Kitchen2000: efficient coordination of a group of robot performing cooking tasks

*Team 24*

Luca Marson, lmarson@kth.se

Alfredo Reichlin, alfrei@kth.se

Peiyang Shi, pyshi@kth.se

Gianluigi Silvestri, giasil@kth.se

### **Abstract**

We designed a multi-agent holonic scheduling system that tackles a NP-Hard jobshop scheduling problem(JSP) with additional constraints in order simulate a real life restaurant kitchen operated by autonomous robots. We implemented two solutions tackling two types of scheduling. We used reactive tabu search to solve a classic JSP and modified genetic algorithm to solve a flexible JSP kitchen. We find the genetic algorithm can handle very complex constraints and flexible job assignment through evolution. Tabu search can find optimal solution much quicker but implementation complexity increases as additional constraints are added.

### Introduction

Kitchen2000 is a robotic chef system that can plan and schedule work in a fully autonomous kitchen. There can be multiple chefs working together forming a holonic multi-agent system. Our goal for this project is to efficiently schedule all robots to work smoothly and fulfill orders. Our robot cooks fall into a subcategory of planning problems called Jobshop Scheduling Problem (JSP), where each robot is an agent, each recipe is a job and the operations of each job are the various steps of the recipe. Each robot is specialized to perform only a specific kind of action (i.e. cutting vegetables, mixing ingredients or cooking food). On top of a standard JSP problem, we have a series of constraints in order to mimic the behaviors of a real kitchen. (e.g. we can’t make tomato sauce before we peel the tomatoes)

JSP problems are notoriously difficult to solve as it is a NP-Hard problem by nature.

It is virtually impossible to find the optimal solution by analyzing all the possible combinations. In order to schedule our NP-Hard kitchen, we need to rely efficient methods to find the optimal solution. In order to mimic the real world kitchen, we mixed in a number of constraints into a typical JSP problem. These constraints are solved using constraint programming heuristics.

Our first approach is using a modified Tabu search for JSP. We began by assigning a set of operations that a robot can do then we schedule our NP-Hard Kitchen’s incoming orders by using Reactive Tabu Search [1]. Then we found that by removing the restriction that an operation had a preassigned robot, we can greatly reduce the scheduling time. Rather than restricting a robot’s operations, we give robots a specialization but they are free to perform any action. However, such flexibility further increases the complexity of the solution space as the assignments of robots become dynamic. By allowing robots to perform all actions, this problem is classified as a holonic system with Flexible Jobshop Scheduling Problem (FJSP). We decided to used a completely different approach to solve FJSP based on added constraints and complexity. Our solution is to use a modified version of a Genetic Algorithm [2]. Genetic algorithm has much greater flexibility to embrace additional constraints in the problem.

### Related Work

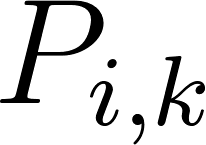
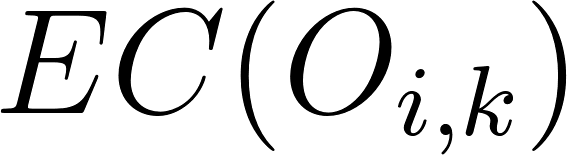
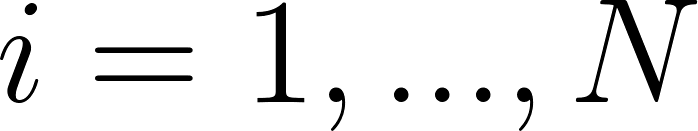
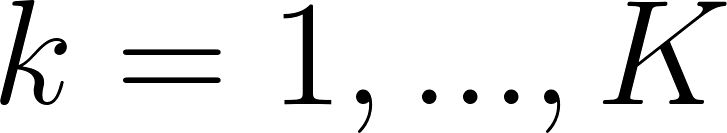
Initially we tried to solve the Planning Domain Description Language. The introduction of Planning Domain Description Language has facilitated the scientific development of planning. It was first proposed in 1998 by Drew McDermott and is now a community standard for planning domain models [3]. Although there has been research on using planning technology to solve realistic problems, there is a gap between the requirements from real-world domains and the possibilities in the original PDDL. Thus PDDL 2.1 has been developed to introduce new functionalities like durative action.

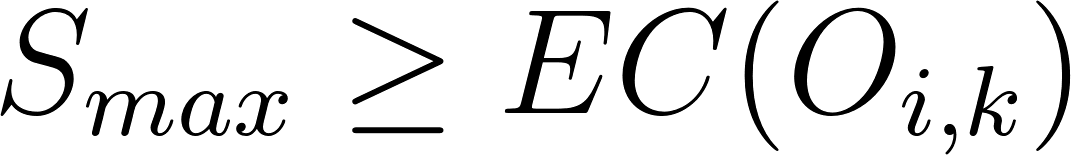
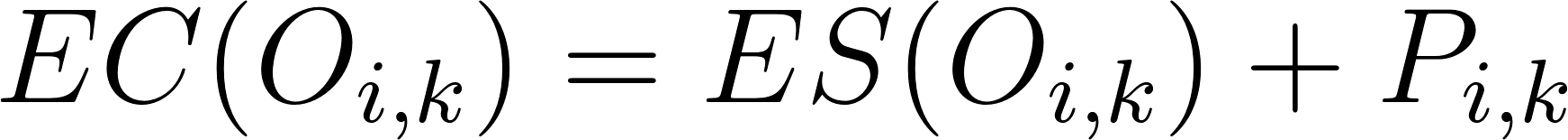
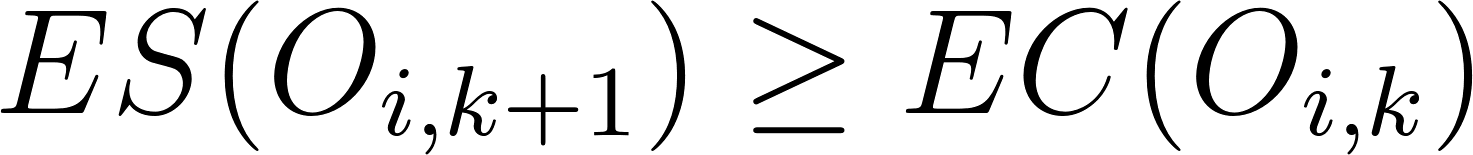
Firstly we tried to modeled our problem using PDDL 2.1, since we needed a way to implement action with duration. We wanted to to solve it with an off-the-shelf solver but we did not find one which supported this feature.We tried then to investigate different resolution techniques for multi-agent problems with time constraints, finding that JSP is an ideal representation of a multi-agent system where planning is centralized. The Reactive Tabu Search algorithm used to solve this problem is taken from [1], while the solution of its extension FJSP with a Genetic Algorithm is taken from[2].

