# A Tabu Search Optimization Module for Scheduling

## Design and Integration in the Open Source Tool LibrePlan for Project Management

A. González-Sieira, A. Bugarín, M. Mucientes
Centro de Investigación en Tecnoloxías da Información (CITIUS)
University of Santiago de Compostela, Spain
{adrian.gonzalez, alberto.bugarin.diz, manuel.mucientes}@usc.es

J. Morán
Igalia S.L.
Pontevedra, Spain
jmoran@igalia.com

*Abstract*—**In this paper we describe a tabu search based approach to the scheduling problem in project management and its integration in the LibrePlan open source software tool. A description of the key elements of the algorithm is provided, together with the execution of three use cases with LibrePlan, that shows how feasible plannings are obtained, achieving a noticeable reduction in the makespan of the projects.**

*Keywords:* ***tabu search, metaheuristics, scheduling, open source, project management software tools.***

## I. INTRODUCTION

A *Flexible Job Shop* problem [5] is a scheduling problem where the tasks are developed in an environment with limiting parallel resources and each task can only be processed by a subset of those resources. Optionally the task may need to be processed more than once. The optimal assignation of the resources demands to minimize the time necessary to complete all tasks and their associated cost. To solve this problem it is necessary to take into account the task dependencies and the date restrictions, both at the beginning and at the end, and the different work load of each resource. In a real project there is also a calendar associated to each resource that defines its availability (due to holidays, maintenance, etc.) which must be considered to generate valid assignations and fit to the restrictions imposed by the project information.

The information about the dependencies between tasks and the resource assignations can be modeled by a directed graph [6] where the nodes are the tasks associated to the project, and the rest of the information is contained in the arcs; thus, a search space which contains the set of graphs obtained from changing the resource assignation arcs is defined.

Different approximations have been described in the literature to solve this type of problems: genetic algorithms [3], hybrid search [2] and ant colonies [7], [9], etc. Genetic algorithms are the most significant approximation, as they explore the search space by combining and modifying different candidate solutions, which allows having more control about the diversity of the optimization process. The main disadvantage of these methods is their high computational cost: a very relevant factor taking in account the environment where the module will be deployed, as the result should be shown to the user in a reasonable time.

This work was supported by Igalia, S.L. and the Xunta de Galicia under grants 2011/CG247 and PGIDIT10DPI031E. Support from Spanish Ministry of Economy and Competitiveness under grants TIN2011-22935 and TIN2011-29827-C02-02 is also acknowledged. A. González-Sieira is supported by a FPU grant (ref. FPU12/05712) from the Spanish Ministry of Education, Culture and Sports. M. Mucientes is supported by the Ramón y Cajal program of the Spanish Ministry of Economy and Competitiveness. This work was also supported in part by the European Regional Development Fund (ERDF/FEDER) under the project CN2012/151 of the Galician Ministry of Culture, Education and Universities.

Another approximation is to use a descent search algorithm to do the optimization starting from any point of the search space. The candidate solution is used to generate a neighborhood. The search space is explored iteratively, selecting the most promising neighbor as in an informed search algorithm. To keep the diversity of the exploration process and avoid local minimums, a tabu search algorithm may be used. Thus, the selection of the seed for the next iteration is limited by the history of the changes made in previous iterations. In this way both the exploration loops and the local minimums are avoided as the exploration direction is forced to take other less promising paths to preserve diversity. The main advantage of this technique is that, if the neighborhood size is at the same time small enough and representative, the algorithm may converge to the optimal solution very fast, thus reducing the execution time. This is the most important reason for using this approximation to solve the scheduling problem.

In this paper we describe the implementation of a scheduling module that was integrated into LibrePlan [8], a web based open source tool for project management. The main function of this module is to optimize the time, and indirectly the cost, for completing a given set of tasks in a limiting resource environment, obtaining the best resource assignations, and observing all the restrictions introduced by the user.

The optimization is done by an iterative process that performs re-assignations on the tasks in the critical path to shorten it. It takes in account elements managed by LibrePlan such as: resource calendars, restrictions of which resources can be assigned to each task, dependencies, etc. Once the optimization process is completed, the assignations can be inspected and modified before saving the changes. The module was developed in J2EE and integrated into LibrePlan, and the optimization algorithm was developed in Java.

