IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 9, Issue 1 (Jan. - Feb. 2013), PP 73-80
www.iosrjournals.org

Evaluation on a Simulated Annealing based Model for Solving Job Shop Scheduling Problem

S. Jayasankari [1], Dr. A. Tamilarasi [2], Dr. G. M. Nasira [3]
[1] *Assistant Professor, Department of MCA, Vivekanandha Institute of Information and Management Studies, Tiruchengode, India*
[2] *Professor and Head, Department of MCA, Kongu Engineering college, Perundurai, Erode, India*
[3] *Assistant Professor, Department of Computer Science, Chikkanna Govt. Arts College, Tirupur – 2, India*

Abstract: *The stochastic optimization techniques were commonly used in solving several optimization problems. There are different types of these algorithms that were addressed in several previous works. In our previous work, we implemented a GA and PSO based stochastic optimization model for solving the job shop scheduling problem (JSSP). The impacts of these two kinds of initial condition on the performance of these two algorithms were studied using the convergence curve and the achieved makespan. In this work we will implement a simulated annealing (SA) based model for solving JSSP. We will compare the performance of SA based method with the two other methods which we presented in our earlier works.*
Keywords: *Jop Shop Scheduling, JSSP, GA, PSO, Gant-Chart, Simulated Annealing.*

I. INTRODUCTION

In the modern competitive environment in manufacturing and service industries, the effective sequencing and scheduling has become an essential for survival in the marketplace. Companies have to produce their product untimely as opposed to due date. Otherwise, it will impinge upon reputation of a business. At the same time, the activities and operations need to be scheduled with the intention that the available resources will be used in an efficient manner. As a result, there is a great good scheduling algorithm and heuristics are invented. Most of the prevailing practical scheduling problems exist in stochastic and dynamic environment.

Stochastic is a problem where some of the variables are uncertain while dynamic problem is when jobs arrive randomly. On the other hand, the problems with ready time is known and fixed are called problems static and for problem where all the parameter such as processing times are known and fixed is called deterministic problems (French, 1982). In spite of this, it is quite impossible to predict exactly when jobs will become available for processing. Additionally, the understanding of scheduling problems where there is no uncertainty involved will help us towards the solution of stochastic and dynamic problems.

The main objective in solving the job shop scheduling problem is to find the sequence for each operation on each machine that optimizes the objective function. The most common objective function that has been used in scheduling the job shop problem is minimization of makespan value or the time to complete all jobs. It has been the principal criterion for academic research and is able to capture the fundamental computational difficulty which exists unconditionally in determining an optimal schedule (Jain and Meeran, 1999).

The Types of Related Scheduling Problems

We can group the main classical scheduling problems in five distinct classes:

- **Workshops with only one machine**: There is only one machine which must be used for scheduling the given jobs, under the specified constraints.
- **Flowshop :** There is more than one machine and each job must be processed on each of the machines - the number of operations for each job is equal with the number of machines, the j^{th} operation of each job being processed on machine j.
- **Jobshop :** The problem is formulated under the same terms as for the flowshop problem, having as specific difference the fact that each job has associated a processing order assigned for its operations.
- **Openshop :** The same similarity with the flowshop problem, the processing order for the operations being completely arbitrary - the order for processing a job's operations is not relevant; any ordering will do.
- **Mixed Workshop:** there is a subset of jobs for which a fixed processing path is specified, the other jobs being scheduled in order to minimize the objective function.

Problem Definition

The job shop scheduling problem (JSP) may be described as follows: Given n jobs, each composed of several operations that must be processed on m machines. Each operation uses one of the m machines for a

fixed duration. Each machine can process at most one operation at a time and once an operation initiates processing on a given machine it must complete processing on that machine without interruption. The operations of a given job have to be processed in a given order. The problem consists in finding a schedule of the operations on the machines, taking into account the precedence constraints, that minimizes the makespan (C_{max}), that is, the finish time of the last operation completed in the schedule.
We focus on job-shop scheduling problems composed of the following elements:

