UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

GAME THEORETIC MULTI-AGENT ALGORITHMS FOR THE JOB SHOP SCHEDULING PROBLEM

BY

ORWA HORACE OWITI

SUPERVISOR

DR. ELISHA OPIYO OMULO

THIS REPORT HAS BEEN SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE IN THE UNIVERSITY OF NAIROBI

SEPTEMBER,2014

Declarations

The research project presented in this report is my original work and has not been presented for any other university Award

Signature:_____ Date:_____

Orwa Horace Owiti

Registration No: P58/63388/2011

This research project report has been submitted in partial fulfillment of the requirement of Master of Science in Computer Science in the University of Nairobi with my approval as Supervisor.

Signature:_____ Date:_____

Dr Elisha. O. Omulo

ABSTRACT

Job shop scheduling problem is a problem of scheduling n jobs on m machines with each job having a set of equal number of operation that are to be process in unique machine routes. The Job Shop Scheduling (JSSP) is one of the hardest combinatorial optimization problems and has been researched over the decade. This study proposes a new approach to solve a Job Shop Scheduling problem by structuring the problem as multi-agent system (MAS) and using 3 game theoretic algorithms to achieve the scheduling objectives. The objective of this study is to minimize the makespan. This approach is meant to achieve feasible schedules within reasonable time across different problem instances. This research solves the scheduling of operation on different machine and defines the sequence of operation processing on the respective machine. Job Scheduling problem is a resource allocation problem is mainly apparent in manufacturing environment, in which the jobs are allocated to various machines. Jobs are the activities and a machine represents the resources. It is also common in transportation, services and grid scheduling. The result and performance of the proposed algorithms are compared against other conventional algorithms. The comparison is on benchmark data used across multiple studies on JSSP.

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

CONTENTS

CHAPTER ONE: INTRODUCTION

1.1.JOB SCHEDULING

A Scheduling problem can be defined as the problem of allocation of limited shared resources over time to competing activities. Scheduling problems have over the years attracted interest in much research and has been the subject of a significant amount of literature in the operations research and Artificial intelligence fields. A huge amount of emphasis has been on investigating machine scheduling problems where jobs represent activities and machines represent resource and each machine can process at most one job at a time. This has kind of problem is apparent in a multitude of diverse real world domains e.g. scheduling of task in an assembly lines, scheduling of jobs in multi-processors/ multi-core machine, assignment of tasks to employees, job scheduling in distributed computing ,etc. We can categorize real world scheduling application areas as follows,

- **Demand scheduling for customers**: problem of assigning customers to a definite time for an order or service.
- **Workforce scheduling for employees**: problem of determining when employees work.
- **Operations scheduling**: combines workforce scheduling with job scheduling.
    1. Assigning jobs to workstations.
    2. Assigning people to workstations.
    3. Assigning people to jobs.
- **Distributed computing:** assigning jobs to processors time in multi-processor or multi-computer environment.

There are diverse variations of scheduling problems that have been formulated in machine scheduling, the simplest of which is a single machine scheduling problem. In the single machine scheduling problem involves trying to schedule a finite number of jobs onto one machine. Other variations depend on of the following factors.

- Machines can be related, independent, equal
- Machines can require a certain gap between jobs(recovery time) or no idle-time
- Machines can have sequence-dependent setups, that is, each machine processes a single stage of processing cycle.
- Jobs may have constraints, for example a job i needs to finish before job j can be started
- Jobs and machines have mutual constraints, for example, certain jobs can be scheduled on some machines only
- Set of jobs can relate to different set of machines
- Jobs can have different operations and machines can only process a single operation.
- Deterministic (fixed) processing times or probabilistic processing times.
- Scheduling can be non-pre-emptive, that is, processing of a job on a machine can be interrupted after it has started
- Jobs can have deadlines in which they need to be processed.
- Scheduling can be static, that is, all jobs are presented for scheduling at the same time or it can be deterministic that is jobs appear at different intervals and are scheduled as they appear. In this case processing and scheduling are concurrent.

Reasons for scheduling complexity include (Fox and Sadeh 1990):

Scheduling is a feasibility problem. The final solution must accomplish all the problem constraints. Another objective to be satisfied is the optimization of an evaluation function, adjusting to certain criteria as cost, lateness, process time, inventory time, etc.

Some scheduling problems have many constraints due to the unavailability of resources, due dates, etc.

Constraint representation cannot express the importance of the value domains. The number and identity of tasks that require a resource over a particular time interval is a key piece of information that can suppose the basis for heuristic variable and value orderings.

1.1.1. CLASSIFICATION OF SCHEDULING PROBLEM

Classification of scheduling problems depends on one or more variation of the above parameters. The most common classification of scheduling problems is as follows.

Open-shop scheduling problem (OSSP) is a scheduling problem where, given *n* jobs and *m* workstations, each job has to be processed on a workstation at least once. Job might have operations but there is no ordering precedence on the operation. However, some of these processing times may be zero. This problem becomes an NP-Hard when three or more machine are involved but can be solved in polynomial time if

Only two machines are involve,

All the jobs have the same length.

Flow-shop scheduling problem (FSSP) is a scheduling problem where, there are ***m*** machines and *j* jobs where **m>1**, each job has a set of operations *o* and the j^{th} operation of the job must be processed by j^{th} *machine*. The number of operations on each job is equal with the number of machines; each job must be processed on each of the machine.

Job-shop scheduling problem is a scheduling problem where, there are ***m*** machines and ***j*** jobs where **m>1**, each job has a set of operations *o* and has associated a processing order assigned for its operations. Unlike in flow-shop scheduling, the precedence sequence for operation in a job may differ from job to job. Job-Shop scheduling is a known NP-Hard.

1.1.2. OBJECTIVE OF SCHEDULING PROBLEMS

The objective of any job scheduling algorithm can have any of the following objectives;

Minimize the Makespan- The Makespan is the total length of the schedule, that is, the time it takes all jobs finish processing. This is formulated as

$$\mathbf{M = max\{ C_1 \ldots\ldots\ldots C_n \}}$$

EQUATION 1: *Minimize The Makespan*

Where,

C_j= the earliest time job ***j*** finishes processing.

Minimize Tardiness- In situations where the jobs $\boldsymbol{j}$ have deadlines $\boldsymbol{d_j}$, tardiness is the duration of time delays past its deadline

Then tardiness $\boldsymbol{t_j}$,

$$\boldsymbol{t_{j=max\{c_j-d_j\ ,0\}}}$$

The tardiness of the schedule **T** is,

$$\mathbf{T} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{max\{c_j - d_j\ , 0\}}$$

Therefore,

$$\mathbf{T} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{t_j}$$

EQUATION 2 *Minimize Tardiness*

Minimize lateness- Lateness for a job is defined as,

$$\boldsymbol{l_j = C_j\ - d_j}$$

The Lateness of the schedule **L**

$$\mathbf{L} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{max\{c_j - d_j\}}$$

$$\mathbf{T} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{t_j}$$

EQUATION 3 *Minimize Lateness*

1.2.JOB SHOP SCHEDULING

Job-shop scheduling is one of the most commonly researched about problems in the domain of scheduling problems. In this section we outline the main attributes of this kind of scheduling problem.

1.2.1. THE CLASSICAL JOB-SHOP SCHEDULING PROBLEM

In assembly lines, staff roasters, manufacturing, or production planning, The production of a good involves a number of processing steps that have to be performed in a set order. The decision to further process some good can only be taken, if all preceding steps are completed. In most cases, however, it is usually a common scenario that not just a single, but a variety of products is assembled concurrently. This means that, an appropriate sequencing and scheduling of individual processing operations is crucial, if maximal joint productivity is desired. This type of problems can be formulated as a classical job-shop scheduling problem.

The most generalized formulation of job-shop scheduling is a follows, there is an existence of n jobs that must be processed on m machines in a pre-determined order. Each job j consists of o_{ji} operations such that job j is $j(o_1 \ldots . o_n)$, each of the operation of a job must be processed by a specific machine, $p(o_{ji}, m_k)$ and processing of the job on the machine can take a certain duration $p(o_{ji}, m_k)$ A processing of a job is completed after completion of processing of its last operation, the completion of a job is denoted as c_j.

equation 1.3: *Formalizing a job shop scheduling problem*

A problem instance $P=(M,O,J)$ in job shop scheduling consists of

- A set M of Machines,
- A set O of operations o, each associated with a machine $M(o) \in M$ and having a duration $d(o) \in N$ and

- A set ***J*** of jobs **(o_1……o_n)**(each operation has exactly one occurrence.)

A Schedule ***S*** for ***P*** assigns to every operation ***o*** a time ***b(o)***:

1. ***b(o)≥0*** for all ***o Є O***
2. ***b(o)≥ b(o') + d(o')*** for operations ***o'*** preceding o in the same job.

A Schedule has cost ***T*** if ***b(o)+d(o)≤ T*** for all ***oЄ O.***

(Takeshi Yamada and Ryohei Nakano, 1997) describes a grant chart as a convenient way of visually representing a solution of the JSSP. An example of a solution for the 3 X 3 problem depicted in table 1 can be represented as shown on Figure 1.0

job	Operations routing (processing time)		
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

FIGURE 1 : *A 3 X 3 JSSP*

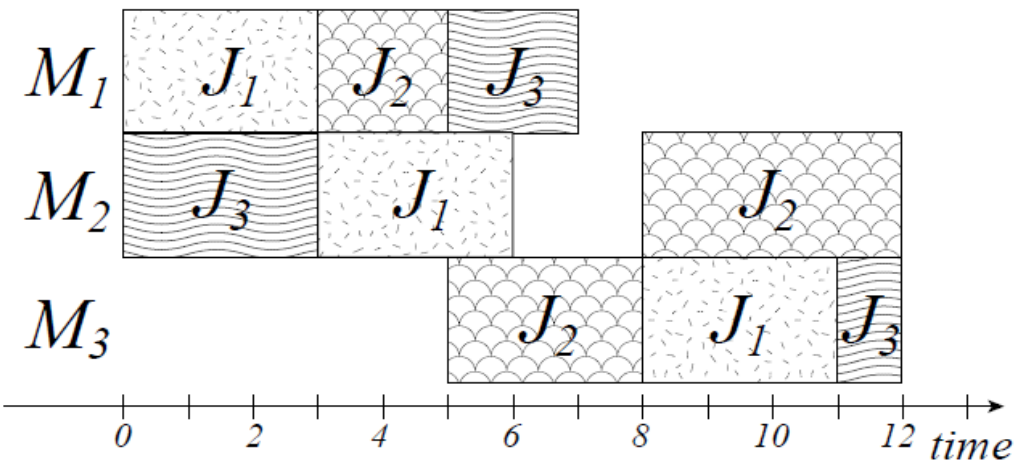


FIGURE 2: GRANTT CHART REPRESENTATION OF A JSSP SCHEDULE

They further illustrate that JSSP can be formally described by a disjunctive graph

G = (V, C U D), where;

V is a set of nodes representing operations of the jobs together with two special nodes, a *source* ***(0)*** and a *sink* *******, representing the beginning and end of the schedule, respectively.

C is a set of conjunctive arcs representing technological sequences of the operations.

D is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines.

The processing time for each operation is the weighted value attached to the corresponding nodes. Figure 1.2 shows this in a graph representation for the problem given in Table 1.0

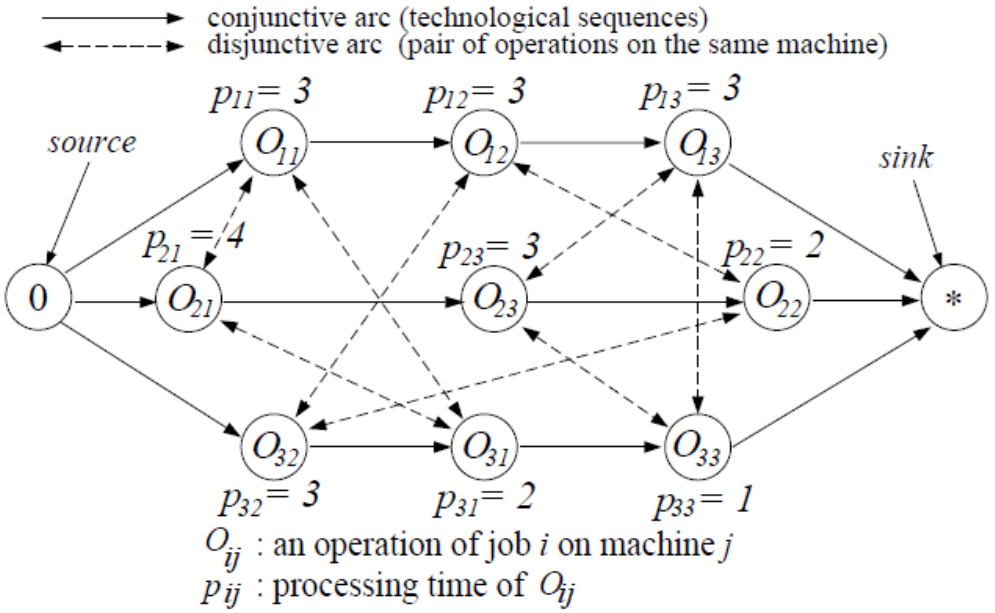


FIGURE 3: DISJUNCTIVE GRAPH REPRESENTATION OF A JSSP

Disjunctive graph helps in visualizing and understanding the structure of a JSSP problem.

The JSSP is not only N P-hard, but it is one of the worst members in the class. Even with a 3 X 3 problem where each job has 3 operations the search space can be as big as

Search Space= $(3^3)^3$

An indication complex a JSSP is, is the fact that one 10 X 10 problem formulated by (Muth and Thompson, 1963) remained unsolved for over 20 years before it was finally settled in 1985.

1.3. PROBLEM STATEMENT

1.3.1. RESEARCH OBJECTIVE

This study has the following objectives.

- Model JSSP as a game theoretic Multi-agent environment in which agents interact to achieve global optima. We seek to define games that govern agent strategy/actions in these environments.
- Visualization of the algorithms the JSSP and the algorithms that will be defined,
- Evaluating the algorithms using the available benchmark data for JSSP. Example of which is the compilation of important provided by the Operations Research Library (Beasley, 2005) .

1.3.2. RESEARCH QUESTIONS

The research problem in this thesis is defined as a job shop problem with precedence constraints on job operations denoted by definition one. The scheduling problem to be solved involves determining an optimal assignment of the operations of independent **n** jobs, where **n>0** that are to be processed on **m** non identical machines so that the total processing time for all the jobs, the makespan, is minimized. The jobs and the machines the following properties;

- Machine don't require idle time between jobs
- The schedule must be non-preemptive. That is, once a machine begins processing a stage of a job, it must complete that stage before doing anything else.
- Each job ***j*** consists of o_{ji} operations such that job ***j*** is $j(o_1 \ldots . o_n)$, and
- there is a precedent constraint on the operation such that operation **o**' preceding and operation **o**, should be processed before **o** is processed,

- each of the operation of a job must be processed by a specific machine, $p(o_{ji}, m_k)$ and
- processing of the job on the machine can take a certain duration $p(o_{ji}, m_k)$
- A processing of a job is completed after completion of processing of its last operation, the completion of a job is denoted as c_j.

The study aims to address the following questions,

- How can JSSP be modeled as game theoretic multi-agent environment. Modeling the problem multi-agent systems provides a number of advantages compared to centralized solution approaches. Among those is the ability to distribute the required computations over a number of entities, an increased amount of robustness, flexibility, and scalability due to the possibility of exchanging individual agents, or the benefit of allowing for spatial distribution of the work.Using the variation of games modeled by (Opiyo et al, 2009) in solving the parallel machine scheduling problem, these are, potential games and Random choice games. We seek to determine.How can the games be modified in order be able to provide feasible solutions to the Job Shop scheduling problems. The aims of algorithms defined here would be to reduce/minimize the makespan of a JSSP. The makespan(***J// Cmax***) is defined as the time the last job finishes processing, that is,

  If C_j is the time job j finishes processing then

$$C_{max} = \max_{j=1.....J} C_j$$

EQUATION 4: FINDING CMAX

The objective that needs satisfying becomes the search of C^* such that,

$$C^* = \min C_{max}$$

EQUATION 5: STATISFACTION OBJECTIVE

1.4.SIGNIFICANCE OF STUDY

Scheduling and sequencing have always been crucial decision-making tasks to support and enhance the productiveness of manufacturing organizations as well as logistics and service

providers. Job Shop Scheduling Problem (JSSP) has emerged as one most studied scheduling problem because of its common occurrence and complexity. The classical JSSP is well-known as an NP-hard problem. Most of the proposed evolutionary and operation research based solutions are sequential in their decision making and with current computational capabilities are able to provide solutions for small problems cases but as the job get larger it becomes computationally infeasible to achieve a feasible solution. Use of multi-agents to distribute decision making thus allowing for the possibility of distributing the computational resource as well as including fault tolerance.

Apart from the offering scalability advantage due to is distributed decision making nature. Multi agent systems also offer adaptability. (Opiyo et al, 2009) also state that The issue with the OR approaches is that most solutions are limited to each class of the scheduling problem that is solved. This makes it necessary to seek the invention of algorithms or heuristics for different problem classes. For example algorithms for **1||Cmax** are not guaranteed to solve the **3||Cmax** or the **Q||Cmax** problems. The agent-based approaches are different. The schedules are generated according to the agent behavior. This associates the qualities of schedules that are produced with the behavior of the agents. This shifts the burden of the scheduling problem from the invention of algorithms to determining the agent behavior that would lead to good schedules. The main advantage of using the agent-based approach is that in the extreme case that the problem class is unfamiliar the agents can learn the behavior that leads to good schedules on their own. (Opiyo et al, 2009) defined such algorithms for the parallel machine scheduling problem using game theoretic multi-agents. This study seeks to use the same concept in order to achieve adaptable algorithms for JSSP in attempt to solve for most **J||Cmax** instances.

CHAPTER TWO: LITERATURE REVIEW

Job shop scheduling is among the hardest combinatorial optimization problems and is NP-complete (Garey and Johnson, 1979). An NP-complete or NP-hard problem is where no algorithm exists (unless P=NP) that in polynomial time is able to solve all possible instances of the problem. Hence, the solution time risks increasing exponentially with the number of jobs. (Karin Thörnblad, 2013). According to (Karin Thörnblad, 2013) JSSP remains a NP-complete problem despite the objective function selected. As noted in section 1, the following can be the objective of a job shop scheduling problem.

Minimize the Makespan- The Makespan is the total length of the schedule, that is, the time it takes all jobs finish processing. This is formulated as

M= max{ C_1…….C_n}

Where,

C_j= the earliest time job ***j*** finishes processing.

EQUATION 6: MINIMIZE MAKESPAN

Minimize Tardiness- In situations where the jobs $\boldsymbol{j}$ have deadlines $\boldsymbol{d_j}$, tardiness is the duration of time delays past its deadline

Then tardiness $\boldsymbol{t_j}$,

$$\boldsymbol{t_j = max\{c_j - d_j\ , 0\}}$$

The tardiness of the schedule **T** is,

$$\mathbf{T} = \sum_{j=1}^{n} \boldsymbol{max\{c_j - d_j\ , 0\}}$$

Therefore,

$$\mathbf{T} = \sum_{j=1}^{n} t_j$$

EQUATION 7: MINIMIZE TARDINESS

Minimize lateness- Lateness for a job is defined as,

$$l_j = C_j - d_j$$

The Lateness of the schedule **L**

$$\mathbf{L} = \sum_{j=1}^{n} max\{c_j - d_j\}$$

$$\mathbf{T} = \sum_{j=1}^{n} t_j$$

EQUATION 8: MINIMIZE LATENESS

The objective that is most often utilized for scheduling problems is the minimization of the makespan , i.e., the time between the start of the first operation and the completion of the last operation of the schedule.

(Metta Haritha, 2008) noted that the nature of the scheduling environment plays a vital role in determining the job Schedules. she differentiated between two environments, A static environment, where the number of jobs and the arrival times are known in advance and a dynamic environment, where the arrival times of jobs are unknown at time of scheduling and scheduling is usually done as processing continues.(Madureira et al., 2001) observed that a dynamic scheduling system encounters the difficulties of randomness such as machine breakdowns, unexpected job orders etc. which are experienced in real world problems.

2.1. RESEARCH ON JOB SHOP SCHEDULING

Job Shop scheduling has been the subject of a significant amount of literature in the operations research and artificial intelligence. Research in scheduling theory has evolved over the past forty years and has been the subject of much significant literature. This is because, this problem is not only NP-hard it is also has the well-earned reputation of being one of the most computationally stubborn combinatorial problems considered to date. Over the past forty years different solution approaches have been proposed to address the JSSP. These approaches can be categorized in two, these are ;

Optimization Algorithms: These are usually mathematical programming based approaches that work toward achieving optimal solutions. According to *(Azizizoglu and Kirca 1999a)* they involve the process like formulating Mathematical models for the problem, and using exact algorithm such as branch-and-bound algorithms or mathematical formulation to solve the problem.

Approximation Algorithms: These are usually heuristic/Meta-heuristic algorithms based approaches that aim to give an approximately near optimal solution rather than the optimal solution. We look at the main classification of approximation algorithms , that is, priority dispatch rules, bottleneck based heuristics, artificial intelligence and local search methods.

In the following sub-sections we review the above approach, illustrating past and recent studies on each.

2.1.1. OPTIMIZATION ALGORITHMS

These methods simply build an optimum solution from the problem data by following a simple set of rules which exactly determine the processing order. Optimization algorithms have been known to solve a given problem optimally with a requirement that increases polynomial with respect to the size of the input. Optimization approaches usually process like formulating Mathematical models for the problem. These approaches form the earliest of approaches in solving scheduling problems, The first example of an efficient method and probably the earliest work in scheduling theory is (Johnson ,1954) who develops an efficient algorithm for a simple two machine flow shop whose objective function was to minimizes the maximum flow time. The two most common methods in these approaches are;

- Branch-and-bound algorithms
- Mathematical formulation.

2.1.1.1. BRANCH AND BOUND

Branch and Bound (B&B) is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems. The general definition of branch and bound is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. According to (A S Jain and S Meeran,1998) , A branch-and-bound algorithm consists of a systematic enumeration of all candidate solutions. The algorithm searches the complete space of solutions for a given problem for the best solution. However, explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions. The use of bounds for the function to be optimized combined with the value of the current best solution enables the algorithm to search parts of the solution space only implicitly (Jens Clausen,199). (A S Jain and S Meeran,1998) explains that in a typical branch and bound algorithm, large subsets of fruitless candidates are discarded en masse, by using upper and lower estimated bounds of the quantity being optimized. They state that Branch and Bound (BB) algorithms use a dynamically constructed tree structure as a means of representing the solution space of all feasible sequences. The search begins at the topmost (root) node and a complete selection is achieved once the lowest level (leaf) node has been evaluated. At any point during the solution process, the status of the solution with respect to the search of the solution space is described by a pool of yet unexplored subset of the tree and the best solution found so far. Initially only one subset exists, namely the complete solution space, and the best solution found so far is 1. The unexplored subspaces are represented as nodes in a dynamically generated search tree. Each node at a level p in the search tree represents a partial sequence of p operations. As implied by their name a branching as well as a bounding scheme is applied to perform the search. From an unselected (active) node the branching operation determines the next set of possible nodes from which the search could progress.

One of the most popular branch and bound is based on work of (Brucker et al, 1994) which was later extended by (Brucker and Thiele, 1996) where They consider is the general shop problem

with sequence-dependent setup time. The method is based on the disjunctive graph representation. E.g. for an instance of the job-shop scheduling problem the disjunctive graph

$G = (V, C \cup D)$

is defined as follows.

- V is the set of nodes, representing the operations of the jobs.
- There are two special nodes, a source 0 and a sink *. Each node *i* has a weight which is equal to the processing time p_i of the corresponding operation, whereby p. and p* are equal to 0.
- C is the set of conjunctive arcs which reflect the job-order of the operations. For every pair of operations that require the same machine there is an undirected, so-called disjunctive arc.
- The set of all these arcs is denoted by D.

The basic scheduling decision on this model is to define an ordering between all those operations which have to be processed on the same machine, i.e. to fix precedence relations between these operations. This branch and bound algorithm solved the famous 10 x 10 benchmark problem in less than 19 min on a workstation. The algorithm proved unsuitable for benchmark problems larger than the 10 X 10 problem.

More recently,(A. AitZai and M. Boudhar, 2013) proposed a parallel version of a branch-and-bound method based on an implicit enumeration, that further improved the speed of solving instances smaller or equal to the 10 X10. Though the algorithm still proved inefficient for case larger than this.

2.1.1.2. MATHEMATICAL FORMULATION

These methods usually involve finding the optimal solution by describing the JSSP as some mathematical model. Mathematical modeling as a solution technique was made popular after works (Wagner, 1959) and (Manne, 1960) who both introduced an integer linear-programming model for machine scheduling. Integer linear-programming is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers the aim of linear programing is to achieve the best outcome) in a mathematical model whose requirements are represented by linear relationships.

Since them other models have been devised to solve for scheduling problems. Most notably Lagrangian relaxation (LR) approaches devised by (Van De Velde, 1991) and (H. Chen and P.B. Luh, 2003). Lagrangian relaxation is a relaxation method which approximates a difficult problem of constrained optimization by a simpler problem. It works on the assumption that a solution to the relaxed problem is an approximate solution to the original problem.

Mathematical optimization models with currently available resources have proved successful when present with simple problem instance but their complexity grows polynomially with the increase with the instance size and the algorithms become computationally infeasible.