### Solution and methods

In the following section we first describe the problems of Job Shop Scheduling and Flexible Job Shop Scheduling, and then provide the description of the used algorithms Reactive Tabu Search and Genetic Algorithms.

#### Jobshop Scheduling Problem (JSP)

The job shop problem can be formulated as an optimization problem:   
  
 - number of jobs  
 - number of machines  
- a job  
 - a machine  
[](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D) - [](https://www.codecogs.com/eqnedit.php?latex=k%5E%7Bth%7D) operation of job [](https://www.codecogs.com/eqnedit.php?latex=i)  
[](https://www.codecogs.com/eqnedit.php?latex=P_%7Bi%2Ck%7D) - processing time of operation [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D)  
[](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%7D)) - operation [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D) start time  
[](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)) - operation [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D) end time  
[](https://www.codecogs.com/eqnedit.php?latex=S_%7Bmax%7D) - maximum end time among all operations [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D)  
 [](https://www.codecogs.com/eqnedit.php?latex=i%3D1%2C...%2CN)  
 [](https://www.codecogs.com/eqnedit.php?latex=k%3D1%2C...%2CK)

Given a job composed of several operations, all its operations must be processed subsequently. Each operation can be processed only by one machine, and a machine can process several operations.   
  
[](https://www.codecogs.com/eqnedit.php?latex=S_%7Bmax%7D%5Cgeq%20EC(O_%7Bi%2Ck%7D))  
[](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%7D)%5Cgeq%200)  
[](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)%20%3D%20ES(O_%7Bi%2Ck%7D)%20%2B%20P_%7Bi%2Ck%7D%20)  
[](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%2B1%7D)%5Cgeq%20EC(O_%7Bi%2Ck%7D))  
  
Objective function:  
we want to minimize the maximum makespan.  
[](https://www.codecogs.com/eqnedit.php?latex=%5Cmin%20S_%7Bmax%7D)

#### Reactive Tabu Search

We solved the job shop problem using the Reactive Tabu Search algorithm described by Kawaguchi and Fukuyama in [1]. Tabu Search is a metaheuristics proposed by F. Glover [5] able to solve combinatorial optimization problems. It is an improvement of simple local search methods since it adds a memory element, the tabu list, which allows to escape from local minima by classifying certain moves as forbidden.

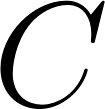
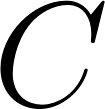
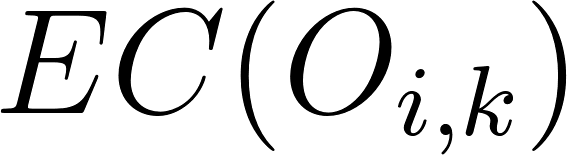
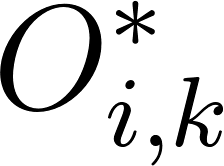
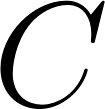
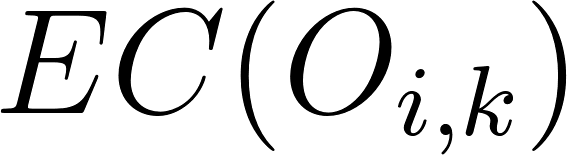
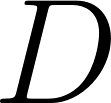
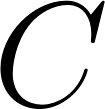
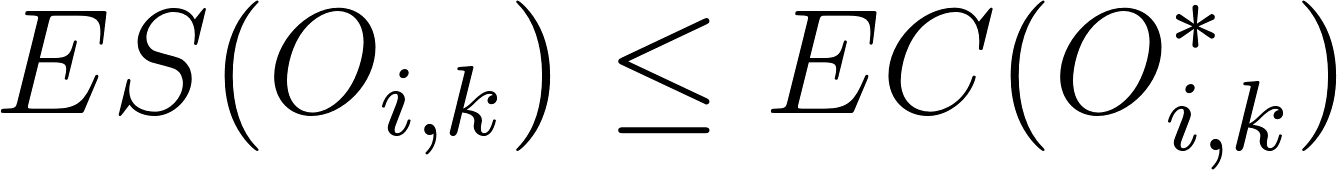
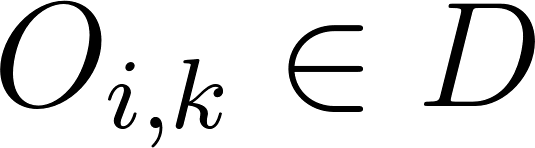
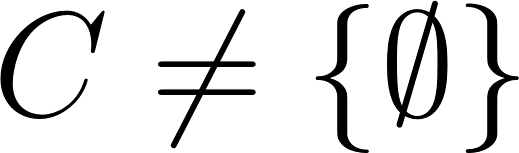
RTS differs from the standard tabu search in that the tabu list length is modified dynamically during execution in this way:

1. Save all the visited states in a saving list.
2. If the current solution is in the saving list and it has been visited in the past CYCLEMAX iterations, then increase the tabu list length.
3. If the tabu list length has not been modified for a number of iterations greater or equal than cycleMoveAve, decrease it.

