**­­Beam-ABC – hybridizing artificial bee colony with beam search for solving open shop-scheduling problem**

*A BACHELOR’S THESIS*

*submitted in partial fulfillment*

*of the requirements for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

**INFORMATION TECHNOLOGY**

**(B.Tech in IT)**

*Submitted by*

*Vikash Gupta (IIT2009088)*

*Under the Guidance of:*

**Dr. K.P. Singh**

Assistant Professor

IIIT-Allahabad



**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY**

**ALLAHABAD – 211 012 (INDIA)**

May 6, 2013

**CANDIDATE’S DECLARATION**

I hereby declare that the work presented in this thesis entitled “**Beam-ABC – hybridizing artificial bee colony with beam search for solving open shop scheduling problem**”, submitted in the partial fulfillment of the degree of Bachelor of Technology (B.Tech), in Information Technology at Indian Institute of Information Technology, Allahabad, is an authentic record of my original work carried out under the guidance of **Dr. K.P. Singh** due acknowledgements have been made in the text of the thesis to all other material used. This thesis work was done in full compliance with the requirements and constraints of the prescribed curriculum.

Place: Allahabad **Vikash Gupta**

Date: 6/May/2013 IIT2009088

**CERTIFICATE FROM SUPERVISOR**

I do hereby recommend that the thesis work prepared under my supervision by Vikash Gupta titled “Beam-ABC – hybridizing artificial bee colony with beam search for solving open shop scheduling problem” be accepted in the partial fulfillment of the requirements of the degree of Bachelor of Technology in Information Technology for Examination.

Date: 6/May/2013 **Dr. K.P. Singh**

Place: Allahabad Assistant Professor, IIITA

Committee for Evaluation of the Thesis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ACKNOWLEDGEMENTS**

I would like to thank **Dr. K.P. Singh** who has been a constant source of inspiration and guidance for me. I also owe a great deal of gratitude project evaluation committee members, for their valuable insights that help me view the project in the new light. Further I take this opportunity to express my gratitude to all my teachers and project guides during the course of my study at IIITA, to help me learn so much in a short span of four years at IIITA – learning memories, enough to last a lifetime.

Place: Allahabad **Vikash Gupta**

Date: 6/May/2013 B Tech Final Year, IIITA

**ABSTRACT**

Artificial Bee Colony (ABC) algorithm is one of the most recently introduced swarm-based algorithms. The ABC algorithm is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm. In this thesis we propose a hybridization of ABC algorithm with Beam Search (BS), which is well known heuristic tree search method. We call this approach Beam-ABC. The usefulness of Beam-ABC and ABC is demonstrated by their application to Open Shop Scheduling Problem (OSSP) with the objective of minimizing the makespan time (time required for all operations to complete their process). Finally, we compare the computation results of ABC and Beam-ABC on well-known benchmarks. Results show that the performance of the Beam-ABC is better than ABC algorithm.

**Keywords:** open shop scheduling, artificial bee colony algorithm, beam search, minimum makespan.

**TABLE OF CONTENTS**

1. **Introduction………………………………………………………………………………1**
   1. Currently existing technologies…………………………………………………..1
   2. Analysis of previous research in this area………………………………………...2
   3. Problem definition and objectives.………………………………………………..4
   4. Formulation of the present problem……………………………………………….4
   5. Organization of the thesis………………………………………………………....5
2. **Description of Hardware and Software Used………………………………………….6**
   1. Hardware………………………………………………………………………….6
   2. Software…………………………………………………………………………...6
3. **Theoretical Tools – Analysis and Development………………………………………..7**

3.1 Artificial bee colony algorithm……………………………………………………7

3.2 Open shop scheduling problem.............................................................................10

3.3 Beam search……………………………………………………………………...12

1. **Development of Software ………………………………………....................................13**
   1. Initialization………………………………………………………………………13
   2. Employed bees phase……………………………………………………………..15
   3. Onlooker bees phase……………………………………………………………...15
   4. Scout bees phase………………………….............................................................16
   5. Beam search phase………………………………………………………………..16
2. **Testing and Analysis…………………………………………………………………….18**
   1. Testing……………………………………………………………………………18
   2. Analysis…………………………………………………………………………..24
3. **Conclusions……………………………………………………………………………...28**
4. **Recommendations and Future Work……………………………………………….....28**

**Appendix - Explanation of the Source Code………………………………………......29**

**References…………………………………………………………………………….....36**

1. **Introduction**
   1. *Currently existing technologies*

The shop scheduling problem (SSP) can be simply introduced as a problem of redistribution of resources or a problem of rearrangement of operations order. SSP is mainly divided into following three types: Flow shop scheduling problem (FSSP), Job shop scheduling problem (JSSP), and Open shop scheduling problem (OSSP).

OSSP is more difficult than JSSP and FSSP because its operations have no predefined order. When number of machines is greater than or equal to 3, OOSP is proved as a NP-complete problem [14]. OSSP is a combinatorial optimization (CO) problem that can be describe in terms of a set of jobs, each with one or more operations. The operations of a job have to be processed in any sequence on a specific set of machines. The time required for all operations to complete their processes is called the makespan. The objective of OSSP to minimize the makespan value [3].

Many approaches using both mathematical formulations and heuristic method have been developed to solve this problem. In recent years, many heuristic rules for solving OSSP have been presented. One of the earliest heuristic rules is the branch and bound method, which regards the solution space as a tree with limited branches. Through the bound design of scientists such as lower bound (the lower bound of a solution), the searching range of the solution space can be effectively downsized. The advantage of branch and bound method is that the solution space is a clear tree structure, whose branches examined without omission to find a feasible solution. Its weakness is that the changes of feasible solutions are concentrated on certain branches, thus its feasible solution has an inherent lack of changeability.

Intelligence heuristic rules for various colonies have aroused widespread exploration and application in recent years, such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC) and their combinations. These heuristic rules integrate the techniques of random a number generator, parallel operations, probability rules, fitness functions, tabu serial, etc., and in the face of a problem with large solution space, attempt to solve problems, where a feasible solution lacks changeability, and to effectively exclude bad solutions more quickly, in order to come close to the optima or even obtain the optima.

