Job-shop scheduling applied to computer vision

J.M[a].Sebastián[a], F.Torres[b], R.Aracil[a], O.Reinoso[a], L.M.Jiménez[b], D. García[a]

[a]Dept. Engineering System and Automatica, UPM, Spain
[b]Dept. Engineering System and Communicactions, UA, Spain

ABSTRACT

This paper presents a method for minimizing the total elapsed time spent by n tasks running on m differents processors working in parallel. The developed algorithm not only minimizes the total elapsed time but also reduces the idle time and waiting time of in-process tasks. This condition is very important in some applications of computer vision in which the time to finish the total process is particularly critical -quality control in industrial inspection, real-time computer vision, guided robots, ... The scheduling algorithm is based on the use of two matrices, obtained from the precedence relationships between tasks, and the data obtained from the two matrixes. The developed scheduling algorithm has been tested in one application of quality control using computer vision. The results obtained have been satisfactory in the application of different image processing algorithms.

Keywords: scheduling, computer vision, real-time, parallel processing, priority algorithm.

1. INTRODUCTION

According to Henry Winston's definitions, a plan is a description of a sequence of acts that, in the case it is followed, the relationships between objects will change in order to reach a specific goal[1]. The nature of this goal can be of different kind. It can consist on reducing to a minimum the number of processors, or reducing the global time spent by the set of tasks, or obtaining the maximum occupation in all the existing processors, or minimizing the dead times in only some of them, etc. The theory of planning or scheduling was originally developed in the industrial context. The aim was to find the most suitable distribution between the different machines capable of performing a job and all the different operations than were needed to be accomplished. However, in most of the cases the aim consisted in obtain the higher degree of occupation in the different machines. Afterwards, with the appearance of the parallel computation, new scheduling techniques were developed in order to distribute the operations between the different processor that were present. The application of these techniques does not remain restricted to the mentioned disciplines and can also be extended to others such as real time systems[2]. Stankovic defines the computation in real time as a system that not only depends on the results obtained in the execution of a process, but also on the amount of time in which this is performed[3]. The requirement of time constitutes the principal difference between these systems with respect to more classic ones. Some applications of computer vision constitute a clear example of real time systems, since the result not only depends on the quality of the data obtained in the processing, but also because this result must be obtained within a required span of time. In this context, there are examples of computer vision systems in the area of automated visual inspection, guidance of mobile vehicles, processing and pattern recognition in movement, etc.

In this paper we present a scheduling algorithm that solves the problem of planning in computer vision. In order to perform this planning, all the specific characteristics of computer vision systems, many of them different to those of other real time systems, are taken into account. It is important to emphasize that the solution we have obtained solves the problem and minimizes the total processing time spent by the system.

2. PROBLEM SET OUT

Just like in other real time systems, in the applications of computer vision in the field of industrial inspection the requirements of time present a known value, which is generally constant, and which determine them as systems directed by time. However, there are other cases such as pattern recognition in movement, guidance of vehicles, etc… in which the system can be directed by events. This is the reason why the scheduling techniques have a direct application in computer vision systems.

2.1. Types of tasks

Just like in other real time systems, in the area of computer vision, the tasks can be classified[4] in periodic and asynchronous, preemptive and nonpreemptive, ... However, in computer vision there is a new concept that allows one to establish a new classification. Such concept is the place in which the tasks are executed. Generically, the architecture that is usually employed in computer vision is composed of central processing units (CPU'S) and Image Acquisition and Processing Boards (IAPB'S). Depending on the place in which a task is executed, three different types of tasks[5] can be distinguished: T_CPU/IAPB; T_CPU and T_IAPB.

A T_CPU/IAPB task makes use of the two system resources jointly. Such task requires the full-time of a CPU and an IAPB. Furthermore, once they have begun its execution they can not be interrupted by other task. Therefore, they present exclusion relationships. The information exchange that needs to be established between the CPU and the IAPB in both directions constitutes the prototype of this kind of tasks. Operations such as data reading of an image from the video-RAM of the IAPB towards the CPU, reading of a histogram for its interpretation, etc, are examples of this type of tasks.

T_CPU tasks are characterized by being executed in the system CPU. They consist in abstract operations located at the higher levels of processing. Also, certain tasks that actually correspond to lower levels of processing can be considered among this kind of tasks, due to the fact that they are executed in the CPU in order to obtain some kind of improvement as higher precision, computation speed, etc. This tasks are also characterized by its preemptive nature, or in other words, the nonexistence of exclusion relationships.

The T_IAPB tasks correspond with the tasks at the lowest levels of processing, and they are usually executed in the IAPB. Their principal characteristic consists in the existence of exclusion relationships for the tasks that are executed in this hardware. Another characteristic of this type of tasks consists in the need of establishing at its beginning a T_CPU/IAPB task, in order to allow the CPU to indicate to the IAPB the start of execution. However, generally the amount of time required to run this task is negligible. This fact produces that at the moment of beginning of a T_IPAB the CPU must be available or al least in a nonpreemptive state.

From the previous paragraphs, it is clear the need of accomplishing an adequate space-time distribution of all the tasks between the existing processors in the system in order to achieve an optimization of the application.