This paper is organized as follows: first, the tabu search based optimization algorithm is detailed, including all the components that make up it. Section 3 describes the integration of the algorithm into LibrePlan, and Section 4 shows a use case. Finally the conclusions of this work are presented.

## II. OPTIMIZATION ALGORITHM

To use a tabu search process two main elements need to be determined: the neighborhood function and the estimation cost

function. As the solution of the scheduling problem is represented as a directed graph, its cost can only be reduced by introducing changes in the critical path. So, the neighborhood of a candidate solution will contain graphs obtained by re-assigning critical tasks to other resources, or to the same resource but in different processing order. It is necessary to keep reduced the number of neighbors to improve the efficiency of the algorithm, but always maintaining the representativeness of the set of changes.

The estimation cost function acts as a heuristic to guide the exploration process through the most promising zones of the search space. It was implemented following the principle that the temporal marks of the predecessor and successor tasks of the one to be re-assigned will not change, so accumulating these marks to the processing time in the new resource is a good estimation for the new cost of the solution.

The tabu tenure is also an important concept managed by the algorithm. It represents the number of iterations that the prohibition of moving to a neighbor will last. This parameter should not be static: it will be set taking into account the complexity of the neighborhood in order to avoid a different behavior depending on a heuristically defined parameter.

The following subsections detail the structure of the solution graph, the generation of the initial seed, the neighborhood function, the estimation cost function and the complete tabu search process.

### A. Solution graph

The dependencies between tasks and the order of the assignations can be modeled as a directed graph [6], called the *solution graph*. Each node in the graph represents a task, the atomic work elements of the algorithm. Each task is defined by the number of work hours to complete it and the calendar associated to the task. It is a common situation that all tasks of the project share this calendar, but it is not mandatory (there may be tasks developed in other regions with different holiday days). Two artificial nodes, *0* and * nodes, are added to the graph to represent the beginning and the end of the execution of the tasks. These nodes do not have any work load, neither can be assigned to any resource.

The arcs of the graph codify different information depending on their type: *P* arcs represent dependencies between tasks (the algorithm must respect the order given by these arcs to perform valid assignations); *M* arcs represent the processing order of the tasks assigned to the same resource (modifications over these arcs represent re-assignations of the tasks to other resources, or in the same resource in different processing order); and *D* arcs connect each task with nodes 0 and * when there is no equivalent connection of different type, to keep the connectivity of all tasks in the solution graph.

To execute the optimization algorithm, it is necessary to generate an initial graph, where the assignations are done stochastically, containing all the relevant input data for the optimization process. To generate this graph the following information is used:

- The **set of tasks**, each one with its calendar and associated work hours. Not always the whole set of tasks is used to generate the solution graph, since in LibrePlan the tasks with lower relevance can be grouped into higher-level tasks and be omitted in the optimization algorithm. Tasks considered in the optimization process are called *planning points*.
- The list of **resources**: in LibrePlan some assignation criteria (location, category, etc.) can be imposed to the tasks, so only the resources that match those criteria can do the work.
- The list of **dependencies** between tasks.

### B. Initial graph generation

To generate the initial graph, first the artificial nodes *0* and * are added to define the opposite sides of the graph. Then, a node for each task that is also a planning point is created. In each node, the algorithm stores the number of working hours and the calendar associated to the task. The initialization process is detailed in Algorithm 1, where $t_0$ is the initial task, $t_*$ the final one, $SJ[t]$ are the dependent tasks of $t$ through $P$ arcs and $PM[t]$ is the previous one in the processing queue of $t$.

Each node in the graph is labeled with the earliest starting time of the associated task, $s_x = l(0,x)$, the later tail time, $t_x = l(x,*)$, and the processing time of the task in the assigned resource, $p_{x,r}$. The cost of a solution graph, also called *makespan*, is calculated by adding all of the processing times of the tasks in the longest path that connects the nodes *0* and *. This is the critical path of the graph, $l(0,*)$