Among the approximate methods for solving the CO problems we can identify two large groups: tree search methods and local search methods. The nature of tree search methods is constructive. The solution construction mechanism maps the search space to a tree structure, where path from root node to a leaf corresponds to the process of constructing a solution. Then, the search space is explored by repeated or parallel solution to solution constructions. In contrast, local search methods explore a search space by moving from solution to solution on a landscape that is imposed by a neighborhood structure on the search space. The simplest example is a deepest descent local search that moves at each step from the current solution to the best neighbor of the current solution. Most of the classical tree search methods have their origin in the fields of operations research (OR) or artificial intelligence (AI). Examples are greedy heuristic, backtracking methods, and beam search (BS) [11].

Goal of this project is to minimize the maximum completion time (makespan) of all jobs in OSSP using Beam-ABC algorithm.

Motivation

The shop scheduling problem has attracted many researchers’ attention in the past few decades, and many algorithms based on heuristic algorithms genetic algorithms, PSO, ACO and ABC algorithms have been presented to solve it. Many comparative studies showed that the performance of the ABC algorithm was competitive to other population based algorithm like PSO and ACO, with advantage of employing fewer control parameters in the continuous space. ABC algorithm is a newest algorithm in among all given population based algorithm yet. Hybridizing beam search with ABC is never done before.

* 1. *Analysis of previous research in this area*

The ABC algorithm is relatively new swarm intelligence based optimizer. It mimics the cooperative foraging behavior of a swarm of honey bees. ABC was initially proposed by Karaboga in 2005 [6] for optimizing multi-variable and multi-modal continuous functions. The latest research has revealed some good properties of ABC. Especially, the number of control parameters in ABC fewer than of other population based algorithms, which makes it easier to be implemented. Meanwhile the optimization performance of ABC is comparable and sometimes better to the other algorithm. Therefore ABC has aroused much interest and has been successfully applied to different kinds of optimization.

Maniezzo V. (1999) [11] propose an ant colony optimization (ACO) algorithm for the quadratic assignment problem (QAP) as an approximate non-deterministic tree search procedure. The results of this approach are compared to both exact algorithm and BS techniques.

I. Sabuncuoglu, M. Bayiz (1999) [15] propose a beam search method for solving the job shop scheduling problem with minimizing makespan time. The proposed algorithm is also compared with other well- known search methods and dispatching rules for a wide variety of problems and the results indicate that the beam search technique is a very competitive and promising tool which deserves further research in the scheduling literature.

Prins (2000) [14] suggested that, the characteristics of individuals in a GA colony were differed from each other, and there chromosomes could be rearranged to be applicable to the direction of a global optima from quasi optimal schedules, meaning a feasible solution with diversity, found through GA, could meet the needs of feasible solutions from large solution space.

Liaw (2000) [13] developed a HGA (hybrid genetic algorithm) that incorporate a local improvement procedure, based on Tabu Search (TS), for solving the open shop scheduling problem. The incorporation of the local improvement procedure enabled the algorithm to perform a genetic search over the subspace of the local optima, and then, TS performed the local improvement procedure, which costs heavy computing time on a computer, and therefore, it is only performed in several round of the local improvement procedure.

Since 2005, D. Karaboga and his research group have been studying the ABC algorithm and its applications to real world problems. Karaboga and Basturk [1] have investigated the performance of the ABC algorithm on unconstrained numerical optimization problems and its extended version for the constrained optimization problems and Karaboga et al. applied ABC algorithm to neural network training.

Blum (2005) [11] hybridize ant colony optimization (ACO) with a beam search to overcome difficult combinatorial optimization problems. This method caused the probability ACO mechanism to produce a group of complete solutions, and then used the beam search to perform improvement procedures of partial solutions. This method could randomly search the solution space and direct the solution to optimal branches.

Chong et al. (2006) [12] proposed a novel approach that used the honeybee foraging model to solve job shop-scheduling problems. This approach decided whether the branch from node A or node B was formed by the side length ratio between nodes A and B, as well as heuristic distance between nodes A and B. The branching probability was calculated according the average profitability of the previous round.

Sha and Hsu (2008) [10] modified the particle position representation using priorities, and the particle movement using an insert operator, and implemented a modified parameterized active schedule generation algorithm (mP-ASG) to decode a particle position in a schedule. In mP-ASG, the search area between non-delay schedules and active schedules could be reduced or increased by controlling the maximum delay time allowed.

Wong et al. (2008) [12] publish the Big Valley Landscape Exploitation (BVLE) method, which proposes to define a sole search space, called a landscape, when a heuristic search approach was applied to a combinatorial optimization problem. Exploring the search space with different numbers of search operations could create different landscapes, as the content of the landscape might change with the number of heuristic operations. The structures of these landscapes could help search for the global optima. In the BVLE structure, the local optima in some clusters may tend to appear near other local optima, and every cluster would form a valley centered on the global optimum. BVLE suggested that the new starting point for search should be based on the previous local optimum, other than random point search, because good candidate solutions can often be found near a local optimum.

Julia A. Bennell, Xiang Song (2010) [16] publish the beam search implementation for the irregular shape packing problem. They represent the problem as an ordered list of pieces to be packed where the order is decoded by a placement heuristic. They implemented beam search algorithm to search the packing order. Results for benchmark problems show that the algorithm generates highly competitive solutions in significantly less time.

In 2010, Hadidi et al. [7] employed an Artificial Bee Colony (ABC) Algorithm based approach for structural optimization. In 2011, Zhang et al. [8] employed the ABC for optimal multi-level thresholding, MR brain image classification, cluster analysis, face pose estimation, and 2D protein folding.

Ling Wang et al. (2012) [3] proposed a better approach to improve the effectiveness of artificial bee colony algorithm (ABC) to solve the flexible job-shop scheduling problem with the criterion to minimize the maximum completion time (makespan). They called this improved ABC algorithm as effective artificial bee colony algorithm.