- Jobs. $J = \{J_1, \cdots, J_n\}$ is a set of n jobs to be scheduled. Each job J_i consists of a predetermined sequence of operations. $O_{i,j}$ is the operation j of J_i. All jobs are released at time 0.
- Machines. $M = \{M_1, \cdots, M_m\}$ is a set of m machines. Each machine can process only one operation at a time. And each operation can be processed without interruption during its performance on one of the set of machines. All machines are available at time 0.
- Constraints. The constraints are rules that limit the possible assignments of the operations. They can be divided mainly into following situations:

- Each operation can be processed by only one machine at a time (disjunctive constraint).
- Each operation, which has started, runs to completion (non-preemption condition).
- Each machine performs operations one after another (capacity constraint).
- Although there are no precedence constraints among operations of different jobs, the predetermined sequence of operation for each job forces each operation to be scheduled after all predecessor operations (precedence/conjunctive constraint).
- the machine constraints emphasize the operations can be processed only by the machine from the given set (resource constraint).

- Objective(s). Most of the research reported in the literature is focused on the single objective case of the problem, in which the objective is to find a schedule that has minimum time required to complete all operations (minimum makespan). Some other objectives, such as flow time or tardiness are also important like the makespan.

Different Approaches for Solving Scheduling Problems

Job shop scheduling is fundamentally a NP-hard problem with no easy solution. Branch-and-bound, Tabu search, GA, Swarm Intelligence and other stochastic model such as simulated annealing algorithm were proposed for achieving possible solutions to complex problems like job shop scheduling problems. During the last few periods, Evolutionary Computing (EC) has emerged as a powerful methodology for attempting the often highly complex problems of modern society, such as optimizing engineering design, job shop scheduling, and transport systems. Such real-world optimization problems typically are characterized by huge, ill-behaved solution spaces which are infeasible to exhaustively search and defy traditional optimization algorithms because they are for instance non-linear, non-differentiable, non-continuous, or non-convex. EC encompasses a class of stochastic, population-based, optimization algorithms inspired by biological evolution and genetics which have been shown to perform well on problems with huge, ill-behaved solution spaces.
Historically JSP has been primarily treated using the following approaches:

- **Exact methods:** Giffler and Thompson (1960), Brucker et al. (1994) and Williamson et al. (1997)
- **Branch and bound:** Lageweg et al. (1977), Carlier and Pinson (1989, 1990), Applegate and Cook (1991) and Sabuncuoglu and Bayiz (1999). Carlier and Pinson (1989) have been successful in solving the notorious 10´10 instance of Fisher and Thompson proposed in 1963 and only solved twenty years later; Heuristic procedures based on priority rules: French (1982), Gray and Hoesada (1991) and Gonçalves and Mendes (1994)
- **Shifting bottleneck:** Adams et al. (1988).

  Problems of dimension 15´15 are still considered to be beyond the reach of today's exact methods. Over the last decade, a growing number of metaheuristic procedures have been presented to solve hard optimization problems.

II. THE JSSP AND SA BASED MODEL CONSIDERED FOR SOLVING JSSP

Mathematical Representation of the JSSP

Let $J = \{0, 1, \ldots, n, n+1\}$ represent the set of operations to be scheduled and $M = \{1,..., m\}$ the set of machines. The operations 0 and n+1 are dummy, have no duration and represent the initial and final operations. The operations are interconnected by two kinds of constraints. First, the precedence constraints, which force each operation j to be scheduled after all predecessor operations, P_j, are completed. Second, operation j can only be scheduled if the machine it requires is idle. Further, let d_j denotes the (fixed) duration (processing time) of operation j.

Let F_j denotes the finish time of operation j. A schedule can be represented by a vector of finish times (F_1, , F_m, ... , F_{n+1}). Let A(t) be the set of operations being processed at time t, and let $r_{j,m} = 1$ if operation j requires machine m to be processed and $r_{j,m} = 0$ otherwise.