Reactive Tabu Search main steps:

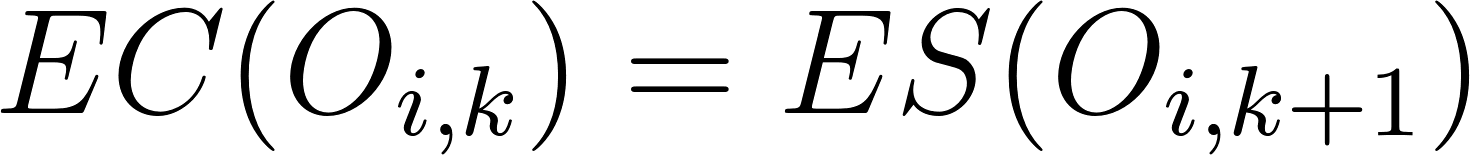
1. Set the initial tabu length, CYCLEMAX and the maximum number of iterations to a predetermined value. Set cycleMoveAve to 0.
2. Generate an initial schedule, set it as the current solution and store it in the tabu list and in the saving list.
3. Generate a neighborhood of the current solution.
4. Choose the best neighbor not contained in the tabu list and set it as the current solution.
5. Store the current solution in the tabu list and in the saving list.
6. Modify the tabu list length if needed.
7. If the number of iterations has not exceeded the predetermined maximum, go to step 3, else stop.

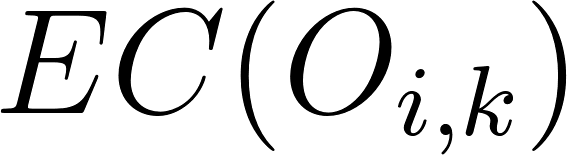
Initial feasible solution generation:

1. Find the set [](https://www.codecogs.com/eqnedit.php?latex=C) of operations for which the start time [](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%7D)) has not been evaluated yet and that should be the next operation for each job.
2. For all the operations in [](https://www.codecogs.com/eqnedit.php?latex=C), find [](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%7D)) and [](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)) such that [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D) can be started as early as possible and all constraints are respected.
3. Find the operation [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D%5E*) in [](https://www.codecogs.com/eqnedit.php?latex=C) with the earliest [](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)) and set the relative machine to [](https://www.codecogs.com/eqnedit.php?latex=M_k%5E*).
4. Find the set [](https://www.codecogs.com/eqnedit.php?latex=D) of operations [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D) processed by [](https://www.codecogs.com/eqnedit.php?latex=M_k%5E*), belonging to [](https://www.codecogs.com/eqnedit.php?latex=C) and such that [](https://www.codecogs.com/eqnedit.php?latex=ES(O_%7Bi%2Ck%7D)%20%5Cleq%20EC(O_%7Bi%2Ck%7D%5E*)).
5. Select an operation [](https://www.codecogs.com/eqnedit.php?latex=O_%7Bi%2Ck%7D%5Cin%20D) and set it as the next operation of [](https://www.codecogs.com/eqnedit.php?latex=M_k%5E*).
6. If [](https://www.codecogs.com/eqnedit.php?latex=C%20%5Cneq%20%5C%7B%5Cemptyset%5C%7D) go to step 1, else stop.

Neighborhood generation:

We generate a neighborhood of schedules using ne notion of critical blocks.

A critical path is a series of consecutive operations that form a makespan in a schedule. A critical block is a pair of operations belonging to the critical path, executed by the same machine and such that [](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)%20%3D%20ES(O_%7Bi%2Ck%2B1%7D)).

1. Starting from the operation with the latest [](https://www.codecogs.com/eqnedit.php?latex=EC(O_%7Bi%2Ck%7D)) and going backward in time, find a critical path. We consider only the first one that we encounter.
2. Locate all the critical blocks inside the critical path and swap them to generate new feasible solutions.

#### Flexible JobShop Search Problem(FJSP)

The flexible JobShop Search Problem is an extension of the JSP problem. In which the operations are not strictly bound to a certain agent, however, each agent will have different efficiency per same operation. This allows more flexible schedule, but also exponentially more calculation complexity for the optimal solution comparing to the classic JSP problem.

#### Genetic Algorithm

A Genetic Algorithm (GA) is an evolutionary algorithm inspired by the mechanism of natural selection [4], where the better individuals are more likely to survive and their characteristics will be propagated to the next generations.In the genetic algorithms each individual is called *chromosome*, and the single units of a chromosome are called *genes.* To each chromosome is associated a value called *fitness.*

At each step of the algorithm the “parents” genes are combined to generate offspring in the next generation, with the goal of obtain individuals with better fitness. On the first step of the algorithm a predefined number of chromosomes is randomly generated and the fitness value is calculated for each of them. A chromosome represent a potential solution to the problem. The algorithm will then be executed until a certain condition is reached, for example the execution of a predefined number of cycles. At each cycle we choose several couples of chromosomes to be combined in order to generate the offspring. This choice must be done in a way to select among the best individuals, to ensure that the best genes will be transmitted.

The combination procedure is usually divided in two phases: *crossover* and *mutation.* In the crossover a random position in the chromosome is chosen, and the two parents are split in that position. The two parts at the left of the cut are then exchanged to create two children chromosomes, to whom the mutation is applied. In the mutation each gene has a small probability to be exchanged with another.

The probabilities of crossover and mutation must be chosen basing on the problem to solve, and usually are set between 0.6 and 1 for crossover and less than 0.1 for mutation. An example of crossover and mutation can be seen in fig.1 and fig.2.