* 1. *Problem definition and objectives*

An interesting example of a meta-heuristic that can be seen as a probabilistic tree search method is artificial bee colony (ABC) [1]. In ABC algorithms, artificial bees construct solution from scratch by probabilistically making a sequence of local decisions. At each construction step a bee chooses exactly one of possibly several solutions of extending the current partial solution. The rules that define the solution construction mechanism in ABC implicitly map the search space of the considered problem (including the partial solutions) onto a search tree. This view of ABC as a tree search procedure allows us to put ABC into relation with classical tree search methods such as beam search (BS) [11]. One of interesting features of BS is that it works on a set of partial solutions in parallel, extending each partial solution – in contrast to ABC – at each step in several possible ways. However, in BS the extension of partial solutions is usually done by using a deterministic greedy policy with respect to weighting function that gives the weights to the possible extensions. The idea of this project is to hybridize the solution construction mechanism of ABC with BS, which results in general approach that we call Beam-ABC. We apply Beam-ABC to open shop scheduling problem (OSSP). We show that Gantt chart representing the exact sequence on each machine, total makespan for input and Beam-ABC improves on the results obtained by the best standard ABC approach for OSSP.

Objective of this project is to minimize the maximum completion time (makespan) of all jobs in OSSP using Beam-ABC algorithm.

* 1. *Formulation of the present problem*

To describe the formulation of the problem, I introduce the following notations:

Symbol definition

Ji = job number I;

Mk = machine number k;

Oij = operation number j of Ji;

Problem parameters

n = number of jobs for processing at time zero;

m = number of machines in the shop;

pik = the processing time of Ji on Mk;

rik = 1 if Ji requires Mk; 0 otherwise;

Ni = number of operations of Ji, that is, Ni =;

Decision variables

Cmax = maximum completion time or makespan; Cmax = maxi=1 to n(Ci);

Gir = the starting time of the rth operation of Ji;

wirk = 1 if rth operation of Ji is scheduled on Mk; O otherwise;

The binary wirk specifies rth operation of Ji is processed on Mk. The following model describes the open shop scheduling problems.

Objective of the problem is minimize makespan Cmax (1)

wirk Mrik i = 1,2,…..,n; r = 1,2,….,Ni; k = 1,2,……,m (2)

wirk = rik i = 1,2,…..,n; k = 1,2,…….,m (3)

gir Mwirk  i = 1,2,…..,n; r = 1,2,…..,Ni; k = 1,2,……,m (4)

gir + (pik \*wirk) gi,s i = 1,2,…..,n; r = 1,2,…..,Ni-1; r = 1,2,…..,Ni-1 (5)

gi,Ni + (pik \*wi,Ni,k) Cmax i = 1,2,…….,n; (6)

In above model, constraint sets (2) and (3) describe the feasible value of wirk. Constraint set (4) enforces gir = 0 when wirk = 0. Constraint set (5) guarantees that the starting time of Oi,s is later than Oi,r. Constraint set (6) gives the definition of Cmax which is to be minimized in the objective function (1).

* 1. *Organization of the thesis*

The outline of the thesis is as follows:

Section 2 describes the hardware and software support required by the application. Specifically 2.1 deals with hardware and 2.2 deals with software support required.

Section 3 defines theoretical tools applied during analysis and development. These include fundamentals of the problem and algorithm which have been used in this project. In section 3.1 artificial bee colony (ABC) algorithms is defined and flowchart for ABC algorithm has shown. In section 3.2 open shop scheduling problem is defined with example. In section 3.3 beam search algorithm is defined.

Section 4 describes how this project has been developed. In section 4.1 initialization phase for developing the ABC algorithm is defined. In section 4.2 employed bees phase is defined for ABC algorithm. In section 4.3 onlooker bees phase is defined for ABC. In section 4.4 scout bees phase is defined for ABC algorithm. In section 4.5 beam search implementation is defined for improving the solution provided by ABC algorithm.

In section 5, I have described testing and analysis methods used in work. Section 5.1 deals with testing part and section 5.2 deals with the analysis part.

Section 6 includes conclusion derived from the work.

Section 7 includes recommendation and future work suggestions.

In Appendix, I have described the source code of the project.

At last references are listed.

1. **Description of Hardware and Software used**
   1. *Hardware*

I have built this application on Intel core 2duo processor system with 2GB RAM memory and 30GB storage memory. But this application can run on any system which supports Windows/Linux/Mac-OS/Solaris operating system. I have built on Windows 7.

* 1. *Software*

I have used Java programming language to build this application using IntelliJ IDEA IDE (Integrated development environment).

Java is general-purpose, concurrent, class-based, object oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers “write once, run anywhere” (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use. First time Java released by Sun Microsystem in 1995. The language derives much of its syntax from C and C++.

Java Development Kit (JDK) is used for running java code on the system. This is work as loader for Java application. This tool is an interpreter and can interpret the class files generated by the javac compiler. Javac is the Java compiler, which converts source code into java bytecode.

IntelliJIdea is a Java IDE by JetBrains, available as Apache 2 Licensed community edition and commercial edition. Community edition is free for all and I have used community edition for this application. The first version of IntelliJ IDEA was released in January 2001, and at the time was one of the first available Java IDE with advanced code navigation and code refactoring capabilities integrated.

1. **Theoretical Tools – Analysis and Development**

This project is divided into following phases:

* Understanding Artificial Bee Colony (ABC) algorithm.
* Understanding Beam Search algorithm.
* Understanding the Open Shop Scheduling Problem (OSSP).
* Understanding hybridizing ABC with Beam Search (Beam-ABC).
* Implementing ABC to solve the OSSP.
* Implementing Beam-ABC to solve the OSSP.
  1. *Artificial Bee Colony Algorithm* *[1][3]*

In the ABC algorithm, the artificial bees are classified into three groups: the employed bees, the onlookers and the scouts. A bee that is exploiting a food source is classified as employed. The employed bees share information with the onlooker bees, which are waiting in the hive and watching the dances of employed bees. The onlooker bees will then choose a food source with probability proportional to quality of that food source. Therefore, good food sources attract more bees than the bad ones. Scout bees search for new food sources randomly in the vicinity of the hive. When a scout or onlooker bee finds a food source, it becomes employed. When a food source has been full exploited, all the employed bees associated with it will abandon the position, and may become scouts again. Therefore, scouts bees perform the job of “exploration”, whereas employed and onlooker bees perform the job of “exploitation”.