The conceptual model of the JSP can be described the following way:

Minimize F_{n+1} (C_{max}) …….(1)

Subject to:

$F_k \leq F_j - d_j$ j-1,…,n+2 ; k □ P_j …….(2)

$\sum_{jA(t)} r_{j,m} \leq 1$ m □ M ; t ≥ 0…….(3)

$F_j \geq 0$ j=1,….,n+1 …….(4)

The objective function (1) minimizes the finish time of operation n+1 (the last operation), and therefore minimizes the makespan. Constraints (2) levy the precedence relations between operations and constraints (3) state that one machine can only process one operation at a time. Finally (4) forces the finish times to be non- negative. The JSP is amongst the hardest combinatorial optimization problems. The JSP is NP- hard (Lenstra and Rinnooy Kan, 1979), and has also proven to be computationally challenging.

Simulated Annealing

Simulated annealing (SA) is a random-search technique which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system; it forms the basis of an optimization technique for combinatorial and other problems.

Simulated annealing was developed in 1980 to deal with highly nonlinear problems. SA approaches the global maximization problem similarly to using a bouncing ball that can bounce over mountains from valley to valley. It begins at a high temperature which enables the ball to make very high bounces, which enables it to bounce over any mountain to access any valley, given enough bounces. As the temperature declines the ball cannot bounce so high and it can also settle to become trapped in relatively small ranges of valleys. A generating distribution generates possible valleys or states to be explored. An acceptance distribution is also defined, which depends on the difference between the function value of the present generated valley to be explored and the last saved lowest valley. The acceptance distribution decides probabilistically whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on the temperature. It has been proved that by carefully controlling the rate of cooling of the temperature, SA can find the global optimum. However, this requires infinite time.

Simulated Annealing (SA) is moved by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis etc al in 1953 [Metropolis, 1953). The algorithm in this paper simulated the cooling of material in a heat bath. This is a process called annealing. If you heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled slowly, large crystals will be formed. However, if the liquid is cooled quickly (quenched), the crystals will contain imperfections. Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, frozen state. In 1982, Kirkpatrick et al (Kirkpatrick, 1983) took the idea of the Metropolis algorithm and applied it to optimization problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution.

Ever since its introduction, independently by Kirkpatrick, Gelatt and Vecchi, simulated annealing algorithm has been applied to many combinatorial optimization problems. The algorithm can be considered as a generalization of the well-known iterative improvement approach to combinatorial optimization problems, and it can be viewed as an analogue of an algorithm used in statistical physics for computer simulation of the annealing of a solid to the state with minimal energy. SA approach can be viewed as an enhanced version of local search or iterative improvement, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. SA randomizes this procedure in a way that allows occasional alterations that worsen the solution in an attempt to increase the probability of leaving a

local optimum. The application of SA as a local search algorithm assumes a cost function calculated for each possible solution, a neighbourhood comprising alternative solutions to a given solution and a mechanism for generating possible solutions.

Simulated annealing refers to the annealing process done on a computer by simulation. In this model, a parameter T, equivalent to temperature in annealing, is reduced slowly.

The law of thermodynamics state that at temperature, t, the probability of an increase in energy of magnitude, dE, is given by

$$P(\delta E) = \exp(-\delta E / kt) \qquad \ldots\ldots\ldots(5)$$

Where k is known as Boltzmann's constant.

The simulation in the Metropolis algorithm calculates the new energy of the system. If the energy has decreased, then the system moves to this state. If the energy has increased then the new state is accepted using the probability returned by the above formula. A certain number of iterations are carried out at each temperature and then the temperature is decreased. This is repeated until the system freezes into a steady state.

This equation is directly used in simulated annealing, although it is usual to drop the Boltzmann constant as this was only introduced into the equation to cope with different materials. Therefore, the probability of accepting a worse state is given by the equation

$$P = \exp(-c/t) > r \qquad \ldots\ldots\ldots(6)$$

Where

c = the change in the evaluation function
t = the current temperature
r = a random number between 0 and 1

The probability of accepting a worse move is a function of both the temperature of the system and of the change in the cost function. It can be appreciated that as the temperature of the system decreases the probability of accepting a worse move is decreased. This is the same as gradually moving to a frozen state in physical annealing. Also, that if the temperature is zero then only better moves will be accepted which effectively makes simulated annealing act like hill climbing.

