	5	360,55	1039,47	87	378	**191,16**	138,10	99,20	241,40	46,98%
	10	433,73	1731,16	87	417	**200,18**	143,48	101,60	258,80	53,85%
dur_arrival	1	590,92	2134,54	145	578	**292,15**	173,22	166,20	376,40	50,56%
	2	368,12	814,83	119	400	**188,49**	87,79	124,60	234,00	48,80%
	3	177,9	384,81	40	196	**97,14**	56,05	54,80	127,60	45,40%
cross_training	0,25	377,3	1237,15	92	390	**198,80**	144,23	102,00	252,00	47,31%
	0,5	383,16	1440,07	86	383	**191,64**	140,19	98,60	239,80	49,99%
	0,75	378,2	1367,38	81	378	**187,34**	139,20	94,40	238,00	50,47%
restrictiveness	0,25	379,56	1357,43	86	385	**192,73**	139,88	99,00	244,60	49,22%
	0,5	377,96	1234,77	86	381	**193,19**	144,34	98,20	241,60	48,89%
	0,75	381,15	1451,5	87	385	**191,85**	139,63	98,00	244,60	49,67%
redundancy	0,25	346,42	1518,39	82	355	**187,30**	142,23	92,80	234,40	45,93%
	0,5	377,75	973,67	86	390	**193,08**	142,23	98,00	245,40	48,89%
	0,75	414,53	1489,25	92	410	**197,38**	139,24	104,40	249,20	52,39%
CST	0,25	404,47	1359,67	89	400	**198,60**	145,33	100,80	249,80	50,90%
	0,5	379,43	1331,2	86	381	**191,75**	143,14	97,00	244,40	49,46%
	0,75	354,75	1360,73	84	372	**187,42**	134,99	97,00	237,00	47,17%

Table VI

Tabla VI. A2 vs SR, considering TARDCOST(S_{final}) over SET_T.

		SR				A2				dec
		Average	**Desv.**	**25**	**75**	**Average**	**Desv.**	**25**	**75**	
num_area _types	3	28603,50	67811,96	3461,00	25229.5	**10566,45**	12472,45	3512,40	12679,60	63,06%
	5	24707,49	65353,28	4183,00	20232,00	**10606,54**	10880,51	4005,80	13560,20	57,07%
	10	24590,20	69661,61	5279,00	21025,00	**11808,92**	9064,41	5115,80	16291,40	51,98%
num_task _types	1	25334,63	57037,44	3081,00	25330,00	**10834,50**	12888,26	3349,40	13326,80	57,23%
	5	27137,67	71314,28	4307,00	20229,00	**10089,57**	9284,60	4111,40	12902,40	62,82%
	10	25303,82	72178,51	5388,00	20808,00	**12057,84**	10140,10	5389,60	16306,80	52,35%
max_mtask_size	5	11360,72	16880,16	3163,00	12249.5	**6310,44**	4840,07	2935,60	8206,00	44,45%
	10	23649,06	46548,22	4770,00	21806,00	**9905,51**	6970,77	4552,20	13504,20	58,11%
	15	44343,24	106138,81	6368,00	38517,00	**16765,95**	15121,04	6562,20	21678,60	62,19%
num_worker_types	3	18739,90	40127,02	3953,00	18787,00	**10798,62**	10707,20	4053,20	13965,80	42,38%
	5	22383,73	52584,14	4251,00	20076,00	**11032,88**	10720,46	4231,80	14508,40	50,71%
	10	36850,11	96183,70	4857,00	27632,00	**11150,40**	11290,33	4340,40	14466,80	69,74%
dur_arrival	1	29006,10	83562,34	6200,00	21554,00	**16167,99**	14585,74	7249,40	19648,40	44,26%
	2	39784,87	76270,22	8439,00	38626,00	**12542,61**	7839,60	7067,20	16248,60	68,47%
	3	8810,53	18893,48	2364,00	7497,00	**4271,31**	2920,76	2441,80	5291,60	51,52%
cross_training	0,25	22535,82	46526,29	4633,00	22401,00	**12441,29**	11301,06	4881,20	16459,80	44,79%
	0,5	25084,31	65968,18	4251.5	20993,00	**10733,50**	10658,21	4143,60	13913,20	57,21%
	0,75	30066,52	84509,34	4081,00	21602,00	**9807,11**	10593,63	3749,80	12528,00	67,38%
restrictiveness	0,25	26104,04	67702,08	4395,00	21697,00	**11088,33**	11139,54	4180,60	14354,60	57,52%
	0,5	25914,19	68462,21	4297,00	21821,00	**11068,45**	10797,68	4234,20	14380,20	57,29%
	0,75	25649,42	66368,81	4263,00	21522.5	**10825,12**	10788,23	4215,40	14188,60	57,80%
redundancy	0,25	25874,76	64922,11	4280,00	21887,00	**11268,91**	11921,85	4191,60	14502,20	56,45%
	0,5	25984,28	73356,07	4319,00	21491,00	**10833,01**	10542,74	4144,00	14100,60	58,31%
	0,75	25809,20	63872,36	4341,00	21687,00	**10879,98**	10183,99	4258,20	14318,60	57,84%
CST	0,25	26260,81	67610,69	4308,00	21698,00	**11115,78**	10690,16	4213,60	14575,80	57,67%
	0,5	26046,74	68863,61	4286,00	21691,00	**11055,24**	11541,94	4180,60	14375,80	57,56%
	0,75	25360,24	66044,46	4349,00	21657,00	**10810,89**	10467,10	4228,20	13991,60	57,37%

Table VII

Tabla VII. A2 vs SR, considering TARDCOST(S_{final}) over SET_NMT.

		SR				*A2*				
		Average	**Desv.**	**25**	**75**	**Average**	**Desv.**	**25**	**75**	**dec**
num_area _types	3	3828,9	6012,55	608	4333	3202,25	891,80	3912,00	3499,54	16,37%
	5	6147,38	11251,56	1228	5593	5026,15	1601,60	7662,40	5365,39	18,24%
	10	9038,33	13828,08	2570	9739	8178,77	2634,20	10352,80	8284,59	9,51%
num_task _types	1	3227,12	6541,87	437	2745	2713,08	513,80	3181,60	3466,36	15,93%
	5	6142,85	10465,53	1726	6478	5249,12	2100,00	6191,80	5524,56	14,55%
	10	9644,63	13949,82	2895	9719	8444,98	3182,80	10861,40	7953,72	12,44%
num_worker_types	3	7818,39	12231,36	1721	7933	6474,72	1852,60	8769,00	6992,33	17,19%
	5	6631,33	11415,49	1304	7174	5679,88	1717,40	6135,00	6622,68	14,35%
	10	4564,89	9070,12	714	5225	4252,57	891,80	5192,00	5213,85	6,84%
dur_arrival	1	4924,05	5982,49	1754	6034	4333,95	2126,40	5310,40	3059,99	11,98%
	2	11865,09	16658,5	2628	13928	10085,37	3741,00	14404,40	8722,48	15,00%
	3	2225,46	2177,63	667	2984	1987,85	1109,80	2736,00	1381,13	10,68%
cross_training	0.25	8723,75	14482,71	1766	9369	7087,78	2176,80	8994,00	7951,33	18,75%
	0.5	6116,18	10636,59	1311	6580	5306,06	1477,00	6856,40	6275,80	13,25%
	0.75	4174,68	5855,66	908	5189	4013,33	1473,80	5192,00	3881,93	3,87%