2.2. Relationships between tasks

The precedence relationship[4] is strongly related to any computer vision system. Obviously an image can not be processed without a previous acquisition task. Neither an object can be classified without performing a previous image segmentation. Therefore, the precedence relationship exists and so it has to be considered in the subsequent scheduling of the tasks to be performed.

The exclusion relationship[4] is a special characteristic of the computer vision systems with respect to other real time systems. The difference lies in the coexistence of tasks with exclusion relationship together with other tasks that do not possess this relationship. That is to say, tasks that can be interrupted as opposed to others that can not be. The acquisition of an image constitutes an example of a task with exclusion relationship, since it can not be interrupted by any other task. Generally, all those tasks that correspond to the lower levels of processing and which are executed in the IAPB are characterized by possessing exclusion relationships. This condition is imposed by the constructive characteristics of the board employed in each application, although it can be different in certain cases. However, the tasks located at the higher levels of processing tend to be developed in a generic processor that allows it to maintain the execution of tasks without exclusion relationships.

2.3. Priorities assignment

In any computer vision system which has important restrictions of time, the factor to be minimized is the decrease of the total time response of the system. In quality control, guidance of vehicles, etc., the lag in time response leads to unsatisfactory results in the first case or catastrophic consequences in the second case. For this reasons, the variable to be taken into account at the moment of accomplishing the scheduling of a computer vision system is the minimization of the global time response of the system. Considering this fact, it is possible to obtain some benefits such an increase of inspection speeds, or an

improvement of the quality of the system, an increase of the speed and features in the case of guided vehicles, etc. If the aim consists in minimizing the time response of the system, the assignment of priorities to each one of the tasks is performed based on this approach.

Def. critical task at a certain instant. Such task that, considering the existing conditions at that moment, will be the last to end its processing. Once this task has been identified, the maximum priority is assigned to the task that presents some precedence relationship with that critical task or with other tasks that have preceding relationship with the critical task. That is to say, such a task that, starting from it in the precedence graph, it is possible to get to the most critical node. In the event of existing more than one task that presents these characteristics, the assignment of the maximum priority will be performed on the task that presents the greatest cost to get to the critical task.

3. LINK MATRIXES

In the resolution of a problem, different precedence relationships appear between the tasks. The representation of these relationships for n tasks is accomplished with the aid of a precedence graph, from which it is possible to build a link matrix $\mathbf{C}^{nxn}$ [7].

3.1. Link Matrix C^{nxn}

The elements c_{ij} of the matrix **C** are binary numbers. The element c_{ij} takes the value 1 when a precedence relationship between the tasks i and j exists, that is to say, when i precedes j. On the contrary, if it does not exist a precedence relationship between both tasks, it takes the value 0. In short, it is a matrix that collects the different links that exist between tasks in the precedence graph in each one of its elements. However, the impossibility of existence of some precedence relationships force some elements of **C** to the value 0. Obviously, a task does not need of itself to start execution. For example, the acquisition of an image does not depend on the data generated in the quantification and sampling processes in the captation stage, since it would not be feasible such a task. This fact implies that the elements c_{ii} of **C** must take the value 0.

Neither it is possible that a task i needs the occurrence of another j in order to start execution, and at the same time, the task j has the need of i to start. For example, the task of captation of an image does not have any kind of dependency with subsequent processing on it such a segmentation process. This particularity leads to the condition expressed in equation 1:

$$\text{if } c_{ij} = 1 \quad \Rightarrow \quad c_{ij} \neq c_{ji} \tag{1}$$

If an element of **C** takes a value equal to 1, its symmetrical element with respect to the principal diagonal can not take the value 1. Also, it is impossible the existence of any set of relationships that leads to a closed loop between different tasks. With the aid of an adequate representation of both the tasks and the relationships that configure the precedence graph, it is possible to assign the value 0 to every element placed above the principal diagonal (equation 2).

$$C = \begin{bmatrix} 0 & 0 & 0 & . & . & 0 \\ c_{21} & 0 & 0 & . & . & 0 \\ c_{31} & c_{32} & 0 & . & . & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ c_{n1} & c_{n2} & c_{n3} & . & . & 0 \end{bmatrix} \tag{2}$$

3.2. Total link matrix T^{nxn}

Starting from the matrix **C** it is possible to build a new matrix $\mathbf{T}^{nxn}$, designated as *total link matrix*[8]. Just like in the case of **C**, every element of the matrix **T** is binary and each element placed in and above the principal diagonal is 0. If it is necessary the occurrence of the task i in order to start the execution of the task j, the element t_{ij} of the matrix **T** takes a value equal to 1. That is to say, if we follow the precedence graph starting from the task j and going in opposite direction of that established by the precedence relationships, until we get to the initial task, and also the task i belongs to the path, the element t_{ij} of the matrix **T** will take the value 1. Otherwise, the value of the element will be 0. The values associated with the matrix **T** are obtained by applying the total link algorithm[8] to the matrix **C**.