The following algorithm is taken from (Russell, 1995), will be able to find similar algorithms.

Function SIMULATED-ANNEALING(*Problem*, *Schedule*) **returns** a solution state
Inputs :*Problem*, a problem
Schedule, a mapping from time to temperature
Local Variables :*Current*, a node
Next, a node
T, a "temperature" controlling the probability of downward steps
Current = MAKE-NODE(INITIAL-STATE[*Problem*])
For t = 1 **to** ∞ **do**
T = *Schedule*[t]
If Termination Condition **then**
return C*urrent*
Next = a randomly selected successor of *Current*
ΛE = fittness[*Next*] – fittness[*Current*]
if ΛE > 0 **then**
Current = *Next*
else if *(*exp(-ΛE/T) > probability **then**
Current = *Next*

The Representation of JSSP Solutions in Stochastic Models

The authors give the formal definition of string representation and then, in order to show that the string representation is a valid encoding for schedules, they formulated two most important theorems which are the foundation of this stochastic models.

Definition 1. String Representation.

Let us consider three finite sets, a set ***J*** of jobs, a set ***M*** of machines and a set ***O*** of operations. For each operation a there is a job ***j(a)*** in ***J*** to which it belongs, a machine ***m(a)*** in ***M*** on which it must be processed and a processing time ***d(a)***. Furthermore for each operation a its successor in

the job is given by ***sj(a)***, except for the last operation in a job. The representation of a solution is a string consisting of a permutation of all operations in **O**, i.e. an element of the set:

StrRep = { ***s*** belongs to O^n | $n = |O|$ and for all ***i, j*** with $1 \leq i < j \leq n$: $s(i) \neq s(j)$ } ……..(7)

Now we can define legal strings. Formal for ***s*** in ***StrRep***:

Legal(***s***) = For all a, ***sj(a)*** belongs to ***O***: a ~< ***sj(a)*** ……..(8)

Where ***a*** ~< ***b*** means: ***a*** occurs before ***b*** in the string **s**.

Theorem 1. (Feasible Solution Legal String)

Every feasible solution can be represented by a legal string. More than one legal string corresponding to the same feasible solution may exist.

Theorem 2. (Legal String Feasible Solution)

Every legal string corresponds exactly to one feasible solution. To explain the JSSP and valid/legal and invalid/illegal solutions, we have chosen the simplest 3x3 JSSP presented in this example of a 3x3 JSSP is given in Table 1. The data includes the routing of each job through each machine and the processing time for each operation in parentheses. For example, "2(3)" in the third row represents the operation one of the Job 3 and the 2 in "2(3)" represents that that operation should be scheduled to machine 2 and the operation will consume 3 units of time.

Table 1 : A 3x3 JSSP

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)

The one of the known optimum schedule of the above problem is [$J_{1,1}$, $J_{3,1}$, $J_{2,1}$, $J_{1,2}$, $J_{3,2}$, $J_{2,2}$, $J_{2,3}$, $J_{1,3}$, $J_{3,3}$].Here, for example, $J_{1,2}$ represents the operation 2 of the Job 1. Figure1shows one of such a optimum solution for the problem represented by "Gantt-Chart".

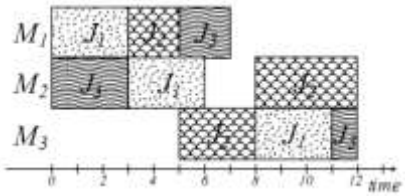

Figure 1 : The Gantt-Chart Representation of the Solution of the above 3x3 Problem

If we denotes the operations of the job as follows,

Job1: Op1, Op2, Op3

Job2: Op4, Op5, Op6

Job3: Op7, Op8, Op9

Or simply

1 2 3

4 5 6

7 8 9

then, the schedule [$J_{1,1}$, $J_{1,2}$, $J_{1,3}$, $J_{2,1}$, $J_{2,2}$, $J_{2,3}$, $J_{3,1}$, $J_{3,2}$, $J_{3,3}$] or simply [1, 2, 3, 4, 5, 6, 7, 8, 9] will be the one of the known worst case schedule which will satisfy all the conditions of the JSSP. But in this case, the makespan will not be optimum.