**Algorithm 1** Graph initialization

add $t_0$ y $t_*$
add $t \; \forall \; t \in project \cap planning$
add P arcs $(t, t') \; \forall \; t \in dependents(t)$
add P arcs $(t_0, t) \; \forall \; t/PJ(t) = \emptyset$
add P arcs $(t, t_*) \; \forall \; t/SJ(t) = \emptyset$
$assignables = SJ(t_0)$
**while** $assignables \neq \emptyset$ **do**
  extract randomly $t_a \in assignables$
  select randomly $m_a \in resources(t_a)$
  **if** $PM(t_a) = \emptyset$ **then**
    add arc M $(t_0, t_a)$
  **else**
    add arc M $(PM(t_a), t_a)$
  **end if**
  $assignables = assignables \cup (SJ(t_a) - t_*)$
**end while**
**for** $m \in resources$ **do**
  **if** $last\_assignation(m) = \emptyset$ **then**
    add arc M $(t_0, t_*)$
  **else**
    add arc M $(last\_assignation(m), t_*)$
  **end if**
**end for**

**Algorithm 2** *generatePathP*(*criticalPaths*)

$op$ = first task of a random critical path
**while** $op \neq t_*$ **do**
  **if** $SJ[op] \neq \emptyset$ and $SJ[op]$ is critical task **then**
    $P = P \cup SJ[op]$
    $op = SJ[op]$
  **else if** $SJ[op] \neq \emptyset$ and $SM[op] \neq \emptyset$ **then**
    $P = P \cup SM[op]$
    $op = SM[op]$
  **end if**
**end while**

### C. Neighborhood function

The neighborhood function used in this work was extracted from the work by Mastrolilli et al. [4]. The neighbors of a solution graph are obtained changing the assignments of tasks to other resources, or in the same resource with different processing order. In order to minimize the *makespan* of the neighbors with respect to the original graph, only the tasks belonging to the critical paths are re-assigned. Otherwise the value of the *makespan* could increase or remain the same, since the changes made on the graph do not guarantee the reduction of the number of critical tasks and, in the worst case, this could increase it.

To reduce the number of neighbors, a special definition of critical path is used: the path *P*. This path is obtained by selecting tasks belonging to any critical path in the graph in a random way, as detailed in Algorithm 2, where $SM[t]$ is the next task in the processing order of the resource assigned to $t$. To generate the neighborhood of a solution graph, the tasks contained in its path *P* are reassigned to other resources capable to process them, or in the same resource but with different processing order. The length of this path gives the number of neighbors obtained from the graph. Each re-assignation is defined by a task, $v$, and a resource, $k$. This is also called $k$-insertion of $v$. To perform an insertion of $v$ in resource $k$ two actions must be done: first, the *M* arcs adjacent to the task $v$ are removed; then $v$ is assigned to the resource selecting an adequate processing order, adding the corresponding *M* arcs to the graph.

**Algorithm 3** *Nopt1*(*S*)

$P = generatePathP$(critical paths in $S$)
$F_{vk} = \emptyset$
**for** $v \in P$ **do**
  **for** $k \in M_v$ **do**
    $S' =$ copy of $S$
    remove arc $PM[v] \longrightarrow v$ in $S'$
    remove arc $v \longrightarrow SM[v]$ in $S'$
    $\mu(v) = \emptyset$
    $s_v^- = s_{PJ[v]} + p_{PJ[v]}$
    $t_v^- = t_{SJ[v]} + p_{SJ[v]}$
    $R_k = (x \in Q_k | s_x + p_x > s_v^-)$
    $L_k = (x \in Q_k | p_x + t_x > t_v^-)$
    **if** $L_k \cap R_k \neq \emptyset$ **then**
      $\check{S}_{vk}$ = $k$-insertion of $v$ as succesor of $L_k \backslash R_k$ or predecesor of $R_k \backslash L_k$ in $S'$
    **else**
      $\check{S}_{vk} = k$-insertion of $v$ in any position after $L_k \backslash R_k$ and before $R_k \backslash L_k$ in $S'$
    **end if**
    $F_{vk} = F_{vk} \cup \{\check{S}_{vk}\}$
  **end for**
**end for**
**return** $F_{vk}$

The neighborhood function, *Nopt1* (Alg. 3), is defined as the set of re-assignations, $F_{vk}$, obtained from inserting the tasks $v \in P$ after all operations in $L_k \backslash R_k$ and before all operations in $R_k \backslash L_k$, where $R_k$ and $L_k$ are the subsets of tasks located before and after the removed operation $v$, contained in the processing queue of the resource $k$, $Q_k$.