The interpretation of **C** is quite different to that of the matrix **T**. The matrix **C** collects every existing relationship of its related graph, that is to say, the sum of all the elements c_{ij} indicates the number of relationships that actually exists in the graph. In the case of **T** this last statement is not valid, but there are some new concepts that can be obtained from it (Section 4).

4. SIMULTANEOUS OCCURRENCE BETWEEN TASKS

The concept of simultaneous occurrence appears at the moment in which it is possible that several tasks may be running at the same time sequence. In the application that we have accomplished there are two possible situations in which simultaneous occurrence exists. The first is related to the specific characteristics of the problem due to the fact that it is a computer vision system and therefore, it is extrinsic to the matrix **C**. The second situation depends on the values of the matrix **T** and therefore, it is intrinsic to the precedence graph.

4.1. For type of task

Def. simultaneous ocurrence by type of task between two tasks i and j. It is the case in which the two processors involved in the execution of each task are different. If this condition is fulfilled, both tasks can be executed at the same instant of time. Examples of simultaneous ocurrence by type of task are a T_CPU with a T_IAPB, or with a T_CPU/IAPB, provided that the CPU processors needed by each task are different. Similarly, a T_IAPB and a T_CPU, or a T_IAPB'S and a T_CPU/IAPB'S using different IAPB or even two T_CPU/IAPB provided that, in all cases, the CPU's and IAPB's involved are different.

4.2. For precedence relationship

Def. two tasks i and j are concurrent by relationship of precedence if the elements of the matrix **T** comply with the next relationship:

$$\forall\, k \;\Big|\; t_{ki} = t_{kj} = 1 \qquad \Rightarrow \qquad t_{ij} = 0 \quad \text{and} \quad t_{ji} = 0 \tag{3}$$

Hence, the existence of two incoming precedence relationships in the task k can induce to the simultaneous occurrence between the i and j tasks, as shown in figure 1.a. However, the second condition of equation 3 is the one which actually determines the simultaneous ocurrence existence by precedence relationship. That is to say, it depends of the existence of either one of the two relationships, a or b, between the i and j tasks (Figure 1.b).

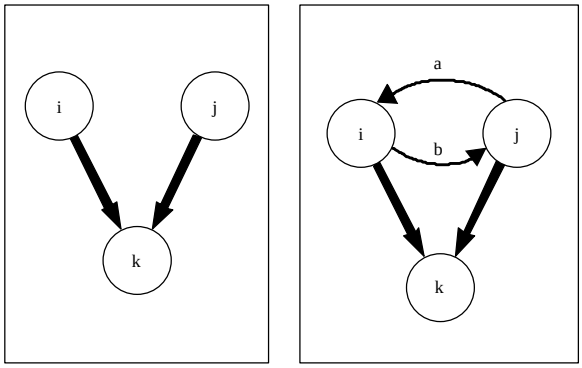


Figure 1. a) Possible simultaneous ocurrence. b) Without simultaneous ocurrence.

As it is clear, if two tasks, i and j, verify the simultaneous ocurrence condition by precedence relationship of the equation 3, also they must fulfill the simultaneous ocurrence condition by type of task in order to be actually concurrent. For example, the tasks i and j of the figure 1.a are potentially concurrent by precedence relationship. However, the existence of real simultaneous occurrence is imposed by the nature of each one of the tasks. If both tasks are of type T_IAPB and they are executed in the same IAPB, it will not be possible to execute them in a concurrent way, even though the condition of precedence relationship is fulfilled. On the contrary, two tasks i and j can be actually concurrent by type of task without the need of existence of any kind of relationship between them.

5. STATES OF A PROCESSOR

In order to be able to execute a certain task in a processor, this processor must be available. We can observe important differences between a computer vision system and other kind of systems. The coexistence of preemptive and nonpreemptive tasks together shows the hybrid character of the system, that is to say, the characteristics of both types of tasks appear in the same system. The availability of the processor, in case of being busy, depends on the nature of the task that is running on it at that moment.

If there is not any task being executed in that processor, then the processor is in a free state. In any other case the state of the processor depends on the processor itself and the task that it is running on it. If the processor considered is the system CPU, it is the task being executed the one who decides the state. If this task is a T_CPU task, it will be preemptive since the task in execution can be interrupted. On the contrary, if we consider a T_CPU/IAPB task, the particular nature of this kind of task in the context of computer vision provokes the processor to remain busy. Therefore, the task can not be interrupted and if there is any other task that is ready to be executed, such task will have to wait until the processor gets to a free state. If we consider a situation in which the processor is a IAPB the only possible state is a busy state since any task that can be executed in such processor, T_IAPB or T_CPU/IAPB, is defined as nonpreemptive. The following chart summarizes all the possible states that can be considered according to the type of processor and the task being executed.

Processor	Task			
	No hay	T_CPU	T_IAPB	T_CPU/IAPB
CPU	Free	Preemptive	Impossible	Busy
IAPB	Free	Impossible	Busy	Busy

6. DATA OBTAINED FROM MATRIXES C Y T

6.1. Data obtained from matrix C

Each element of the matrix **C** shows the presence of a relationship that links two tasks. Therefore, this matrix collects information related to every task together with the relationships that link them. The matrix **C** allows us to obtain some characteristic of the graph.