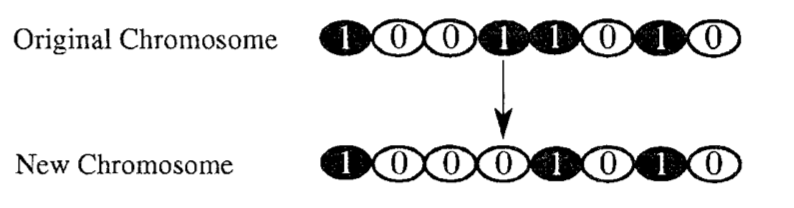
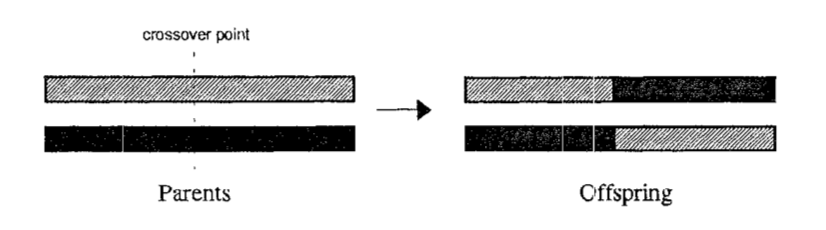
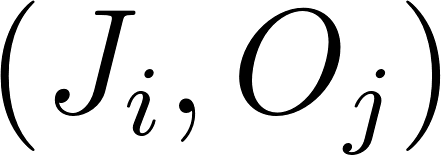
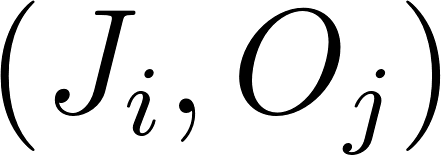
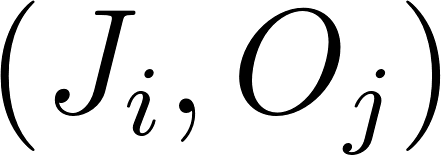
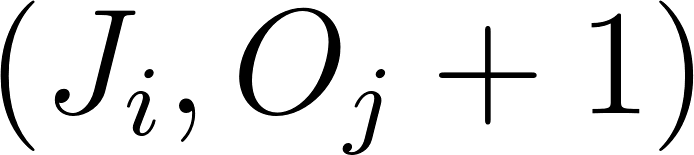


Fig. 1 Fig. 2

#### GENETIC ALGORITHM FOR FLEXIBLE JOB SHOP SCHEDULING PROBLEM

To solve the FJSP using a genetic algorithm we have used a simplification of the procedure implemented by Pezzella et al in [2].

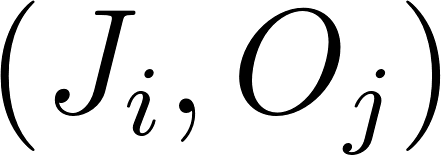
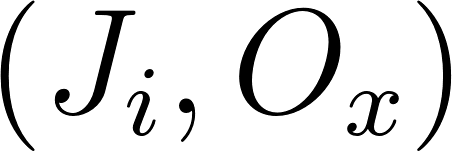
Since each chromosome must represent a potential solution of the problem a chromosome is a solution of the scheduling problem, that is the sequence of execution of all the operations performed by different machines. Each gene contains the following information: a tuple including a tuple[](https://www.codecogs.com/eqnedit.php?latex=%20(J_i%2CO_j)) and a value [](https://www.codecogs.com/eqnedit.php?latex=M_k). The tuple [](https://www.codecogs.com/eqnedit.php?latex=(J_i%2CO_j)) is the operation j of the job i, and [](https://www.codecogs.com/eqnedit.php?latex=M_k) is the machine executing that operation.

The initial population is chosen randomly, but making sure that the order of execution of the operations is respected ( [](https://www.codecogs.com/eqnedit.php?latex=(J_i%2CO_j)) is executed before [](https://www.codecogs.com/eqnedit.php?latex=(J_i%2CO_j%2B1)) ).

The fitness of each chromosome is the makespan of the scheduling, thus the best chromosomes will be the ones with lowest fitness.

At each operation we have to select which parent use for the generation of the offspring. This is done by using a Binary tournament: we randomly choose two individuals, and the one with better fitness value is selected for reproduction.

The offspring generation is then obtained by two different mechanisms called assignment operators and sequencing operators, and for each generation we choose one of them with equal probability. In case of assignment operators the only thing that changes is which machine perform which operation, but the sequence of operations remains the same. The offsprings are thus generated with a crossover on the values [](https://www.codecogs.com/eqnedit.php?latex=M_k) followed by a mutation with probability 0.01.

In case of sequencing operators only the sequence of operations is changed. In this case is not possible to use crossover and mutation in the standard way, because the constraints on operations order must be respected. To avoid this problem, we use the Precedence Preserving Order-based crossover (POX) [6]. In POX a random operation [](https://www.codecogs.com/eqnedit.php?latex=(J_i%2C%20O_j)) is chosen, and all the operations [](https://www.codecogs.com/eqnedit.php?latex=(J_i%2CO_x)) belonging to the job [](https://www.codecogs.com/eqnedit.php?latex=J_i) in the first parent will be copied to the child in the same order. The remaining genes will be filled with the genes in the second parent in the same order. The symmetric procedure will be done for the second child. In this case we don’t perform any mutation, because the probability of successfully performing a mutation maintaining the precedence constraints of the operations is very low.

The stop criterion is the execution of a predefined number of cycles, and during the execution of the whole algorithm the chromosome with the best fitness is stored and updated at every cycle if a better individual is found.

### Case Study & Results and Analysis

In this section we provide an analysis of the results of the Reactive Tabu Search and the GA on some test case. We then formulate a comparison between the two problems of JSP and FJSP, trying to find an optimal solution for the cooking task.

The Reactive Tabu Search algorithm has been tested using benchmark tests of different difficulties. The simpler one is a 6x6 JSP, with 6 jobs, 6 machines and 6 operations per job with a known optimal solution of 55. The more difficult one one is a 15x15 JSP, with 15, jobs, 15 machines and 15 operations per job with a known optimal solution of 1233.