It is demonstrated that the set of solution graphs obtained after inserting $v$ after all operations of $L_k \backslash R_k$ and before all operations of $R_k \backslash L_k$ contains an optimal change. The best solutions are always obtained after inserting $v$ as a successor of the last task in $L_k \backslash R_k$, or as a predecesor of the first task in $R_k \backslash L_k$. If $L_k \cap R_k = \emptyset$, all possible insertions are optimal, and one of them is selected stochastically.

**Algorithm 4** *makespan* estimation

calculate $L_k$ and $R_k$
**if** $L_k \cap R_k = \emptyset$ **then**
  $makespan = s_v^- + p_{vk} + t_v^-$
**else**
  order $x \in \pi | \pi = L_k \cap R_k$ by $s_x$
  **if** $k \neq \mu(v)$ **then**

$$makespan = p_{vk} + \begin{cases} s_v^- + p_{\pi(1)} + t_{\pi(1)} & if\ i = 0 \\ s_{\pi(i)} + p_{\pi(i)} + p_{\pi(i+1)} + t_{\pi(i+1)} & if\ 1 \leq i < l \\ s_{\pi(i)} + p_{\pi(i)} + t_v^- & if\ i = l \end{cases}$$

  **else**
    $makespan = lpath(L_k \backslash R_k \cap R_k \backslash L_k, k)$
  **end if**
**end if**

### D. Estimation cost

The optimization process consists in an iterative search based in the tabu metaheuristic. This process uses a heuristic value to select the most promising direction to explore the search space. The exact cost of the longest path in the graph can be calculated in $O(N)$. Although this is not a high computational complexity process, repeating it for all neighbors generated in each iteration of the algorithm means a high cost. To keep the execution time as low as possible and give the user a quick response, an estimation of the cost is used instead of the actual value. Using this estimation as heuristic, the actual cost of the solution graph is calculated only once per iteration.

The estimation of the *makespan* is explained in [4] and detailed in Algorithm 4. The value obtained is an upper bound of the real *makespan* of a solution graph after performing a $k$-insertion and the average deviation of the values obtained is not more than 1% from the real ones.

The value is obtained by adding the early beginning time of the predecessors of $v$, $s_v^-$; the later tail time of the successors, $t_v^-$; and the processing time of the task after its re-assignation, $p_{vk}$. This is a good estimation of the solution cost after performing the re-assignation since the time marks after the removal of $v$ do not change. A different situation is when the re-assignation is done in the same resource but in a different position in the processing queue. As the processing order of the tasks assigned to the resource $k$ will change when performing the $k$-insertion, the time marks of the predecessor and successors of $v$ will be outdated. In this case, a different method is used to calculate the *makespan* estimation, described in [1] and detailed in Algorithm 5.

### E. Tabu search

The optimization algorithm is a search process executed iteratively that stores a list of tabu motions, *TM*, that are forbidden $k$-insertions for a certain number of iterations. This list allows having more diversity in the exploration of the search space, avoiding local minimums. This also maximizes the probability of finding the optimal solution. Iteratively, the neighborhood of the current solution graph is generated and ordered by ascending estimated *makespan*. The neighbors that are tabu motions are filtered: if non-tabu neighbors were generated, one of the best two is selected; if only tabu neighbors are available, one of both the two with the prohibition near to expire is selected. There is also an aspiration criterion: if the most promising of the generated neighbors is better than the best solution found at the current iteration, that $k$-insertion is selected without checking the tabu list content. Every time that a $k$-insertion is selected to be the seed of the next iteration, the tabu list is updated by adding that motion and the prohibition expiration iteration. The expiration value is calculated dynamically, depending on the size of the neighborhood and the number of alternatives of re-assignation of the selected motion. The higher the number of neighbors or the number of resources capable to execute the re-assigned task, the higher is the number of iterations that repeated $k$-insertions will be forbidden. The complete process is detailed in Algorithm 6.