Def. identification of start tasks. If we multiply the matrix **C** and the n-dimensional unitary vector $\mathbf{X}^T = \{1, 1, 1, ..., 1\}$, we obtain as a result a new n-dimensional vector $\mathbf{O}^T$ whose components o_i represent the number of precedence relationships whose destination task is i (Equation 4). This value represents the addition of the values of the matrix **C** by rows and indicates the number of tasks from which the task i depends directly.

$$o_i = \sum_{j=1}^{i-1} c_{ij} \tag{4}$$

Thus, it is possible to identify the start tasks in the precedence graph, that is to say, the tasks associated with the component i of the vector $\mathbf{O}^T$ whose value is equal to zero. Such tasks are identified as start tasks in the graph, since they do not posses any precedence relationship with itself.

Def. identification of destination tasks. Just like in the previous situation, we can multiply the matrix $\mathbf{C}^T$ and the unitary vector $\mathbf{X}^T$. The result obtained is a new n-dimensional vector $\mathbf{D}^T$ whose element d_i represents the number of precedence relationships whose start task is i. Therefore, each component of $\mathbf{D}^T$ represents the addition of the values of **C** by columns, that is, the numbers of tasks that depend directly on the task i to start execution (Equation 5).

$$d_i = \sum_{j=i+1}^{n} c_{ji} \quad (5)$$

In this case, from the values of the components of this vector, it is possible to determine the end tasks in the precedence graph. Such tasks are those whose value in the vector $\mathbf{D}^T$ is equal to 0.

Def. start and completion time of a task. A task may go through several states throughout its life. But in order for this to occur, this task must be previously created and furthermore, this creation must happen in a certain instant of time. From the values of the matrix **C,** it is possible to know the moments in which each task is executed. This matrix also collects information related to the tasks that must have finished its execution in order to allow other tasks to execute. If we know the amount of computation time spent by each task involved in the application, it is possible to obtain the start and completion times of each one in the best cases, that is to say, in the case in which the maximum theoretical simultaneous occurrence exists. Therefore, we will assume that the computation time vector $\mathbf{t\_cp}^T$ is known. Each component t_cp_i of this vector represents the computation time of the task i. We can also define the completion time vector $\mathbf{t\_fin}^T$ as a vector whose components t_fin_i take the following value:

$$t\_fin_i = t\_cp_i + \max_{j=1,2,...i-1} t\_fin_j \quad (6)$$

that is to say, the amount of time that certain task may spend depends on the time in which its immediate predecessor task ends in the precedence graph. However, if we take into account the case in which the task i be preceded by more than one task, the value of t_fin_i will represent the sum of the maximum time spent by all the tasks which precedes this task and its computation time. In this case, we assume that the maximum simultaneous occurrence takes place, and therefore, all the tasks that precede the task i are executed simultaneously. Once the vector $\mathbf{t\_fin}^T$ has been determined, it is possible to obtain the start time vector $\mathbf{t\_crea}^T$, whose components represent the times in which each task is created in the best of the possible cases.

$$t\_crea_i = t\_fin_i - t\_cp_i \quad (7)$$

This operation must be accomplished once the completion time of each task is known since the completion time of the task i depends on the computation times of the i-1 previous tasks. The value of the component t_crea_i represents the time that must elapse until the creation of the task i in the best of the possible cases. A value equal to 0 in any of the components of the vector $\mathbf{t\_crea}^T$ indicates that the task related to that component is ready to be executed, and the only necessary condition for this to occur is that the processors in which this task must be executed must be available (free or preemptive).

6.2. Data obtained from matrix T

As detailed in the previous subpart, starting from matrix **C,** it is possible to determine the number of incoming and outgoings relationships for a given task. Considering the matrix **T** the addition of the values of the elements by rows or by columns have different meaning.

Def. tasks needed by the task i in order to occur. The product of the unitary vector $\mathbf{X}^T$ and the matrix **T**, just like in the case in which this vector is multiplied by the matrix **C**, allows us to identify the start tasks in the graph. However, the new values of the components of the vector $\mathbf{O}^T$ are:

$$o_i = \sum_{j=1}^{i-1} t_{ij} \quad (8)$$

In this case, the component o_i represents the total number of tasks necessary to end in order to allow the task i to start execution, in contrast to the information given by the matrix C which indicates the task that precedes the task i.

Def. tasks that need the occurrence of the task i. The new values of the components of the vector $\mathbf{D}^T$ are obtained similarly to the case of the matrix **C**, that is to say, by the product of the matrix $\mathbf{T}^T$ and the vector $\mathbf{X}^T$.

$$d_i = \sum_{j=i+1}^{n} t_{ji} \tag{9}$$

Just like in the preceding case, it is possible to identify the end tasks associated with the precedence graph. However, the value of each one of the components of $\mathbf{D}^T$ has a new meaning. In this case, the value of the component d_i indicates the number of tasks that needs of the completion of the task i, provided that they need this completion to occur in order to start execution.