2.1.2. APPROXIMATION ALGORITHMS

These are usually heuristic/Meta-heuristic algorithms based approaches. A Meta-heuristic is a higher-level procedure or heuristic designed to find or generate procedure or heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem. These methods are usually preferred and are better for larger problem/dynamic problems/ problems with multiple constraints as they are more likely to converge to a good enough solution much earlier than optimization methods can achieve an optimal solution. In most problem instance successful algorithm have shown that the solution derived from approximation approaches are usually close to enough to the optimal solution. Since the

solution is close to optimal and generated in much less time, (Blum and C.Roli, A. 2003) argue that the benefit of having using far less resources outweighs the disadvantage of not arriving to an absolute optimal solution. We review four main categories of approximation technique are considered: priority dispatch rules, bottleneck based heuristics, artificial intelligence and local search methods.(Karin Thörnblad, 2013) explained that the major disadvantage of metaheuristics is that there is often no other stopping criteria than a maximum allowed number of iterations, or a maximum computation time. She also states that because of this the quality of the solution obtained is often unknown.

In the following sub section we look at early and recent research on the four main common classifications of approximation methods, that is, priority dispatch rules, bottleneck based heuristics, artificial intelligence and local search methods.

2.1.2.1. PRIORITY DISPATCH RULES

Priority dispatch rules is a technique of finding a near optimal solution for scheduling problems by applying heuristic dispatching rules. Dispatch rule in a simplistic approach would involve assigning priorities to jobs/operations based on criteria which could be a task corresponding to longest/shortest operation time; most/least successors; or ranked positional weight, i.e., sum of operation times of its predecessors, the jobs or operations deadline, etc. The priority is used to assign jobs/operations to machine whenever they become available.

Priority dispatch rules perform reasonably well in a wide range of environments, and are relatively easy to understand. They also need only minimal computational time, which allows them to be used even in real-time, on-line scheduling environments. (Torsten Hildebrandt et al, 2010). Hundreds over approaches based on priority dispatch rules have been proposed in handling job shop scheduling. A summary of over 100 classical dispatching rules can be found in (Panwalkar and Iskander, 1977). The Earliest being work being by (Jackson, 1955). More recently (H Ingimundardottir and P Runarsson, 2010) introduced a priority dispatch rules approach for job shop scheduling based on Supervised Learning.

2.1.2.2. BOTTLENECK BASED HEURISTICS

The most common of these approaches is the shifting bottleneck heuristics. These algorithms work on the assumption that in cases where jobs/machines are competing with each other for the same resources (machines), there is always be one or more resources that act as a 'bottleneck' in the processing. The algorithm then works to reduce/minimize the bottleneck. This algorithm has proven very efficient for solving for instances equal or less to those of 30 machines and 50 jobs. It was first introduced for the reducing makespan for JSSP problems by (Adams et al., 1988) and this was later extended by (Balas et al. 1995). Because of is relatively good performance the approach has been extended to other performance measures in solving a JSSP like total weighted tardiness by (Pinedo M, Singer M, 1999) and maximum lateness by (Demirkol et al,1997) and (Uzsoy R and Wang C S., 2000). More recently (Gokhale et al, 2011) address a scheduling problem for minimizing total weighted tardiness in JSSP.

2.1.2.3. LOCAL SEARCH METHODS

Local search algorithms have been around for over forty years and are in evolution through many research papers. The algorithms work on the fundamental idea that given an initial or set of initial solution, a best fit can be obtained making small improvement on solution, this is done over and over until a certain criteria is met. How the initial set selected allows with the improvement/search methodology and the evaluation functions have been subject of much research. Currently the two most popular approaches in local searches include;

Tabu search algorithms; Tabu searches where introduced by (Glover, 1986) . (Metta Haritha, 2008) works defines as follows. "*A Tabu search as the Tabu search algorithm stores the previous search history (list of obtained solutions) in its memory. When the search process is carried out in a new neighborhood the algorithm tries to find the best solution by excluding earlier solutions stored in the memory. Therefore this procedure forbids/ tabus moves in new neighborhoods, by guiding the search process away from solutions that resemble previous ones*".

Simulated Annealing is based on the works of (Kirkpatrick, et al. ,1983) and (Metropolis, et al, 1953) the technique coined the methodology from the analogy between annealing process and the search for the minimum in a general process. (Metta Haritha, 2008) describes the algorithm as follows. The algorithm starts with a randomly generated set of initial solutions and at a high starting temperature 'T'. The algorithm replaces the present solution with a solution from its neighborhood if that solution is better than the current one. A better solution in this algorithm could be the one whose objective function value is less than latter solutions. The value of temperature gradually decreases during the search process, thereby the solutions are replaced more number of times at the beginning and then toward the end. The above steps are repeated until a termination criterion is reached. In most case once the termination criteria is achieved, the best out of the current set of solution is selected as the near optimal solution.

The most notable recent studies on local search algorithms is Zhang et al.'s hybrid tabu search / simulated annealing algorithm (Zhang et al., 2008) . Local Search methods are known to be simplistic, easy to implement and very efficient in regards to use of computing resources. The major disadvantage is that local search emphasis fails to consider effects at a global scale, a fact that sometimes lead to poor solutions in larger test instances.

2.1.2.4. ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is the subfield of computer science concerned with building software that can think and act rationally and independently as if to exhibit human intelligence. Artificial intelligence has been used as problem solving mechanism in different fields. In JSSP, hundreds of artificial intelligence approaches have been developed over the last 40 years. Common artificial intelligence employed for JSSP includes use of;

- Genetic Algorithms; (Davis, 1985)
- Artificial Immune System; (Coello et al 2003)
- Artificial Nueral Networks; (Yu and liang 2001)
- Reinforcement Learning ; (G Weiss, 2013)
- Multi-agents Systems; (Opiyo et al, 2009)

- Ant colony optimization; (Zhang et al 2006)

Artitifial Intelligence approaches have be identified by to have the following four main advantages.

- They employ both quantitative and qualitative knowledge in the decision-making process.
- Second, they are capable of generating heuristics that are significantly more complex problem instances than the earlier approaches
- The third is that the selection of the best heuristic can be based on information about the entire JSSP and not localization like in local search policies. They can also adapt in a dynamic JSSP to the change in state or configuration of the JSSP e.g machine break downs, additional job arrivals.
- They can model complex relationships in elegant new data structures and have techniques that can be used for powerful manipulation of the information in the data structures.

This research will try to employ an Artificial intelligence approach to the classic JSSP problem, the approach used is based on Multi-agents and game theory. The approach is extends to the works of (Opiyo et al, 2009) which employed game theoretic multi-agents in solving the parallel machine scheduling problem and has relations to agent based model techniques reviewed by (G Weiss, 2013). As was earlier mentioned, the benefit of using this approach as, first, the schedules are generated according to the agent behavior. This associates the qualities of schedules that are produced with the behavior of the agents. This shifts the burden of the scheduling problem from the invention of algorithms to determining the agent behavior that would lead to good schedules. The main advantage of using the agent-based approach is that in the extreme case that the problem class is unfamiliar the agents can learn the behavior that leads to good schedules on their own. (Opiyo et al, 2009).Secondly, Use of multi-agents enables to distribute decision making thus allowing for the possibility of distributing the computational resource as well as including Profit from inherent properties of distributed systems like robustness, fault tolerance, parallelism and scalability.(G Weiss, 2013).

2.2. MULTI-AGENT SYSTEMS

A multi-agent system (M.A.S.) is a computerized system composed of multiple interacting intelligent agents within an environment. Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Because of its nature it lends its self to solving problems where distributed decisions are necessary. Multi-agent systems are centered on the concept of a rational agent. An agent is anything that can perceive its environment through sensors and act upon that environment through actuators (Russell and Norvig, 2003). Flexibility and rationality are achieved by an agent on the basis of key processes such as problem solving, planning, and decision making, and learning. As an interacting entity, an agent can be affected in its activities by other agents and perhaps by humans (S. Russell, 2003). Multi-agent systems consist of multiple agents and their environment. MAS systems are used to model real world problems where distributed decision making is need to achieve the solutions.

2.2.1. CHARACTERISTICS OF AGENTS IN MAS.

The following are the main characteristics of agents in a multi-agent system.

1. **Autonomous:** An agent is capable of acting independently, exhibiting control over their internal state. Agents collaborate/cooperate or compete with other agents in their environment in order to maximize/optimize a certain gain. The gain can be an individual gain or a social cumulative gain.
2. **Reactive:** An agent maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful). If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure – program just executes blindly. But this is not usually the case in real world application, where the environment is usually dynamic. This necessitates for intelligent software entities like agents to be reactive based on the environmental states.
3. **Pro-active:** Agents should be capable of generating and attempting to achieve goals; not driven solely by events; taking the initiative and/or recognizing opportunities.

4. **Social ability:** Social ability in agents is the ability to interact with other agents (and possibly humans) via some kind of *agent-communication mode*, e.g Messaging or bulletin board in order to satisfy their design objectives.
5. **Mobility:** the ability of an agent to move around an electronic network
6. **Veracity:** an agent will not knowingly communicate false information
7. **Benevolence:** agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it
8. **Rationality:** agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit
9. **Learning/adaption:** agents improve performance over time

According to (G WeiB, 2000) interest in multi-agent systems is largely founded on the insight that many real world problems are best modeled using a set of agents instead of a single agent. In particular, multi-agent modeling makes it possible to Cope with natural constraints like the limitations of the processing power of a single agent or the physical distribution of the data to be processed and profit from inherent properties of distributed systems like robustness, fault tolerance, parallelism and scalability.

(V lesser,1995) state that The current set of multi-agent applications can be classified into three broad areas. First, distributed situation assessment Applications, such as distributed network diagnosis, emphasize how (diagnostic) agents with different spheres of awareness and control (network segments)should share their local interpretations to arrive at consistent and comprehensive explanations and responses.

Secondly distributed expert systems applications, such as concurrent engineering, emphasize how agents negotiate over collective solutions (designs) given their different expertise and criteria. The next generation of applications alluded to will probably involve all the emphases of these generic applications and more.

Finally ,as in our case, Distributed resource planning and allocation applications, such as distributed factory scheduling, emphasize how (scheduling) agents (associated with each work cell) should coordinate their schedules to avoid and resolve conflicts over resources and to maximize system output.

2.3.GAME THEORY

Game Theory was launched by (Neumann and Oskar Morgenstern,1944) in their book Theory of Games and Economic Behavior. They state that Game theory is an economic theory that models interactions between rational agents as games of two or more players that can choose from a set of strategies and the corresponding preferences. It is the mathematical study of interactive decision making in the sense that the agents involved in the decisions take into account their own choices and those of others. Choices are determined by stable preferences concerning the outcomes of their possible decisions, and agents act strategically, in other words, they take into account the relation between their own

(Osborne and Rubinstein, 1994) described multi-agent decision making as a subject of game theory. (Vlassis, 2005) described the following, although originally designed for modeling economical interactions, game theory has developed into an independent field with solid mathematical foundations and many applications. The theory tries to understand the behavior of interacting agents under conditions of uncertainty, and is based on two premises, First that the participating agents are rational. Second, that they reason strategically, that is, they take into account the other agents' decisions in their decision making.

Depending on how an Agent selects its action two classic types of game can be distinguished,

- **Strategic game**, here each agent is allowed chooses their strategy once at the start of the game, and then all agents take their actions simultaneously. The normal (or strategic form) game is usually represented by a matrix which shows the players, strategies, and pay-offs as depicted by figure 1.4 below.

	Player 2 chooses *Left*	Player 2 chooses *Right*
Player 1	**4**, **3**	**–1**, **–1**

chooses *Up*		
Player 1 chooses *Down*	**0**, **0**	**3**, **4**

Table 1: *Normal form or payoff matrix of a 2-player, 2-strategy game*

In this example (borrowed from Wikipedia) there are two players; one chooses the row and the other chooses the column. Each player has two strategies, which are specified by the number of rows and the number of columns. The payoffs are provided in the interior. The first number is the payoff received by the row player; the second is the payoff for the column player. Suppose that Player 1 plays Up and that Player 2 plays Left. Then Player 1 gets a payoff of 4, and Player 2 gets 3. This game assumes all players make moves simultaneously.

- **Extensive game**, here the agents take their actions in turn and agents actions can be based on their actions of preceding agents action. The extensive form can be used to formalize these games with a time sequencing of moves. Games here can be depicted as if they are played on decision tree as show by **figure 1.3** Here each vertex (or node) represents a point of choice for a player. The player is specified by a number listed by the vertex. The lines out of the vertex represent a possible action for that player. The payoffs are specified at the bottom of the tree.

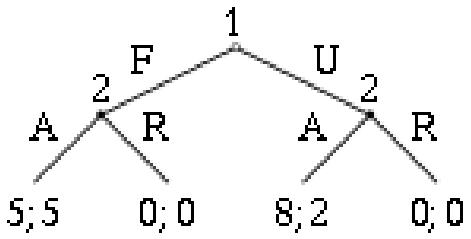


FIGURE 4:GAME THEORY IN EXTENSIVE NORM(BORROWED FROM WIKIPEDIA)

The extensive form can be viewed as a multi-player generalization of a decision tree. (Fudenberg and Tirole, 1991) .The above figure represents a game where there are two players. Player 1 moves first and chooses either F or U. Player 2 sees Player 1's move and then chooses A or R. Suppose that Player 1 chooses U and then Player 2 chooses A, then Player 1 gets 8 and Player 2 gets 2.

(Opiyo et al, 2008) in seeking to devise game theoretic multi-agent based algorithms for solving parallel machine scheduling problem. In defined three types of games these are,

Dispersion games: These are those in which the agents win positive payoffs when they choose distinct actions. This game is a form of anti-coordination games described by (Trond et al. 2002). He describes dispersion games as a game in which agents prefer to be disperse over their actions in that they choose deferent actions than those chosen by other agents. Dispersion Games are used to model real world problems, the classical example is a load balancing problem this problem can be modeled as a Dispersion game in which the agents are the users, the possible actions are the resources, and the equilibria of the game are the outcomes in which agents are maximally dispersed. (Opiyo et al, 2008) gives natural examples such as setting up new businesses in areas where there are no similar businesses and choosing to drive on streets with low traffic, are some of the activities that can be modeled by dispersion games.

In dispersion games the desired end state is a Nash equilibrium, a Nash equilibrium is a state in which all players are relatively satisfied with the choices they've made, that is, If each player has chosen a strategy and no player can benefit by changing strategies while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitute a Nash equilibrium. Therefore dispersion games seek to identify a Nash equilibrium.

Potential games: In these games the key is to achieve/learn a social policy known as the potential function which will guide the actions of the players. It is assumed that the learned function will guided the players/agents in making decision that will ensure a social good. (Sandholm 2001) describes these games as those in which the incentive of all players to change

their strategy is expressed in one global function called the potential function. The progressive actions of the participants lead to a stable state. (Opiyo et al, 2008) describes the use of taxes or public charges to influence the decisions of people are a form of potential game.(Riedmiller, et al, 2009) describe reinforcement learning based approaches to achieve a potential function, they describe a policy based search algorithm based on Markov Decision Processes to achieve a potential function.

Random Games, these are game in which players/agents randomly select an action with hope of achieving a near optimal state. These are done in a predefined number of iterations and the overall best state from the iterations is selected.

CHAPTER THREE: METHODOLOGY

In this section, research methods for this study are described

3.1. RESEARCH APPROACH

Our first objective is to model JSSP as a game theoretic Multi-agent environment in which agents interact to achieve global optima. We seek to define games that govern agent strategy/actions in these environments, evaluate and review Multi-agent environment/archi-type to discover the best archi-type to adopt, this will also involve evaluating other models of multi-agents adopted in solving scheduling problems. We will further design/ definition of our multi-agent environment and finally review of the 3 games discussed in (Opiyo et al, 2009) to be able to extend or modify in order to be applicable to our problem.

The second objective is to provide visualization of the algorithms the JSSP and the algorithms that will be defined. This will involve review of the available development toolkit and their appropriateness to use in our cause, designing a conceptual model of the visualization tool defining our multi-agents environment and finally developing the visualization tool/realize the conceptual model

Our third objective involves evaluating the algorithms using the available benchmark data for JSSP. Example of which is the compilation of important provided by the Beasly's Operations Research Library (Beasley, 2005) .This would involve a review on benchmark criteria in JSSP Including data instances and use of benchmark for each of our defined games.

3.2. RESULT PRESENTATION

In order to analyze the result the visualization tool will present the data and derivation of the solutions; we will then proceed to tabulate the result against available performance benchmarks. Will then document our finding and conclusion from the study for each of the defined games

3.3. TOOLS

For Visualization we will use the following Visual Studio 2012.

CHAPTER FOUR: ALGORITHM FORMULATION

In this chapter we define our job Shop algorithm as a Multi-agent system environment and we formulate the game theoretic algorithms to solve the problem. In the next chapter we will perform experiments on our algorithms and evaluate using the defined benchmark data.

4.1.JOB SCHEDULING AS A MAS ENVIRONMENT

In Chapter we formalized the job shop scheduling problem as follows;

A problem instance ***P= (M, O, J)*** in job shop scheduling consists of

1. A set ***M*** of Machines,
2. A set ***O*** of operations ***o***, each associated with a machine ***M(o)Є M*** and having a duration ***d(o) Є N*** and
3. A set ***J*** of jobs **J(o_1……o_n)**(each operation has exactly one occurrence.)

A given Schedule ***S*** for ***P*** assigns to every operation ***o*** a starting time ***T(o)***: on the relevant machine time

3. ***T (o)≥***0 for all ***o Є O***
4. We define an operations processing time ***P(o)*** *as*

   P(o)= T (o)+ d(o)
5. We define a precedent constraint on ***T (o')on T (o)*** such that

   T (o)≥ T (o')+ d(o')

   for operations ***o'*** preceding **o** in the same job.

Our objective function in the problem is to minimize the makespan in search of a near optimal schedule. We defined the Makespan as the time the last machine finishes process the last operation, therefore the makespan of a schedule ***M(S),*** can be defined as

$$M(S) = MAX(T(o_1)+ d(o_1) , T(o_2)+ d(o_2),.....T(o_n)+ d(o_n))$$

EQUATION 9: CALCULATE MAKESPAN

To define this as a multi-agent system, We adopt and extend [Opiyo et al,2009]'s definition of a multi-agent system for parallel machine scheduling. Same as their study had, we make the following considerations; First, A multi agent system to be a system that consists of the agents, the agents act as autonomous entities that can sense and react to the changes in their environments.

Secondly, game theory as the study of interactions in contexts where the participants make the choices to affect the overall status in the game. A game is a structure that consists of a set of the agents, a set of the agent actions or choices and a set of the agent payoffs associated with their actions. A situation where schedules are generated by agents as they choose machines can be considered as a game [Opiyo et al. 2008b].

4.2. RANDOM TOKEN GAME

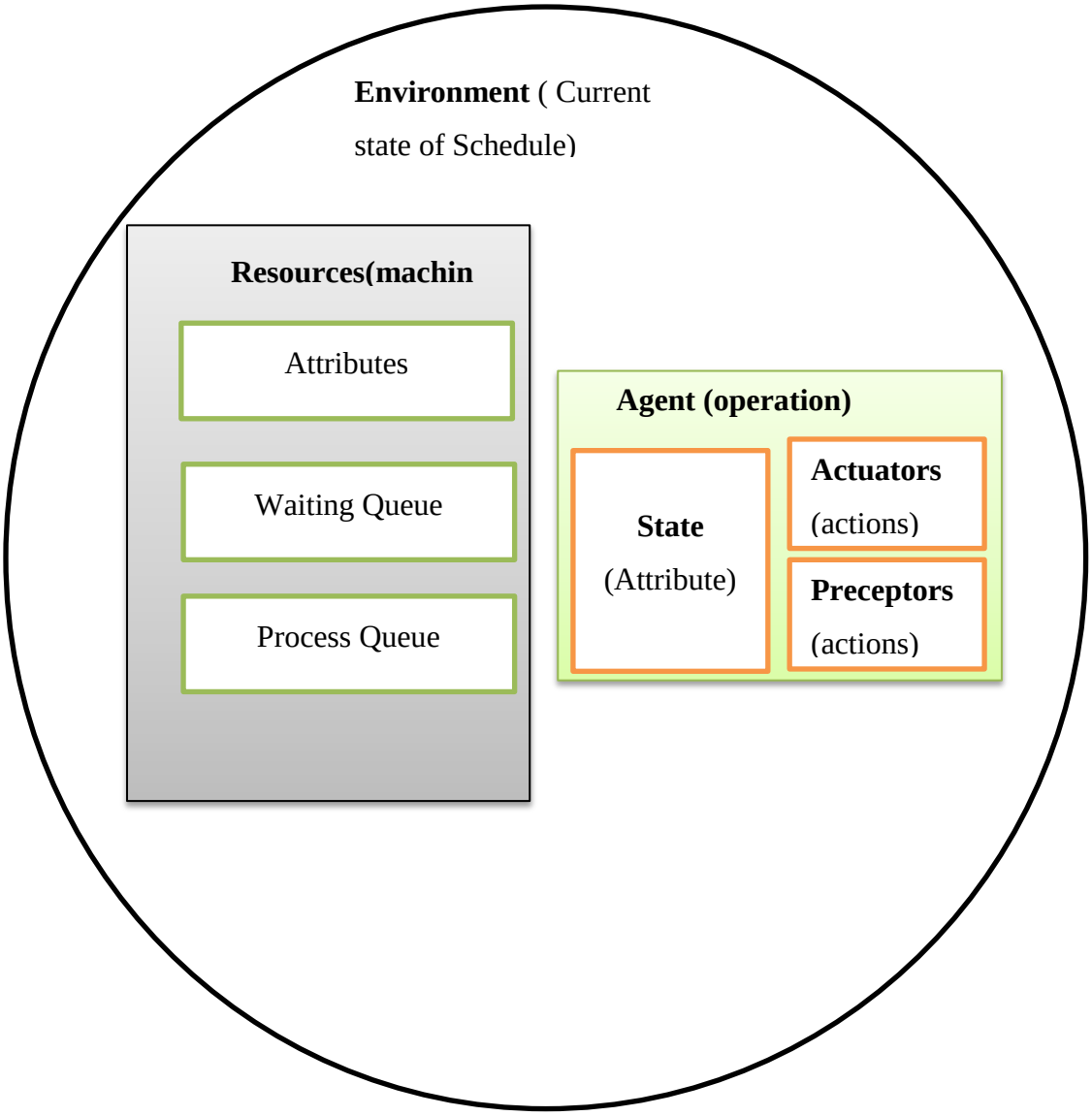
4.2.1. MAS ENVIRONMENT FOR RANDOM TOKEN GAME

From the above perspective of a multi-agent system, we redefine the job shop scheduling problem as MAS environment to suite our random token game as follows this follows;

We define jobs as a categorization of agents, where agents represent a single operation. We define all operation as agents that will either compete or cooperate with each other in order to achieve a schedule. Each Agent belongs to a particular categorization/ job. The current state of a schedule defines the agent's external environment, while the agent's internal state is defined by its attributes as shown by the table below. The Agent can also perform the actions as illustrated by the table below to affect its internal status. Some Actions act as it actuators to affect both the environment and its internal state, some act as preceptors to sense the state of

the environment and there is also a messaging action to enable the agent communicate with other agents. The diagram below depicts an Agents environment while the table that follows illustrates the agents attributes(internal State) and actions(actuators and Perceptors)

FIGURE 5:SIMPLE ILLUSTRATION OF A 1X1 MAS ENVIRONMENT



AGENT(OPERATION)		
Attirbutes		
	OperationID	*The Id of the operation*
	PredecesorID	*ID of the operation's predecessor*
	SuccesorID	*ID of the operation's successor*
	MachineID	*ID of the machine which the operation is to be processed*
	ProcessingTime	*The processing time on the machine*
	Status	*The status of the operation e.g* •*Waiting:- The Agent is idle and waiting for a turn,* •*Active:- The Agent is allowed to make a move* •*Scheduled;-The agent as achieved time share on a schedule;*
	JobID	*The id of the job/ categorization of the operations*
	StartTime	*The current processing start time if scheduled*
	MessageQueue	*A queue for all incoming messages*
Actions		
	Move	*Action allowing agent to make a choice*
	Message	*Action allowing agent to send message to other agents*
	Read	*Action allowing agent to sense its environment*

TABLE 2: SIMPLE ILLUSTRATION OF THE AGENT'S ACTIONS AND ATTRIBUTES

Agents can compete or cooperate in order to get processing time on the machines, this action lead to formulation of a schedule. We consider machines as a resource on which agents compete for processing time on. We also consider a schedule complete when the all the agents have made their turn and have acquired a time share allocation on a machine. To achieve this,

the agents act by employing a strategy in the moves, the strategy is based on the algorithms we define the sections that sections follow.

As show by the diagram we consider machines as resources in the environment for which the agents compete for. Machines have two key attributes;

- **Waiting Queue;** this is a queue that holds an operation to be performed on the machine before they are allocated a time share on the machine.
- **Process Queue;** this is a queue that the schedule of processing on the machine. The Queue holds the list of Agents that have been allocated a time share on the machine and the respective start time.

We also define a referee agent in some games that responsible in marshaling the games, the role and structure of the referee and role of the referee will depend on the type of game environment that defined in.

4.2.2. DEFINING RANDOM TOKEN GAME

In defining algorithms for parallel machine scheduling [Opiyo et al, 2008] define random choice games are those in which the agents make choices at random without considering any other matters. In their definition an agent are allowed to make moves in turn and each agent in its turn makes random decision which machines they would like to be processed on and select the earliest available time slot on the machine. After all the agents have made their move the resultant schedule is evaluated. This process is repeated in several rounds and at the end the most suitable/ shortest schedule is select as a feasible solution. This work was able to demonstrate that it is possible to achieve a relatively feasible schedule using random select of schedule in a schedule search space. It gives us great in on the distribution of solution in the search space. We try to define a similar algorithm for job shop scheduling.

