Job Shop Scheduling via Disjunctive Boolean Formulas

Guillermo De Ita, Yolanda Moyao, Juan Carlos Pérez, Josúe Pérez

Universidad Autónoma de Puebla, México
deita@ccc.inaoep.mx, ymoyao@cs.buap.mx, jeancarlopv@hotmail.com, d.josue.pl@gmail.com

Abstract. Given a project formed by a serie of tasks, we present a method which allow us to build the working teams to perform the project. The teams are formed by employees and according to the ability of them for satisfying the logical requirements of the tasks. The leader of the project determines for each task, logical contraints indicating the different employees who can perform the task. Each employee has associated a cost for participating in a task, so that we have different ways to perform the total project and with different costs.
We present a method for processing the logical constraints associated to each task. Our method builds all the possible working teams and also it shows the cost associated with each working team. The method also finds the working team with minimum cost for doing the total project.
Keywords: Satisfiability Problem, Job Shop Problem, Working Teams, Disjunctive Forms

1 Introduction

During the seventies, computer scientists discovered the scheduling as a tool for improving the performance of computer systems. Furthermore, scheduling problems have been investigated and classified with respect to their computational complexity. During the last few years, new and interesting versions of scheduling problems have been formulated in connection with flexible manufacturing and logic proposals [1].

Scheduling is the problem of allocating limited resources to tasks (activities) over time. Scheduling is also considered as a decision-making process that has as a goal the optimization of one or more objectives.

The commonly used Critical Path Method (CPM), assumes that unlimited resources are available, and that activities requiring a common resource can be carried out in parallel [7]. But when the resources are limited and have to be shared during the performance of the project, then an important problem is the allocation of those scarce resources for competing activities in order to minimize overall project duration.

Here, we consider scenarios where a set of tasks are performed through specialized equipment or by specialized employees. We consider equipment or specialized personnel such as the common resources to be used in order to accomplish the project.

In this article, we consider the Job-Shop problem without restrictions of the time windows tasks. So, we want to schedule a set of m tasks, and where the tasks share common resources, in this case, a resource means an employee who must participate in a working team for performing each task. Then, different sets of conflicting taks are formed dinamically according with the order of performing of the previous tasks and the employees performing those tasks.

This version of the Job-Shop problem continues being a NP-hard problem since the possibilities of permutations of conflicting employees performing the tasks. In this class of instances, the explosive number of permutations for executing the tasks depend on the number of subteams involved in each task. We present here, a novel method for solving this class of Job-shop problem, our method is based in the description of each task through disjunctive forms.

We are interested in computing all the different working teams that can be formed to develop a project in large corporative companies. The leader of the project determines the logical constraints for each task in the project. Such contrainsts are translated to disjunctive forms. Thus, the constraints for any task in the project will be associated with a disjunctive form.

Joining all disjunctive forms of the tasks in the project, a new conjunctive boolean formula Σ is formed. And in this case, the set $SAT(\Sigma)$ (the set of satisfied assignments of Σ) determines the different working teams that can be formed to develop the project effectively. While $\#SAT(\Sigma)$ (the number of satisfied assignments) give us how many different teams can be formed to develop the project.

We have designed a method to build the set $SAT(\Sigma)$. Thus, we find the working teams which can perform the project and also, we determine the working team with minimum cost, assuming that the participation of each employee in a task has a fixed cost. This kind of automatization will help to the companies to save money and time, by making the working team selection process more efficient, transparent and dynamic.

2 Notation and Preliminaries

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n projects. One project:$P_i = (t_{i1}, \ldots, t_{im_i})$ formed by a set of m tasks . Each project $P_i \in \mathcal{P}$ is performing by a working team, and each working team is a set of employees.

Each project $P_i \in \mathcal{P}$ consists of a series of consecutive and interdependent tasks: $(t_{i1}, \ldots, t_{im_i})$. There is an order among the tasks of a project given by the interdependency of them, in such a way that certain tasks cannot begin until all the tasks they depend on are completed.

A dependency graph or precedence graph (DAG) $G_i = (\mathcal{T}, E(G_i))$ is built for representing each project $P_i \in \mathcal{P}$. The nodes in G_i represent tasks and we join the last task of the project with a special node labeled as P_i. Each edge $(v, w) \in E(G_i)$ meaning that the task w depends directly on the task v. The precedence constraints between tasks are represented on this DAG through the edges. Then, an edge represents a precedence relationship between its tasks.