**6x6 JSP**

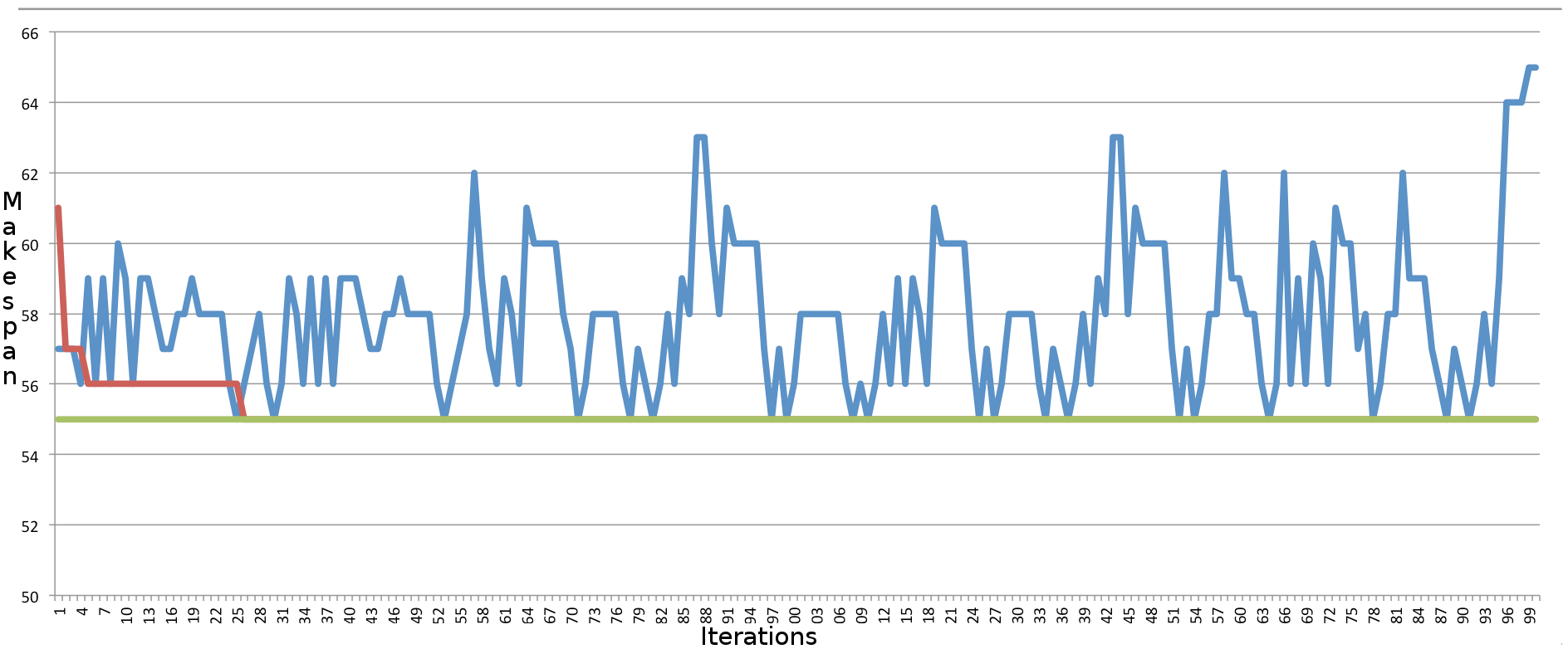


Fig 3.