Def. weight associated with each process. The matrix **T** collects information related to the number of tasks that must have ended in order to allow another task to be created. Out of this concept, it is possible to introduce a new measure of time associated with each task, which is represented in the vector $\mathbf{t\_peso}^T$, whose component t_peso_i takes the following value:

$$t\_peso_i = t\_fin_i + \sum_{j=1}^{n} t_{ij} \cdot t\_cp_j \tag{10}$$

and represents the amount of time that will be spend, in the best of the cases, by each task that needs the completion of task i in order to execute. As it is observed, it is a value that, indicates the degree of urgency for the completion of a given task.

7. SCHEDULING POLICY

The aim of this algorithm is the minimization of the total time spent by a set of tasks involved in an application of a computer vision system in real time. It is necessary to implement a set of rules of decision in order to allow the algorithm to decide, at the moments of creation of each one of the tasks, which task can start executing at that instant. Thus, the algorithm must be capable of assigning the maximum priority to certain tasks, those which can start executing in a specific instant. The method of selection studied assigns the highest priority to the task that generates useful information for the execution of other tasks with larger completion time, that is to say, those tasks linked to the most critical task of the system, according to the values of **T**. Considering the pre-selected tasks, we can distinguish the task with the largest weight time ($\mathbf{t\_peso}^T$), and thus, the tasks located at the branch of the graph with the highest computational cost. Nevertheless, the application of this model to the set of tasks makes necessary to consider a dynamic algorithm due to the fact that the data obtained is dependent on the decisions taken in the previous cycle.

7.1. Alterations in the completion time

As was pointed out above, the completion time ($\mathbf{t\_fin}^T$) is established considering the best of all the possible cases. Actually, this assumption is not feasible. This fact implies a modification of the completion time value based on the current conditions at each moment of the planning. These modifications may be motivated by two different situations:

A lack of simultaneous occurrence. The completion time of a task satisfies the equation 6, what implies the maximum simultaneous occurrence assumption. However, a more realistic situation occurs when tasks that are assumed to be concurrent will not behave in such a way due to any of the conditions established in section 4. Therefore, it is necessary to associate a variable t_ret to each task. This variable will accumulate the amount of time that a task delays its start time. Evidently, this value must be considered at the moment of calculating the completion time associated to each task. Equation 6 can be written now as:

$$t\_fin_i = t\_cp_i + t\_ret_i + \max_{j=1,2,\ldots i-1} t\_fin_j \tag{11}$$

The accumulation of a delay in the variable t_ret of the task i may be motivated by two different reasons. The first reason is the lack of simultaneous ocurrence with another task j, which shares the same instant of creation as i, but has a higher priority. In this case the variable t_ret is increased with the value t_cp_j. However, the lack of simultaneous ocurrence between two tasks, i and j, is also important in the case in which the times of creation are different. If t_crea_i happens during the execution of the task j, there will be also some delay. Now the variable t_ret will accumulate the value t_fin_j - t_crea_i. In this case the delay corresponds only to the remaining computation time of the task j.

A task can be interrupted. Continuing with the previous reasoning, if the task j is of T_CPU type, that is to say, it can be interrupted, the lag may fall on either task i or j. Thus, at this moment it is necessary to evaluate such interruption in order to determine which decision may be of higher interest for the system. Either the task j can continue executing or, on the contrary, it can be interrupted to allow task i to execute. Quantitatively, the first alternative is similar to the second solution described in the previous subpart. In the second alternative, the remaining computing time of the task j is detained a period of time equal to the computing time of the task i. These alternatives serve to emphasize the need for a new variable associated to each task. Such variable will indicate the amount of time that a certain task has been waiting from the moment in which another task interrupted its execution. Again, this new variable, t_inte, modifies the value of the completion time of each task, as shown in the next equality:

$$t\_fin_i = t\_cp_i + t\_ret_i + t\_\mathrm{int}e_i + \max_{j=1,2,..i-1} t\_fin_j \qquad (12)$$

7.2. Tasks ready to be executed

Subsequently, it is necessary to determine which task must have the higher priority to be first executed. This selection is made among those tasks that can be potentially executed in a specific moment. Therefore, the algorithm must determine which tasks are ready for execution. The scheduling algorithm keeps information of every task that has started already its execution. These values are stored in a new n-dimensional vector $\mathbf{F}^T$. Thus, this vector determines whether a task is executing or not. Its components are binary numbers, that is, a value equal to 1 is assigned to those that have not started execution yet and 0 to those which are already executing. When every single component of this vector has taken the value 0, the scheduling algorithm has concluded. Starting from the matrix **C,** it is possible to determine the start tasks by adding the values of this matrix by rows. However, as the different tasks are being executed, if we perform a logical AND operation between the $\mathbf{F}^T$ vector and each one of the rows of the matrix **C**, the values related to the tasks that have started executing are updated and therefore reduced. Starting from this new values it is possible to determine which tasks are ready for execution. Such tasks are indicated in a new n-dimensional vector, $\mathbf{L}^T$. The components of this vector are dynamically updated according to the tasks being executed. Each one of its components is a binary value. It acquires the value 1 when the task has already been created, and it is waiting for execution. A task is said to be ready when the addition of the values by rows of the matrix **C**, after the operation with $\mathbf{F}^T$, turns out to be 0. Initially, the components of the $\mathbf{L}^T$ vector that possess the value 1 correspond to the start tasks of the precedence graph. They are the only tasks which do not need to wait for the end of another task to start executing. The subsequent modification of a 0 towards 1 is carried out each time a task is executed.