Unlike in parallel machine scheduling, job shop scheduling as the following complications when trying to employ a pure random strategy in selection of a feasible solution from the search space;

- Agents/operations are tired to a machine, that is, the machine is already pre-selected. Unlike in parallel machine scheduling where agents act by selecting the machines.
- The is a precedence constraints among agents, that is the start time ***S(A)***of an agent ***A*** ,

  S(A) ≥ S(A')+ T(A')

EQUATION 10: PRECEDENT CONSTRAINT

Where ***A'*** is the preceding Agent with a processing time of ***T(A')***. The availability of this constraint preempts the possibility of having a pure random strategy.

These constraints limit the flexibility of an agent in machine selection and put a constraint its selection of a time slot on a machine. To achieve similar a random selection of solution in a such space with the above constraints in job shop scheduling, we introduce a random token notion. The Random token randomizes the playing turn for the agents. This works as follows; we divided the game in two stages for all rounds, the stages are as follows;

Selection Stage; this stage allows random selection of agent turns with will result in random ordering of agents in the machine. The Process flows is as follows; First we introduce a single token in the environment. The token generated by the referee agent for the round and is assigned by the referee agent to an agent at random at least once in a single round. A round is instances of a game where all the agents have made a single move and a complete schedule can be define. The game start with all agent state with a waiting status, when an agent receives the token, their status changes to active and they are allowed to make a move to the assigned machine. Once an agent selects a machine they are add to waiting queue of a machine in a priority of first come first served. The agent then release the token to the referee agent which then assigns it to another agent at random and the selection process continues until all the agents have had the token and have made their selection. We then proceed to the allocation stage.

Allocation Stage; the allocation stage was motivated by shift bottle neck paradigm. In shift Bottleneck, an initial selection of a schedule is selected as we have done in the selection stage without actual time share allocation. If we were to evaluate the schedule as it is now with the agents arranged in a first come first serve order, the schedule will have multiple delays among

the operations and we will have unnecessary idle time on the machine. In fact the initial schedule would be among the worst performers in the search space. An example would be in an instance of 10**X**10 problem where the last agent of job gets the random token first among all the other agents on other jobs that share the same machine instance, this agent will select the machine, the machine will remain idle until all the other agents of the same job have finished processing, this in turns cause delay on the other agents waiting for that machine and the ripple effects can will spread across the schedule. This bottleneck can also cause a deadlock with the schedule. The shift bottle neck algorithm recognizes that in a schedule the is always at least one point/bottleneck that affects its performance. The aim of the shift bottleneck is slow minimize/shift the bottle in several iterations. We adopt a similar iterative approach but in our algorithm it's the agent that makes the decision whether to shift or stay based on their internal states, The agent act for the social good and if an agent consider itself a possible bottleneck, it shifts self to remove the bottle neck if not its stays . The allocation stage proceeds as follows; After the selection stage all the agents would have acquired a priority on the machine's waiting queue. Once a agent has selected a machine its status is changed back to 'waiting'. since this its first come first serve, the ***n^{th} Agent to make the selection of the machine will receive n^{th} priority on the machine.*** To formally state this, If O_{mn} Represents an agent O with processing time on machine m and it was the n^{th} agent to make a selection on the machine, then its priority value $P(O_{mn})$ (lower value signifying higher priority) is;

$$P(O_{mn}) \leq P(O'_{m(n+1)}) \leq P(O''_{m(n+2)})$$

EQUATION 11:AGENT PRIORITY SETTING

Where O' and O'' followed agent O in selection of the machine in that respective order. The Initial selection in most case will not be a candidate solution in the search space as the paths cannot be represented by a directed graph. The subsequent steps refine the selection into a candidate schedule.

In the next step, all the waiting agents with the highest priority on each machine's waiting queue are allowed to a turn, there status changes to active and they are allowed to evaluate their position. If an agent see that they could be possible bottlenecks they will choose to move to the back of the queue assume the lowest priority on the queue and status change back to waiting. The Agent suspects it may bottleneck using the following criteria; One, is if an agent ***A*** has a predecessor and the predecessor has not been schedule yet (acquired a time share), then ***A*** knows it's a might be a bottleneck on a machine if there exists other agents on the machine with a lower priority. In this case the agent will move to the back of the queue.

Secondly, If an agent ***A'*** has a predecessor ***A*** that has already been scheduled and its difference between ***A*** 's expected processing end time , ***P(A)*** and the *'next available start time'* on the machine **M** , **E(M)** is twice as big as the average processing time of the all the agents queued on the machine, then the agent suspect itself to be a bottleneck. A Machines *'next available start time'*, ***E(M)***, is the sum of all the agents that have been scheduled on the machine. If a machine M has 3 agents scheduled on it, An agent defined as $A_{(job, machine)}$.

$$E(M)= (P(A_{1M}) + P(A_{2M}) + P(A_{4M}))/3$$

EQUATION 12:NAST

if (A'_{1m}) is the one evaluating it situation and it has a predecessor **A**, the processing end time of **A,**

$$P(A) =S(A)+D(A)$$

Where;-

S(A) is the processing start time of ***A,***

D(A) is the processing time/duration of ***A.***

A at this point would consider its 'possible' processing start time, $S(A'_{1m})$, as equal to the processing end time , ***P(A),*** of ***A***

$$S\ (A'_{1m}) = P(A)$$

EQUATION 13: PROCESING END TIME

(A'_{1m}) consider itself a bottleneck in the schedule, and move to the back of the machines waiting queue.

Thirdly, If the agent (A'_{1m}) has evaluated its situation and does not consider itself a bottleneck, the agent will be scheduled on the machine by selecting the earliest possible start time on the machine. This would be the greater of ***P(A)***and ***E(M).*** That is ;

If $P(A) \geq E(M)$ then $S(A'_{1m}) = P(A)$

else

$S(A'_{1m}) = E(M)$

Once an agent has been schedule the priority listing of all other jobs in the waiting queue is adjusted.

The same steps are repeated for the number of Iteration needed till all the agents have been successfully scheduled.

A complete **selection stage** followed by a complete **allocation stage** constitute a round in the game, each round produces a candidate schedule from the search space. At the end of the game the refree agents evaluates the makespan m_s of all the candidate solution $s \in S$ where S represents the search space and selects a feasible solution ***f(s)*** using the following criteria.

$$f(m_s) = MIN(m_1, m_2, m_3, \ldots\ldots m_s)$$

EQUATION 14: FEASIBLE SCHEDULE FROM SEARCH SPACE

Because we achieved a random initial selection by using a randomized token. We can say that we are selecting schedules at random from the search space and thus we have achieved a similar effect that [Opiyo, et al] achieved with their random games in parallel machine scheduling. Therefore we can state for a typical job shop problem there is a random distribution of solution on the search space.

4.2.3. ILLUSTRATION OF A RANDOM TOKEN GAME.

In this section we illustrate the random token game using a simple example of a 2X3 Job shop scheduling problem. In our example we use the following syntax to represent an operation,

Job (machine, processing time)

We have the following job and their operations in order of processing sequence.

Job X: x(3,5) , x(1,6), x(2,2)

Job Y: y(2,3), y(3,4), y(1,2)

A disjunctive representation of the graph is as follows.

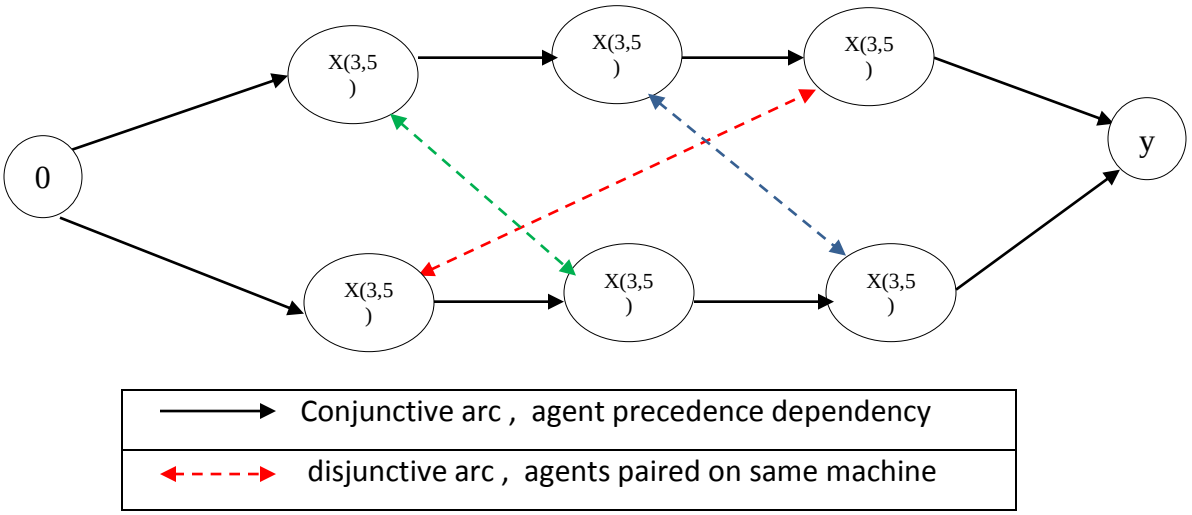


FIGURE 6: DISJUNCTIVE GRAPH

Once an agent has been scheduled we introduce a third value to represent its processing start time (in minutes), that is

Job (machine, processing time, processing start time)

e.g. **x(3,5,4)**

We have 3 machines each machine will have a waiting queue and schedule queue as we had earlier defined. Suppose after the random selection stage we had the following arrangement on waiting queue. Note that the schedule queue will always start empty with the NAST (next available start time of each machine set to time 0. Also note that agents arranged in the queue in order of first come first serve, therefore the jobs in the first cell always have the priority and

are next in turn to move. If they suspect themselves to be bottle necks they will shift themselves to the furthers cell, allowing other agents to shift forward

Waiting Queue			Schedule Queue			
						NAST
Machine 1	y(1,2)	x(1,6	**Machine 1**			**0**
Machine 2	x(2,2)	y(2,3	**Machine 2**			**0**
Machine 3	y(3,4)	x(3,5	**Machine 3**			**0**

FIGURE 7: RTG INTERATION 0

We begin iteration in the allocation stage as follows;

- **ITERATION 1**

From the initial arrangement, all the agents with priority on each machine evaluated their position. In this case all the agents deemed themselves bottlenecks (based on the earlier defined rule and shifted their positions

Waiting Queue			Schedule Queue			
						NAST
Machine 1	x(1,6)	y(1,2	**Machine 1**			**0**
Machine 2	y(2,3)	x(2,2	**Machine 2**			**0**
Machine 3	x(3,5)	y(3,4	**Machine 3**			**0**

FIGURE 8:RTG INTERATION 1

- **ITERATION 2**

In this Iteration y(2,3) and x(3,5) having no predecessors considered themselves not to be bottlenecks and acquired a schedule on their respective machine while x(1,6) still deemed itself a bottle neck and shifted. The results are as follows. Also note change on the machines NAST

Waiting Queue			Schedule Queue			
						NAST
Machine 1	y(1,2)	x(1,6	**Machine 1**			**0**
Machine 2	x(2,2)		**Machine 2**	y(2,3,0		**3**
Machine 3	y(3,4)		**Machine 3**	x(3,5,0		**5**

FIGURE 9:RTG INTERATION 2

- **ITERATION 3**

In this Iteration y(3,4) having a scheduled predecessor and there being no other agents left on the machine, acquired a schedule on it respective machine. Note that even though its predecessor's processing time P(A)=3, it acquire a start time of 5. This is because the for its scheduled predecessor, y(2,3,0), which is calculated as

P(A) =S(A)+D(A)

=0+3

=3

Where;-

S(A) is the processing start time of ***A,***

D(A) is the processing time/duration of ***A.***

is lower than the y(3,4)'s respective machine's NAST , $E(M_3)$. machine 3 has a

E(M$_3$)=5

And since our game rules say that,

- If ***P(A)≥ E(M)*** then ***S (A'$_{1m}$) = P(A)***

  else

  S (A'$_{1m}$) = E(M)

  For **A'** which is preceded by **A**

Then the processing start time for y(3,4), S(y(3,4))= 5 or y(3,4,5). This also affected its respective machines NAST as illustrated on the diagram.

Also on this y(1,2) and x(2,2) deemed themselves bottlenecks. While y(1,2) shifted, x(2,2) had no need to shift because there was no agent on its respective machine waiting queue with a lower priority. The result is as shown by the illustration below

Waiting Queue			Schedule Queue			
						NAST
Machine 1	x(1,6)	y(1,2	**Machine 1**			**0**
Machine 2	x(2,2)		**Machine 2**	y(2,3,0		**3**
Machine 3			**Machine 3**	x(3,5,0	y(3,4,5	**9**

FIGURE 10:RTG INTERATION 3

- **ITERATION 4**

This Iteration saw x(1,6) and x(2,2) get scheduled while y(1,2) remained as the lone agent machine 1 waiting queue . Note the changes In NAST and also note x(1,6) 's eventual processing start time created a 5 min idle time on machine 1 .

Waiting Queue			Schedule Queue			
						NAST
Machine 1	y(1,2)		**Machine 1**	x(1,6,5		**111**
Machine 2			**Machine 2**	y(2,3,0	x(2,2,11	**13**
Machine 3			**Machine 3**	x(3,5,0	y(3,4,5	**9**

FIGURE 11:RTG INTERATION 4

- **ITERATION 5**

This Iteration saw the last agent being scheduled y(1,2) and there being no agent in any of the machine's waiting list the Round ended . The results are as illustrated below.

Waiting Queue			Schedule Queue			
						NAST
Machine 1			**Machine 1**	x(1,6,5	y(1,2,11	**13**
Machine 2			**Machine 2**	y(2,3,0	x(2,2,11	**13**
Machine 3			**Machine 3**	x(3,5,0	y(3,4,5	**9**

FIGURE 12:RTG INTERATION 4

At the end of the round we now have a complete candidate schedule. The makespan of the schedule is calculated as follows

$$makespan = Max(E(M_1), E(M_2), E(M_3))$$

$$= MAX(13, 13, 9)$$

$$= 13$$

The candidate solution can represent in a Gantt chart as follows.

FIGURE 13:RTG SAMPLE GANTT CHART

Once a candidate schedule has been generated the game the schedule is noted by the referee agent and other rounds of turns are performed for a predetermined number of rounds. The number of rounds will depend on the scale of the problems. It's expected that problem with larger problems will require more alteration to increase the probability of achieving a near optimal schedule.

4.3. POTENTIAL GAMES

[Opiyo et al, 2008] described potential games as those in which the incentive of all players to change their strategy is expressed in one global function called the potential function. The progressive actions of the participants lead to a stable state. In this section we defined a game that behaves in this way. In our interpretation we define a function that reward's/penalize agents based of the action it takes in the environment. As agents take actions the gain a bit of appreciation of their environment as their actions are reinforced by their reward/penalty system. To achieve this we borrow concepts from reinforcement learning, which transform our

game into a policy search function, That is, the aim of the game is meant to teach an agent what to base their actions (what policy to use) and at the end of a learning phrase is able to make decision on a certain state based on their experience with on that particular state. We start introducing the reinforcement learning concepts we borrowed and then proceed to modeling our job shop problem as multi-agent system environments that will enable us define this game.

4.3.1. AGENT BASED REINFORCEMENT LEARNING

(Sutton and Barto, 1998) describe reinforcement learning as follows.

Its concept that follows the idea that an autonomously acting agent obtains its behavior policy through repeated interaction with its environment on a trial-and-error basis. In each time step a reinforcement learning agent observes the environmental state and makes a decision for a specific action, which incur some immediate reward (also called reinforcement) generated by the agent's environment and, on the other hand, transfers the agent into some successor state. The agent's goal is not to maximize the immediate reward, but its long-term, expected reward. To do so, it must learn a decision policy that is used to determine the best action for a given state. Such a policy is a function that maps the current state the agent ands itself in to an action from a set of viable actions. (Thomas Gambel, 2008) describes the basic idea of learning through interaction within an agent's environment in following steps that must be performed by the agent.

- **Step 1.** The agent perceives an input state.
- **Step 2.** The agent determines an action using a decision-making function (policy).
- **Step 3.** The chosen action is performed.
- **Step 4.** The agent obtains a scalar reward from its environment (reinforcement).
- **Step 5.** Information about the reward that has been received for having taken the recent action in the current state is processed.

Reinforcement Learning methods explore the environment over time to come up with a desired policy.

(Yailen Martínez Jiménez, 2012), formally describe a generic reinforcement learning model as as an agent is connected to its environment via perception and action. In each interaction step, the agent perceives the current state ***s*** of its environment, and then selects an action **a** to change this state. This transition generates a reinforcement signal ***r***, which is received by the agent. The task of the agent is to learn a policy for choosing actions in each state to receive the maximal long-run cumulative reward. The diagram below provides a simple illustration.

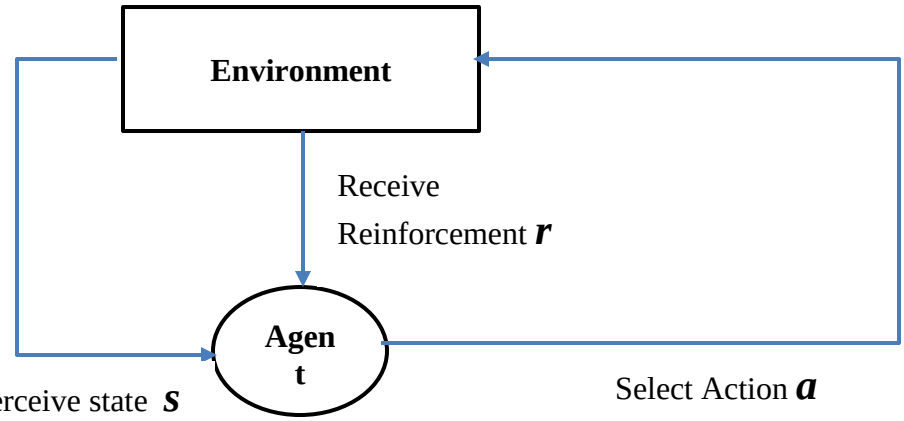


FIGURE 14:

(Yailen Martínez Jiménez, 2012) model can be formally described as follows.

- a set of environment states ***S***;
- a set of actions ***A***;
- a set of scalar rewards in ***R***;
- a transition function ***T***.

At each time *t*, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment the new state S_{t+1} and a reward r_{t+1} , this means that the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. In words, it is the probability of selecting action ***a*** in state ***s*** at time ***t***. The *reward function* defines the goal in a reinforcement learning problem. It maps each perceived state (or state-action pair) of the environment to a single

Numerical value, a *reward*, indicating the intrinsic desirability of that action in that state. The objective of a reinforcement learning agent is to maximize the total reward it receives in the long run, that is, an agent will prefer action that maximize reward in the long run rather than one than an that gives a good reward at the current state only.

4.3.2. MARKOV DECISION PROCESS

Markov Decision Process provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning. Markov Decision Process is a good framework to use to model a decision process in an optimization problem where the search space is finite. Job shop problem has a finite search; the only limiting factor is that the search space the search space is becomes very big in large problem instance. MDP can still be used to model a Job shop Scheduling problem by limiting the decision process to either a subset of the search space. The subset search space can be determined at random.

A Markov Decision Process (MDP) is a 4-tuple ***[S, A, T, R]*** where:

- $S = s^1, \ldots, s^n$ denotes a finite set of states;
- Set of actions ***A***, and $A(s) \in A$, where A(s) is the finite set of available actions in state $s \in A$;
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function, **T(s, a, s')** specifies the probability of ending up in state s0 when performing action α in state ***s***;
- $R : S \times A \times S \rightarrow R$ is the reward function, **R(s, a, s')** denotes the expected reward for the transition from state ***s*** to state ***s'*** after taking action α.

For MDPs, the Markov property assures that the transition from **S** to ***s'*** and the corresponding reward **R(s, a, s')** depend only on the state ***s*** and the action α, and not on the history of previous states and actions.

The formulation of MDP above assumes that an agent has full awareness of the environment which in the real world its rarely so. In most really world environment the agents usually have a partial observation of the Environment. The same is true for the Job shop scheduling problem where we expect an agent to only have local knowledge of its environment and not the global knowledge. We borrow a concept of reinforcement learning known a Q-learning that can help with partially observable environment. We describe the concept below.

4.3.3. Q-LEARNING

We borrow our description of Q-learning as describe by (Yailen Martínez Jiménez, 2012) and (Thomas Gambel, 2008). They describe Q-learning as a well-known reinforcement learning algorithm is Q-Learning (QL), as a reinforcement learning technique based on learning an action-value function that gives the expected utility of taking a given action in a given state. They describe the core of the algorithm as

"Simple value iteration update, each pair ***(s, a)*** has a Q-value associated. When the action **a** is selected by the agent located in state **s**, the Q-value for that state-action pair is updated based on the immediate reward received when selecting that action, and the best Q-value for the subsequent state **s'** ". The update rule for the state action pair (s, a) is the following:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{\mathbf{a}'} Q(s',a') - Q(s,a))$$

EQUATION 15: UTILITY FUNCTION

Where;

$Q(s,a)$ - The utility of state **s** defined recursively the update rule above

α - is a learning rate.

γ - Discount rate of subsequent action.

r- Reward of taking action ***a*** on state ***s***

The Equation we can see that the utility of pair $Q(s,a)$ is not only based on the current reward or penalty achieved by taking action **a** in state **s** but we also consider the subsequent $Q(s',a')$. This will help use the agents converge at a optimal policy set at each time slice. To achieve this

we look at all the subsequent alternative $Q(s',a')$ and pick the route with the maximum utility, which is

$$\max_{a'} Q(s',a') - Q(s,a)$$

A discount γ is usually applied to the subsequent actions utility so we have

$$\gamma \max_{a'} Q(s',a') - Q(s,a)$$

We also apply a learning rate α which depends on the size of our search space.

$$\alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Finally we add this to the utility of the single action ***a*** in state ***s*** to achieve the update rule below,

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

We give a simple illustration of Q-learning below using a simple chart.

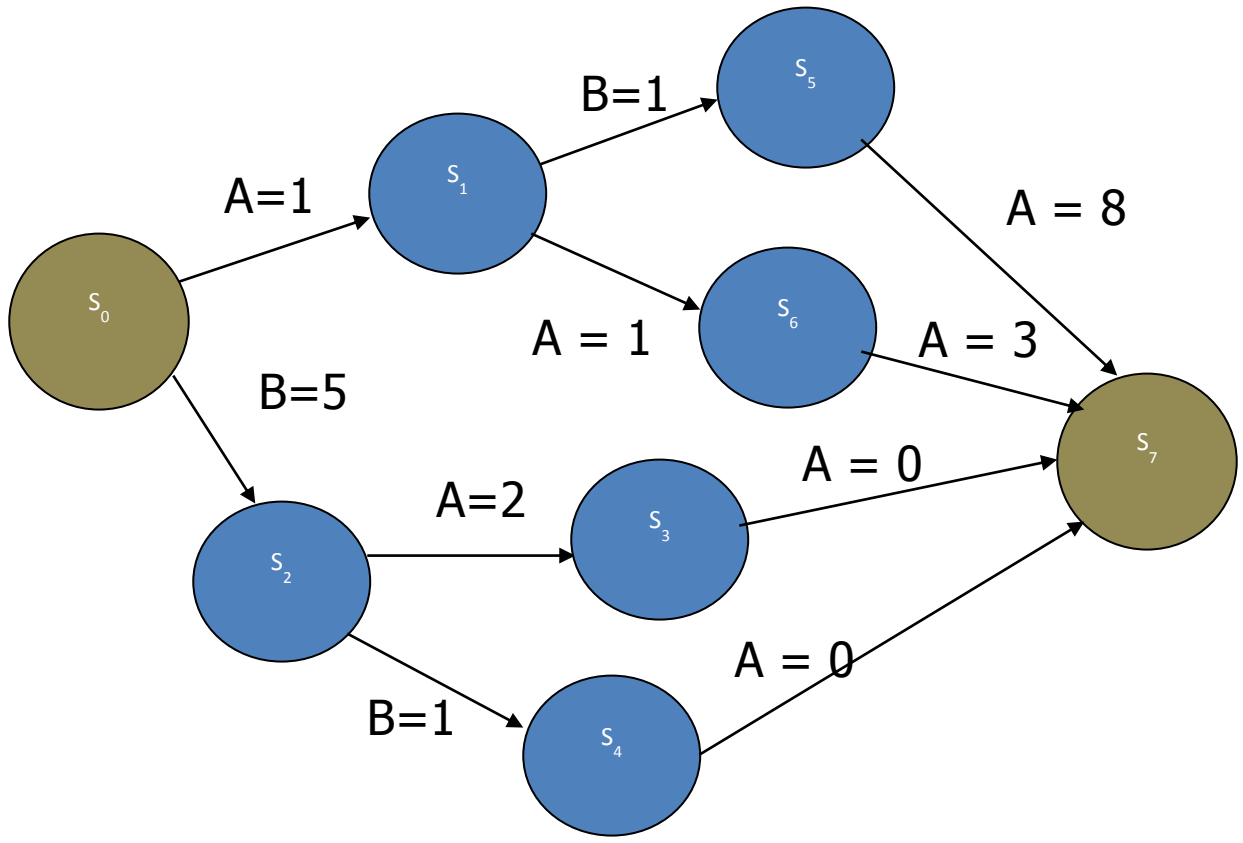


FIGURE 15:POTENTIAL GAME SAMPLE DECISION TREE

The figure below shows a subset of an environment that has been learnt by an agent. The interest is to get from state $\mathbf{S_0}$ to state $\mathbf{S_7}$. The agent previously had moved from state to state using either policy **A** or **B** at each state and learnt a reward the various possible policies taken at each state. The reward is determined by a value system and in our case a potential function. With this knowledge an agent can recursively calculate the utility of each state as it continues with its exploration and during the decision making stage the agent will use the utility values to know which action will lead to maximize the total reward. Suppose the learning rate and discount is as follows;

α = 1

γ =1.