The schedule [$J_{1,1}$, $J_{3,1}$, $J_{2,1}$, $J_{1,2}$, $J_{3,2}$, $J_{2,2}$, $J_{2,3}$, $J_{1,3}$, $J_{3,3}$] or simply [1, 7, 4, 2, 8, 5, 6, 3, 9] will be the one of the best know optimum solution. Here the strings "1, 2, 3, 4, 5, 6, 7, 8, 9" and "1, 7, 4, 2, 8, 5, 6, 3, 9" represents solutions and known as valid strings.

In GA, a legal string or a illegal string (of numbers) which represent the order of the schedule can be represented by a chromosome. For example, the known worst case solution can be represented as a chromosome of GA by a string "1, 2, 3, 4, 5, 6, 7, 8, 9" . Similarly, the chromosome of GA "1, 7, 4, 2, 8, 5, 6, 3, 9" will represent a legal string which is an optimal solution of JSSP.

And for example, the chromosome "3, 9, 4, 2, 1, 5, 6, 7, 8" will be a invalid string which correspond to a illegal operation or schedule since this schedule will not satisfy the conditions of JSSP.

So, if we select the initial chromosomes of GA or initial points of PSO with random values, then there will be lot of invalid strings in the initial guess. The scope of the evolutionary algorithm is to permute the most optimal string to better most optimal string which will hopefully make that string as a legal string in

proceeding generations/steps and finally we will end up with a string belongs to a better solution or optimal schedule with minimum makespan.

This assumption will be good and can produce meaningful solutions for lower order scheduling problems such as 3x3 JSSP or 4x4 JSSP. But, it may produce illegal solutions even after very long runs in the case of higher order scheduling problems like 15x15 JSSP. Because, if we randomly chose initial population then there will be much chance for getting all illegal strings in the initial set which belongs to no nearby solution. So the fitness calculation methods will lead to meaningless fitness values and the selection method will also be incapable of selecting a better solution in each generation or step. So in each step of the evolutionary process there will not be any guaranty of getting progressive solution.

So we believe that the random selection of initial solution/seed in an evolutionary algorithm will not lead to a better result in higher order scheduling problems such as 15x15 JSSP. So in this work, we have evaluated the performance of two evolutionary optimization techniques GA and SA with different initial conditions.

III. RESULTS AND ANALYSIS

In the experiments experiment, we have given the known worst case solution as initial "seed" for the evolutionary process. We expect that, the system will be capable of producing at latest one meaningful legal string in every generation/step/iteration and hence there will be a much good probability of achieving a better solution in the succeeding generations or steps.

Analysis of Performance with Different Problem Size

The GA was run for 100 generations with population size of 100. The PSO was run for 100 steps with 100 particles. Refer our previous papers and for the further information about this methods. The Proposed SA was run for 3000 iterations since this simple SA will only handle one solution at a time (but in the case of GA and PSO, 100 solutions are evaluated at each generation or step).

The Analysis with 3x3 JSSP

In the following figures we are presenting the Gantt-Chart found by the SA based method for the 3x3 problem presented in table 1. In the following schedules, the forth one is solution already discussed in figure 1. The code developed for drawing Gantt-Chart will display the chart in color.

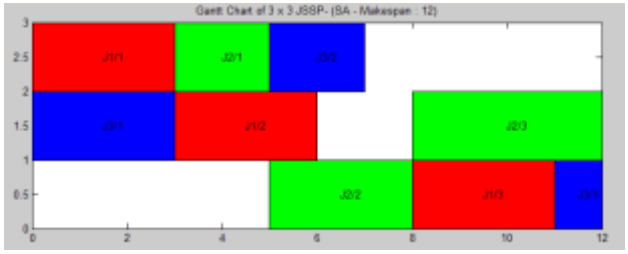

Figure 2 : The Gantt-Chart Representation of the Solutions found by SA based algorithm for the previously mentioned 3x3 Problem

The following figure shows the Gantt-Chart found by the SA based method for a 4x4 problem.