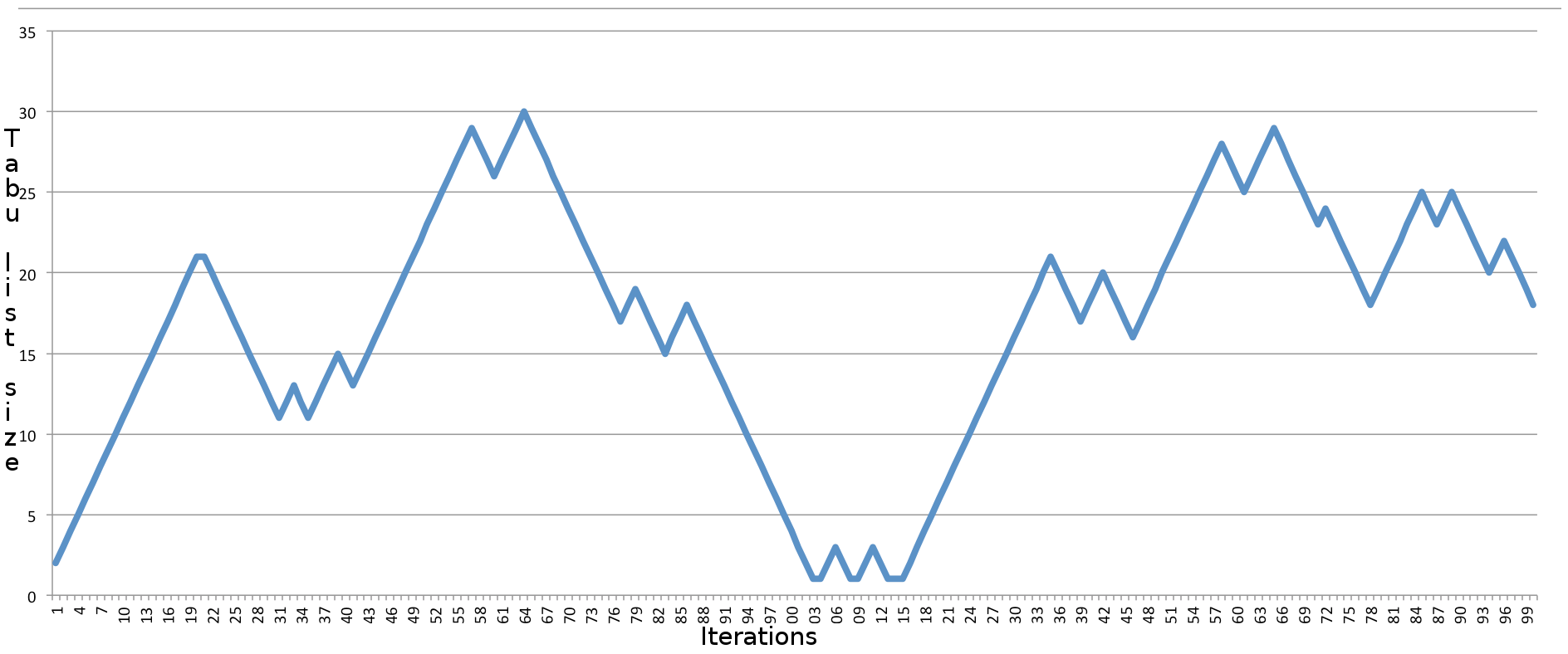


Fig 4.

The red line in Fig 3. shows the optimal makespan. The algorithm is able to find the known optimal solution (green line) for the 6x6 problem. The execution time is 0.239s.

**15x15 JSP**

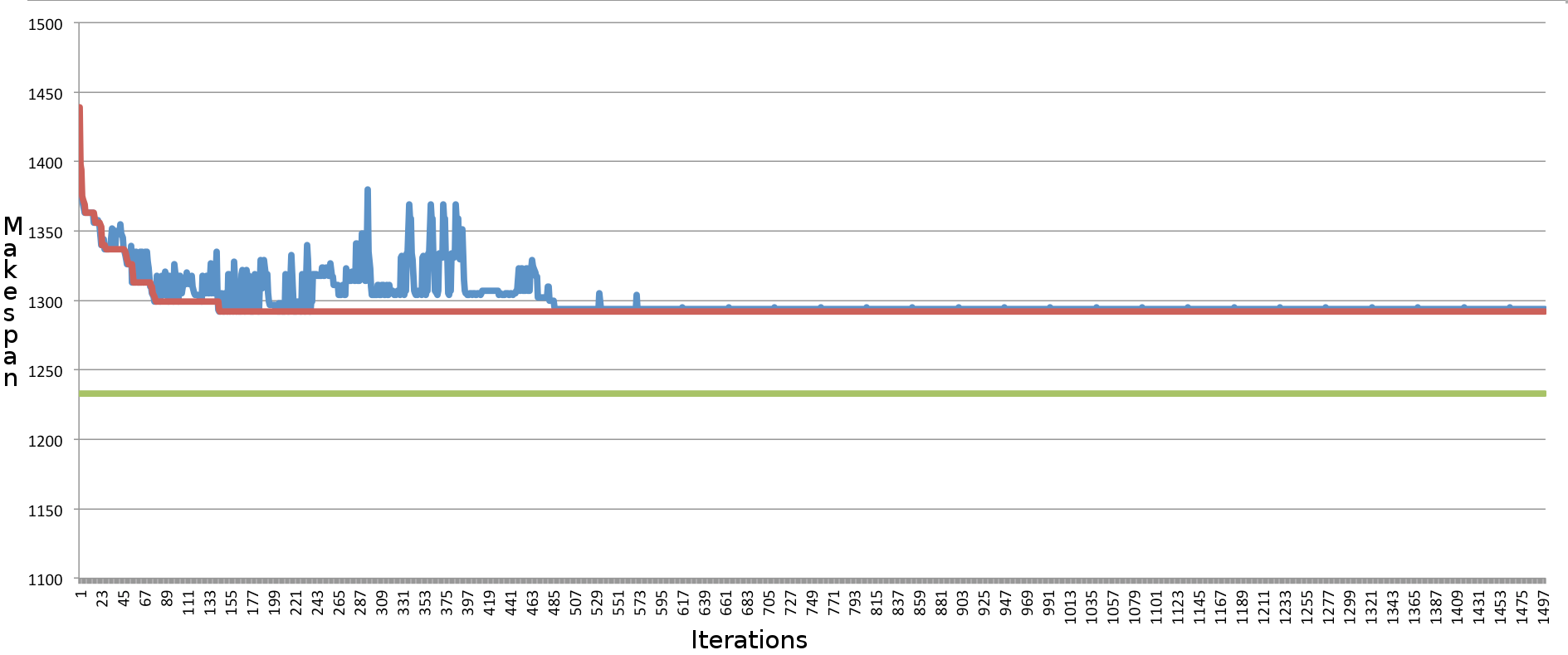


Fig 5.