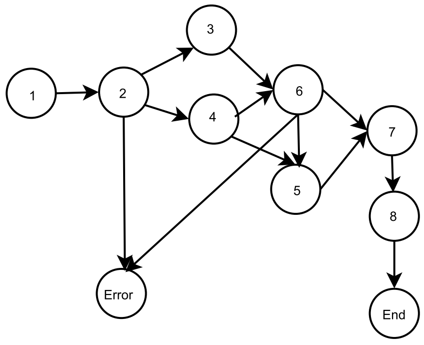

Fig. 1. A DAG for a project

We denote by t_{ik} the k-th task from the project P_i. Usually, in a DAG three nodes are common, the initial task (or sometimes the initial tasks), a node representing when an error has been peformed (the Error node), and the final task, in our figure, it is represented by the *End* label node. The duration of each task is known, and the task requires a set of employees throughout its performance.

Let $X = \{x_1, x_2, ..., x_n\}$ be a set of n boolean variables, each variable $x_i \in X$ represents an employee from a company. A *literal* is either a variable or a negated variable. The bar over the variable, $\overline{x}$ for instance, will denote a negated variable. Sometimes, we also use $\neg$ as the negated operator. Let $L = \{x_1, \neg x_1, ..., x_n, \neg x_n\}$ be the set of literals over a set of variables X, and let $\bot$ and $\top$ be two constants that represent the *false* and *true* values, respectively. $|A|$ denotes as usual, the number of elements of a set A. We use $v(l)$ to indicate the variable involved by the literal l.

The disjunction of different literals is called a *clause*. While the conjunction of different literals is called a *phrase*.

For $k \in \mathbb{N}$, a $k-clause$ ($k-phrase$) is a *clause* (*phrase*)consisting of exactly k *literals*. A variable $x \in X$ appears in a clause or a phrase c if x or $\overline{x}$ is an element of c. Let $v(c) = \{x \in X : x \textit{ appears in } c\}$.

A conjunctive form (CF) is a conjunction of *clauses*. A k-CF is a CF containing only k-clauses. A disjunctive form (DF) is a disjunction of *phrases*. A k-DF is a DF containing only k-phrases.

An assignment s for a boolean formula F is a boolean function $s : v(F) \rightarrow \{0, 1\}$. An *assignment* can be considered also as a set of non complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l *true* and $\bar{l}$ *false*.

Considering both a clause c and an assignment s as a set of literals, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$, and if for all $l \in c$, $\bar{l} \in s$ then s falsifies c. Assuming also a phrase p as a set of literals, p is satisfied by an assignment s if and only if for all literal $l \in p$ then $l \in s$. Then, considering p and s as a set of literals; if $p \subseteq s$ then the phrase p is satisfied by s.

A CF Σ is satisfied by an assignment s if each *clause* in Σ is satisfied by s and Σ is contradicted if it is not satisfied. A DF Σ is satisfied by an assignment s if any *phrase* in Σ is satisfied by s and Σ is contradicted if all phrase in DF is not satisfied by s. We call s a model of Σ if s is a satisfied assignment of Σ.

If F represents a literal, a clause, an assignment, a CF or a DF then $v(F)$ will denote the set of variables contained in F. For example, if $F = \{\neg x_1, x_2\}$ or $F = \neg x_1 \vee x_2$, then $v(F) = \{x_1, x_2\}$.

Let Σ be a boolean formula, $SAT(\Sigma)$ is the set of models that Σ has over $v(\Sigma)$. Σ is a contradiction or unsatisfiable if $SAT(\Sigma) = \emptyset$. Let $\#\mathrm{SAT}(\Sigma) = |SAT(\Sigma)|$ be the cardinality of $SAT(\Sigma)$. The SAT problem consists in determining if Σ has a model. The $\#SAT$ counting problem consists of counting the number of models of Σ defined over $v(\Sigma)$. The SAT problem is a classic NP-complete problem.

3 Defining a Job Shop Model

A Job-shop scheduling problem can be formulated as a set of n jobs (multi-project) $P = \{P_1, \ldots, P_n\}$ to be scheduled on m machines, or by m employees. Each job (or project) P_i is formed by n_i consecutive tasks $P_i = (t_{i1}, \ldots, t_{in_i})$, so there is a sequential order among the tasks in the same project. The task (t_{ik}) represents the k-th task of the job P_i.

Each task t_{ij} has associated a processing time w_{ij}, and each project P_i must be achieved before a due time dt_i. Let s_{ij} be the start time for scheduling the task t_{ij}. For $i = 1, \ldots, n$, the total time that a project P_i needs for completing all its tasks is its completion time and it is denoted as CT_i, While TD_i denotes the total tardiness spent by A_i in order to achieve the last task P_i [4].