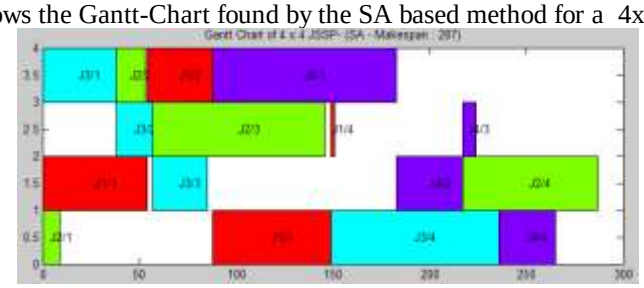

Figure 3 : The Gantt-Chart Representation of the Solutions found by SA based algorithm for a 4x4 Problem

The following figure shows the Gantt-Chart found by the SA based method for a 6x6 problem.

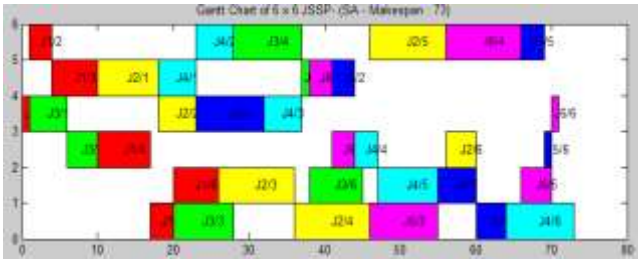

Figure 4 : The Gantt-Chart Representation of the Solutions found by SA based algorithm for a 6x6 Problem

Performance in terms of speed

To measure the performance in terms of speed, with problems of different sizes, the model was run with problems of different sizes.

Table 2 : The time taken for different JSSP size

Sl.No	JSSP Size	Time Taken(sec)		
		GA	PSO	SA
1	3x3	2.87	1.29	8.44
2	4x4	3.61	1.70	9.14
3	6x6	5.70	2.72	10.51
4	10x10	16.75	13.63	12.72
5	15x15	58.28	33.47	19.82

The following figure shows the performance in terms of time with respect to different JSSP problem size

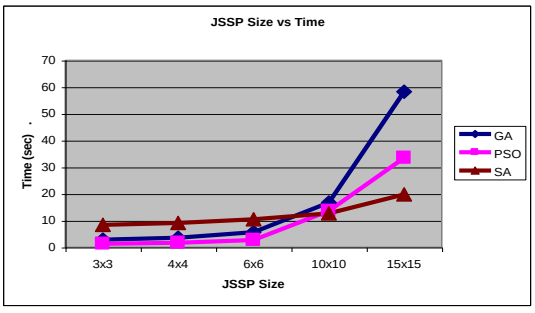

Figure 5 : JSSP Size vs Time Chart

Convergence Capability of the Algorithms

The convergences measured in terms of makespan for different size of the problem were tabulated below. We only considered the convergence upto the first 100 iterations in the case of GA and 100 steps in the case of PSO. SA was run for 3000 iterations. Even though the 3000 iterations seems to be high, SA only consider one solution at a time and search the best among 3000 solutions during its run.

Table 3 : Startup with known worst case solution

Sl.No	JSSP		Achieved Optimal Solution (Makespan)		
	Size	Known best optimum value	GA	PSO	SA
1	3x3	12	12	12	12
2	4x4	272	272	286	290
3	6x6	55	68	72	75
4	10x10	902	2214	2899	3258
5	15x15	1268	6771	8241	8871

The following figure shows the performance in terms of Makespan with respect to different JSSP problem size

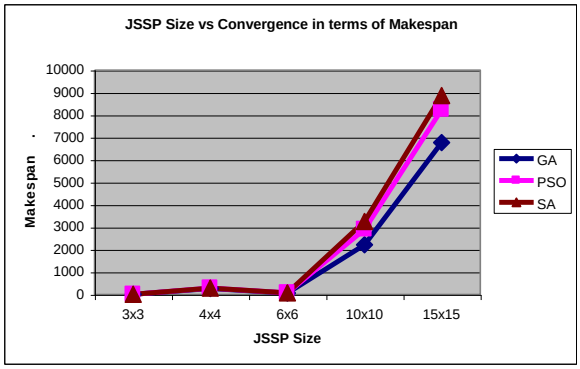

Figure 6 : JSSP Size vs Makespan Chart

Even though the arrived solution is far from the optimal solution (since we run this for low generations), the better performance in the case of GA is very obvious. Even though the SA is consuming much lesser time, it is not producing better results like GA. PSO also producing better results than SA.