In ABC, the first half of the colony consists of employed bees and the other half are onlookers. The number of employed bees is equal to the number of food sources (SN) because it is assumed there is only one employed bee for each food sources. The main procedure of ABC can be described as follows:

Step 1: Initialize the food sources.

Step 2: Each employed bee start to work on a food source.

Step 3: Each onlooker bee selects a food source according to the nectar information shared by the employed.

Step 4: Determine the scout bees, which will search for food sources in a random manner.

Step 5: Test whether the termination condition is met. If not, go back to Step 2.

The detailed description for each step is given below:

1. The initialization phase:The SN initial solutions are randomly generated D-dimensional real vectors. Let xi = {xi,1, xi,2, ….., xi,D} represent the i-th food sources, which is obtained by

*xi,d = xd(min) + r × ( xd(max) - xd(min) ),* d = 1,… , D (1)

where *r* is uniform random number in the range [0,1], and *xd(min)* and *xd(max)* are the lower and upper bounds for dimension d, respectively.

1. The employed bee phase: In this stage, each employed bee is associated with a solution. She exerts a random modification on the solution (original food source) for finding a new solution (new food source). This implements the function of neighborhood search. The new solution vi is generated from xi using a differential expression:

*vi,d = xi,d + r′ × ( xi,d - xk,d )* (2)

where *d* is randomly selected from {1, … , D}, *k* is randomly selected from {1, … , SN} such that *k != i* and r*′* is a uniform random number in the range [-1,1].

Once *vi* is obtained, it will be evaluated and compared to *xi.* If the fitness of *vi* is better than that of *xi* (i.e, the nectar amount of the new food source is higher than the old one), the bee will forget the old solution and memorize the new one. Otherwise, she will keep working on *xi.*

1. The onlooker bee phase: When all employed bees have finished their local search, they share the nectar information of their food source with the onlookers, each of whom will then select a food source in a probabilistic manner. The probability *pi* by which an onlooker bee chooses food source *xi* is calculated as follows:

*pi* = *fi* ***/***

where *fi* is the fitness value of *xi* . Obviously, the onlooker bees tend to choose the food sources with higher nectar amount.

Once the onlooker has selected a food source *xi* , she will also conduct a local search on *xi* according Eq.(2). As in the previous case, if the modified solution has a better fitness the new solution will replace *xi.*

1. The scout bee phase: In ABC, if the quality of a solution cannot be improved after a predetermined number *(limit)* of trials, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout will then produce a food source randomly by using Eq.(1).

Initial food positions

Calculate the nectar amount

Determine neighbors of the chosen food source by the employed bees

Calculate the nectar amount

Selection

All onlookers distributed?

N

Determine a neighbor of the chosen food source by the onlooker bee

Y

Memorize the positions of best food source

Find the abandoned food sources

Produce new positions for the abandoned food sources

Termination criteria met?

N

Y

Final food positions

Fig.1.Flow chart of the ABC Algorithm [2]

* 1. *Open shop scheduling problem [10][11]*

Shop scheduling problems (SSP) can be introduced simply as problems of redistribution of resources, or a problem of rearrangement of operation order. SSP is mainly divided into following three types:

1. Flow shop scheduling problem (FSSP),
2. Job shop scheduling problem (JSSP), and
3. Open shop scheduling problem (OSSP)

As far as difficulty for solving, OSSP is more difficult than JSSP and FSSP, due to the fact that it has no predefined order of operations in the same job.

Suppose the existing resources are m sets of machine, and jobs are j pieces.

1. Each job contains m operations.
2. Each operation must cost time pij.
3. The operations in same job are randomly ordered, but at a time, only an operation can handled.
4. No preemptive, which indicates that no operation can interrupt other operations.
5. At one time, a machine can handle only one operation and each job only be handled by one machine.

Objective of this problem is to minimize the makespan time of OSSP. Makespan time, that is, the time needed for completing all jobs.

Example: 5x5 benchmark OSSP [10]

Table 1: Job Processing Time Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Oper.  Job | Op1 | Op2 | Op3 | Op4 | Op5 |
| Job 1 | 85 | 64 | 31 | 44 | 66 |
| Job 2 | 7 | 14 | 69 | 18 | 68 |
| Job 3 | 1 | 74 | 70 | 90 | 60 |
| Job 4 | 45 | 76 | 13 | 98 | 54 |
| Job 5 | 80 | 15 | 45 | 91 | 10 |

Table 2: Job Machine Sequence Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Op1 | Op2 | Op3 | Op4 | Op5 |
| Job 1 | 4 | 1 | 3 | 5 | 2 |
| Job 2 | 1 | 4 | 2 | 5 | 3 |
| Job 3 | 4 | 1 | 2 | 5 | 3 |
| Job 4 | 2 | 4 | 5 | 3 | 1 |
| Job 5 | 1 | 4 | 2 | 5 | 3 |

Above tables shows a standard 5x5 benchmark OSSP, that is, 5 jobs and 5 machines taken. In the above tables, task 1 of job 1 must go to machine 4 (from table 2) for 85 units of time (from table 1), task 2 of job 1 must go to machine 1 (from table 2) for 64 units of time (from table 1), and so on, with no restrictions on the order in which the tasks for any job are to be processed. The problem is generated a valid schedule with minimal makespan. Below figure shows a minimum-makespan (300) schedule for the benchmark in above table.