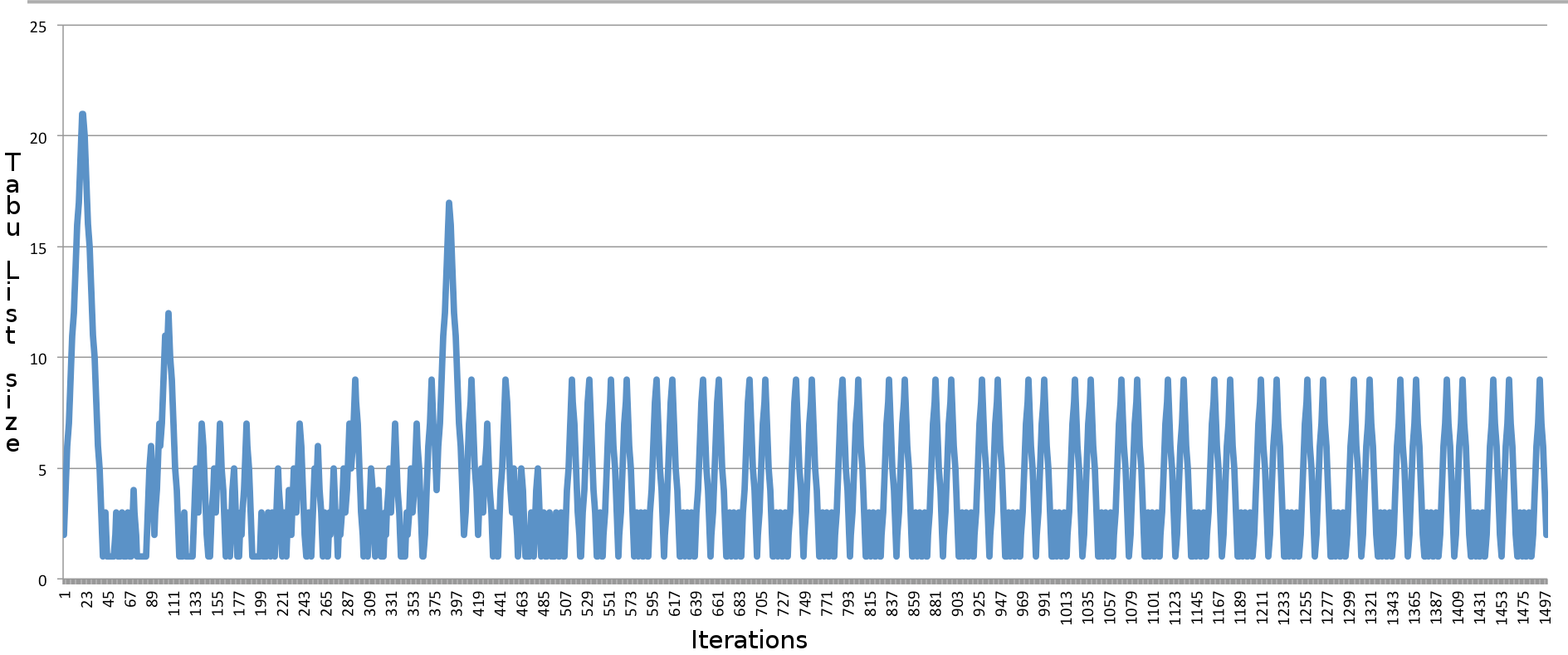


Fig 6.

The algorithm is not able to find the known optimal solution (green line) for the 15x15 problem, but a systematic improvement from the initial solution can be noticed. The execution time is 2.249s.

As it can be seen from Fig 1. And Fig. 3, the makespan oscillates because the algorithm tries to explore the solution space by also considering non improving solutions.

In Fig. 2 And Fig. 4 it can be seen that the tabu list size is increased whenever many states stored in the saving list are reconsidered.

The GA has been tested in two problems of different dimensions where the solutions were already known. The settings used in our GA were: number of individuals=400, number of generations=1000.

The first test has been used to test the ability of our algorithm to effectively find the correct solution of the problem. Thus, the test is composed by two Jobs containing two Operations each and two Machines, and the correct result is found by the GA.

The second test case is a more complex problem, with 15 Jobs, 10 Machines, and a flexible number of Operations for each Job, for a total of 56 Operations. The expected optimal fitness is 12, and the result given by the GA is 13 in 116 seconds. The graph of the execution can be seen in fig 7., where the x axis represent the execution step and the y axis represent the best fitness value found. The red line is the best fitness value expected. This test has proven that the GA is able to find a good solution in an acceptable amount of time for complex problems.