IV. CONCLUSION AND FUTURE WORK

We have successfully implemented three basic evolutionary models for solving JSSP using SA, GA and PSO. The arrived results show that the models produced optimal or near-optimal solutions medium level job shop scheduling problems in a shot duration. While initializing with known, worst case solution, the evolutionary process was capable of converging into meaningful and more optimum solutions.

Further, as shown in the convergence cures in the previous section, The GA was behaved in a very better way than PSO and SA. Future works may address hybrid models for JSSP by combining the several aspects of different evolutionary algorithms. For example, we may combine SA and GA and design an improved GA. Future works may address the issued involved in designing this kind of hybrid models.

REFERENCES

[1] Moraglio , H.M.M. Ten Eikelder, R. Tadei, "Genetic Local Search for Job Shop Scheduling Problem" , Technical Report, CSM-435 ISSN 1744-8050

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys. - Sequencing and scheduling: Algorithms and complexity. - In: S.C. Graves, A.H.G. Rinnoy Kan and P. Zipkin, editors, Handbooks in Operations Research and Management Science 4, North-Holland, 1993.

[3] Dr. Daniel Tauritz, The abstract of the talk "Grand Challenges in Evolutionary Computing - Part II", Missouri S&T

[4] Hongbo Liu, Ajith Abraham,Zuwen Wang, "A Multi-swarm Approach to Multi-objective Flexible Job-shop Scheduling Problems", School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China, Fundamenta Informaticae,IOS Press, 2009

[5] José Fernando Gonçalves, Jorge José de Magalhães Mendes,Maurício G. C. Resende, "A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem", AT&T Labs Research Technical Report TD-5EAL6J, September 2002.

[6] Mahanim Binti Omar, "A Modified Multi-Step Crossover Fusion (Msxf) In Solving Some Deterministic Job Shop Scheduling Problem (Jssp), A thesis work submitted to Universiti Sains Malaysia, 2008

[7] Xiangyang Wang, Jie Yang, Richard Jensenb Xiaojun Liu, , "Rough Set Feature Selection and Rule Induction for Prediction of Malignancy Degree in Brain Glioma ",Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, Shanghai, China and Department of Computer Science, The University of Wales, Aberystwyth, UK

[8] Y. Shi, R. C. Eberhart, Parameter selection in particle swarm optimization, in Evolutionary Programming VII: Proc. EP98, pp. 591-600 (New York: Springer-Verlag, 1998).

[9] Takeshi Yamada and Ryohei Nakano, "Genetic Algorithms for Job-Shop Scheduling Problems", NTT Communication Science Labs, JAPAN, Proceedings of Modern Heuristic for Decision Support, pp.67, UNICOM seminar, March 1997, London

[10] S. Jayasankari, Dr. A. Tamilarasi "Evaluation on GA based Model for Solving JSSP", *International Journal of Computer Applications* Volume 43 - Number 7, Year of Publication: 2012, 10.5120/6113-8248.

[11] S. Jayasankari, Dr. A. Tamilarasi "Analysis of Two Stochastic Optimization Techniques for Solving Job Shop scheduling Problem", European Journal of Scientific Research, ISSN 1450-216X Vol. 88 No 3 October, 2012, pp.365-379.

[12] Bandyopadhyay S., Saha S., Maulik U. & Deb K. A Simulated Annealing-Based Multi-objective Optimization Algorithm: AMOSA, IEEE Transactions on Evolutionary Computation 2008;12(3) 269-283.