7.3. Selection of tasks ready to be executed

Once the vector $\mathbf{L}^T$ has been determined at an instant of time, we must decide to execute the most adequate task in order to optimize the total amount of time spent by the system. If the vector $\mathbf{L}^T$ presents only one task ready to be executed, the selection is automatic. Otherwise, if there are more than one task ready, it is necessary to determine which is the task with the highest priority. In order to perform this operation, we can build a n-dimensional vector $\mathbf{Nfin}^T$ of binary components. The aim of this is to indicate the presence of a task that constitutes an end node in the precedence graph. In this case, its associated component takes a value equal to 1 in this matrix. Otherwise, it takes the value 0. The most critical task t_c at the moment of performing the selection is defined as the one which presents the highest completion time between the tasks that constitute an end in the precedence graph (Equation 13).

$$t_c = \max_{j=1,2,\ldots i-1} (t\_fin_j \cdot Nfin_j) \quad (13)$$

Once the most critical end task has been determined, a previous selection of the tasks ready to be executed is accomplished. The aim is to determine the subset of the tasks that precede in the precedence graph the most critical end task, in order to give a higher priority to the components of that subset as opposed to the remaining tasks. In fact, it consists in checking whether the element of the matrix **T** at the row t_c and at a column associated to the task to be determined, takes the value 1. Out of those tasks, we can chose the one which has the greatest value t_peso of all. The result of this operation is a task that we will call ready task (t_{ls}). However, it is possible the nonexistence of a task that precedes the t_c selected. In this case, the algorithm will compute again the t_{ls} considering a new value for t_c and excluding the first t_c calculated.

7.4. Execution of the task selected

The execution of the task t_{ls} previously selected is structured in several parts. The first consists in testing the processors involved in the execution of t_{ls} in order to determine whether it can be executed or, on the contrary, it must be delayed. In such a case, the task t_{ls} is not canceled in the vector $\mathbf{L}^T$. That is to say, this vector stores the tasks that can begin its execution, but have not started yet. Therefore, the task t_{ls} will continue belonging to the group that certify the lists subset, in order to be evaluated in the following selection. In the case in which the task t_{ls} interrupts another, the interrupted task will change to a waiting state. This task will enter in the following selection process of tasks ready for execution as any other task. Therefore, such a task that was being executed until that moment and is interrupted, will belong to the lists of the $\mathbf{L}^T$ vector, as if it was a new task that is ready to be executed insofar as possible. That is to say, the waiting state of a task j is considered as if it was a new task j1 with computation time value equal to the remaining computation time of the task j, and furthermore, it is preceded by other task j2 build from the part of j already computed. This situation implies that the completion time of the new task j1 increases in a value equal to the corresponding value of computation time of the task j2. When the task t_{ls} starts its execution, the initialization of the start time of t_{ls} (t_crea_{tls}) is performed. Finally, it is necessary to test every single ready tasks in order to delay the tasks which initially were considered as potentially recurrent, and finally they were not.

8. RESULTS

The algorithm presented has been applied to a system of bank notes sheets inspection using computer vision. The structure of the system is composed of a CPU and four IAPB'S. The high volume of data handled by the application together with the strong requirements in the production, demand the use of parallel image processing algorithms. In order to spend the lowest amount of time devoted to the processing, it is necessary the use of different optimization strategies depending on the level of processing considered. In some processes at low level, such as point-to-point operations between images, a division of the total image in regions for each IAPB is accomplished. Each one of them will be computed in parallel with respect to the others. Other processes at low level, such as reading certain parts of an image, imply to make use of the CPU and one IAPB jointly. In this case it is necessary the optimization of the computer resources of the system. On the other hand, the top-level process should be executed in the system CPU. Thus, such tasks will share the processor with the ones described in the previous point.

In the previous paragraph, it is clear the need of accomplishing an adequate spatial and time distribution of the processes between the existing processors, in order to achieve an optimization of the application. It is important to emphasize that this assignment is going to depend on the inspected product itself (bank notes). Therefore, it must be flexible, that is to say, depending on the size and the number of bank notes to inspect the assignment of processes can result different. In the solution presented two independent modules can be distinguished: one for the inspection of the principal face of the sheet and other for the back. Each one of them has the following composition: CPU, IAPB1, IAPB2, IAPB3 and IAPB4. In the table shown below, the numbers of the tasks are related to its corresponding function, computation time and type. The precedence relationship is represented in figure 2.