Let C_{max} be the makespan (total completion time of the multi-project system), and TD be the total tardiness of the system. The multi-objective optimization problem consists roughly in finding a schedule of the n projects that minimizes the makespan and the total tardiness [6]. Both objectives can be formulated as minimizing the following functions:

$$f_1 = C_{max} = Max\{s_{in_i} + w_{im_i} | i \in [1 \ldots n]\}$$
$$f_2 = TD = \textstyle\sum_{i=1}^{n}[max(0, s_{im_i} + w_{im_i} - dt_i)]$$

Corporate companies developing projects have to select in a dynamic way the different working teams to solve the requirements of a project. These companies have a finite staff formed by "n" employees which we will denote by the set: $X = \{x_1, \ldots, x_n\}$.

In practical sceneries, each project $P_i \in \mathcal{P}$ could be accomplished in different ways, that is, each task in the project can be realized by different employees (working teams). And different costs are allocated according to the participation of employees for performing a determined task.

The supervisor or leader of a project P may specify how different employees can form a team to do a determined task. So, a logical formula F_i is associated to each task $t_i \in P$. F_i indicates the employees who must participate, who must not participate and who is not relevat to participate for performing the task t_i.

Project managers propose the staff selection to form working teams based on the restrictions given for each task. Such restrictions are formed according to the capabilities of employees and the requirements that must be accomplished in the project. For example, a kind of restriction could be that two employees have the same abilities, or that two specific employees can not work in cooperative way with each other, etc. In general, a logical formula is established for determining the relationship among employees doing a determined task.

The project manager defines such restrictions (capabilities or requirements)for each task in the project. For example, we can identify the following cases that model constraints between a pair of employees.

Case 1: (x_i). It indicates that it is mandatory that the employee x_i participate in the current task.
Case 2: $(x_i \vee x_j)$. The clause is unsatisfiable when $x_i = 0$ and $x_j = 0$, indicating that either of the employees x_i or x_j must participate in the team, or that both can be on the same team.
Case 3: $(x_i \vee \neg x_j)$. The clause is unsatisfiable when $x_i = 0 \wedge x_j = 1$. It indicates that if x_j is on the team, then x_i should be on the same team. In the same way, if x_i is not in the team then x_j must not be in it.
Case 4: $(\neg x_i \vee \neg x_j)$. The clause is unsatisfiable if $x_i = 1 \wedge x_j = 1$ indicating that either x_i or x_j employees should not participate in the team, both of them should not. Thus, we must avoid having both employees in the same team.

Lemma 1. *Let F be a propositional formula then F can be translated in a logical equivalent formula expressed in disjunctive form. [3]*

According to the previous lemma, the logical constraints given by the leader of the project can be rewritten as disjunctive forms. Then, we associate with each task t_i of a project P a disjunctive form $DF(t_i)$ which denotes the different working teams which can perform the task t_i.

Each disjunctive form $DF(t_i)$ can be codified via vector of n positions of 1,0 and -1 according to the position of each employee he may either participate, not participate, or he must not participate, in the current working team.

The set of vectors, one for each disjunction from $DF(t_i)$ conform the rows of a matrix M_i. Thus, each task t_i has associated a matrix M_i containing the different subteams which can perform the corresponding task.

If we join (with the conjunction operator) the m disjunctive forms, then a new propositional formula $\Sigma = DF(t_1) \wedge DF(t_2) \wedge \ldots \wedge DF(t_m)$ is formed. Σ will be a conjunction of disjunctive forms. Furthermore, any assignment that satisfies Σ represents a way to form a proper working team that will execute the total project.

We also assume that the participation of an employee in a subteam to perform a task has a fixed cost and then, a total cost is formed according to the working team which can develop the total project.

Therefore $SAT(\Sigma)$ contains all the possible teams that can be formed to implement the project effectively. We present in the following section, how to build the set $SAT(\Sigma)$, and also how to determine the model of Σ with minimum cost.

4 A Logical Procedure to Form Working Teams

A particular example of Schedulling for the optimization of resources is to minimize costs. Given just one project $\mathcal{P}=(t_1,\ldots,t_m)$, where each task $t_i \in P$ has associated a set of boolean constraints, we can build a boolean formula Σ whose models represent the teams which can develop the total project. Each working team is represented by a model of Σ.

In those models of Σ the logical value 1 indicates that a particular employee must be a component of the working team, the value 0 expresses that the corresponding employee may not be a part of the working team. We also use the value -1 to indicate that the respective employee must not participate in that working team.