If an agent is at $\mathbf{S_0}$ and needs to know the optimal route/most rewarding route to take to r, it will not only use the learnings from the reward system alone, that is, it will take action **B** to get an immediate result 5 which seem to be better than taking action **A** with reward of **1.** This is because an agent knows that an immediate high reward currently might not end up being the most rewarding route in the long run. To do this agent will use the utility of each successive state-action(Q-pair) to take the route that lead to the (Q-pair) with the highest utility enroute to the terminal state. The agent would have recursively determined the utility of each (Q-pair) using the update function,

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

The Psuedocode of the exploration/learning process would be similar to the following

- Initialize **Q-values** arbitrarily
- for each episode do
    - Initialize **s**
    - for each episode step do
        - Choose **a** from **s**
        - Take action **a**, observe state **s'** and **r**
        - Update

          $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a, Q(s',a') - Q(s,a)]$$
    - end for
- end for

The following table illustrates the utility of the q-pair achieved by the agent in our example. Note that for illustration purposes we calculate utility from the terminal state backward, that's why our $\boldsymbol{Q(s,a)}$ on the table always seems to zero, in the actual process the agent would update it recursively and its value would always be changing for each q-pair until the final state is reached. The table below notes the utility of each q-pair.

No.	q-pair	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$	utility
q1	**$Q(S_5, A)$**	=0+ 1(8+ 1(max(0,0)-0)	8
q2	**$Q(S_6, A)$**	=0+ 1(3+ 1(max(0,0)-0)	3
q3	**$Q(S_1, B)$**	=0+ 1(1+ 1(max(q1,0)-0) =0+ 1(1+ 1(max(8,0)-0)	9
q4	**$Q(S_1, A)$**	=0+ 1(1+ 1(max(q2,0)-0) =0+ 1(1+ 1(max(3,0)-0)	4
q5	**$Q(S_0, A)$**	=0+ 1(1+ 1(max(q3,q4,0)-0) =0+ 1(1+ 1(max(9,4,0)-0)	10
q6	**$Q(S_3, A)$**	=0+ 1(0+ 1(max(0,0)-0)	0
q7	**$Q(S_4, A)$**	=0+ 1(0+ 1(max(0,0)-0)	0
q8	**$Q(S_2, A)$**	=0+ 1(2+ 1(max(q6,0)-0) =0+ 1(2+ 1(max(0,0)-0)	2
q9	**$Q(S_2, B)$**	0+ 1(1+ 1(max(q7,0)-0) =0+ 1(1+ 1(max(0,0)-0)	1
q10	**$Q(S_0, B)$**	=0+ 1(5+ 1(max(q8,q9,0)-0) =0+ 1(5+ 1(max(2,1,0)-0)	7

TABLE 3:Q-PAIR CALCULATION EXAMPLE

Suppose we have now stopped our exploration stage and an agent wants to make a decision that on how to get from state S_0 to the terminal state S_7 , The agents steps will proceed as follows;

- Starting at state **S_0** , the agent will need to look at the utility of Q-pair **$Q(S_0, A)$** and **$Q(S_0, B)$** to decide if to pick action **A** or **B** . Since **$Q(S_0, A)$** has a better utility the agent will choose action **B** to move to the successive state **S_1** .
- At **S_1** the agent is face with two options **$Q(S_1, A)=4$** or **$Q(S_1, B)=9$** . The agent therefore selects **$Q(S_1, B)$** and move to successive state **S_5** .
- At state **S_5** the agent has only one option **$Q(S_5, A)=8$** . The agent selects this to move to the terminal state **S_7** .

The route the agent follows will look as depicted by the illustration below.

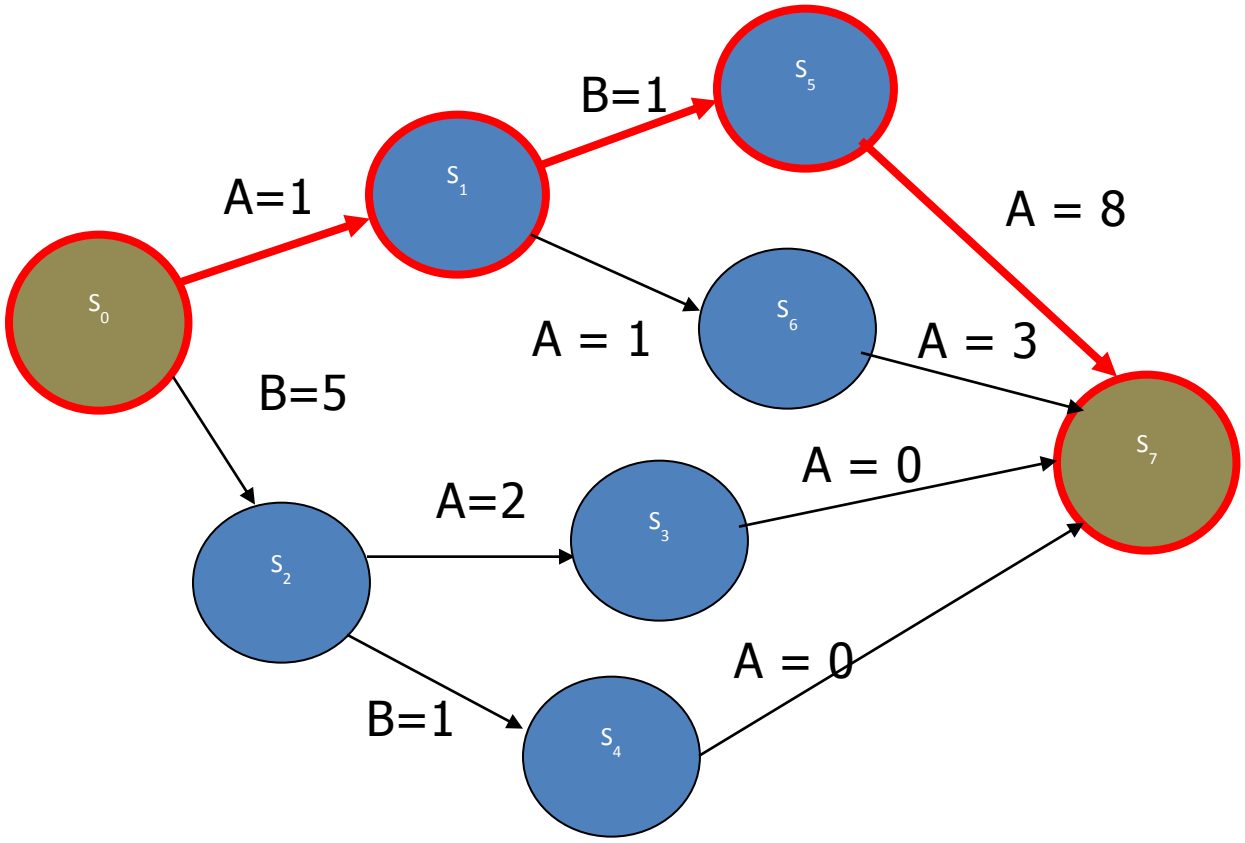


FIGURE 16: SAMPLE POTENTIAL GAME DECISION TREE

Note that because our learning rate $\alpha = 1$ and $\gamma = 1$, We can still achieve an optimal route by adding up the rewards along each route and pick the route with the highest some of rewards as the optimal route.

By using Q-learning in a an environment where the agent has partial observation of the environment, an agent can learn a finite set of the search space and using the potential function, derive utility for each its decisions and finally select a series of decision policy that are beneficial to it. If we utilize Q-learning we are able to achieve (opiyo, et al, 2008)'s description of a potential game where there is a global function that guides agent in decision making. The reward function and utility function act together to guide the agent in decision making. In the section that follows we model our Job shop problem as a Multi-agent system and use Q-learning to define our potential games.

4.3.4. JOB SHOP SCHEDULING AS POTENTIAL GAME WITH Q-LEARNING

We start by modeling the multi-agent game environment for this game we use an agent to represent a machine. In potential games as earlier discussed, there is one global function that governs the behave of all the agents this leads to the game being co-operative game where agent learn as a group and are motivated by the global function to employ similar policies. There we adopt a variation of Actor-Critic agent model where we have;

- Multiple actor only agents that represent a machine and their main function is to choose the next job to be processed by employing a specific policy.
- One Critic only agent whose main function is to evaluate successive policies taken by the agents and give feedback on their suitable using the global function.

Our Agents have one which is to reduce the makespan of processing all the Jobs in the environment. Our environment will therefore consist of the following; a global dispatch queue, this is a queue that holds all jobs before they can be moved to a machines/agents waiting queue. Jobs move to from the dispatch queue to the machine queue when there is no constraint to their processing e.g. they have no predecessor or their predecessor has already been scheduled for processing. This is demonstrated in the illustration of a simple 3X3 problem.

We include an actor only agents, these agents will be responsible for selection of policy during the exploration/learning stage of the game. The agent at each time step in schedule formation will employ a certain policy as they seek to achieve a complete schedule.

We include a critic only agent will responsible for evaluating the policies employed by each agent at every time step and will give feedback to the agents in terms of a reward/penalty. In our algorithm the critic's memory structure will also be responsible for storing the learned utility of each state. Once the exploration is done information learned by actor agents is used by them to select successive policies, as they form what they consider to be an optimal schedule.

We also include action policies. These represent dispatch rules, the rules act as policies that is available for the actor to choose from when selecting the next action, that is, whenever an actor makes a specific move, their move has to be based on a specific policy. At learning stage an actor would tryout one or more policies and will observe the reward/penalty using that policy on that particular state. The aim of our global function is to define a series of policies that an agent can employ that would lead to an optimal schedule We shall define four policies that can be used by agents in any state. This will be discussed further in our 3X3 problem illustration.

- **FIFO**- First In First Out
- **LIFO**-Last In First Out
- **SPT**-Shortest Processing time.
- **LPT**-Longest Processing time.

A global potential function, this function is used by the critic agent to appraise and influence the action of the actors. As we had earlier demonstrated in the Q-learning algorithm, the function assigns a reward on agent actions and defines the utility of each Q-pair. The utility will finally influence the agents' decision on which policy to employ at the selection stage.

A reward structure, the reward structure is used by the critic agent to appraise the agent actions. A reward is quantification of how good the selected policy in the current state is. The several ways that we can determine a reward for a decision. We choose two that we believe would lead convergence of a near optimal solution from the search space. These are;

- Number of jobs that remain in global dispatch queue after all actor agents have selected a single action.
- Total process time in the global dispatch queue. This is a sum of the processing time ***p(o)*** of the ***n*** jobs remaining in the global dispatch queue after all actor agents have selected a single action. That is,

$$\boldsymbol{r} = \sum_{\mathbf{0}}^{\boldsymbol{n}} \boldsymbol{p}(\boldsymbol{o})_{\boldsymbol{n}}$$

EQUATION 16: TOTAL DISPACTH QUEUE PROCESSING TIME

- Total process of all waiting Jobs. This is a sum of the processing time ***p(o)*** of the ***n*** jobs that are waiting global dispatch queue and on waiting queues of machine after all actor agents have selected a single action. That is,

$$\boldsymbol{r} = \sum_{\mathbf{0}}^{\boldsymbol{n}} \boldsymbol{p}(\boldsymbol{o})_{\boldsymbol{n}}$$

EQUATION 17: TOTAL PROCESSING TIME

Using this structure means the appraisal ***r*** to be a penalty/cost rather than a reward. That is, a decision that leads to more jobs or larger total processing time on the dispatch queue is less favorable. Because this is a cost/penalty, we negate the appraisal.

$$\boldsymbol{r} = -\mathbf{1}\left(\sum_{\mathbf{0}}^{\boldsymbol{n}} \boldsymbol{p}(\boldsymbol{o})_{\boldsymbol{n}}\right)$$

EQUATION 18: REWARD/PENALTY FOMULAE

The table below shows the structure of our machine agents.

AGENT(Machine)		
Attirbutes		
	MachineID	*The Id of the Agent*
	Status	*The status of the operation e.g* •*Idle:- The Agent is idle and waiting for a turn,* •*Active:- The Agent is allowed to make a move*
	Waiting Queue	*Queue holding job to be processed on the machine but yet to be scheduled.*
	Process Queue	*Multi-dimensional structure that notes the scheduled*

		jobs and there start and finish time.
Actions		
	Move	*Action allowing agent to make a choice*
	Message	*Action allowing agent to send message to other agents*
	Read	*Action allowing agent to sense its environment*

TABLE 4: POTENTIAL GAME AGENT PROPERTIES

With the above definition we have factored in the main characteristics that have to be considered when modeling the problem are adapted from [Gabel (2009)] and can be summarized as follows:

- **Factored World State**: The world state of a job-shop scheduling problem **J** can be factored: We assume that each resource has one agent **i** associated that observes the local state at its resource and controls its behavior. Consequently, there are as many agents as resources in the JSSP.
- **Local Full Observability**: The local state $\mathbf{s_i}$ of agent **i**, hence the situation of resource $\mathbf{r_i}$, is fully observable. That is, an agent has full view of what is in its queues. Additionally, the composition of all resources fully determines the global state of the scheduling problem. Therefore, the system is jointly observable.

- **Factored Actions**: Actions correspond to the starting of jobs' operations (job dispatching). So, a local action of agent **i** reflects the decision to further process one particular job (more precisely, the next operation of that job) out of the set $\mathbf{A_i}$ of operations currently waiting at $\mathbf{r_i}$.
- **Changing Action Sets**: If actions denote the dispatching of waiting operations on the machines waiting queue for further processing, then the set of actions available to an agent varies over time, since the set of operations waiting at a machine changes.

  Furthermore, the local state $\mathbf{S_i}$ of agent **i** is fully described by the changing set of operations currently waiting at resource $\mathbf{r_i}$ for further processing, thus, $\mathbf{s_i} = \mathbf{A_i}$. We shall

demonstrate changing action sets of agents in our sample 3X3 problem in the next section

- **Dependency Functions**: Since operations have precedence function with other operation on other machines, a dependency function among agent action as imply that, after one agent executes an action (processes one operation), the local state of maximally one further agent is influenced

4.3.5. 3X3 POTENTIAL GAME EXAMPLE

In this section we give a brief illustration of learning demo the learning stage of potential. We diagrams to depict the enviroment and we also use decision trees represent the memory of what has been learnt. Suppose we have a 3X3 matrix,

Job X: x(3,5) , x(1,6), x(2,2)

Job Y: y(2,3), y(3,4), y(1,2)

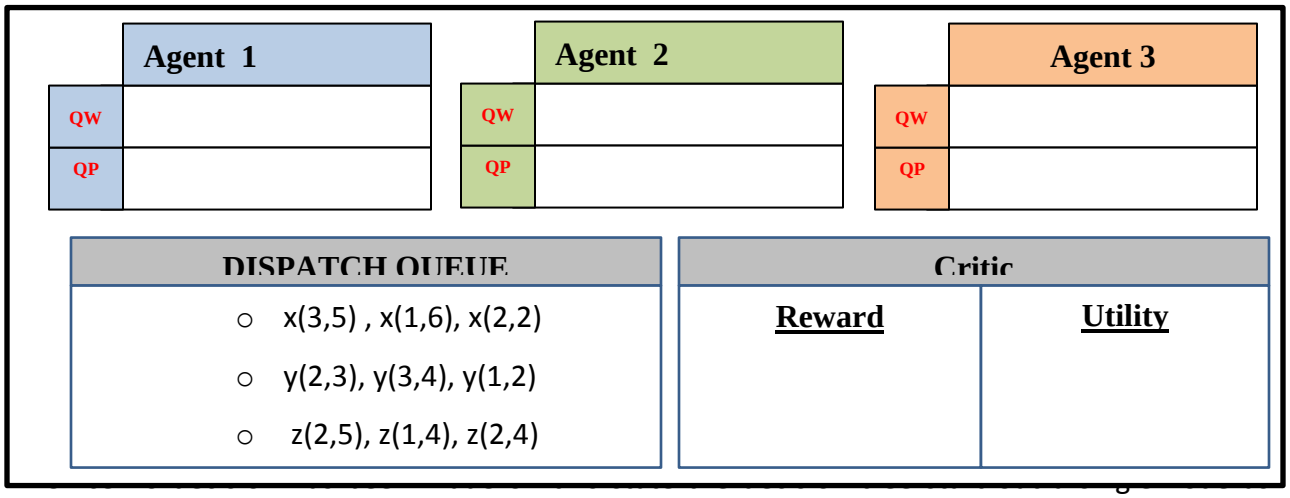
Job Z: z(2,5), z(1,4), y(2,4)

In our enviroment we have the following as had earlier been discussed we have the following components, an actor agents representing machines, a List of Jobs and their operations which have precedence constraints with operations on the same job, a critic agent, a global dispatch and a set of action policies. Agents can base their action on the following dispatch rules that act as action policies, e.g. SPT, LIFO,LPT and FIFO. For demo purposes we will use avail only 2 policies to the agent, that is, **SPT** and **LPT**

At the begin of the game all the jobs are in the global dispatch queue and no agent has an action set , that is, there are no operation in any of the agent's waiting queue. Suppose according to how earlier description of our actor agent structure , we have a waiting and processing queue for each agent and for our ***qw*** and ***qp*** represents agents waiting and process queue respectively .

1.1.1.1. STATE 0

At the start of the game can depicted by as follows,



shown below,

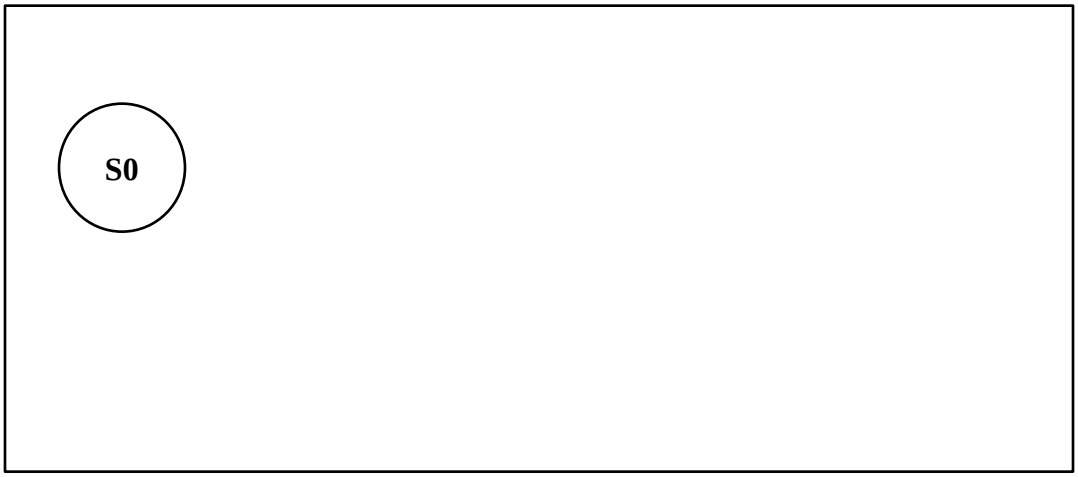


FIGURE 18: SAMPLE POTENTIAL GAME DECICISION TREE GENERATION

After this all the operations with without predecessor or whose predecessors have been scheduled (moved to QP of a machine) will be moved to the respective machine waiting queue. So we move to the next state as shown below. In this State Agent 1 has action set of 0 there for cannot make a move, while Agent 2 and Agent 3 have actions sets of 2 and 1 operations respectively. The State is as shown.

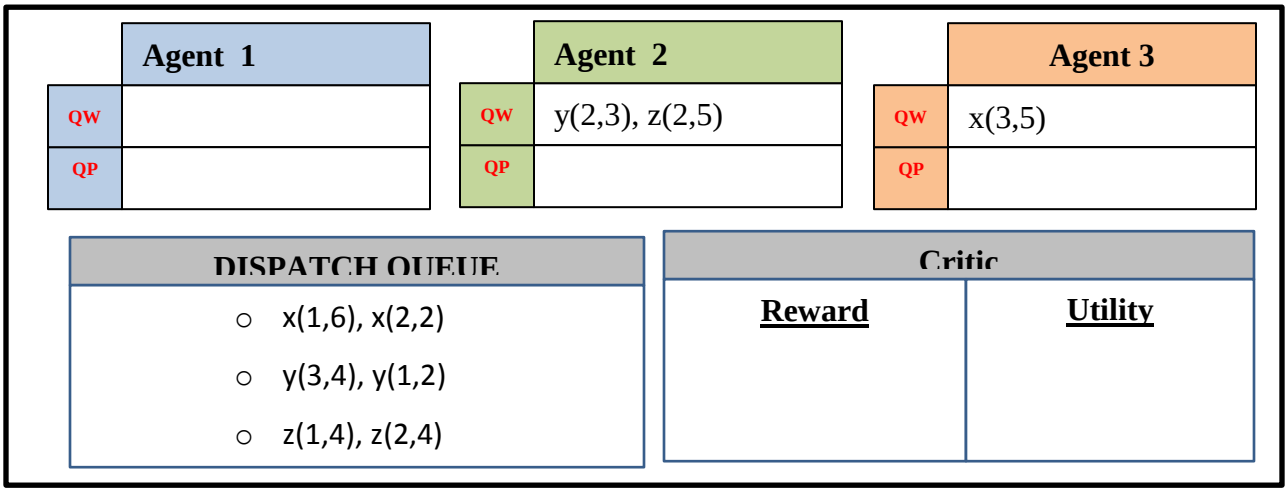


FIGURE 19:POTENTIAL GAME SAMPLE STATEO

At this stage the agents can make a move and might choose any the available policies, SPT or LPT for their next move. Upon which its reward by the critic agent. Our Critic agent uses the negation total processing time remaining in the dispatch queue has a reward. We had earlier formalized the reward as shown below.

$$\boldsymbol{r = -1\left(\sum_{0}^{n} p(o)_n\right)}$$

Our critic will also update the utility of each Q-pair form as the agents learn, this will be done recursively using the update function,

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Earlier we had proved that if we have that if we have the learning rate as $\alpha = 1$ and discount as $\gamma = 1$. Then the utility of a Q-Pair can be achieved by summing the along rewards at each possible route from the Q-pair to the terminal state and selecting the summation of the route that offer the highest sum of rewards as the utility of the Q-Pair. Note that utility of a Q-pair may change multiples times as long as new routes are discovered or developed

1.1.1.2. STATE 1 (Q(S0,SPT))

Suppose our agents decided to employ **SPT** as a policy in state 0 we will arrive to arrive at state 1. This move will for each agent result in scheduling of the job with the shortest processing time from the machines waiting **qw** , into **qp.** It will also result in operations that are now ready to be processed (because the predecessors have been scheduled to process in **qp**) be moved their respective machines waiting queue therefore changing the action set of each machine. By employing policy **SPT** at state **S0** We form a Q-Pair **Q(S0,SPT)** .The enviroment will change as shown below and the critic will appraise the Q-pair with the reward shown. The critic also calculates the utility for this Q-pair as shown. Our decision tree will further grow as follows.

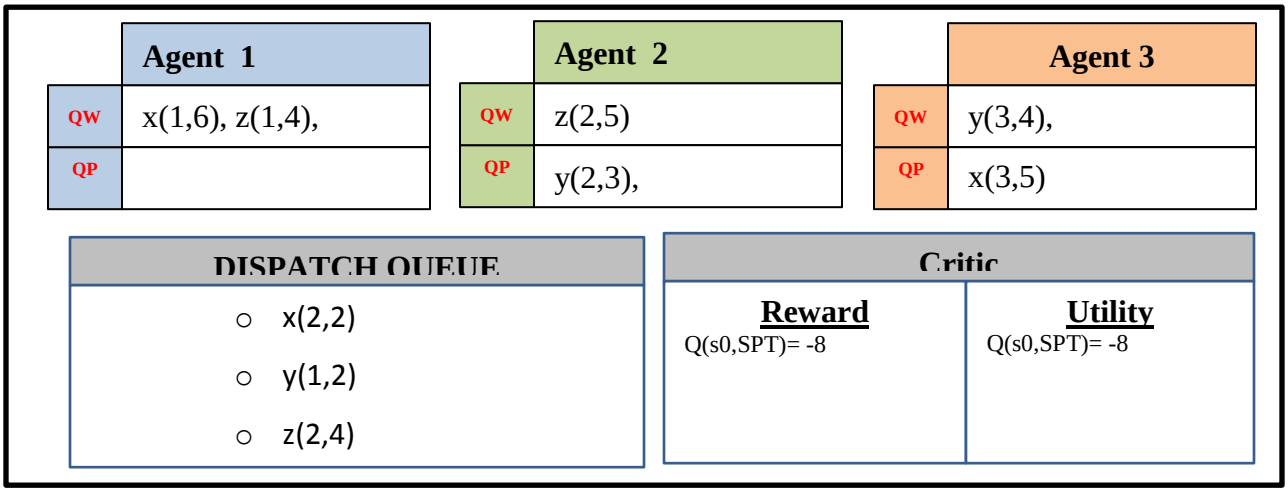


FIGURE 20: POTENTIAL GAME SAMPLE STATE1

Our decision tree then transforms as shown.