**Algorithm 5** $lpath(Q,k)$

**Require:** permutation tasks: $Q = Q_1, ...Q_q$
**Require:** resource $k$ which tasks are assigned to
  $a = Q_1$
  $s'_a = max(s_{PJ[a]} + p_{PJ[a]k}, s_{PM'[a]} + p_{PM'[a]k})$
  **for** $i = 2$ to $size(Q)$ **do**
    $b = Q_i$
    $s'_b = max(s_{PJ[b]} + p_{PJ[b]k}, s'_a + p_a k)$
    $a = b$
  **end for**
  $b = Q_q$
  $t'_b = max(t_{SJ[b]} + p_{SJ[b]k}, t_{SM'[b]} + p_{SM'[b]})$
  **for** $i = q - 1$ to $1$ **do**
    $a = Q_i$
    $t'_a = max(t_{SJ[a]} + p_{SJ[a]k}, t'_b + p_{bk}$
    $b = a$
  **end for**
  **return** $max(r'_{Q_i} + p_{Q_i k} + t'_{Q_i})$

## III. LibrePlan integration

LibrePlan [8] is a collaborative open-source tool to plan, monitor and control projects endowed with a rich web interface which provides a desktop alike user experience. It is an open source software developed by the Galician software company

**Algorithm 6** Tabu search

  $S, S_{best} = s_0$, initial solution
  $iter = 0$
  **while** $iter < MAX\_ITER$ **do**
    $N = Nopt1(S)$
    **if** $N = \emptyset$ **then**
      $S$ is optimal.
      break
    **else if** $min(est(N)) < S_{best}$ **then**
      $C =$ most promising neighbor
    **else if** $N_{tabu} = \emptyset$ **then**
      $s' = N_{tabu}$ with $min(TM)$
    **else**
      $s' = N_{non-tabu}$ with $min(est(s'' \in N))$
    **end if**
    **if** $size(s' > 1)$ **then**
      $pair =$ best two elements of $s'$
      $C =$ random element of $pair$
    **else**
      $C = s'$
    **end if**
    $TM(v_C, k_C) = iter + size(pathP(C)) + size(resources(v_C))$
    $S = C$
    **if** $makespan(S) < S_best$ **then**
      $S_best = S$
    **end if**
    $iter++$
  **end while**

Igalia S.L. as an evolution of the previous alike tool NavalPlan, which initially was developed to satisfy the scheduling needs of the projects in the Galician naval industry. Nowadays the software is oriented to solve general purpose scheduling problems. The integration of this optimization algorithm as a module of LibrePlan provides users an easy way of generating optimal resource assignations with the purpose of minimizing the time necessary to complete the projects and, indirectly, their associated cost.

There are two main changes made on the algorithm behavior to adapt it to run into LibrePlan: the calculation of the real time needed to complete the working hours associated to a task, and the resource assignation limitations that can be managed by the program. Below, these changes are described more in detail.

### A. *From working hours to real time*

The basic data model managed by the algorithm does only support the static definition of the duration of the tasks, where this value depends only on the resource assigned to execute the task. This implies that the result of the algorithm is the same with independence of the beginning date of the project. In LibrePlan, however, a more complex context is defined to know the real duration of the task, as it depends on the following elements:

- The working time necessary to complete the task
- The project calendar, that defines the holidays days associated to national, regional or local feast days, and the maximum dedication time in working days.
- The resource calendar, which contains information about the availability of the resource with independence of the project calendar (due to holidays, maintenance, medical conditions, half day working days, etc.).

With this new scenario, the beginning date of a task becomes a new element to be considered in the optimization process, as its duration will be very sensitive to any change produced in the calendar information: e.g., a 9 hour task typically needs an entire day and one extra hour to be completed, but if the beginning day is Friday, two additional days are added because the weekend corresponds to non working days.

As in the basic data model of the algorithm the real duration of a task is static and only depends on the resource that processes it, it was necessary to improve it and add support for calendar management: the algorithm considers the beginning date of the project and propagates the beginning dates of the tasks to be able to correctly calculate the real duration. As the calendar restrictions come from different information sources (the project calendar, the task calendar and the resource calendar), it is necessary to merge all restrictions to check the real dedication that can be applied by date.

### B. *Resource assignations validation*

Resources in LibrePlan are used to model machines, workers and virtual workers. Also, all these types of resources can contain information about their category, localization, etc. All this information can be used to determine the ability of the resources to process certain types of tasks, e.g. there are high-level tasks that can be only done by project managers due to qualification restrictions.