Machine

J5-O1

J3-O2

J1-O2

J4-O5

J2-O1

1

2

J1-O5

J3-O3

J5-O3

J2-O3

J4-O1

3

J3-O5

J2-O5

J5-O5

J4-O4

J1-O3

4

J1-O1

J4-O2

J5-O2

J2-O2

J3-O1

5

J3-O4

J1-O4

J5-O4

J2-O4

J4-O3

Time

300

250

200

150

100

50

0

Fig.2. Gantt Chart for 5x5 benchmark for OSSP [10]

In above figure Ji-Oj mean jth operation of ith job.

where, i = 1,2,….n and j = 1,2,….m.

where, n = no. of jobs, and

m = no. of machines.

* 1. *Beam search [11]*

Beam Search (BS) is a classical tree search method that was introduces in the context of scheduling, but has since then been successfully applied to many other CO problems. BS algorithms are incomplete derivatives of branch and bound algorithms, and are therefore approximate methods.

BS is like breadth-first search since it progresses level by level without backtracking. But unlike breadth-first search, beam search only moves downward from the best Kbw promising nodes (instead of all nodes) at each level and Kbw is called ‘beam width’. The other nodes are simply ignored. In order to select the best B nodes, promise of each node is determined. This value can be determined in various ways. One way is to employ an evaluation function which estimates the minimum total costs of the best solution that can be obtained from the partial schedule represented by the node.

Beamwidth = 2

Filterwidth = 2

Nodes pruned by local evaluation

Beam Nodes

Nodes selected for global evaluation

Fig.3. Representation of beam search tree

The central idea behind BS is to allow the extension of partial solution in several possible ways. At each step the algorithm extends each partial solution from a set B, which is called the beam, in at most kext possible ways. Each newly obtained partial solution is either stored in the set of complete solution Bc (in case it is a complete solution), or in the set Bext (in case it is a further extensible partial solution). At the end of each step, the algorithm creates a new beam B by selecting up to kbw (called the beam width) solutions from the set of further extensible partial solution Bext. In order to select partial solutions from Bext, BS algorithm use a mechanism to evaluate a partial solutions. An example of such a mechanism is lower bound. Given a partial solution sp, a lower bound computes the minimum objective function value for any complete solution s that can be constructed starting from sp.

Steps for Beam Search:

Step 1: Initialize set of partial solution (B) with sp and set of complete solutions (Bc) with Null.

Step 2: Initialize Bext with Null.

Step 3: For each sp of B we update a set Bext and set Bc based on fitness value of sp.

Step 4: Rank the partial solutions in Bext using lower bound (LB).

Step 5: Update a set B by selecting the min(kbw,|Bext|) highest ranked partial solution from Bext.

Step 6: Repeat step 2 -5 until B is Null.

Step 7: Output is a set of candidate solutions Bc.

Input

* No. of Jobs and Machines
* Processing time matrix in which the processing time for each job on each machine is stored in matrix form. It is a NxM matrix where N is the number of jobs and M is the number of machines.
* Job sequence matrix is table which stores the required sequence of each job in the problem.

Output

The output is expressed as a Gantt chart (Fig.) representing the exact sequence on each machine and the total makespan for input.

1. **Development of Software**

Following steps are used to development of software:

* 1. *Initialization [9]*
* The Initial parameters such as number of employed bees, onlooker bees and maximum cycle number are set.
* Next, the job’s processing time on each machine and the job’s machine sequence will be given at this step.
* In our solution representation, a solution in OSSP is an operation scheduling list, which is represented as a food source(x) in our ABC algorithm.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J1 | J2 | J2 | J3 | J3 | J2 | J1 | J1 | J3 |

Fig.4. Operation Scheduling List for JSSP

This (Fig.4.) operation scheduling list is for 3 Jobs and 3 Machines and it is interpreted as following way:

Interpreting from left to right J1 occurred first time it says 1st operation of J1 will be scheduled on machine, J2 occurred first time it says 1st operation of J2 will be scheduled on machine, J2 occurred second time it says 2nd operation of J2 will be scheduled on machine, and so on.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 3 | 5 | 6 | 8 | 9 | 1 | 4 |

Fig.5. Operation Scheduling List for OSSP

In OSSP sequence of operations of jobs does not matter each operation can schedule on machine independently while in JSSP sequences are fixed. To fulfill this condition of OSSP I have taken operation scheduling list (Fig. 5) as number from 1 to d (d = no. of job x no. of machine). Above operation scheduling list is for 3 Jobs and 3 Machines and it is interpreted as following way:

Interpreting from left to right, First schedule operation that has number 7,

Job number = + 1

where, noj = (no. of job)

Operation number = Reminder() + 1

and like this we can calculate job number and operation number for others schedule operations.

* Each dimension in a food source represents one operation of a job.
* The fitness of each food source f(x) is determined by the inverse of its makespan value (Cmax(x)) which is calculated during the selection of feasible solutions.

Operation scheduling list (d)

a0 a1 . . . . . . . . . . . . . . . . . . an

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X0 |  |  |  |  |  | f(x0) = 1/Cmax(x0)  f(x1) = 1/Cmax(x1)  f(xn-1) = 1/Cmax(xn-1) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1  Food Source(n) |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Xn-1 |  |  |  |  |  |  |

Fig.6.Mapping between the food sources and operation scheduling [9]

* 1. *Employed bees phase [9]*
* The candidate food sources are updated by employed bees.
* Position based crossover (PBX) is the mechanism used to update employed bee’s old food sources.
* In this method, a set of job operations from the old food source is selected randomly.
* Each dimension in the operation scheduling list of the old food source is selected to produce the new position with the probability greater than 0.5. This probability value (0.5) has been chosen based on observations.
* The job operations on the neighboring food source will be selected and placed into empty positions from left to right of the operation scheduling list in the new food source.
* The old food source in the employed bee’s memory will be replaced by the new candidate food source if the new food position has a better fitness value.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **J1** | **J3** | **J2** | **J2** | **J3** | **J2** | **J1** | **J1** | **J3** |

**The old food source (xij)**

**The new food source (vij)**

**The neighboring food source (xkj)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **J1** | **J3** | **J2** | **J2** | **J1** | **J3** | **J1** | **J2** | **J3** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **J2** | **J1** | **J3** | **J3** | **J1** | **J3** | **J1** | **J2** | **J3** |

Fig**.**7**.** Exchanging information with a neighboring food source based on PBX method [9]

* 1. *Onlooker bee phase [9]*
* The employed bees share new solutions that they have found with the onlooker bees, who then select these solutions based on probability (Pi)calculated from an equation below

Pi  =

where f(Vi) is the fitness value of the food source I and SN is the number of food sources.