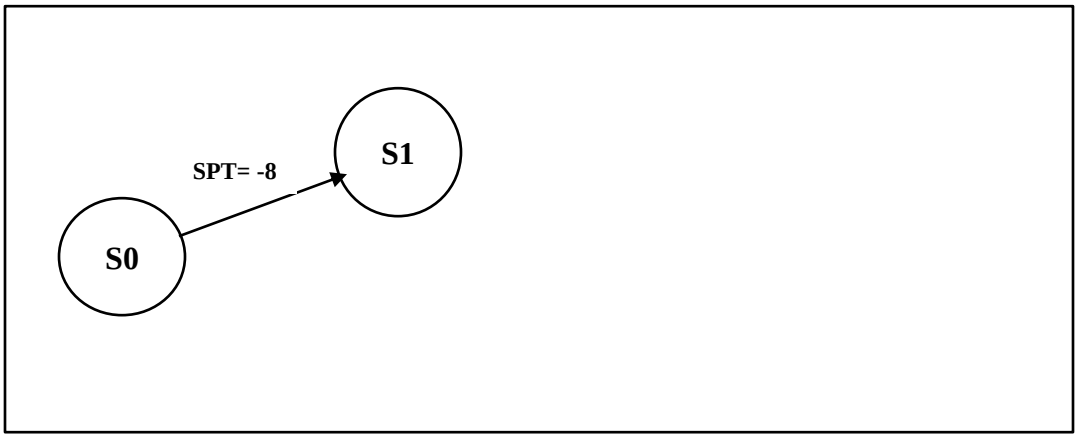
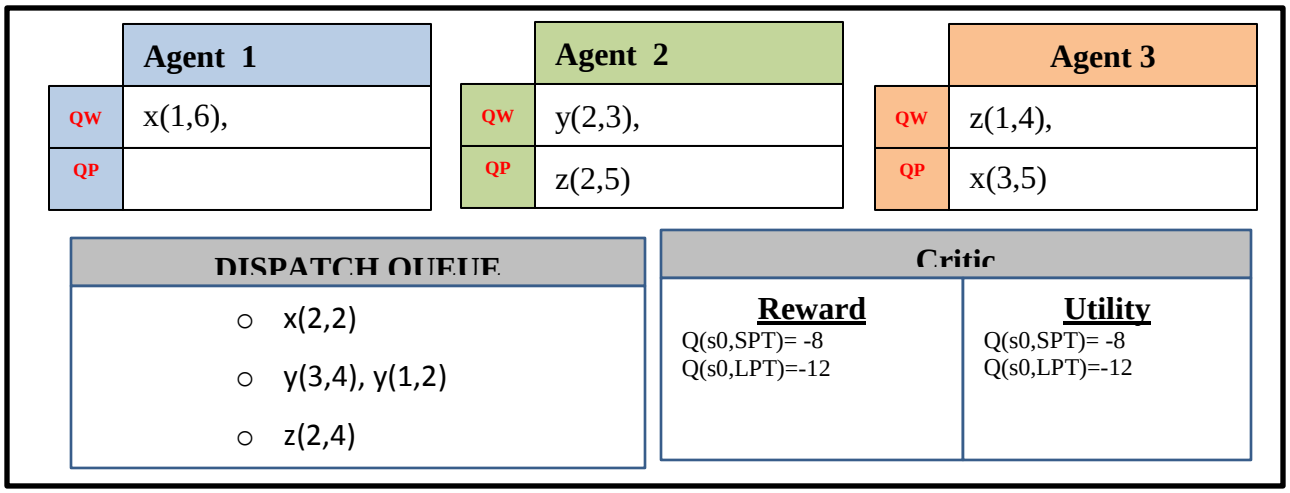


FIGURE 21: SAMPLE DECISION TREE AT STATE 1

1.1.1.3. STATE 2(Q(S0,LPT))

Since our agents are still in the learning phrase, suppose they back track to state **S0** and try out LPT instead of **SPT.** This would create a new Q-pair **q(S0, LPT).** The results will be as follows, note new reward;



And we will transform our decision tree as

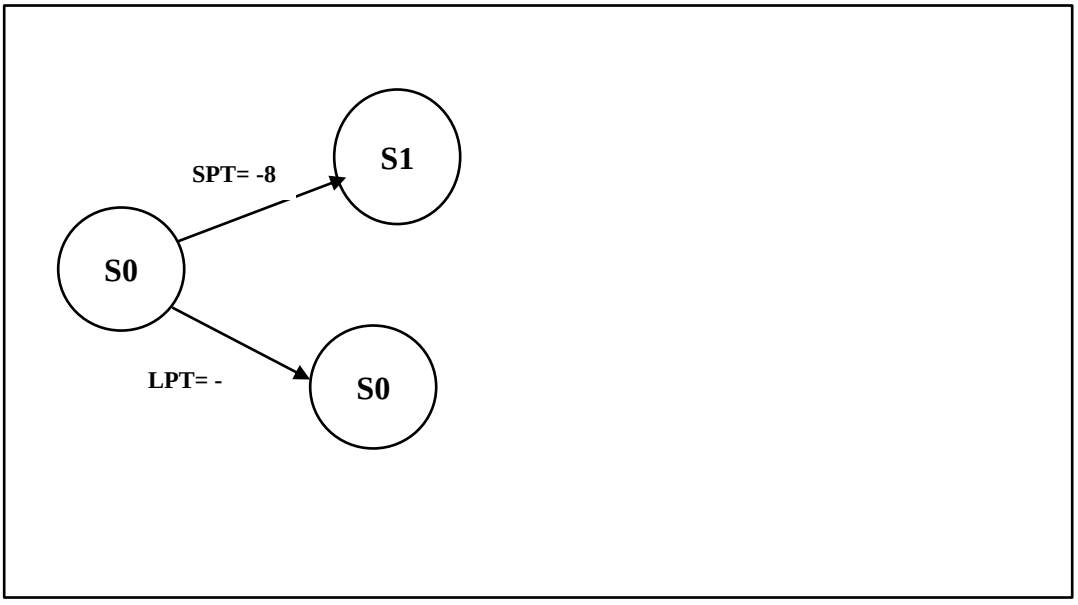


FIGURE 23: SAMPLE POTENTIAL GAME DECISION TREE

1.1.1.4. REST OF THE STATES

Because of the combinatory complexity of the problem, we will assume the agent continued learning by trying different Policies at different states and produced the decision tree below with hypothetical reward values as shown. We assume the agent

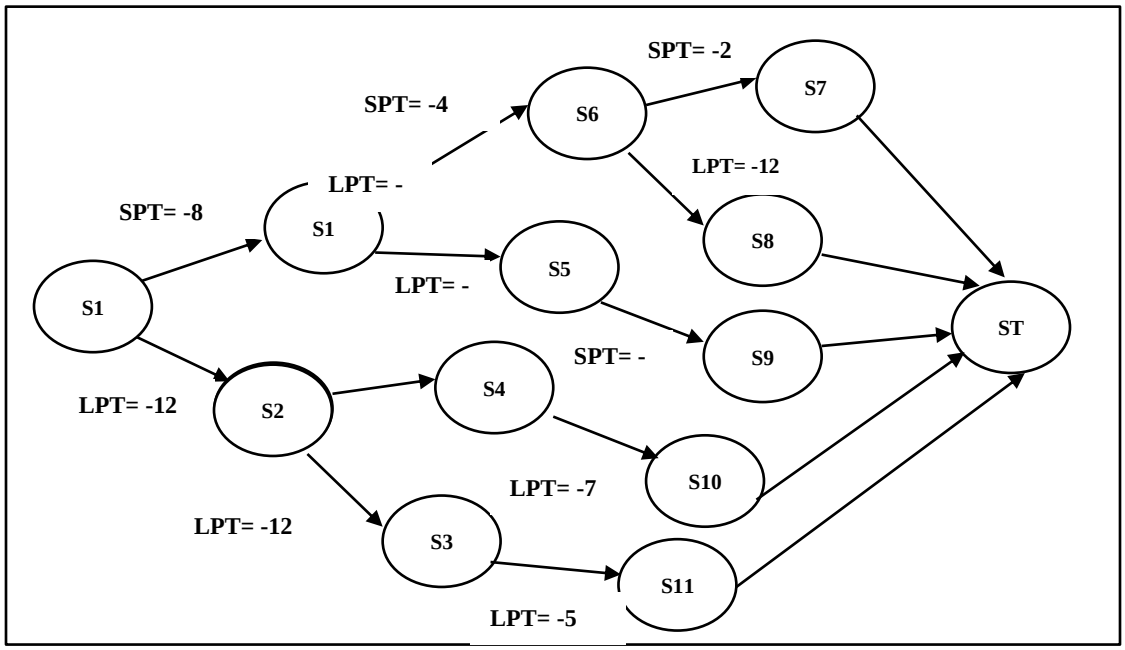


FIGURE 24: COMPLETE SAMPLE POTENTIAL GAME DECISION

As the agent continued to learn it also updated the utility of the each Q-Pair using the update function,

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

To demonstrate this we calculate the utility, we assume the learning rate and discount are both equal to 1, this means the utility of this of a Q-pair would be the route to terminal state with the highest sum of reward as we had earlier proven. So the utility of the Q-Pair **Q(s1,SPT)** would be -6 as opposed to the alternative route through **S8** which has a much lower value of -16. As can be seen as more child and grandchild states emerge from a state, the more complex calculating it's utility becomes. From the decision tree we can also see each route from the initial state to the final state represent complete schedule. After our agents have finished learning they will move to the decision stage where they will start from the initial state on the decision tree and use the utility learnt for each the q-pairs to select the most rewarding route to the terminal state. The combination of policies guided by our utility function along this route would lead to agent performing actions that would lead to a near optimal schedule. Thus achieving a successful potential game.

4.4.RANDOM GAMES

The last game we define is the random games. This game borrows the same concept as potential games where there are actor only agents that represent a machine. The MAS environment is structured as follows;

Our environment will therefore consist of the following; A global dispatch queue, this is a queue that holds all jobs before they can be moved to a machines/agents waiting queue. Jobs move to from the dispatch queue to the machine queue when there is no constraint to their processing e.g. they have no predecessor or their predecessor has already been scheduled for processing and an actor only agents, these agents are responsible for selecting the next operation to process, but unlike the potential game this agents simply select a operation at random.

The diagram illustrates the MAS environment for Potential games for a 3X3 instance

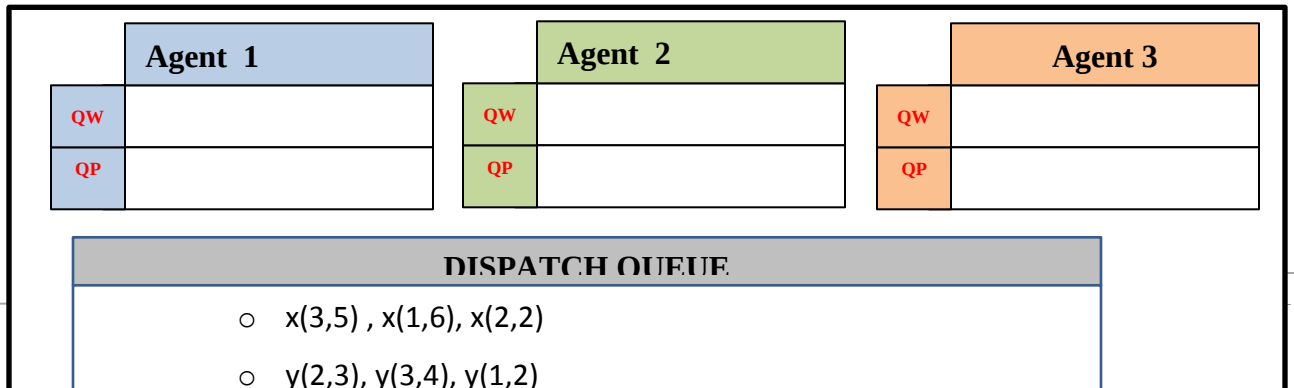


FIGURE 25: SAMPLE RANDOM GAME

This description of a random game is the same as described in (opiyo et al, 2008) where game has multiple preset ***n*** number of iterations and in each iteration an agents select operations at random from there waiting queues until all operations have been schedule. This forms a candidate solution S_0 and its makespan is noted. At the end of the game a candidate solution Si is selected as the feasible solution with the near optimal schedule. The formula below shows the mode of selection of this schedule.

$$\min_{n} S(S_0, S_1, S_2, \ldots \ldots \ldots S_n)$$

Just as shown by (Opiyo et al, 2008). We believe selection of candidate solution at random from the search space there is a high probability of selecting a feasible schedule; this method would thus achieve a feasible solution way much quicker and demand the list of resources.

CHAPTER FIVE: SYSTEM DESIGN AND IMPLIMENTATION

In this section we discuss the design and implementation of the visualization tool. We go through the architecture of the 3 defined games and the implementation decisions made.

5.1.IMPLIMENTATION TOOLS

The following are the implementation tools to be used.

IDE	The Visualization tool was built using Visual studio. This offers comprehensive debug tool for faster development
UI	We choose to use Windows Foundation Pages through its XAML notation for the user interface development because of its ability to represent graphical object and low memory demand and also offers ability to work with primitive data types in definition of graphical objects.
Language	The programing language used is C#. This was chosen because of ability to easily represent complex data structures with a lower memory demand.

TABLE 5 : DEVELOPMENT TOOLS

5.2.SYSTEM DESIGN

In this section we discuss the design for each game in the implementation tool. In the tool we have the following two classifications.

- **Environment**. This is a class object that defines the environment of a game. For each game we create and instance an environment. The environment represents the state of the entire object within it, that is, state of the agents, resources and utilities. In games where learning states are involved an environment at each step represents a state. Therefore in those games we keep track multiple environment instances. So all the

previous environment instance for previous stages are stored in the utility of the current state.

- **Objects.** Both Agents and resources are defined as instance of the various object classes. Agents are defined object with the ability to learn (store experience).

In the sections that follow we go through the design of each of the implemented

5.2.1. POTENTIAL GAMES

The Potential game includes the following objects.

OBJECT : AGENT **CLASS: PGagent** **DESCRIPTION: Agents represent a machine and have the following attributes and Methods.**	
Attributes	
AgentID	Attribute defining the Agent ID, this is similar to the machine number.
NAST	Represent the next available start time of a machine at that particular state
StateID	Represent the current agent state.
List<PGop> **WQ**	A represents the machines waiting queue. It contains a list of the job operations waiting to be scheduled for processing. Operations are members of the class PGop.
List<PGop> **SQ**	A represents the machines Schedule queue. It contains a list of the job operations that have already been scheduled. Operations are members of the class PGop.
Methods	
CryptoRandom **RandomSelection**	This is represent a random function based a CryptoRandom class that we developed that a allows an agent to make a true random action
LPT()	This represent a strategy, *Longest processing time*, that

	can be used by an agent to select a job to schedule next from the available operations in its queue
SPT()	This represent a strategy, *shortest processing time*, that can be used by an agent to select a job to schedule next from the available operations in its queue
LIFO()	This represents a strategy, *last in last out*, that can be used by an agent to select a job to schedule next from the available operations in its queue.
FIFO()	This represents a strategy, *first in first out*, that can be used by an agent to select a job to schedule next from the available operations in its queue.
RANDOM()	This represents a strategy, *random*, that can be used by an agent to select a job to schedule next from the available operations in its queue.
Move()	This function allows an agent to make a move and select which of the above strategies the move will be based on.
getOpStartTime()	Once an agent selects a job operation to be scheduled, as part of the scheduling tasks, this function allows assigning of a time slot on agent to a job's operation.
getNewNAST()	After scheduling an operation, this function is used by the agent to recalculate its new *'Next Available Start Time'*

TABLE 6: PGAGENT PROPERTIES

OBJECT : Operations CLASS: PGop DESCRIPTION: This object represent a single operation	
Attributes	
opID	Identify the operations ID.
JobID	Identify which Job the operation belongs to.
SeqID	Identify the sequence position the operation occupies
predecessor	Identify operation's predecessor(operation that precedes this operation on the same job)
Successor	Identify operation's successor(operation that follows this operation on the same job)
status;	The current status of operation e.g Waiting, Scheduled, Active, e.t.c
processingStartTime	Represents the starting time for processing of the operation on its specific machine.
load	Represents the processing time required to process the operation on its specific machine.
state;	Represent the current environment state of the operation
machineID	Identifies which machine the operation needs to be processed in.

TABLE 7: PGOP PROPERTIES

OBJECT : Enviroment CLASS: PGEnv DESCRIPTION: Represent the potential game enviroment.	
Attributes	
List<PGagent> **PGAgentList**	This represent all the agents available in the environment
List<PGop> **PGOpList**	This represent all the perations available in the environment

List<int> **DispatchQueue**	This represent the dispatch queue in the environment that holds operation that are yet to be selected
List<List<envState>> **allStreams**	This holds instances of all previous states
Methods	
CreateDispatchQueue()	Function used to create the Dispatch Queue.
DispatchExit()	Used to remove operations form the dispatch queue an deliver to the relevant Agent's waiting Queue
agentsAction()	Functions used to prompt an agent to make a move
checkTerminal()	Check if the State arrived at is terminal
penalty()	Provides reinforcement (penalty) an Agent's action.
QValue()	Used to determine the qValue(utility value)of the selected path of actions
MoveToMachineQueue()	Used by the function *'CreateDispatchQueue'* to remove operation from dispatch queue to the relevant machines waiting queue.

TABLE 8: PGENVIROMENT PROPERTIES

OBJECT : State **CLASS: envState** **DESCRIPTION: Represents a state of environment.**	
Attributes	
State	Identifies the state.
strategy	Outlines the strategy selected/employed by the agents I selection on that particular state.
qValue	Stores the utility value of a path of state-action pairs
penalty	Reinforcement given to an agent for making a specific

	action at a specific state.
List<PGagent> TAgentList;	Store all agent states (as list of PGagent objects) for this particular state
Methods	
envState	A constructor used to initialized the state

TABLE 9: ENVSTATE PROPERTIES

The potential game algorithm is implemented according to the following flowchart.

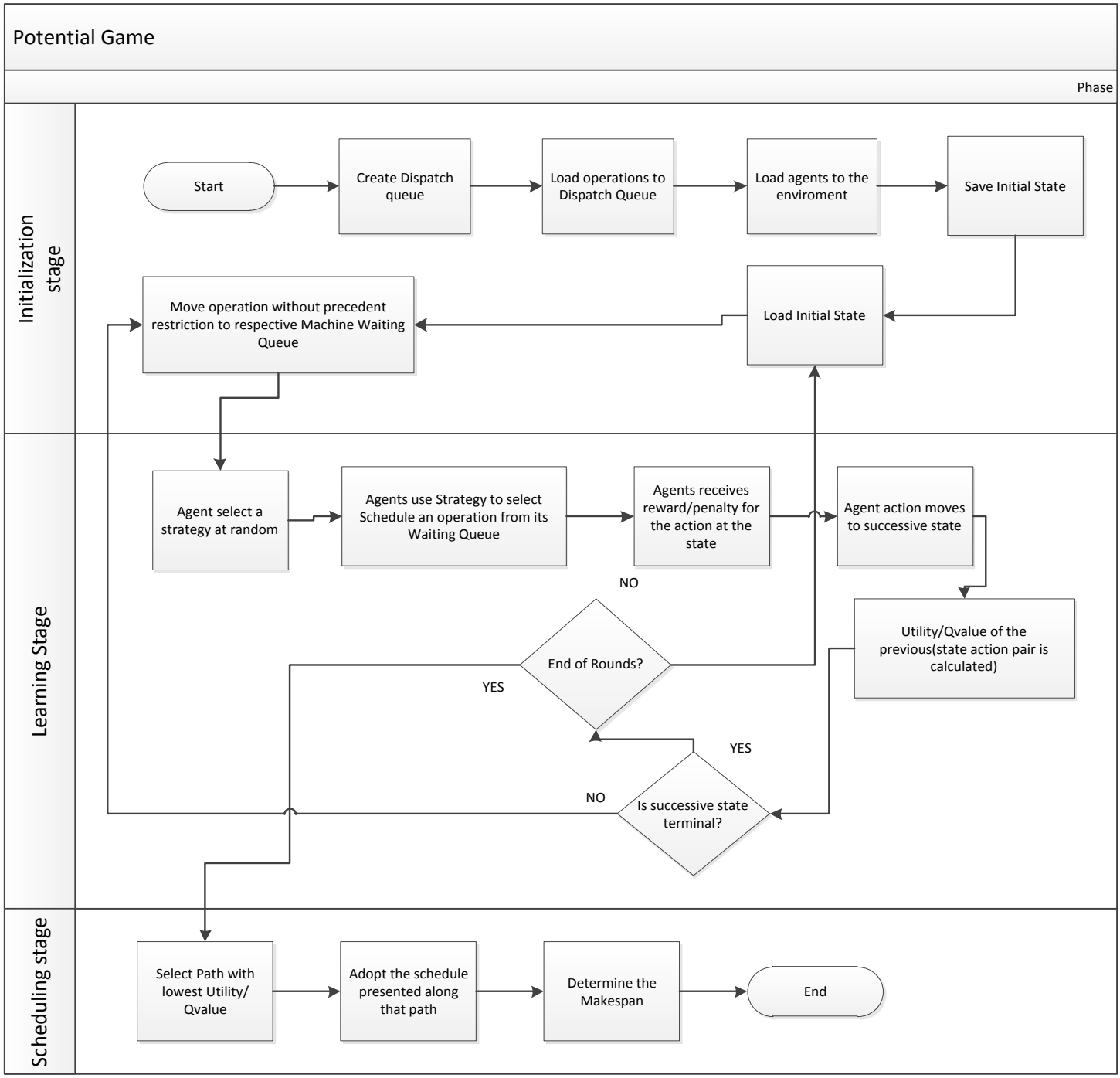


FIGURE 26: POTENTIAL GAME FLOWCHART

5.2.2. RANDOM GAMES

The random game include the following objects (Most items are similar to potential games above)

OBJECT : AGENT **CLASS: PGagent** **DESCRIPTION: Agents represent a machine and have the following attributes and Methods.**	
Attributes	
AgentID	Attribute defining the Agent ID, this is similar to the machine number.
NAST	Represent the next available start time of a machine at that particular state
StateID	Represent the current agent state.
List<PGop> **WQ**	A represents the machines waiting queue. It contains a list of the job operations waiting to be scheduled for processing. Operations are members of the class PGop.
List<PGop> **SQ**	A represents the machines Schedule queue. It contains a list of the job operations that have already been scheduled. Operations are members of the class PGop.
Methods	
CryptoRandom **RandomSelection**	This is represent a random function based a CryptoRandom class that we developed that a allows an agent to make a true random action
RANDOM()	This represents a strategy, *random*, that can be used by an agent to select a job to schedule next from the available operations in its queue.
Move()	This function allows an agent to make a move and select which of the above strategies the move will be

	based on.
getOpStartTime()	Once an agent selects a job operation to be scheduled, as part of the scheduling tasks, this function allows assigning of a time slot on agent to a job's operation.
getNewNAST()	After scheduling an operation, this function is used by the agent to recalculate its new *'Next Available Start Time'*

TABLE 10: PGAGENT PROPERTIES

OBJECT : Operations **CLASS: PGop** **DESCRIPTION: This object represent a single operation**	
Attributes	
opID	Identify the operations ID.
JobID	Identify which Job the operation belongs to.
SeqID	Identify the sequence position the operation occupies
predecessor	Identify operation's predecessor(operation that precedes this operation on the same job)
Successor	Identify operation's successor(operation that follows this operation on the same job)
status;	The current status of operation e.g Waiting, Scheduled, Active, e.t.c
processingStartTime	Represents the starting time for processing of the operation on its specific machine.
load	Represents the processing time required to process the operation on its specific machine.
state;	Represent the current environment state of the operation
machineID	Identifies which machine the operation needs to be processed in.

TABLE 11: PGOPERATION PROPERTIES

OBJECT : Enviroment **CLASS: PGEnv** **DESCRIPTION: Represent the potential game environment.**	
Attributes	
List<PGagent> **PGAgentList**	This represent all the agents available in the environment
List<PGop> **PGOpList**	This represent all the perations available in the environment
List<int> **DispatchQueue**	This represent the dispatch queue in the environment that holds operation that are yet to be selected
List<List<envState>> **allStreams**	This holds instances of all previous states
Methods	
CreateDispatchQueue()	Function used to create the Dispatch Queue.
DispatchExit()	Used to remove operations form the dispatch queue an deliver to the relevant Agent's waiting Queue
agentsAction()	Functions used to prompt an agent to make a move
checkTerminal()	Check if the State arrived at is terminal
penalty()	Provides reinforcement (penalty) an Agent's action.
QValue()	Used to determine the qValue(utility value)of the selected path of actions
MoveToMachineQueue()	Used by the function *'CreateDispatchQueue'* to remove operation from dispatch queue to the relevant machines waiting queue.

TABLE 12: PGENVIROMEN PROPERTIES

OBJECT : State CLASS: envState DESCRIPTION: Represents a state of environment.	
Attributes	
State	Identifies the state.
strategy	Outlines the strategy selected/employed by the agents I selection on that particular state.
qValue	Stores the utility value of a path of state-action pairs
penalty	Reinforcement given to an agent for making a specific action at a specific state.
List<PGagent> TAgentList;	Store all agent states (as list of PGagent objects) for this particular state
Methods	
envState	A constructor used to initialized the state

TABLE 13: ENVSTATE PROPERTIES

The potential game algorithm is implemented according to the following flowchart.