The constraints indicating which employees would develop a task $t_i \in P$ are translated in a disjuntive form $DF(t_i)$. Such $DF(t_i)$ expresses the different ways to perform the task t_i, so the formula includes the different subteams wich can perform the task. Each disjunctive form $DF(t_i)$ is encoded through a row of a matrix of constraints, denoted as Mi.

Now, we present a method to build the models which satisfy all the constraints to do the project. First, we present in table 1, the multiplicative patterns used when two consecutive tasks t_i and t_{i+1} are considered to be performed by the same employee.

The first three rows of the table 1 codify the result obtained when the participation of an employee is necessary in a task and then he is necessary in the total project. Similarly, if an employee must not participate in a task (value -1) consequently he/she will not participate in the project at all (rows 4,5 and 6 in the table 1). On the other hand, the symbol $\perp$ is used when two constraints can not be both satisfied, e.g. when an employee must participate in a task but he has not participate in the following tasks.

Procedure: Building Models for Disjunctive Forms.

Step 1: Let M_i be a matrix containing the dijunctive forms to perform the task t_i. Let $Costs[i] = w_i, i = 1,\ldots,n$ be a vector of weights, each weight w_i represents the cost of the participation of the employee x_i in the project.

Step 2: We apply a special matrix multiplication on the m matrix-tasks, as:

$$MP = M_1 * (M_2)^T * ... * (M_m)^T \tag{1}$$

Table 1. multiplicative patterns

$1 * 1 = 1$
$1 * 0 = 1$
$0 * 1 = 1$
$0 * 0 = 0$
$-1 * 0 = -1$
$0 * -1 = -1$
$1 * -1 = \perp$
$-1 * 1 = \perp$

Where M^T is the transpose matrix from M. When a row versus column matrix multiplication is done, the multiplicative patterns on table 1 are applied.

Step 3: The matrix multiplication in equation (1) is done in iterative way and the rows containing the symbol $\perp$ are removed in each resulting submatrix. Since such rows represent unsatisfiable constraints.

Step 4: After obtaining the final matrix MP without unsatisfiable rows, we associate to each row in MP a cost which represent the charge that the corresponding working team requests to perform on the total project. If a row $r_k \in MP$ is represented by a binary vector: $(b_1, b_2, \ldots, b_n)$ then the cost of that working team will be:

$$Cost(r_k) = \sum_{(b_i=1)\in r_k} w_i$$

Then we will have an equal number of costs as rows there exist in the matrix MP.

Step 5: The resulting rows for matrix MP, are sorted in descending order with respect to the estimated costs, and the row MP with minimum cost will be selected as the working team for carry out the total project.

A simple example illustrates the way in which our method works.

Example 1. Let t_1, t_2 and t_3 be three tasks forming a project P. The constraints which determine how the different employees of the company can perform each task are:

$$t_1 = (x_1 \wedge \neg x_4) \wedge (x_2 \vee x_3) \vee ((x_2 \oplus x_5) \wedge x_3)$$
$$t_2 = (\neg x_1 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_5)$$
$$t_3 = ((x_2 \wedge x_5) \vee (\neg x_1 \wedge x_3)) \wedge x_4$$

A vector of costs is given according to the salary of each employee:

$$Costs = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ 250 & 400 & 550 & 320 & 500 \end{bmatrix}$$

By translating the constraints defined by the leader of the project as disjunctive forms, we obtain the following disjunctive constraints:

$$t_1 = (x_1 \wedge \neg x_4 \wedge x_2) \vee (x_1 \wedge \neg x_4 \wedge x_3) \vee (x_2 \wedge \neg x_5 \wedge x_3) \vee (\neg x_2 \wedge x_5 \wedge x_3)$$

$$t_2 = (\neg x_1 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_5)$$

$$t_3 = (x_2 \wedge x_4 \wedge x_5) \vee (\neg x_1 \wedge x_3 \wedge x_4)$$

And then, such constraints are codified via the matrix M_1, M_2 and M_3, defined as:

$$M_1 = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 & -1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} -1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Considering the multiplicative patterns mentioned in table 1, we obtain the fist resulting matrix $M_1 \times M_2^T$.

$$\begin{bmatrix} & & M_1 & & \\ 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 & -1 \end{bmatrix} \times \begin{bmatrix} & M_2^T \\ -1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} & & MP_i & & \\ \perp & * & * & * & * \\ 1 & 1 & 0 & -1 & 1 \\ \perp & * & * & * & * \\ 1 & 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ 1 & \perp & * & * & * \\ -1 & -1 & 1 & 1 & 1 \\ 1 & \perp & * & * & * \end{bmatrix}$$

The symbol '*' represents a pattern which can be substituted by 1 or 0 as in fact, it doesn't matter what its value is, because the row is unsatisfiable.