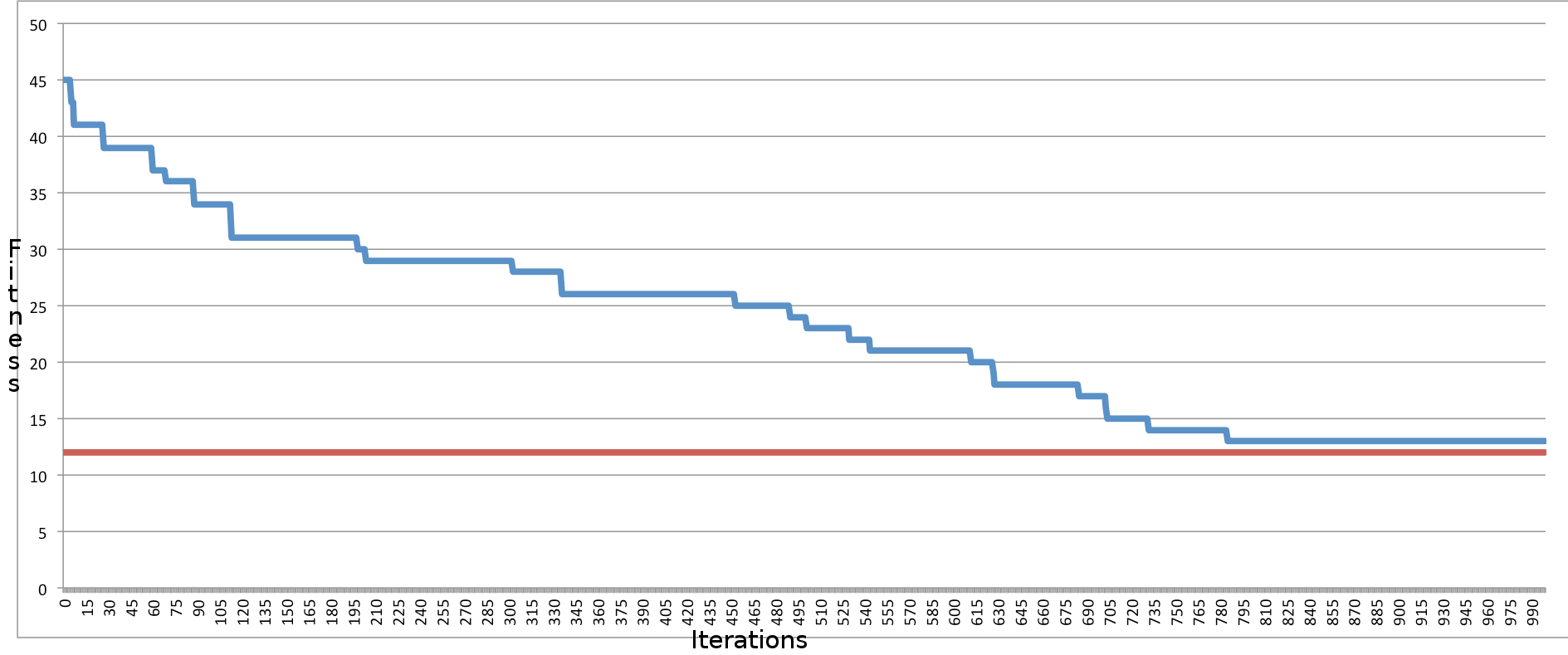


Fig 7.

#### Results comparison

Here we try to understand in which cases is more convenient to use a set of robots capable to perform every task rather than using specialized robots that can only perform certain tasks. In particular, when in the JSP an operation can be performed only by one machine, in FJSP we let that operation executable by every machine, but increasing the time taken to do that. In our test the time is increased by multiplying by 2,3 and 4.

The problem analyzed is the one reported in fig.8.

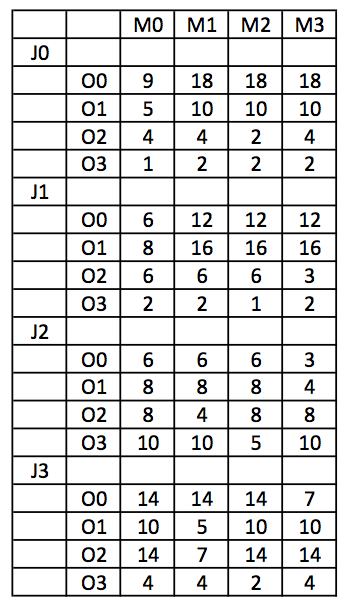
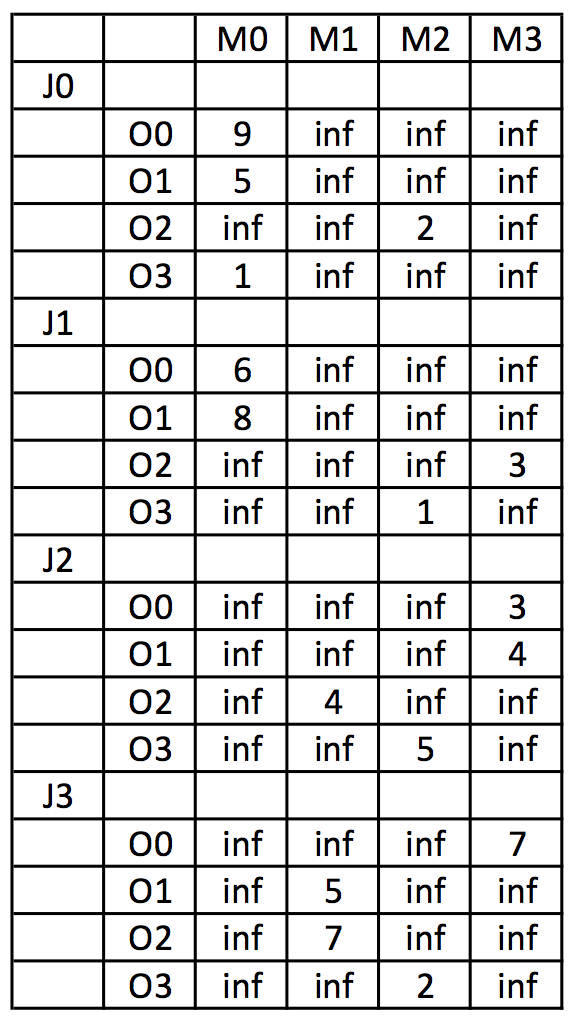


Fig. 8 Fig. 9

Case JSP: The solution found by the Reactive Tabu Search has a makespan of 31.

In the FJSP the previous table is modified, and in the case x2 its values are as in fig.9. The same principle is applied in the cases x3 and x4.

Case FJSP x2: The solution found by the GA has a makespan of 26.

Case FJSP x3: The solution found by the GA has a makespan of 30.

Case FJSP x4: The solution found by the GA has a makespan of 31.

From empirical results we can see that it is better to design all purpose robots the robot can perform most operations with comparable efficiency as a specialized robot. When the penalty for all purpose robot is too high, FJSP scheduling will perform about the same as JSP. And FJSP solving with Genetic algorithm is much slower comparing to Reactive Tabu Search.

### Summary

The aim of this project was to design a multi-agent scheduling system for an efficient organization of a group of robot that can collaboratively cook a recipe. In order maximize the efficiency of the order, we categorized the scheduling in two classes of problems: the Job Shop scheduling Problem and the Flexible Job Shop scheduling Problem. The solution of these approaches has been found using two evolutionary algorithms, the Reactive Tabu Search and a Genetic Algorithm. In the end we provided an analysis on the efficacy of both approaches.

te

REFERENCES:

[1] Kawaguchi S., Fukuyama Y. 2016. Reactive Tabu Search for job-shop scheduling problems

[2] Pezzella F., Morganti G. and Ciaschetti G. 2007. A genetic algorithm for the Flexible Job-shop Scheduling Problem

[3] Fox M., Long D., 2003. An Extension to PDDL for Expressing Temporal Planning Domains

[4] Man K. F., Tang K. S. and Kwong S. 1996. Genetic Algorithms: Concepts and Applications

[5] Glover F., 1989. Tabu Search-Part I

[6] Lee K. M., Yamakawa T. and Lee K. M. 1998. A genetic algorithm for general machine scheduling problems