* Next, we generate a random probability (p) and compare it with Pi , if Pi is greater than p then we assign ith food source to onlooker bee .
* The PBX method is also used to update the old solution of the onlooker bees to the new solution based on the neighboring food sources.
* The old food source in the onlooker bee’s memory will be replaced by the new food source if the new food source has better fitness value.
* Above three steps will continue till each onlooker bee gets a food source.
* After completion of the above steps we memories best food source.
  1. *Scout bees phase [9]*
* Now, those food sources that have not been updated for some given number of iterations (*limit*) then employed bees abandon those food sources and these employed bees become scout bees.
* The scout bees ignore the old solution and randomly search for new solution by using random food generation.
  1. *Beam search phase [15]*
* Let PSt be a partial schedule containing t scheduled operations, St be the set of schedulable operations at stage t.
* Save best solution in St and fix beam-width (b). For each operations j ɛ St that requires machine mk, generate a new node which corresponds to the partial schedule in which operation j is added to PSt.
* Compute global evaluation function values for all the nodes and select the best beam-width (b) number of nodes with lowest global evaluation value

Global evaluation value =

where, Pij – processing time of jth operation of ith job.

and, Pkj – processing time of jth job on kth machine.

* In the next level, generate new nodes from the beam node as we discovered previously and calculate global evaluation function values for all generated node and again select b number of nodes.
* For the partial schedule represented by the beam node update the data set as follows:

1. Remove operation j from St.
2. From St+1 by adding the direct successor of operation j to St.
3. Increment t by one.

* Among the beam-width (b) number of schedules, select the one best objective function value. Compare this value with best solution which saved before beam search phase. If this schedule better fitness value than best schedule than replace best this schedule.
* Otherwise update best schedule by applying PBX method (section 4.2) on best schedule and this schedule.
* Schedule generated by PBX method will be used to update the best solution.

Initialize Food sources

Calculate fitness value of all food sources

Memorize best source

Send employee bees to update their food source by any neighbor source using PBX method

Employee bees share information about food sources to onlooker bees

Onlooker bees calculate fitness value of all food sources and they will select best food sources

Memorize best food source

Scout bees will check for abandoned food sources and search random food source corresponding to abandoned food source

Apply beam search on best food source to improve the solution

No

Yes

Is termination criteria met?

Solution of Instance

Fig.8. Flowchart for the implementation of the problem solution

1. **Testing and Analysis**
   1. *Testing*

I have tested various OSSP benchmarks by Beam-ABC algorithm and ABC algorithm. I have taken well known benchmarks provided by Taillard [17].

The Gantt chart drawn as the output is displayed for different input combination. Gantt chart represents the exact sequence on each machine of operations and the total makespan time for input.

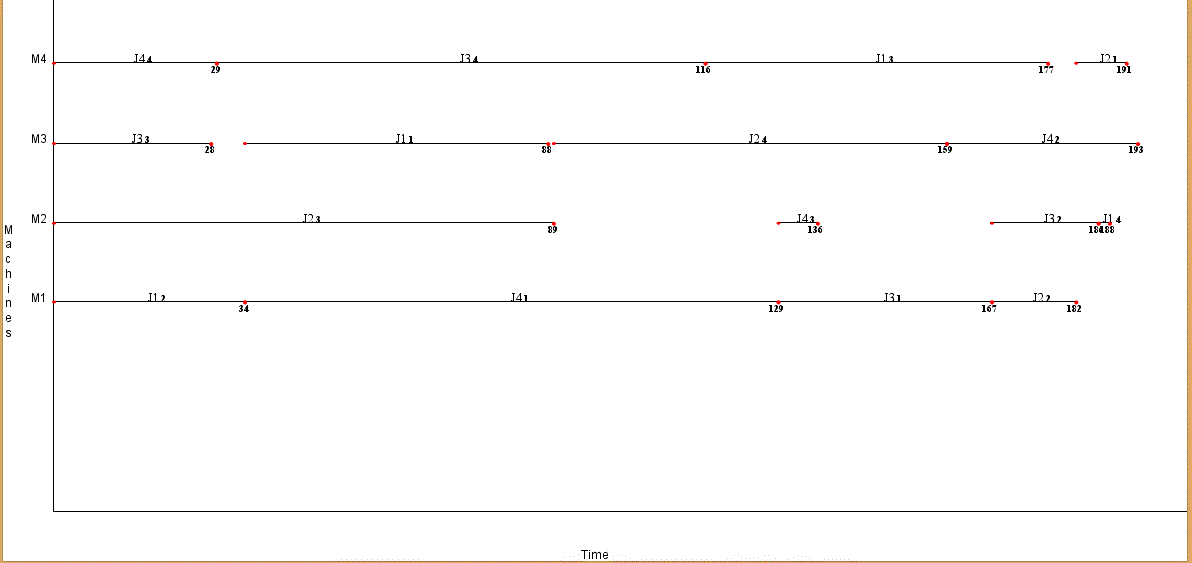
****

Fig.9. Gantt chart for tai4x4\_1 instance

Mi – Shows ith machine on Machines axis (Ordinate)

Jij – means jth operation of Job 3.