Now, keeping only the rows that satisfy the constraints of the project, we form a new matrix that is multiplied by M_3^T.

$$\begin{bmatrix} & & MP_i & & \\ 1 & 1 & 0 & -1 & 1 \\ 1 & 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} & M_3^T \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} & & MP & & \\ 1 & 1 & 0 & \perp & * \\ 1 & \perp & * & * & * \\ 1 & 1 & 1 & \perp & * \\ -1 & 1 & \perp & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & \perp & * & * & * \\ -1 & -1 & 1 & 1 & 1 \\ 1 & \perp & * & * & * \end{bmatrix}$$

And as the final matrix must not contain unsatisfied rows, we reduce the final matrix to:

$$\begin{bmatrix} & MP & & & \\ -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

And finally, we compute the costs associated to the rows of MP, according to the cost given by the participation of each employee in the project.

$$\begin{bmatrix} X_1 & X_2 & X_3 & X_4 & X_5 \\ -1 & 1 & 1 & 1 & -1 = 1270 \\ -1 & -1 & 1 & 1 & 1 = 1370 \end{bmatrix}$$

So, we conclude that the project can be carried out by means of the following working team:

$$(x_2 \wedge x_3 \wedge x_4) \text{ and } (x_3 \wedge x_4 \wedge x_5).$$

And the working team with minimum cost is $(x_2 \wedge x_3 \wedge x_4)$.

In sum, the proposed heuristic, starts by solving the problem of creating the teams that can meet all the project tasks P, searching for the one with the minimal cost. This approach to the problem is very similar to the constraint satisfaction problems [13] which works with graphs and not with disjunctive forms like the one we have just proposed.

5 Conclusions

Nowaday, most companies have to form working teams in order to effectively do a project. The task of a project manager is to consolidate the staff and to create teams that can achieve concrete results, under certain types of restrictions. Such restrictions are concerning with empathy, capabilities, abilities and knowledge among the employees who have to accomplish the project. These relations or restrictions are often complex for the project manager and he must take the best decision.

Our proposal for forming the working teams, is based on processing disjuntive forms specifying the employees who could participate and those who must not, for performing specific tasks. The satisfiability assignments resulting from the boolean formula represent the different ways to perform the total project.

We present a method for processing the disjunctive forms linked to each task. Our method can also determine which working team is associated to the minimum cost, taking into consideration that the participation of each employee in each task has a specific cost.

References

1. Brucker P., *Scheduling Algorithms*, Springer-Verlag, 2007.

2. De Ita G., Polynomial Classes of Boolean Formulas for Computing the Degree of Belief, *Advances in Artificial Intelligence LNAI 3315*,pp.430-440,2004.
3. Gallier J.H., *Logic for Computer Science*, John Wiley & Sons, 1987.
4. Garrido A., Salido A., Barber F., López M.A., *Heuristic Methods for Solving Job-Shop Scheduling Problems*, (2000), citeseer.ist.psu.edu/402791.html
5. Johnsonbaugh Richard, 1993, *Matemáticas Discretas*, Editorial Prentice Hall. 4ta edición.
6. Melab N., Mezmaz M., Talbi E.-G., Parallel cooperative meta-heuristics on the computational grid. A case study: the bi-objective Flow-Shop problem, *Science Direct Parallel Computing* 32, (2006), pp.643-659
7. Patterson J.H. A comparison of exact approaches for solving the multiple constrained resource project scheduling problem, *Management Science*, Vol. 30, (1984), pp. 854-867.
8. Fred S. Roberts, *Graph Theory and Its Applications to Problems of Society*, Edit. SIAM Collection, 1978.
9. Germina K.A., Hameed S.K., On signed paths, signed cycles and their energies, *Applied Mathematical Sciences*, Vol. 4:(70), 2010, pp. 3455-3466.
10. Kota P. S., Subramanya M.S., Note on path signed graphs, *Notes on Number Theory and Discrete Mathematics* 15:(4), 2009, pp. 1-6
11. Roth, D., On the hardness of approximate reasoning, *Artificial Intelligence 82*,pp 273-302,1996.
12. Vadhan, S. P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*,pp. 398-427, V31, N2, 2001.
13. David Poole, Alan Mackworth, Randy Goebel, Computational Intelligence A Logical Approach, *Oxford University Press*,pp 147-150.