The space-time distribution of the tasks for this example (case a in computation times) is indicated in figure 3. It can be observed that for this values the CPU is the most critical processor. This processor only presents deadline times while the

Process	*Tasks*	*Computation Time*	*Type of task*
Acquisition	1, 2, 3, 4	50	T_IAPB
Synchronize	5	1	T_CPU
To read master image ROI	6,14,22,30,38,46,54,62	4	T_CPU/IAPB
To calculate point of the master	7,15,23,31,39,47,55,63	2	T_CPU
To read points on the image to analyze	8,16,24,32,40,48,56,64	10	T_CPU/IAPB
To calculate points on the image to analyze	9,17,25,33,41,49,57,65	a)20 b)12	T_CPU
To Reconstruct the images	10,18,26,34,42,50,58,66	a)15 b)25	T_IAPB
To comparate the images	11,19,27,35,43,51,59,67	a)20 b)25	T_IAPB
To read data of difference	12,20,28,36,44,52,60,68	5	T_CPU/IAPB
To give a bank note result	13,21,29,37,45,53,61,69	2	T_CPU
To give a total result	70	3	T_CPU

image is being acquired by each one of the IAPB'S, and obviously it still does not have any data to operate with. Also, in the time 351 the CPU remains idle due to the fact that it is not able to operate the data of the IAPB4 until the board has finished the process 67.

However, it can occur that in the inspection of a different product (other type of bank notes), the number of points to calculate may descend, and consequently the values assigned to each one of the tasks to accomplish may be different. An assumption of this type has been considered. Therefore, a new table, analogous to the previous one is presented. Now the computation times for the tasks related to points, reconstruction and comparison is designated as case b. With the new assumption, the space - time distribution is different, as it is observed in figure 4. In this case there are processes of T_CPU type which have modified its execution order, in order to optimize the total time spent by the system. This last consideration is the one which justifies the utilization of the scheduling algorithm presented. In this case, job-shop scheduling applied to computer vision.

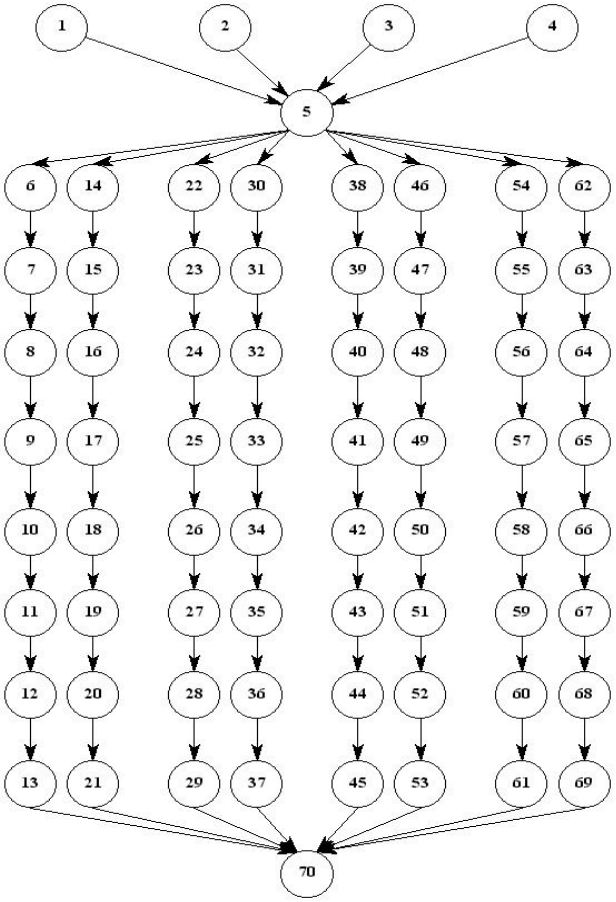
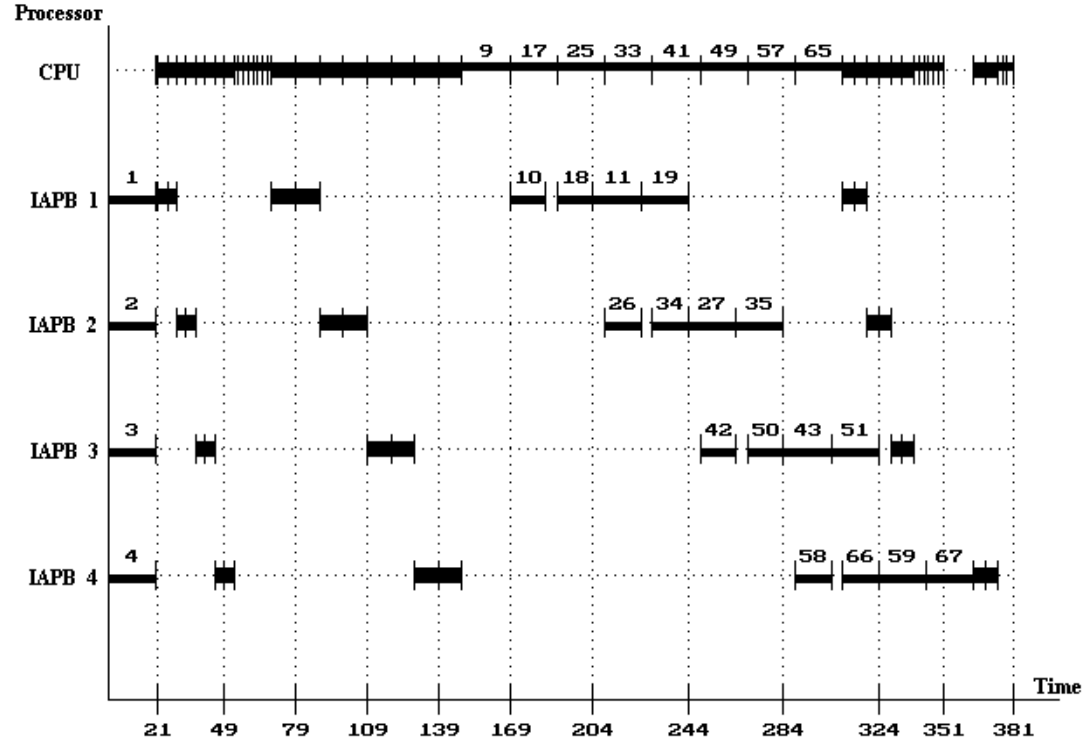