To manage these situations, there is an option in LibrePlan to add resource assignation criteria to each task individually. These criteria act as a filter of which type of tasks can be assigned to each resource, being valid only the assignations where the criteria match the information of the resource. There are some predefined criteria, like the localization, the category or the type of resource (worker, machine, etc.), but the user is free to use his own.

To reduce the execution time of the optimization process, the information about the compatibility between tasks and resources is calculated during the generation of the initial solution graph. From that moment, each task contains the resources that can process it, without considering the calendar restrictions, i.e., the compatibility and the availability of the resources are problems that are solved separately.

## IV. USE CASES

The tabu search algorithm was tested with a set of repositories widely used in the literature to prove the performance and abilities of scheduling algorithms [11]. After concluding the first testing phase, the implementation was modified [10] in order to include the restrictions managed by LibrePlan: the resource and project calendars, the restrictions in the starting and ending dates of the tasks and the assignation criteria. Also the web interface was modified in order to include the option to calculate the optimal assignations of a project and to show the results at the end of the execution.

A subset of three use cases based on the scheduling problems in the standard repositories was selected and introduced in LibrePlan (Fig. 1) to represent different user needs: Mk01 contains short tasks with low diversity of assignations, Mk02 has longer tasks and higher diversity and Mk05 contains long tasks with low assignation variability. The definition of each problem in the repository includes the real duration of each task and each assignable resource. Since the duration of the tasks are calculated in LibrePlan from the

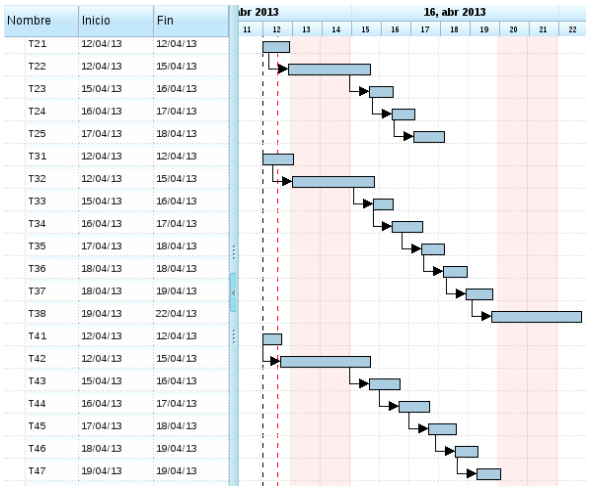

Figure 1. Part of the Gantt diagram that shows the project of this use case.

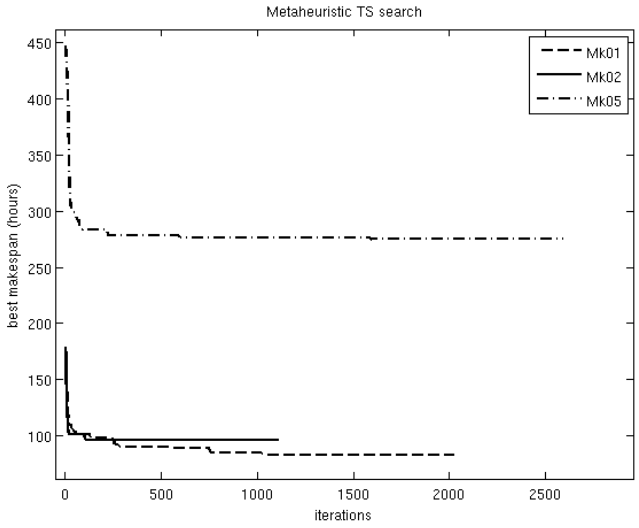


Figure 2. Evolution of the best *makespan* for each use case.

working hours, the resources information and the project calendars, a different interpretation of the input data was made (without losing generality): we took the maximum of the durations specified for each task, and treated it as working hours instead of natural hours. In addition, to introduce differences in the execution time of the tasks depending on the resource, half of them were configured to work part time, 4 hours a day. In Table I the optimization results for each use case are detailed. The values obtained for each problem are the average of 5 executions of the algorithm in each case.