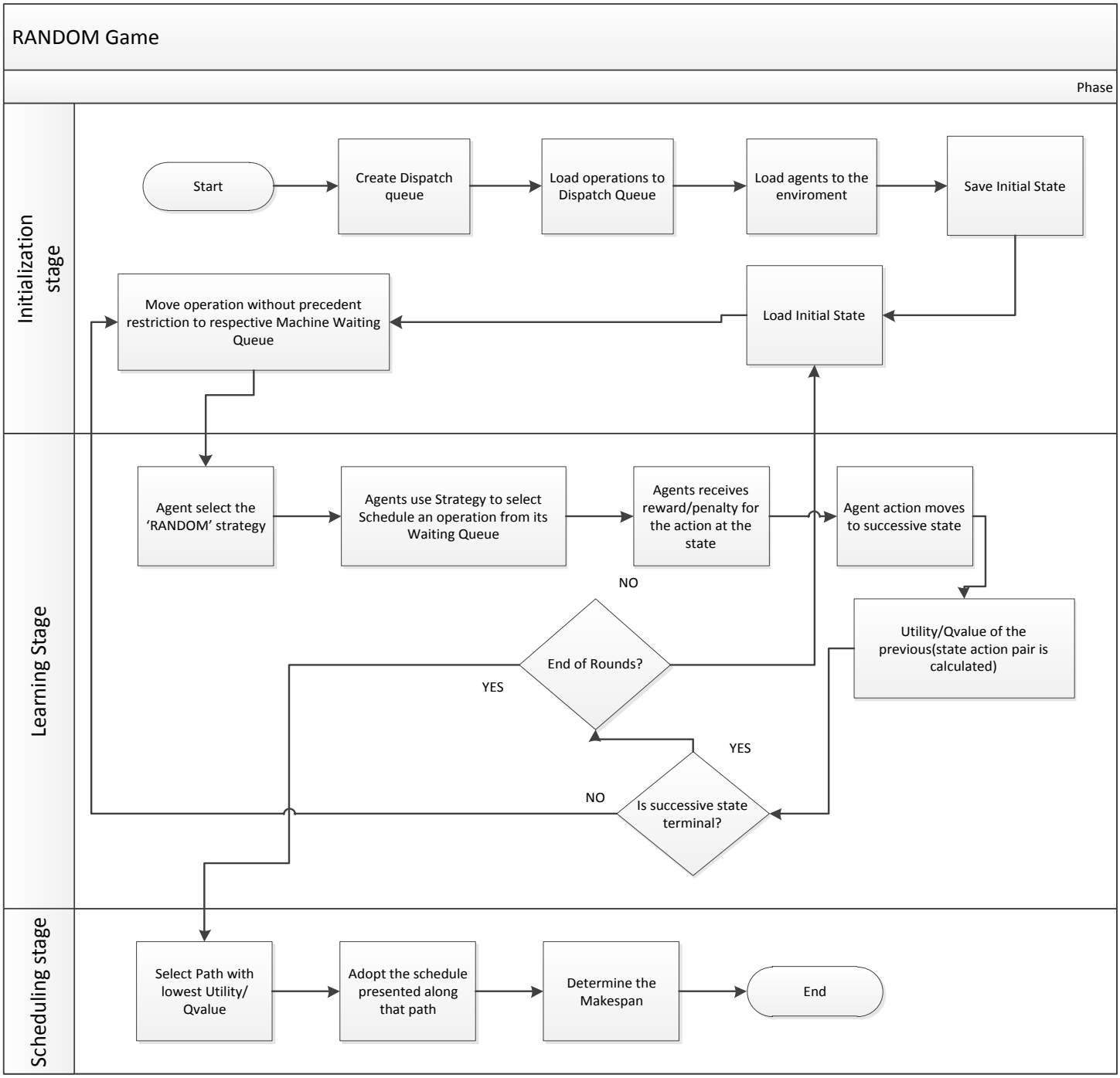


FIGURE 27: RANDOM GAMES FLOWCHART

5.2.3. RANDOM TOKEN GAMES

The random token game includes the following objects.

OBJECT : AGENT **CLASS: RTGagent** **DESCRIPTION: Agents represent a job and have the following attributes and Methods.**	
Attributes	
id	Identifies the operation
JobId	Identifies which job the operation belongs to.
agentId	Identifies the agent
MachineId;	Identifies the machine where the operation is to be processed on.
ProcessingTime	Represents the processing time need to complete processing of the job.
processingStartTime	Represent the time when the processing of the operation will start, once scheduled on it respective machine.
predecessor	Represents the Agents predecessor operation on the same job
successor	Represents the Agents successor operation on the same job
status	Represent which status the agent is currently in. E.g. waiting, scheduled, Active, e.t.c
OriginalMachinePriority	Represent the priority which is given to angent on a machine once it has been scheduled.
Methods	
selectMachine()	Methods allows the agent to select a position on its machines waiting queue
Schedule()	Methods used by agent to schedule itself on a machine once allowed to do so.
checkPredessorStatus()	Method used by agent to check the status of its predecessor(if

	scheduled or not).
getStartTimes()	Method used by agent to calculate its start time based on the machine's NAST and its predecessor's start time

TABLE 14: RTGAGENT PROPERTIES

OBJECT : Enviroment **CLASS: RTGenv** **DESCRIPTION: Represents the environment of an instance of a game.**	
Attributes	
List<rtgAgent> rtgAgentList	A list of Agent objects in the enviroment
List<rtgMachine> rtgMachineList	A list of machine objects in the enviroment
List<int> DispatchQueue	Represents ids of agents that are currently in the dispatch queue
List<string> Derivations	Stores the derivation path to the final solution
List<decimal> Makespan	List the makespan of all the solutions achived at each game round
curentMakeSpan	Holds the makespan of the current round
Methods	
CreateDispatchQueue	Method used to create dispatch queue at the games initialization
AverageQueue	Gets the average processing time of all the agent in all the specific queue
scheduleAgent	Method used to notify an agent to schedule after all the precedence constraints are met.
reprioritize	Method used to reprioritize agent if it fails to meet scheduling criteria's while scheduling.
CryptoRandom RandomSelection	An object of the CryptoRandom class that we defined that to allow facilitation of real random selection

TABLE 15: RTGENVIROMENT PROPERTIES

OBJECT : Machine CLASS: RTGmachine DESCRIPTION: Represents the machines in the games	
Attributes	
List<int> waitingQueue	Holds the ids of the agent that are waiting to be scheduled on the machine. The index represent their priority rating
List<int> sheduleQueue	Holds the ids of the agent that have been scheduled on the machine.
machineID	Identifies the machine.
NAST	Represents the 'Next Available Start Time' on the machine.
Methods	
rtgMachine()	Constructor to initiate the machine at creation.

TABLE 16: RTG MACHINE PROPERTIES

Random token game process is as outlined below. The process is repeated in every iteration/round, and at each game round a candidate solution is generated.

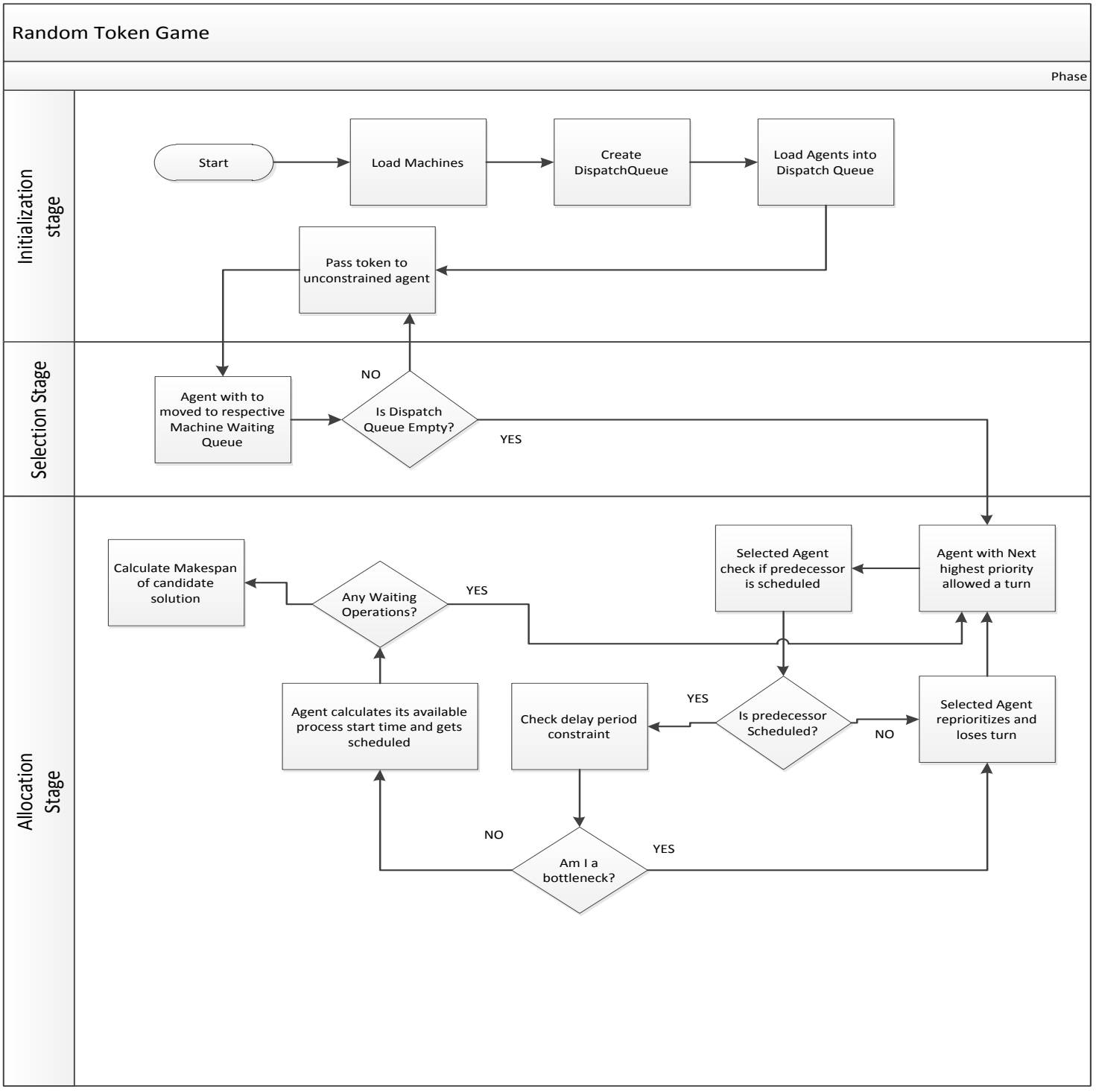


FIGURE 28 : RANDOM TOKEN GAME FLOWCHART

5.3. UI DESIGN

In this section we give an overview of user interface for the visualization tool. The tool has three screen each representing the various games. We show the content of each screen in the sub sections below.

5.3.1. POTENTIAL GAMES AND RANDOM GAMES

The potential and random screens has the following tabs/sections

Parameters Sections

This section allows selection of games parameter and execution of the game.

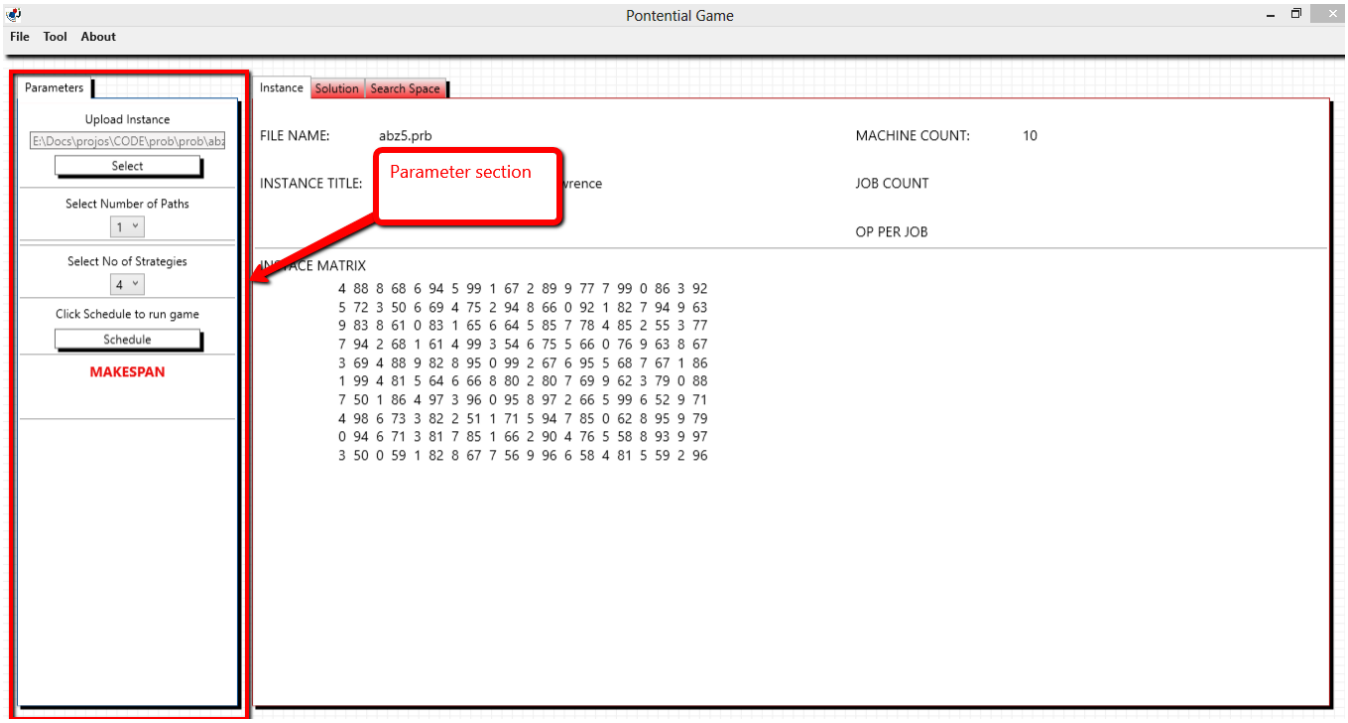


FIGURE 29: POTENTIAL ANDRANDOM GAMES PARMETER SECTION

Instance Sections

This section displays the details of the problem instance that has been selected for scheduling.

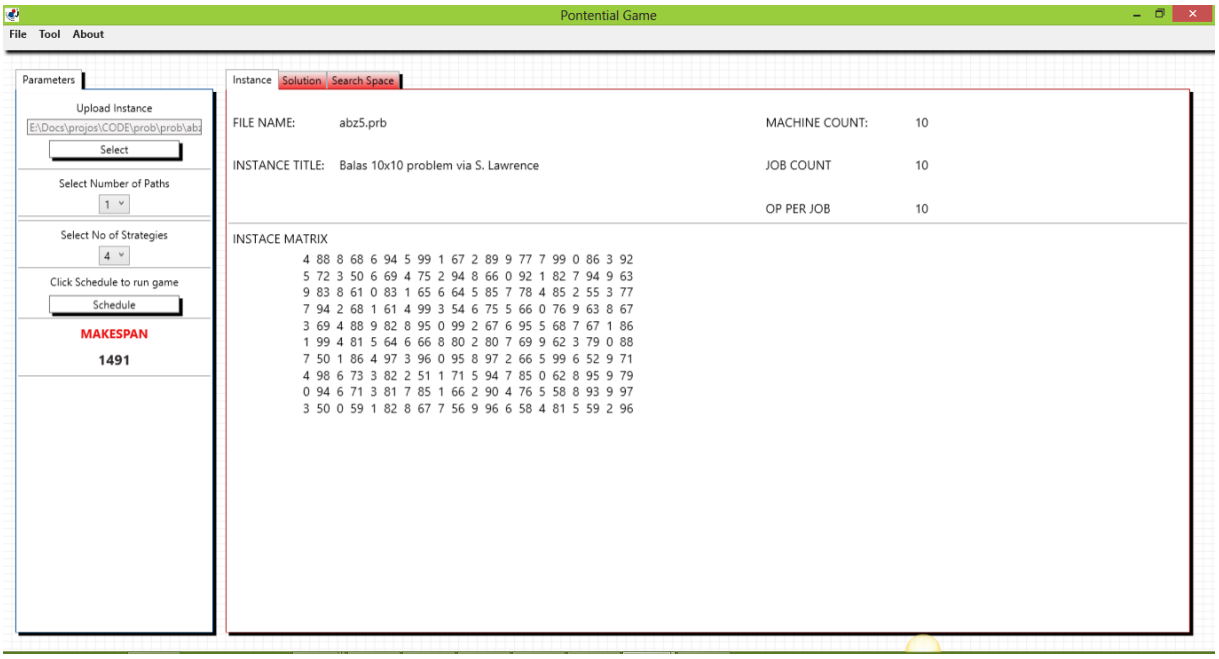


FIGURE 30: POTENTIAL ANDRANDOM GAMES INSTANCE SECTION

Solution Sections

This section displays the details of the feasible solution achieved by the algorithm.

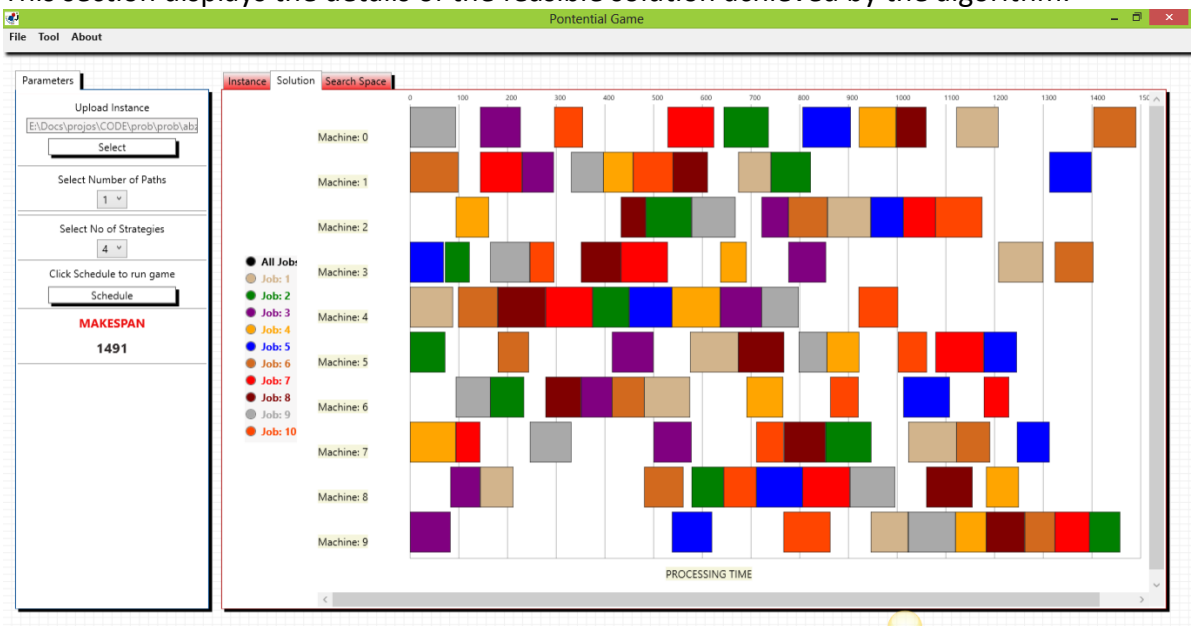


FIGURE 31: POTENTIAL ANDRANDOM GAMES SOLUTION SECTION

Search Space

Gives details about the solution selected and places it in context of all other solutions generated.

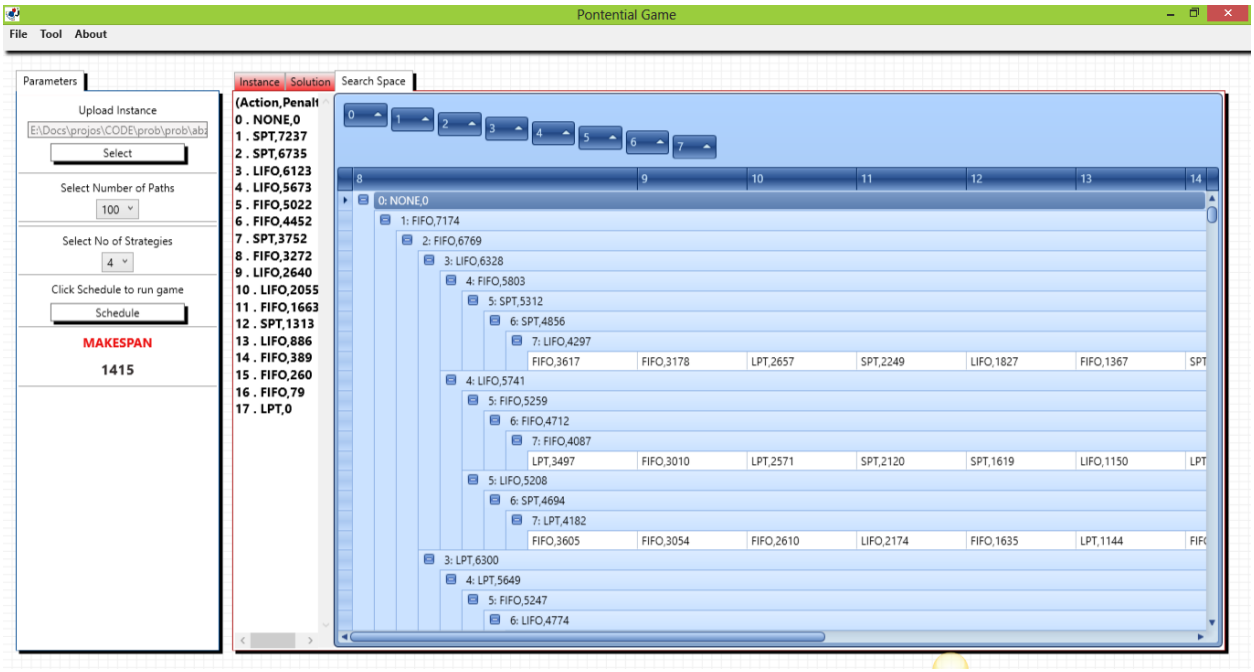


FIGURE 32: POTENTIAL ANDRANDOM GAMES SEACH SPACE SECTION

5.3.2. RANDOM TOKEN GAME

The random token game screen has the following tabs/sections

Parameters Sections

This section allows selection of games parameter and execution of the game.

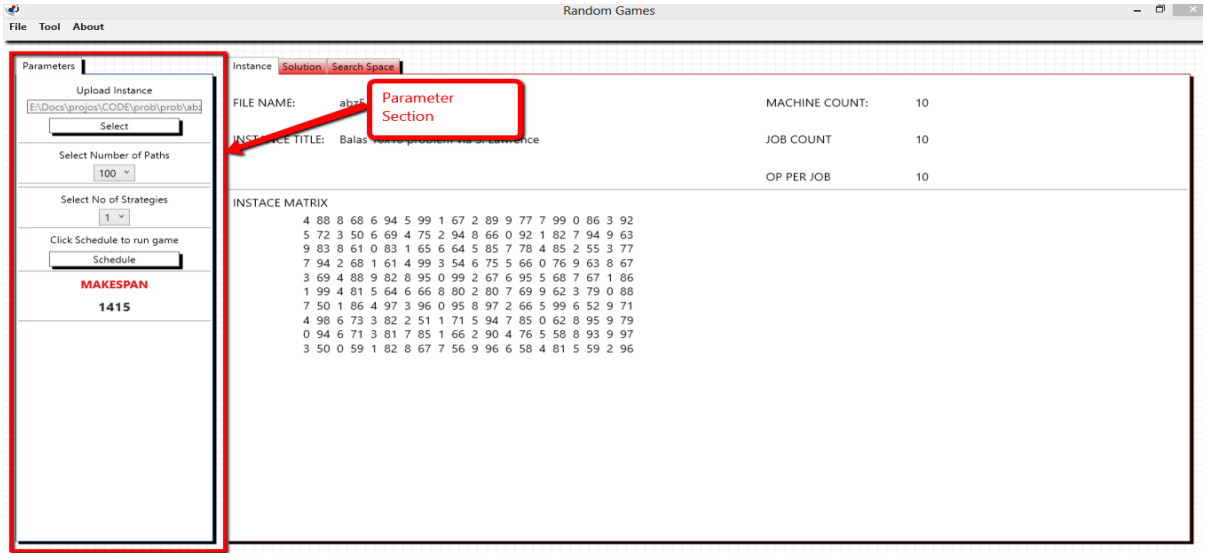
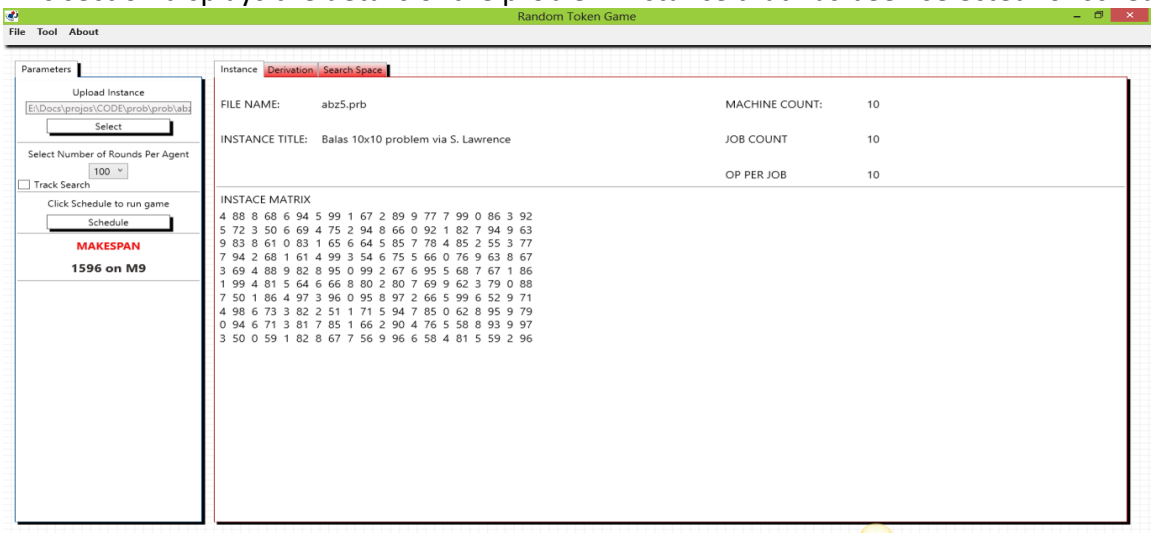


FIGURE 33: RANDOM TOKEN GAMES PARMETER SECTION

Instance Sections

This section displays the details of the problem instance that has been selected for scheduling.



Derivation Sections

This section displays details of how the schedule was achieved through the game.