te – means end time of Jij on particular machine

te

Jij

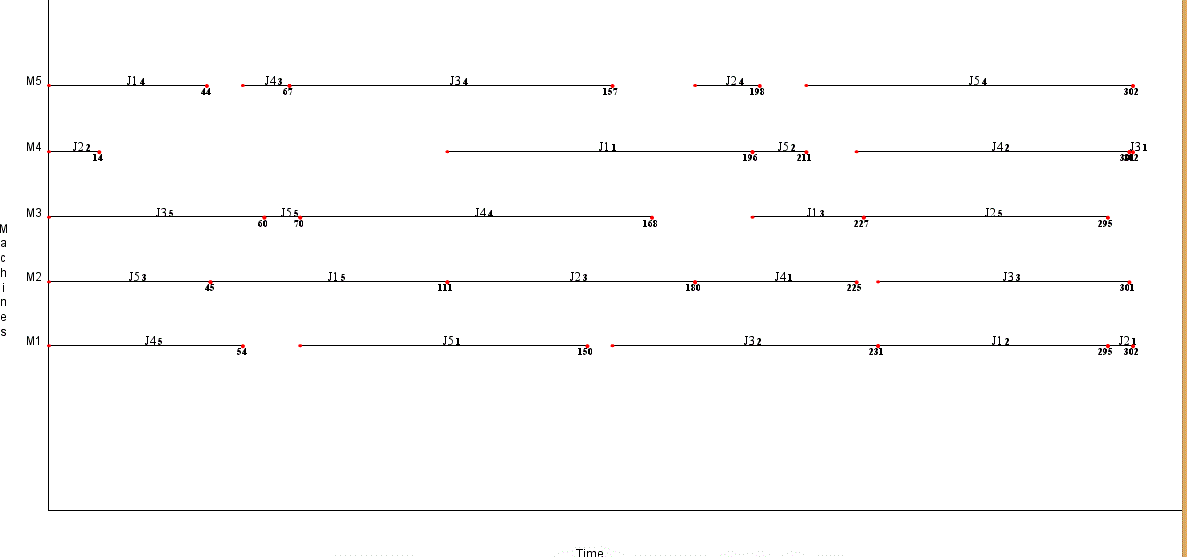


Fig.10. Gantt chart for tai5x5\_1 instance

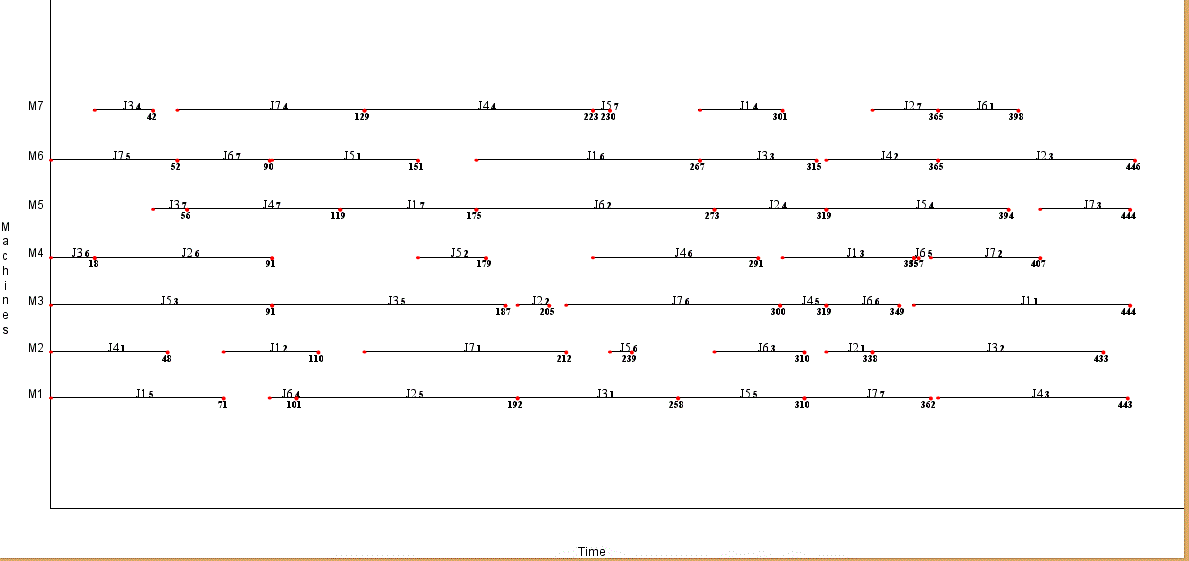


Fig.11. Gantt chart for tai7x7\_1 instance

Table 3: Result of ABC and Beam-ABC for solving the various OSSP benchmark provided by Taillard [17]