Figure 2. The precedence graph.



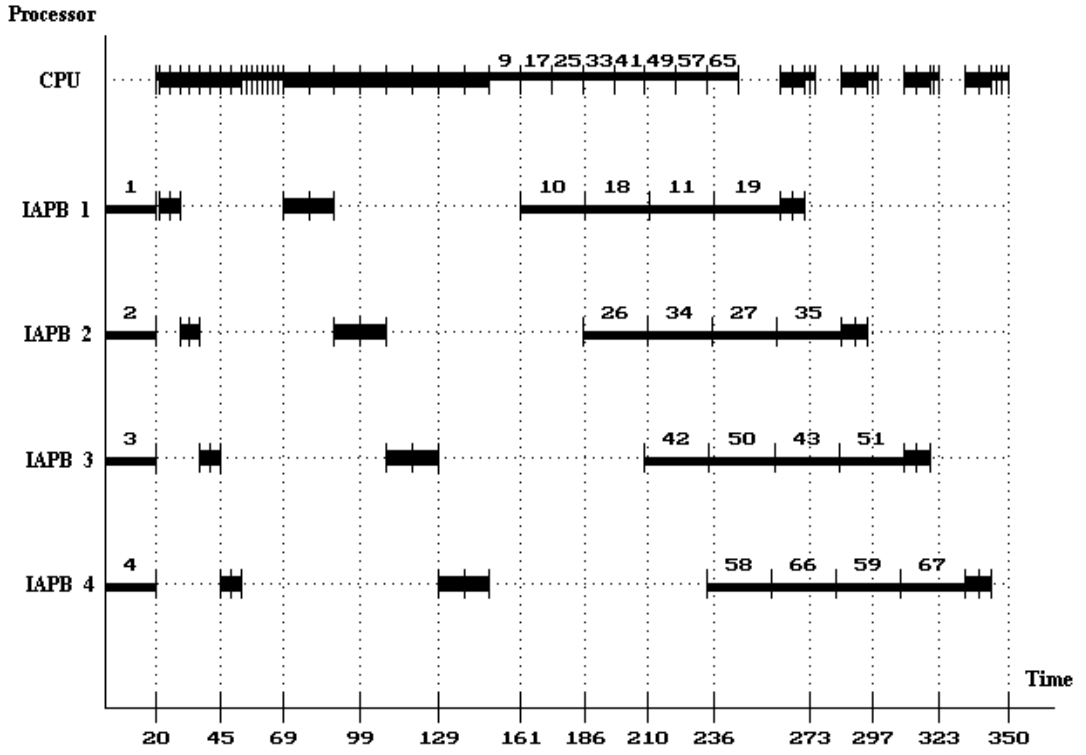


Figure 4. Space-time distribution for the case b.

REFERENCES

1. P.H.Winston, *Artificial intelligence*, Addison-Wesley, 1992.
2. S.X.Bai, J.H.Burhanpurwala, "Real-time scheduling of production with dynamic setup changes", *Proc. of SPIE*, 2063, pp.109-119, 1993.
3. J.A.Stankovic, K.Ramamritham, "Dynamic task scheduling in hard real-time distirbuited systems", *IEEE Software*, 1-3, pp.65-75, 1984.
4. J.Xu, D.L.Parnas, "On satisfying constraints in hard-real-time systems", *IEEE Trans. on Software Engineering*, 19-1, pp.70-84, 1993.
5. J.Mª.Sebastián, F.Torres, O.Reinoso, J.L.Bello, E.Barroso, "Parallel processing and scheduling techniques applied to the quality control of bill sheets", *Proc. 12th IAPR*, IEEE Computer Society, pp.399-403, 1994.
6. L.Sha, R.Rajkumar, J.P.Lehoczky, "Priority inheritance protocols. An approach to real-time synchronization", *IEEE Trans. on computers*, 39-9, pp.1175-1185, 1990.
7. D.I.Moldovan, *Parallel processing from applications to systems*, Morgan Kaufmann Publishers, San Mateo CA, 1993.
8. F.Torres, *Ph.D. Arquitectura paralela para el procesado de imágenes de alta resolución. Aplicación a la inspección de impresiones en tiempo real*, pp.73-75, ETSIIM (UPM), Madrid, 1995.