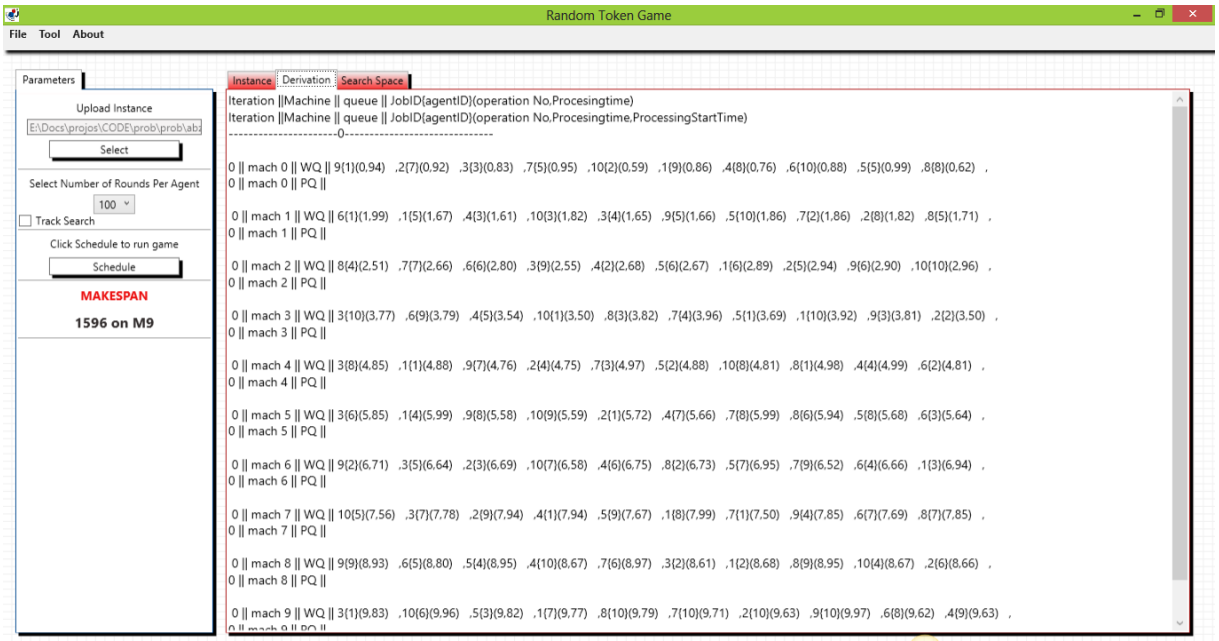


FIGURE 34: RANDOM TOKEN GAMES DERIVATION SECTION

Search Space Sections

This section displays details of how the all schedules have been derived through the game.

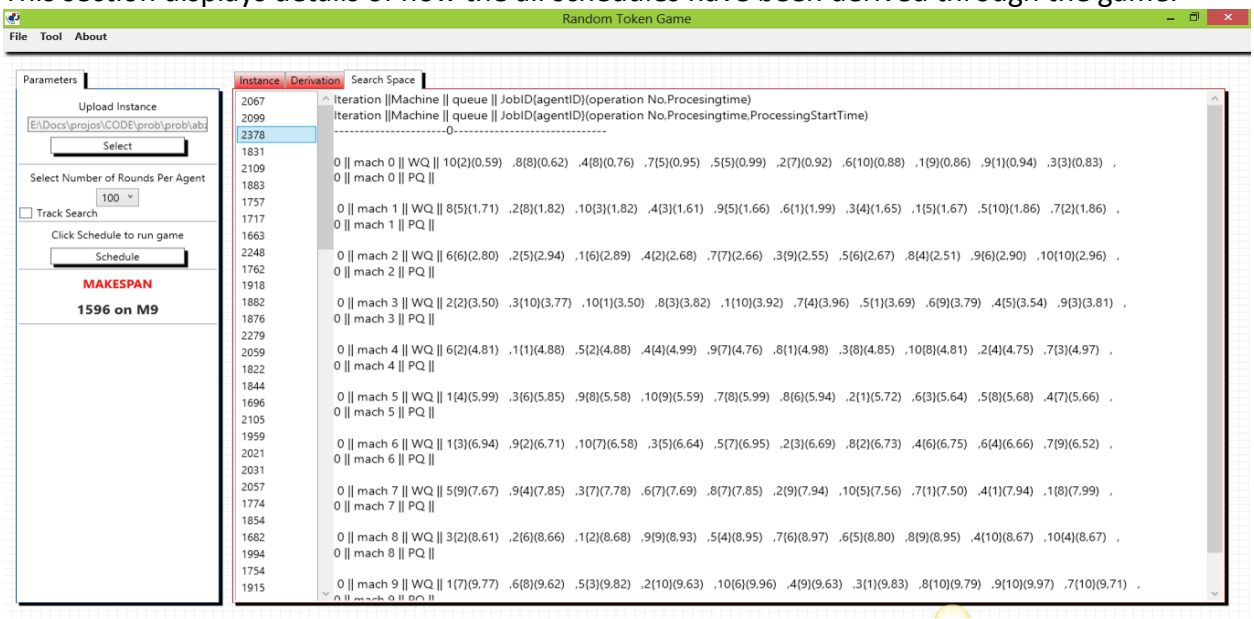


FIGURE 35: RANDOM TOKEN GAMES SEARCH SPACE SECTION

CHAPTER SIX: TEST, RESULTS AND CONCLUSIONS

For our test we use benchmark problems used in Beasley's operation research library complied by Professor Beasley J (Beasley 2005), this is found on Brunel's universities website. The benchmarks offers a list of different instances of job shop problems complied by different researchers in there works. We compare the performance for each our defined games against other similar algorithms. We also compare the performance of the game given the various game parameters and against various instance setting;

5.4. BENCHMARK CASES

5.4.1. BENCHMARK PROBLEM INSTANCES

in benchmarking performance of various algorithms. The benchmark problems contain instance problems of varying size. Each of the benchmark problems has an optimal know solution's makespan defined for them, some of the solution's makespan are known to be optimal, while others are the best known solution. The Benchmark instance consists of the following;

- ABZ 5 problems of 2 sizes proposed by Adams, Balas and Zawack (1989): ABZ 5 and ABZ 6 instances of size 10×10 with processing times from the intervals [50,100] and [25,100] respectively and ABZ 7 – 9 instances of size 20×15 and processing times
- la01-la40 are from "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques" by S. Lawrence.
- mt06, mt10, and mt20 are from "Industrial Scheduling" edited by Muth and Thompson.
- Car1-car8 are from "Ordonnancements a contraintes disjonctives" by J. Carlier.
- orb1-orb10 were generated in Bonn in 1986.

The benchmark problems are each presented in a separate text and each file has the following structure.

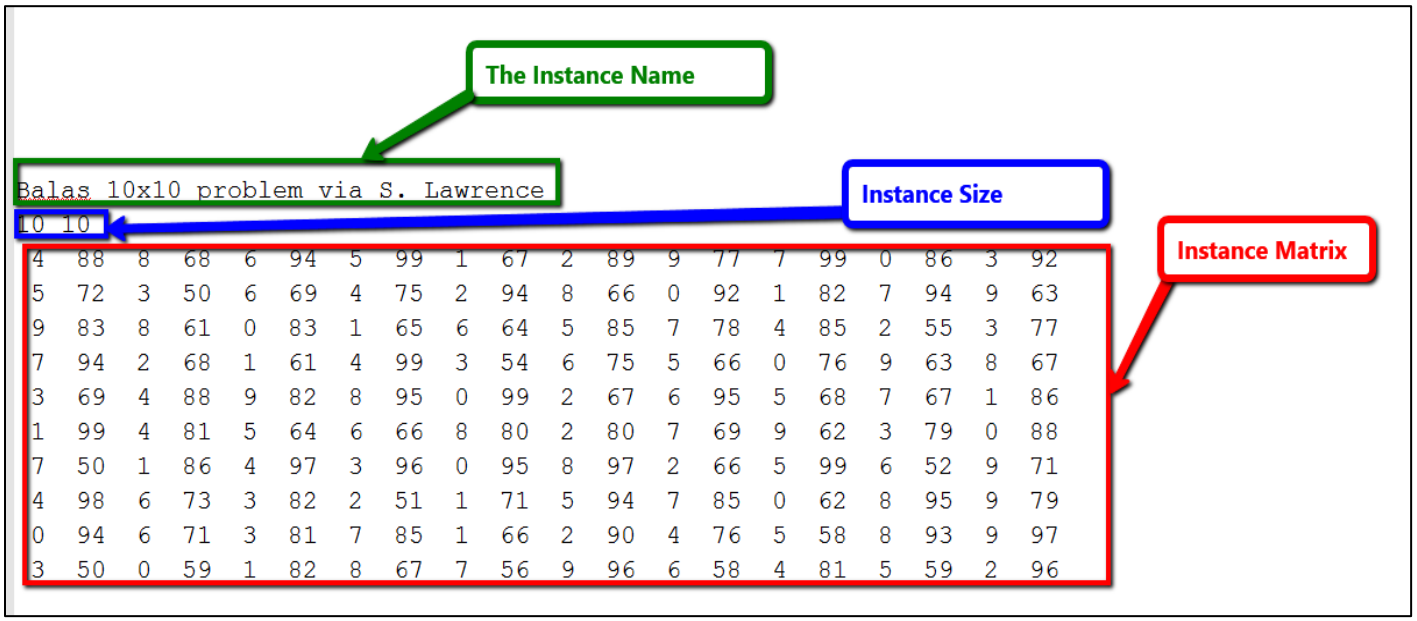


FIGURE 36: BENCHMARK PROBLEM FILE STRUCTURE AND CONTENT

The structure items are detailed as follows;

- ***Instance Name;*** the instance name/title has the following information about the instance.
    - The name of the researcher who created the instance.
    - The source research of where Beasley acquired the instance.
    - It also contains information on the instance size, in the above example 10 X 10, represent the number of Machine and Jobs in the format (no. of *jobs X no. of machines*).
- **Instance Size;** this details the number of jobs and machines the first number represents the number of jobs while the second number represent number of machines.
- **Instance Matrix;** this matrix details the instance itself, it contains a matrix of (machine, operation) pairs. The matrix structure is as follows;
    - Every row represents a Job,
    - In every row we have columns each defining an operation of the job. The operation of the job is describe using the pair (machine, load);

- The Machine, represent the machine where the operation need to be processed on. While,
- Load; represent the amount of time it will take the operation to process on the machine.

5.4.2. BENCHMARK ALGORITHMS

We evaluated the performance of our games and compared them to the results from the following heuristic based studies. We only compare the work to heuristic based algorithms and we also only considered algorithms that have benchmarked with more than one problem instance size. We selected , *"Multi-resource shop scheduling with resource flexibility and blocking."* (Y Mati and X Xie, 2011). This was chosen because the study relatively recent. The algorithms in the study are meant for 10X10 problems only. We will refer to the algorithm in the study has "**MX**" We also compare our results to the mean results of *"Use of an Artificial Immune System for Job Shop Scheduling"*, (CAC Coello et al, 2003). Chosen because it analyses performance across multiple different sizes of the problem instance. We will refer to this study as "**AIS"** and we also compare against *"Job-Shop with Generic Time-Lags: A Heuristic Based Approach".* (P. Lacomme, 2011). This study was chosen because it provides measures against both flow shop and job shop scheduling problems. We abbreviate this as ***"GTL".***

The table below shows characteristics of the comparison algorithms chosen.

Problem	*Symbol*	Types of problem	Instances Sizes Handle
"Multi-resource shop scheduling with resource flexibility and blocking." (Y Mati and X Xie, 2011).	***MX***	Job Shop Scheduling	10 X 10, instance only
"*Use of an Artificial Immune System for Job Shop Scheduling*", (CAC Coello et al, 2003)	***AIS***	Job Shop Scheduling	Multiple
"A contribution to the stochastic flow shop scheduling problem", (M. Gourgand et al, 2003)	***SD***	Flow Shop Scheduling	Multiple
"Job-Shop with Generic Time-Lags: A Heuristic Based Approach". (P. Lacomme, 2011)	***GLT***	Flow Shop Scheduling Job Shop Scheduling	Multiple

TABLE 17: BENCHMARK ALGORITHMS

5.5. RESULTS

5.5.1. POTENTIAL GAMES RESULTS

The following test was done on potential games and the results are as follows.

Test with different configurations of the potential game

The table below represents test on potential games done with different configurations; the table contains the following content;

- ***Problem***- This represents the benchmark problem being solved.
- ***Size***, the number of jobs and machine in the bench mark problem being solved.
- ***Optimal***, the known optimal solution for the benchmark being solved.
- For each of the 3 above we compare against 4 test categorized into two categories and named ***"Best of X try (Y paths)"*** –This means picking the best makespan after Performing ***X*** trials on the benchmark problems with the potential games set to ***Y*** paths.
- For each category we do test using only 2 strategies and all the strategies. For each we note the makespan and the error rate of the solution.

Problem	Size	Optimal	Best of 10 (100 paths)				Best of 10 (1000 paths)			
			(SPT and LPT) strategies only	%Error	All 4 strategies	%Error	(SPT and LPT) strategies only	%Error	All 4 strategies	%Error
la01	10 X 5	666	795	19.37	763	14.56	799	19.97	765	14.86
la02	10 X 5	655	803	22.60	803	22.60	803	22.60	802	22.44
la05	10 X 5	593	652	9.95	620	4.55	727	22.60	671	13.15
abz5	10 X10	1234	1346	9.08	1370	11.02	1358	10.05	1354	9.72
la16	10 X10	945	1093	15.66	1059	12.06	1229	30.05	1118	18.31
la17	10 X10	784	916	16.84	997	27.17	915	16.71	1051	34.06
la18	10 X10	848	1063	25.35	1067	25.83	1043	23.00	1068	25.94
la19	10 X10	842	1020	21.14	1084	28.74	1072	27.32	1055	25.30
la36	15 X 15	1268	1676	32.18	1627	28.31	1716	35.33	1692	33.44
la38	15 X 15	1217	1658	36.24	1617	32.87	1718	41.17	1718	41.17
la39	15 X 15	1233	1662	34.79	1628	32.04	1743	41.36	1677	36.01
la40	15 X 15	1222	1779	45.58	1630	33.39	1707	39.69	1708	39.77
abz7	15 X 20	668	881	31.89	857	28.29	864	29.34	870	30.24
abz8	15 X 20	687	899	30.86	842	22.56	885	28.82	880	28.09

abz9	15 X 20	707	911	28.85	887	25.46	923	30.55	912	29.00
la07	15 X 5	890	1048	17.75	1051	18.09	1063	19.44	946	6.29
la08	15 X 5	863	1067	23.64	1017	17.84	1144	32.56	986	14.25
la09	15 X 5	951	1144	20.29	1072	12.72	1152	21.14	1088	14.41
la10	15 X 5	958	1102	15.03	1035	8.04	1123	17.22	1073	12.00
la28	20 X 10	1216	1610	32.40	1630	34.05	1553	27.71	1551	27.55
la29	20 X 10	1195	1633	36.65	1506	26.03	1649	37.99	1645	37.66
orb02	10 X 10	888	1291	45.38	1154	29.95	1273	43.36	1306	47.07
orb03	10 X10	1005	1496	48.86	1330	32.34	1437	42.99	1426	41.89
orb04	10 X 10	1005	1397	39.00	1356	34.93	1274	26.77	1383	37.61
car1	11 X 5	7038	10418	48.03	8296	17.87	10069	43.07	9463	34.46
car2	13 X 4	7166	10418	45.38	9307	29.88	10354	44.49	9395	31.11
car3	12 X 5	7312	10345	41.48	9557	30.70	10303	40.91	9695	32.59
car4	14 X 4	8003	9318	16.43	8819	10.20	9318	16.43	8990	12.33
car5	10 X 6	7702	10323	34.03	10980	42.56	11200	45.42	11030	43.21
car6	8 X 9	8313	10291	23.79	10449	25.69	10895	31.06	10858	30.61
car7	7 X 7	6558	8228	25.47	8224	25.40	8228	25.47	8404	28.15

TABLE 18 : RESULTS OF THE POTENTIAL GAME WITH DIFFERENT SETTINGS.

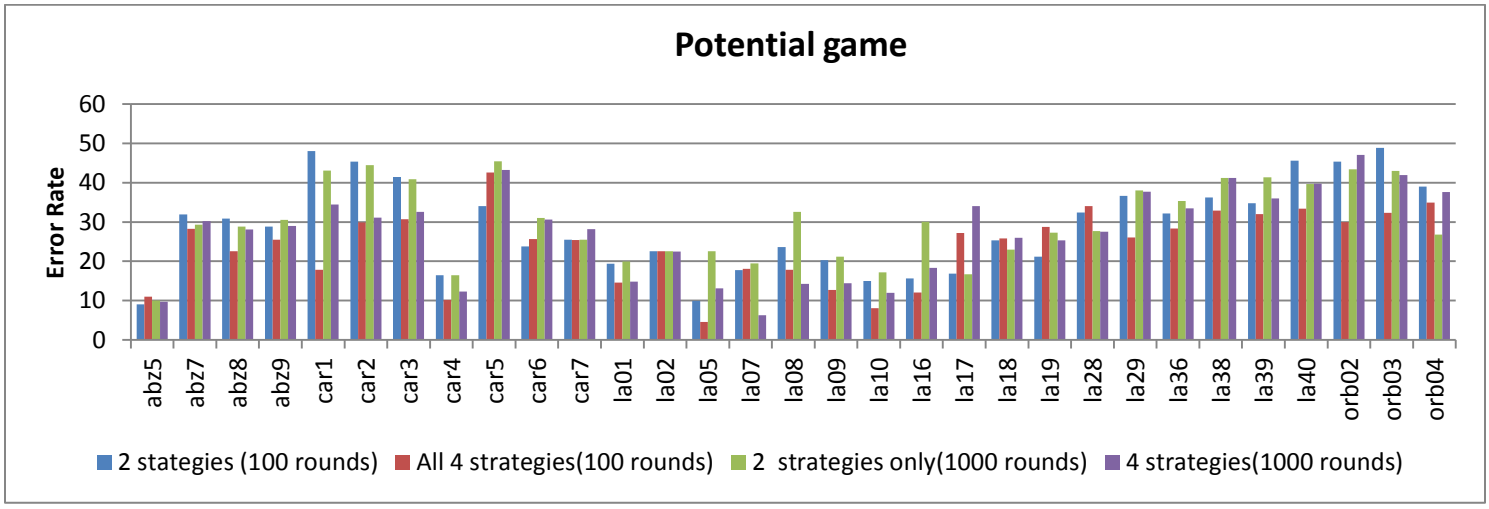


FIGURE 37: POTENTIAL GAME GRAPH GROUPED BY PROBLEM INSTANCE AND SIZE

Performance against selected benchmark algorithms

In this section we compare the potential game with the selected algorithms that were earlier discussed in section *6.1.2* . The comparison is against the best results of the potential game from the above test done with different configuration against the mean results obtained by the other algorithms has detailed in their respective literature.

Problem	Size	Optimal	Potential Game		AIS		MX		GTL	
			Makespan	%error	Makespan	%error	makespan	%error	makespan	%error
la01	10 X 5	666	763	14.56	776	16.46	881	32.28	875	31
la02	10 X 5	655	802	22.44	775	18.34	900	37.40	897	37
la05	10 X 5	593	620	4.55	617	3.96	742	25.13	878	48
abz5	10 X10	1234	1346	9.08	1470	19.10	1705	38.17		
la16	10 X10	945	1059	12.06	1100	16.42	1205	27.51	1599	69
la17	10 X10	784	915	16.71	912	16.30	1020	30.10	1292	65
la18	10 X10	848	1043	23.00	1013	19.49	1156	36.32	0	-100
la19	10 X10	842	1020	21.14	1031	22.42	1191	41.45	1403	67
la36	15 X 15	1268	1627	28.31	1561	23.07	2058	62.30	1747	38
la38	15 X 15	1217	1617	32.87	1548	27.23	2008	65.00	1725	42
la39	15 X 15	1233	1628	32.04	1548	25.57	2046	65.94	0	-100
la40	15 X 15	1222	1630	33.39	1537	25.81	2034	66.45	0	-100
abz7	15 X 20	668	857	28.29	839	25.64				
abz8	15 X 20	687	842	22.56	859	24.96				
abz9	15 X 20	707	887	25.46	884	24.96				
la07	15 X 5	890	946	6.29	961	7.99	1209	35.84	1123	26
la08	15 X 5	863	986	14.25	965	11.81	1261	46.12	0	-100
la09	15 X 5	951	1072	12.72	1019	7.12	1380	45.11	0	-100
la10	15 X 5	958	1035	8.04	982	2.49	1300	35.70	0	-100
la28	20 X 10	1216	1551	27.55	1555	27.85	2381	95.81	1997	64
la29	20 X 10	1195	1506	26.03	1463	22.43	2256	88.79	0	-100
orb02	10 X 10	888	1154	29.95	1070	20.44	1181	33.00		
orb03	10 X10	1005	1330	32.34	1276	26.92	1311	30.45		
orb04	10 X 10	1005	1274	26.77	1221	21.47	1288	28.16		
car1	11 X 5	7038	8296	17.87					13788	96
car2	13 X 4	7166	9307	29.88					0	-100
car3	12 X 5	7312	9557	30.70					0	-100
car4	14 X 4	8003	8819	10.20					0	-100
car5	10 X 6	7702	10323	34.03					13597	77
car6	8 X 9	8313	10291	23.79					0	-100
car7	7 X 7	6558	8224	25.40					10948	66.94

TABLE 19: PONTENTIAL GAMES AGAINST BENCHMARKS

5.5.2. RANDOM GAMES RESULTS

The following test was done on random games and the results are as follows.

Test with different configurations of the random game

The table below represents test on potential games done with different configurations; the table contains the following content;

- ***Problem***- This represents the benchmark problem being solved.
- ***Size***, the number of jobs and machine in the bench mark problem being solved.
- ***Optimal***, the known optimal solution for the benchmark being solved.
- We perform test with 100 rounds and 1000 rounds.

Problem	Size	Optimal	100 rounds		1000 rounds	
			Best of 10	%Error	Best of 10	%Error
la01	10 X 5	666	782	14.83	773	16.07
la02	10 X 5	655	796	17.71	810	23.66
la05	10 X 5	593	652	9.05	695	17.20
abz5	10 X10	1234	1395	11.54	1395	13.05
la16	10 X10	945	1103	14.32	1145	21.16
la17	10 X10	784	980	20.00	954	21.68
la18	10 X10	848	1020	16.86	1087	28.18
la19	10 X10	842	990	14.95	1003	19.12
la36	15 X 15	1268	1696	25.24	1656	30.60
la38	15 X 15	1217	1598	23.84	1697	39.44
la39	15 X 15	1233	1647	25.14	1603	30.01
la40	15 X 15	1222	1654	26.12	1637	33.96
abz7	15 X 20	668	845	20.95	823	23.20
abz8	15 X 20	687	910	24.51	934	35.95
abz9	15 X 20	707	945	25.19	987	39.60
la07	15 X 5	890	1048	15.08	1102	23.82
la08	15 X 5	863	1034	16.54	1111	28.74
la09	15 X 5	951	1132	15.99	1186	24.71
la10	15 X 5	958	1034	7.35	1091	13.88
la28	20 X 10	1216	1648	26.21	1570	29.11
la29	20 X 10	1195	1599	25.27	1680	40.59
orb02	10 X 10	888	1134	21.69	1154	29.95
orb03	10 X10	1005	1137	11.61	1165	15.92
orb04	10 X 10	1005	1404	28.42	1477	46.97
car1	11 X 5	7038	8700	19.10	8787	24.85
car2	13 X 4	7166	8664	17.29	9034	26.07
car3	12 X 5	7312	8769	16.62	8654	18.35
car4	14 X 4	8003	9408	14.93	9304	16.26

car5	10 X 6	7702	9876	22.01	9702	25.97
car6	8 X 9	8313	9867	15.75	9707	16.77
car7	7 X 7	6558	8790	25.39	8669	32.19

TABLE 20: RANDOM GAMES WITH DIFFERENT CONFIGURATION

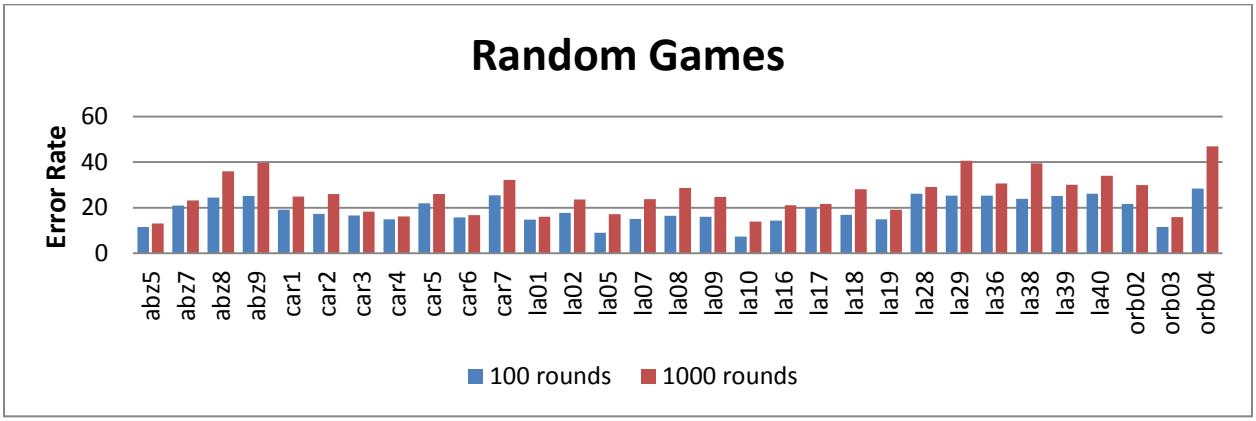


FIGURE 38: RANDOM GAME GRAPH GROUPED BY INSTANCE TYPE AND SIZE

Performance against selected benchmark algorithms

In this section we compare the random game with the selected algorithms that were earlier discussed in section *6.1.2* . The comparison is against the best results of the random game from the above test done with different configuration against the mean results obtained by the other algorithms has detailed in their respective literature.