TABLE I. USE CASES EXECUTION RESULTS

| Case | Input | | | | Mean Result (5 executions) | | |
|---|---|---|---|---|---|---|---|
| | *Working Hours* | *Tasks* | *Resources* | *V*[a)] | *Iter.* | *Optim.* | *Natural Hours* |
| Mk01 | 250 | 55 | 6 | 2 | 2329 | 43 % | 86 |
| Mk02 | 262 | 58 | 6 | 3.5 | 1110 | 48 % | 103 |
| Mk05 | 737 | 106 | 4 | 1.5 | 1644 | 38 % | 281 |

a) V stands for variability (mean of resources assignable to each task)

The algorithm starts the exploration with a random solution generated observing the task assignation restrictions. With the first iterations the *makespan* of the best solution decrease very fast, but the algorithm needs a higher number of iterations to converge to the optimal solution, what is very adequate to do the integration in LibrePlan, because a maximum execution time of the optimization module can be configured, returning the best found solution at that point. This behavior is detailed in Fig. 2, which shows the best found *makespan* respect to the number of iterations.

The output of the optimization module was integrated in the web interface of LibrePlan showing the optimization results when the execution finishes, as shown in Fig. 3.

## V. CONCLUSIONS

In this work, a tabu search algorithm was developed to optimize flexible job shop scheduling problems. The search

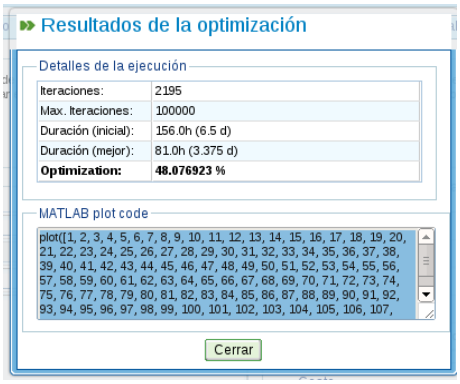


Figure 3. Result window of the web interface with the execution details of the optimization.

space is formed by all the possible resource assignations observing the project restrictions: dependencies, dates and assignation criterions. The algorithm was integrated as a module of LibrePlan, an open source tool for project management, and several use cases were presented showing the optimization abilities of the algorithm for different combinations of tasks length and assignation variability. As part of future work we plan to improve the integration of the module in the web interface and do a more exhaustive validation including corporate projects.

## REFERENCES

[1] Mauro Dell'Amico and Marco Trubian. "Applying tabu search to the job-shop scheduling problem". In Annals of Operations Research, vol. 41(3) pp. 231-252, September 1993.

[2] P. Fattahi, M. Saidi Mehrabad, and F. Jolai. "Mathematical modeling and heuristic approaches to fexible job shop scheduling problems". In Journal of Intelligent Manufacturing, vol. 18(3) pp. 331-342, 2007.

[3] J. Gao, L. Sun, and M. Gen. "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems". In Computers & Operations Research, vol. 35(9) pp. 2892-2907, 2008.

[4] Monaldo Mastrolilli and Luca Maria Gambardella. "Effective neighborhood functions for the flexible job shop problem". In Journal of Scheduling, vol. 3(1) pp. 3-20, 1999.

[5] Michael L. Pinedo. "Scheduling: Theory, Algorithms, and Systems" (Chapter 2). In American Society of Mechanical Engineers, Springer Verlag, 2008 pp. 13-34.

[6] Michael L. Pinedo. Scheduling: "Theory, Algorithms, and Systems" (Chapter 7). In American Society of Mechanical Engineers, Springer Verlag, 2008, pp. 179-216.

[7] A. Rossi and G. Dini. "Flexible job-shop scheduling with routing exibility and separable setup times using ant colony optimisation method". In Robotics and Computer-Integrated Manufacturing, vol. 23(5) pp. 503-516, 2007.

[8] Igalia S.L. Libreplan online demo. *http://demo.libreplan.org*, October 2012.

[9] L.N. Xing, Y.W. Chen, P. Wang, Q.S. Zhao, and J. Xiong. "A knowledge-based ant colony optimization for flexible job shop scheduling problems". In Applied Soft Computing, vol. 10(3) pp. 888-896, 2010.

[10] A. Gonzalez-Sieira, A. Bugarín, M. Mucientes and M. Rego. Optiplan for LibrePlan online demo. *http://demos.citius.usc.es/optiplan4libreplan*, December 2012.

[11] Monaldo Mastrolilli. Problem instances and computational study. *http://www.idsia.ch/~monaldo/fjsp.html*, 1998.