Problem	Size	Optimal	Random games		AIS		MX		GTL	
			Best of 10	%Error	Makespan	%error	makespan	%error	makespan	%error
la01	10 X 5	666	782	14.83	776	16.46	881	32.28	875	31
la02	10 X 5	655	796	17.71	775	18.34	900	37.40	897	37
la05	10 X 5	593	652	9.05	617	3.96	742	25.13	878	48
abz5	10 X10	1234	1395	11.54	1470	19.10	1705	38.17		

la16	10 X10	945	1103	14.32	1100	16.42	1205	27.51	1599	69
la17	10 X10	784	980	20.00	912	16.30	1020	30.10	1292	65
la18	10 X10	848	1020	16.86	1013	19.49	1156	36.32	0	-100
la19	10 X10	842	990	14.95	1031	22.42	1191	41.45	1403	67
la36	15 X 15	1268	1696	25.24	1561	23.07	2058	62.30	1747	38
la38	15 X 15	1217	1598	23.84	1548	27.23	2008	65.00	1725	42
la39	15 X 15	1233	1647	25.14	1548	25.57	2046	65.94	0	-100
la40	15 X 15	1222	1654	26.12	1537	25.81	2034	66.45	0	-100
abz7	15 X 20	668	845	20.95	839	25.64				
abz8	15 X 20	687	910	24.51	859	24.96				
abz9	15 X 20	707	945	25.19	884	24.96				
la07	15 X 5	890	1048	15.08	961	7.99	1209	35.84	1123	26
la08	15 X 5	863	1034	16.54	965	11.81	1261	46.12	0	-100
la09	15 X 5	951	1132	15.99	1019	7.12	1380	45.11	0	-100
la10	15 X 5	958	1034	7.35	982	2.49	1300	35.70	0	-100
la28	20 X 10	1216	1648	26.21	1555	27.85	2381	95.81	1997	64
la29	20 X 10	1195	1599	25.27	1463	22.43	2256	88.79	0	-100
orb02	10 X 10	888	1134	21.69	1070	20.44	1181	33.00		
orb03	10 X10	1005	1137	11.61	1276	26.92	1311	30.45		
orb04	10 X 10	1005	1404	28.42	1221	21.47	1288	28.16		
car1	11 X 5	7038	8700	19.10					13788	96
car2	13 X 4	7166	8664	17.29					0	-100
car3	12 X 5	7312	8769	16.62					0	-100
car4	14 X 4	8003	9408	14.93					0	-100
car5	10 X 6	7702	9876	22.01					13597	77
car6	8 X 9	8313	9867	15.75					0	-100
car7	7 X 7	6558	8790	25.39					10948	67

TABLE 21: RANDOM GAMES AGAINST BENCHMARK ALGORITHMS

5.5.3. RANDOM TOKEN GAMES RESULTS

The following test was done on random token games and the results are as follows.

Test with different configurations of the random token game

The table below represents test on random token games done with different configurations; the table contains the following content;

- ***Problem***- This represents the benchmark problem being solved.
- ***Size***, the number of jobs and machine in the bench mark problem being solved.

- ***Optimal***, the known optimal solution for the benchmark being solved.
- We perform test with 100 rounds and 1000 rounds.

Problem	Size	Optimal	100 rounds		1000 rounds	
			Best of 10	%Error	Best of 10	%Error
la01	10 X 5	666	735	10.36	716	7.51
la02	10 X 5	655	657	0.31	675	3.05
la05	10 X 5	593	593	0.00	593	0.00
abz5	10 X10	1234	1409	14.18	1482	20.10
la16	10 X10	945	1111	17.57	1065	12.70
la17	10 X10	784	980	25.00	920	17.35
la18	10 X10	848	1011	19.22	1009	18.99
la19	10 X10	842	1093	29.81	1008	19.71
la36	15 X 15	1268	1704	34.38	1684	32.81
la38	15 X 15	1217	1704	40.02	1682	38.21
la39	15 X 15	1233	1610	30.58	1519	23.20
la40	15 X 15	1222	1756	43.70	1724	41.08
abz7	15 X 20	668	846	26.65	816	22.16
abz8	15 X 20	687	867	26.20	901	31.15
abz9	15 X 20	707	920	30.13	942	33.24
la07	15 X 5	890	999	12.25	968	8.76
la08	15 X 5	863	994	15.18	1007	16.69
la09	15 X 5	951	1101	15.77	1001	5.26
la10	15 X 5	958	1016	6.05	1067	11.38
la28	20 X 10	1216	1643	35.12	1667	37.09
la29	20 X 10	1195	1602	34.06	1598	33.72
orb02	10 X 10	888	1089	22.64	999	12.50
orb03	10 X10	1005	1023	1.79	1043	3.78
orb04	10 X 10	1005	1189	18.31	1091	8.56
car1	11 X 5	7038	10016	42.31	9750	38.53
car2	13 X 4	7166	9960	38.99	9680	35.08
car3	12 X 5	7312	10089	37.98	9878	35.09
car4	14 X 4	8003	11189	39.81	10587	32.29
car5	10 X 6	7702	10234	32.87	9996	29.78
car6	8 X 9	8313	9678	16.42	9453	13.71
car7	7 X 7	6558	8913	35.91	8167	24.53

TABLE 22: RANDOM TOKEN GAME TEST WITH DIFFERENT CONFIGURATION

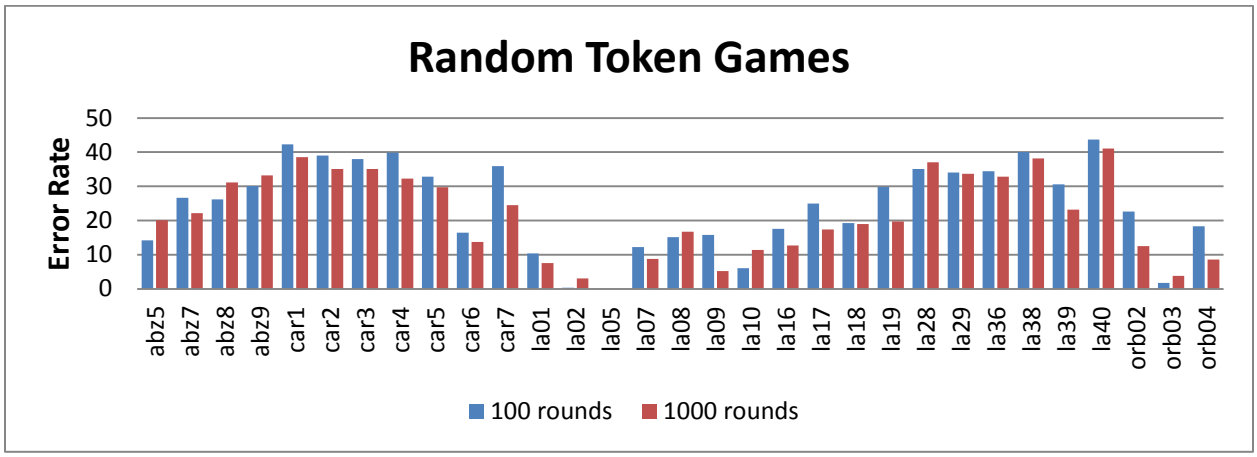


FIGURE 39: RANDOM TOKEN GAME GRAPH GROUPED BY PROBLEM SOURCE AND SIZE

Performance against selected benchmark algorithms

In this section we compare the random token game with the selected algorithms that were earlier discussed in section *6.1.2* . The comparison is against the best results of the random token game from the above test done with different configuration against the mean results obtained by the other algorithms has detailed in their respective literature.

Problem	Size	Optimal	Random token Game		AIS		MX		GTL	
			Makespan	%error	Makespan	%error	makespan	%error	makespan	%error
la01	10 X 5	666	716	7.51	776	16.46	881	32.28	875	31
la02	10 X 5	655	657	0.31	775	18.34	900	37.40	897	37
la05	10 X 5	593	593	0.00	617	3.96	742	25.13	878	48
abz5	10 X10	1234	1409	14.18	1470	19.10	1705	38.17		
la16	10 X10	945	1065	12.70	1100	16.42	1205	27.51	1599	69
la17	10 X10	784	920	17.35	912	16.30	1020	30.10	1292	65
la18	10 X10	848	1009	18.99	1013	19.49	1156	36.32	0	-100
la19	10 X10	842	1008	19.71	1031	22.42	1191	41.45	1403	67
la36	15 X 15	1268	1684	32.81	1561	23.07	2058	62.30	1747	38
la38	15 X 15	1217	1682	38.21	1548	27.23	2008	65.00	1725	42
la39	15 X 15	1233	1519	23.20	1548	25.57	2046	65.94	0	-100
la40	15 X 15	1222	1724	41.08	1537	25.81	2034	66.45	0	-100
abz7	15 X 20	668	816	22.16	839	25.64				
abz8	15 X 20	687	867	26.20	859	24.96				
abz9	15 X 20	707	920	30.13	884	24.96				
la07	15 X 5	890	968	8.76	961	7.99	1209	35.84	1123	26

la08	15 X 5	863	994	15.18	965	11.81	1261	46.12	0	-100
la09	15 X 5	951	1001	5.26	1019	7.12	1380	45.11	0	-100
la10	15 X 5	958	1016	6.05	982	2.49	1300	35.70	0	-100
la28	20 X 10	1216	1643	35.12	1555	27.85	2381	95.81	1997	64
la29	20 X 10	1195	1598	33.72	1463	22.43	2256	88.79	0	-100
orb02	10 X 10	888	999	12.50	1070	20.44	1181	33.00		
orb03	10 X10	1005	1023	1.79	1276	26.92	1311	30.45		
orb04	10 X 10	1005	1091	8.56	1221	21.47	1288	28.16		
car1	11 X 5	7038	9750	38.53					13788	96
car2	13 X 4	7166	9680	35.08					0	-100
car3	12 X 5	7312	9878	35.09					0	-100
car4	14 X 4	8003	10587	32.29					0	-100
car5	10 X 6	7702	9996	29.78					13597	77
car6	8 X 9	8313	9453	13.71					0	-100
car7	7 X 7	6558	8167	24.53					10948	66.94

TABLE 23: RANDOM GAME TOKEN COMPARISION WITH OTHER ALGORITHMS

Finally we compare all our algorithms against each other.

Problem	Size	Optimal	Random token Games		Random Games		potential Games	
			makespan	%Error	makespan	%Error	makespan	%Error
la01	10 X 5	666	716	6.98	782	17.42	763	14.56
la02	10 X 5	655	657	0.30	796	21.53	802	22.44
la05	10 X 5	593	593	0.00	652	9.95	620	4.55
abz5	10 X10	1234	1409	12.42	1395	13.05	1346	9.08
la16	10 X10	945	1065	11.27	1103	16.72	1059	12.06
la17	10 X10	784	920	14.78	980	25.00	915	16.71
la18	10 X10	848	1009	15.96	1020	20.28	1043	23.00
la19	10 X10	842	1008	16.47	990	17.58	1020	21.14
la36	15 X 15	1268	1684	24.70	1696	33.75	1627	28.31
la38	15 X 15	1217	1682	27.65	1598	31.31	1617	32.87
la39	15 X 15	1233	1519	18.83	1647	33.58	1628	32.04
la40	15 X 15	1222	1724	29.12	1654	35.35	1630	33.39
abz7	15 X 20	668	816	18.14	845	26.50	857	28.29
abz8	15 X 20	687	867	20.76	910	32.46	842	22.56
abz9	15 X 20	707	920	23.15	945	33.66	887	25.46
la07	15 X 5	890	968	8.06	1048	17.75	946	6.29
la08	15 X 5	863	994	13.18	1034	19.81	986	14.25
la09	15 X 5	951	1001	5.00	1132	19.03	1072	12.72
la10	15 X 5	958	1016	5.71	1034	7.93	1035	8.04

la28	20 X 10	1216	1643	25.99	1648	35.53	1551	27.55
la29	20 X 10	1195	1598	25.22	1599	33.81	1506	26.03
orb02	10 X 10	888	999	11.11	1134	27.70	1154	29.95
orb03	10 X10	1005	1023	1.76	1137	13.13	1330	32.34
orb04	10 X 10	1005	1091	7.88	1404	39.70	1274	26.77
car1	11 X 5	7038	9750	27.82	8700	23.61	8296	17.87
car2	13 X 4	7166	9680	25.97	8664	20.90	9307	29.88
car3	12 X 5	7312	9878	25.98	8769	19.93	9557	30.70
car4	14 X 4	8003	10587	24.41	9408	17.56	8819	10.20
car5	10 X 6	7702	9996	22.95	9876	28.23	10323	34.03
car6	8 X 9	8313	9453	12.06	9867	18.69	10291	23.79
car7	7 X 7	6558	8167	19.70	8790	34.03	8224	25.40

TABLE 24: COMPARISON AMOUNGST THE GAMES

5.6. DISCUSSIONS

The following are the observations from the test on the potential games above.

Test of algorithms with different configurations

The potential games algorithm perform relative better with more strategies used. This is because it increases breadth of choice and actions available to an agent. This increases the learning experience of an agent and increases the chance of learning a more favorable solution. The different in quality of solution produce when using only SPT and LPT compared to all four strategies, increase with the sizes of the instance. This is because using only 2 strategies limits the game to a subset of solutions in the search space. Limiting the experience scope of the agents.

Both the Potential game and Random games do not show improvement the quality of solution designed when the number of paths was increased significantly from 100 to 1000. This is because the quality of schedule generated for these more on the number of strategies used as they increase the breadth of choose or scope of learning for the agent. Increasing the number

learned paths learned without increasing the number of strategies available to the agents only leads the agent to learn multiple similar schedules, thus the agents is already limited to a certain range quality of solutions they can achieve. From further test we discovered that the quality solution increase gradually with increase of number of rounds until the number of round get to approximately 200 rounds for most problem instance, then quality reduces as the rounds increase. This can be attributed to the fact that increasing the number of rounds past a certain point, we also increase the chance of learning a false path, where the agent get good reinforce but decisions made lead to a poor schedule. This occurrence does expose a limitation in our reward structure of using the total remaining processing time.

Increasing the number of rounds in the random token game does show improvement in the quality of schedule generated. This is because it increases the number of solution the algorithms has to choose from the search space increase the probability of selecting a more favorable solution.

The tests on Potential and Random games also shows relatively poor performance on flow shop problems(car1-car7) compared to the job shop problem this can be attributed to the fact that because of the nature of a flow shop problem which leads to some agents having a larger action set(operations to choose from) than others. Machines/agents the process the initial operations of the jobs end up being the only ones playing at the beginning of the game. The lower the number of agent learning at each stage reduces the learning experience and also reduces the chances of achieving favorable solutions.

Comparison amongst the 3 games and the selected benchmark

The test shows that quality of solution of Random Games and Potential Games are affecting by the sizes of the problem instance. Quality reduces when dealing with large problem instances. This can be attributed to the fact increasing the size of instance significantly increases the size of search space. Since this games are based on learning a subset of the search space based on

the strategies selected and searching for a solution within that subset, the large the search space the harder it is to get a quality subset.

Only the Random Token Game doesn't show better adaptation to change in instance problem size. This can be attributed to the fact that it works by selecting solutions from the workspace at random and refining them, thus not greatly affected by the size of the search space.

Our algorithms have shown relatively good performance compared to the selected benchmark problems. On average we achieved better or equal performance across all problem instances.

5.7. CONCLUSIONS

This paper deals with defining 3 game theoretic algorithms for solving job shop scheduling problems. Our algorithms have shown relatively good performance on the benchmark data and we were able to converge to a feasible solution in relatively good time. We have also been able to demonstrate through visualization that by defining the job shop problem as a multi-agent system we are able to provide algorithms that provide good solution across different sizes of problem instances.

5.8. RECCOMENDED FURTHER WORK

The following are our recommendation for further work;

- This paper has dealt with job shop scheduling where scheduling is static and job are scheduled as a batch. In the real world problems tend to be further research work can be done to the algorithms to apply the two dynamic job shop scheduling.
- Our study choose a basic where of structuring the reward/reinforcement function based on total processing time of un-scheduled jobs at any given point. Further work can be done to refine the algorithm by defining better reward structure to improve the learning of an agent.

CHAPTER SEVEN: REFERENCES

1. A. AitZai and M. Boudhar, (2013). “Parallel branch-and-bound and parallel PSO for the job shop sche duling with blocking”, Int. J. Operational Research, vol. 16, No. 1.
2. Adams J, Balas E, Zawack D. (1988).The shifting bottleneck procedure for job shop scheduling. Management Science, 34(3): 391-401.
3. Anant Singh Jain and Sheik Meeran, (1998). A State-Of-The-Art Review Of Job-Shop Scheduling Techniques Department of Applied Physics, Electronic and Mechanical Engineering University of Dundee, Dundee, Scotland, UK.
4. Balas E, Lenstra J K, Vazacopoulos A. (1995).The one machine problem with delayed precedence constraints and its use in job shop scheduling. Management Science, 41(1):94{109.
5. Beasley J (2005) Or-library http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.
6. Blum, C.; Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison 35 (3). ACM Computing Surveys. pp. 268–308.
7. CAC Coello et al, (2003), “Use of an Artificial Immune System for Job Shop Scheduling” .Department of electrical engineering. National Polytechnic Institute, Mexico.
8. Chu, C., Portmann, M. C. and Proth, J. M. (1992) A Splitting-Up Approach to Simplify Job-Shop Scheduling Problems, International Journal of Production Research 30(4), 859-870.
9. Demirkol E, Mehta S V, Uzsoy R. A computational study of shifting bottleneck procedures for shop scheduling problems. Journal of Heuristics, 1997, 3(2): 111{137.
10. Elisha T. O. Opiyo, Erick Ayienga, Katherine Getao, William Okello-Odongo, Bernard Manderick, and Ann Nowé. Game Theoretic Multi-Agent Systems Scheduler for Parallel Machines. International Journal of Computing and ICT Research, Special Issue Vol. 1, No. 1, pp. 21-27
11. Fox, M. S.; and Sadeh, N. (1990). Why is Scheduling difficult? A CSP Perspective. In *9th European Conferenceon Artificial Intelligence*.
12. Fudenberg, Drew; Tirole, Jean (1991), Game theory, MIT Press, ISBN 978-0-262-06141-4. Acclaimed reference text.

13. G WeiB. (2009) Learning to Coordinate Actions in multi-agent Systems, Institut fur lnformatik, Technische Universitat Miinchen Arcisstr. 21, 8000 Miinchen 2, Germany
14. G Weiss (2013). A Modern Approach to Distributed Modern Approach to Artificial Intelligence, Multiagent Systems , MIT press , Cambridge , Massachusetts, USA.
15. Garey, M. R. and Johnson, D. S. (1979) Computers and Intertractability: A Guide to the Theory of NPCompleteness, W. H. Freeman, San Francisco.
16. Glover, F., (1986), "Future Paths for Integer Programming and Links to Artificial Intelligence", Journal of Computer and Operations Research, Vol. 13, No. 5, pp.533-549.
17. Gokhale NA, Zaremba A, Shears SB. Receptor-dependent compartmentalization of PPIP5K1, a kinase with a cryptic polyphosphoinositide binding domain. Biochem J. 2011;434:415–426
18. H. Chen, P.B. Luh, (2003). European Journal of Operational Research 149 (2003) 499–512-2003
19. Helga Ingimundardottir, Thomas Philip Runarsson (2010).Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling, School of Engineering and Natural Sciences, University of Iceland
20. J.F. Muth and G.L. Thompson. (1963) Industrial Scheduling. Prentice-Hall, Englewood Cliffs, N.J.
21. Jackson, J. R. (1955). Scheduling a Production Line to Minimise Maximum Tardiness, Research Report 43, Management Science Research Projects, University of California, Los Angeles, USA.
22. Jens Clausen, (1999). Branch and Bound Algorithms -Principles and Examples. Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark.
23. Johnson, S. M. (1954). Optimal Two- and Three-Stage Production Schedules with Set-Up Times Included, Naval Research Logistics Quarterly, vol 1, 61-68.
24. Karin Thörnblad(2013), Mathematical Optimization in Flexible Job Shop Scheduling: Modelling, Analysis, and Case Studies, Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg.

25. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., (1983), “Optimization by Simulated Annealing”, Journal of Science, Vol. 220, No. 4598, pp. 671 - 680.
26. M Azizoglu and O Kirca , (1999). On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 113 (1), 91-100-1999.
27. M. Gourgand et al, 2003. A contribution to the stochastic flow shop scheduling problem, European Journal of Operational Research 151 (2003) 415–43
28. Madureira, A., Ramos, C., and Silva, S.C. , (2001). “A Genetic Approach for Dynamic Job-Shop Scheduling Problems”, 4th Metaheuristics International Conference 2001, Porto, Portugal, July 16-20.
29. Manne, A. S. (1960) On the Job-Shop Scheduling Problem, Operations Research, vol 8, 219-223.
30. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E., (1953), “Equation of State Calculations by Fast Computing Machines”, The Journal of Chemical Physics, Vol. 21, Issue 6, pp. 1087-1092.
31. Metta, Haritha,(2008), "ADAPTIVE, MULTI-OBJECTIVE JOB SHOP SCHEDULING USING GENETIC ALGORITHMS" . University of Kentucky Master's Theses.Paper 518.
32. Nikos Vlassis(2005). A Concise Introduction to Multiagent Systems and Distributed AI, Intelligent autonomous Systems Informatics Institute University of Amsterdam.
33. Osborne, M. J. and Rubinstein, A. (1994). A Course in Game Theory. MIT Press.
34. P. Brucker and O. Thiele. (1996). A branch and bound method for the general shop problem with sequence-dependent setup times. Operations Research Spektrum, 18:145-161.
35. P. Brucker, P. Jurisch, and B. Sievers. (1994). A fast branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics, 49:107-127.
36. P. Lacomme, N. Tchernev and M.J. Huguet. (2012). Job-Shop with Generic Time-Lags: A Heuristic Based Approach”. 9th International Conference of Modeling, Optimization and Simulation - MOSIM’12 June 06-08, 2012 – Bordeaux - France
37. Panwalkar, S., Iskander, (1977). A Survey of Scheduling Rules. Operations Research 25(1) 45–61

38. Pinedo M, Singer M.(1995) A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. Naval Research Logistics, 46(1): 1-17.
39. R. Sutton and A. Barto. Reinforcement Learning. An Introduction. MIT Press/A Bradford Book, Cambridge, USA, 1998.
40. S. Russell and P. Norvig(2003). Artificial Intelligence { A Modern Approach. Prentice Hall, Englewood Cliffs, USA.
41. SANDHOLM W. H. (2001). Potential Games with Continuous Player Sets. In Journal of Economic Theory, Volume 97, pp. 81 – 108, 2000.
42. Takeshi Yamada and Ryohei Nakano. (1997)Chapter 7: Job-shop scheduling (pp. 134–160). Genetic algorithms in engineering systems. IEE control engineering series 55. The Institution of Electrical Engineers.
43. Thomas Gabel. (2009).Multi-Agent Reinforcement Learning, Approaches for Distributed,Job-Shop Scheduling Problems, Tag der wissenschaftlichen Aussprache: 26.06.
44. Torsten Hildebrandt , Jens Heger, Bernd Scholz-Reiter, (2010). Bremen Institute of Production and Logistics – BIBA at the University of Bremen Hochschulring 2028359 Bremen, Germany.
45. Trond Grenegar, Rob Powers, Yoav Shoham. (2002) Dispersion Games: General Definitions and Some Specific Results. In proc. AAAI02, pages 398-403. AAAI Press.
46. Uzsoy R, Wang C S. Performance of decomposition procedures for job shop scheduling problems with bottleneck machines. International Journal of Production Research, 2000, 38(6): 1271-1286.
47. V Lesser(1995), Mullti-agent. Systems: An Emerging Sub-discipline of AI. ACM Computing Surveys, Vol 27, No 3, September 1995
48. Van De Velde, S. (1991) Machine Scheduling and Lagrangian Relaxation, Ph. D. Thesis, CWI Amsterdam, The Netherlands.
49. von Neumann, J., Morgenstern, O., Theory of Games and Economic Behaviour, Princeton University Press, 1944.
50. Wagner, H.M .(1959) "An integer linear-programming model for machine scheduling." Nav. Res. Logist. Quart. (6) 131-140.

51. Y Mati and X Xie “Multi-resource shop scheduling with resource flexibility and blocking.” (2011) ,IEEE transactions on automation science and engineering.
52. Yailen Martínez Jiménez. (2012). A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems. Brussels University Press
53. Zhang, C.Y., P. Li, Y. Rao, Z. Guan. 2008. A very fast TS/SA algorithm for the job shop scheduling problem. Computers and Operations Research 35 282–294.

APPENDIX ONE: GLOSSARY

Machine Time Share	Processing time slot on a machine
Token	Indicator of the agent that's making a choice
Algorithmic deadlock	When to process are blocking each other from proceeding.
Processing end time	The time an operation will complete processing
Processing time	The processing load of an operation on a machine
Processing start time	The time when an operation will start processing
Possible start time	A probable time when processing of an operation will start.
Bottleneck	An operation that cause delays in a schedule while laying idle
Makespan	Time it takes to complete processing all the jobs in a problem
Next available start time (NAST).	The next estimated time when a machine will be available to process another job.
Waiting Queue,	A queue on a machine that holds the operations that are waiting to be scheduled for processing
Schedule queue	A queue on a machine that holds the operations that have been scheduled for processing
MDP	Markov Decision Process
Q-learning	Reinforcement learning technique based on learning an action-value function that gives the expected utility of taking a given action in a given state.
Q-pair	Pair defining the utility of a decision.
SPT	Shortest Processing Time
LPT	Longest Processing Time
FIFO	First in First Out
LIFO	Last In Last Out
Action set	Set of available action for an agent
Flow shop Scheduling problem	A variation of job shop scheduling problem where all the operation on the jobs follow the same processing sequence.