|  |  |  |  |
| --- | --- | --- | --- |
| Instance | Best Known | ABC  Best Average S.D RPE | Beam-ABC  Best Average S.D. RPE |
| tai4x4\_1 | 193 | 193 197.80 5.96 0.00 | 193 195.46 1.03 0.00 |
| tai4x4\_2 | 236 | 236 242.46 4.85 0.00 | 236 240.13 2.13 0.00 |
| tai4x4\_3 | 271 | 272 274.46 3.92 0.30 | 271 273.23 2.03 0.00 |
| tai4x4\_4 | 251 | 252 254.89 2.47 0.40 | 251 254.41 2.31 0.00 |
| tai4x4\_5 | 295 | 297 302.93 2.74 0.70 | 295 300.46 3.09 0.00 |
| tai4x4\_6 | 189 | 189 195.55 4.51 0.00 | 189 193.66 4.42 0.00 |
| tai4x4\_7 | 201 | 203 204.46 2.03 0.50 | 201 204.40 1.74 0.00 |
| tai4x4\_8 | 217 | 217 222.33 3.05 0.00 | 217 220.86 2.30 0.00 |
| tai4x4\_9 | 261 | 261 270.13 5.70 0.00 | 261 268.33 4.81 0.00 |
| tai4x4\_10 | 217 | 217 224.53 3.45 0.00 | 217 223.13 3.32 0.00 |
| tai5x5\_1 | 300 | 301 318.86 7.92 0.30 | 300 314.46 7.89 0.00 |
| tai5x5\_2 | 262 | 263 278.73 9.17 0.40 | 262 276.00 6.55 0.00 |
| tai5x5\_3 | 328 | 333 348.07 9.58 1.15 | 331 350.66 5.81 0.91 |
| tai5x5\_4 | 310 | 325 337.47 10.24 4.80 | 318 333.16 7.20 2.58 |
| tai5x5\_5 | 329 | 336 355.26 10.03 2.10 | 335 354.20 7.82 1.82 |
| tai5x5\_6 | 312 | 322 334.40 8.14 3.20 | 314 335.40 6.61 0.64 |
| tai5x5\_7 | 305 | 318 329.86 8.95 4.20 | 312 324.73 6.75 2.30 |
| tai5x5\_8 | 300 | 307 318.93 6.92 2.30 | 301 317.60 6.71 0.33 |
| tai5x5\_9 | 353 | 365 376.60 9.20 3.30 | 358 375.06 8.33 1.41 |
| tai5x5\_10 | 326 | 338 348.94 8.50 3.70 | 333 346.33 8.40 2.14 |
| tai7x7\_1 | 438 | 451 479.93 13.07 2.98 | 444 469.40 9.63 1.83 |
| tai7x7\_2 | 449 | 469 490.13 14.95 4.40 | 462 483.18 12.42 2.90 |
| tai7x7\_3 | 479 | 505 529.80 12.79 5.40 | 505 522.86 10.51 5.40 |
| tai7x7\_4 | 467 | 495 505.33 12.70 6.00 | 486 504.53 9.31 4.10 |
| tai7x7\_5 | 419 | 438 456.20 9.36 4.50 | 428 454.53 8.33 2.15 |
| tai7x7\_6 | 460 | 489 511.47 13.34 6.30 | 481 506.93 10.77 4.56 |
| tai7x7\_7 | 435 | 461 492.87 14.25 5.97 | 458 484.45 11.78 5.28 |
| tai7x7\_8 | 426 | 455 470.07 10.57 6.80 | 448 464.26 8.87 5.16 |
| tai7x7\_9 | 460 | 470 502.40 9.87 2.18 | 467 492.40 10.68 1.52 |
| tai7x7\_10 | 400 | 411 439.33 11.80 2.75 | 407 438.53 10.86 1.75 |
| tai10x10\_1 | 652 | 683 719.60 17.89 4.75 | 676 707.33 16.37 3.99 |
| tai10x10\_2 | 596 | 627 649.00 17.02 4.76 | 627 646.73 16.45 4.76 |
| tai10x10\_3 | 617 | 645 669.47 14.03 4.53 | 643 656.86 12.87 4.21 |
| tai10x10\_4 | 581 | 615 646.46 15.93 5.85 | 608 642.46 15.89 4.65 |
| tai10x10\_5 | 657 | 694 732.80 13.81 5.63 | 681 698.20 11.34 3.65 |
| tai10x10\_6 | 545 | 565 601.60 20.45 4.12 | 560 577.93 11.27 2.75 |
| tai10x10\_7 | 623 | 658 693.60 13.30 5.62 | 653 664.46 10.22 4.81 |
| tai10x10\_8 | 606 | 639 675.47 16.98 5.44 | 631 649.80 14.83 4.12 |
| tai10x10\_9 | 606 | 640 673.93 13.20 5.94 | 640 674.10 13.35 5.94 |
| tai10x10\_10 | 604 | 644 665.93 7.77 6.62 | 630 656.26 13.11 4.30 |
| tai15x15\_1 | 956 | 1026 1066.93 21.60 6.28 | 1011 1048.06 21.50 5.75 |
| tai15x15\_2 | 957 | 1030 1073.03 21.10 6.59 | 1019 1048.89 20.25 6.48 |
| tai15x15\_3 | 899 | 966 1005.87 28.03 6.34 | 946 986.80 18.65 5.23 |
| tai15x15\_4 | 946 | 1016 1054.0 20.91 6.34 | 991 1023.20 21.59 4.75 |
| tai15x15\_5 | 992 | 1067 1113.4 20.95 6.55 | 1038 1083.20 20.10 4.63 |
| tai15x15\_6 | 959 | 1027 1056.13 21.76 7.19 | 1019 1046.86 22.46 6.25 |
| tai15x15\_7 | 931 | 1006 1068.30 28.18 6.98 | 993 1044.86 21.54 6.65 |
| tai15x15\_8 | 916 | 990 1049.10 21.61 8.07 | 977 1031.86 19.20 6.66 |
| tai15x15\_9 | 951 | 1022 1060.73 19.81 7.47 | 1001 1059.53 20.02 5.23 |
| tai15x15\_10 | 935 | 998 1018.67 16.85 6.73 | 983 1015.40 14.95 5.13 |

* In the table3, “Instance” means the problem name, “taiIxJ\_K” means problem has I jobs, J machine and Kth instance of taiIxJ. “Best known” means the best known solution for the instance, “Best” and “Avg.” means the best and average solution found by each algorithm respectively over 15 runs.
* S.D means standard deviation of solution found by each algorithm over 15 runs. This is calculated by:

S.D

where, xi = answer of ith run of algorithm,

and, x’ = mean or average

* S.D shows a deviation from expected value and average value.
* RPE
* The performance of ABC and Beam-ABC algorithm is evaluated by testing them on the above 50 benchmark problems provided by Taillard. This is well-known benchmark for OSSP.
* The objective is to find the minimum makespan value from these benchmark problems.
* In ABC algorithm total population size is set to 50. Both the number of employed bees and number of onlooker bee are set equal i.e. 25.
* The maximum cycle number (MCN) is set to different for different machine. For 2 to 5 machines is set 2500, for 7 to 9 machines 3500 and greater than 9 machines is set 5000.
* MCN is how many times algorithm try to update the solutions by employed, onlooker and scout bees.
* Each of the experiment is repeated 15 times with different random seeds.
* From table3, we can observe Beam-ABC algorithm gives better result comparison than ABC algorithm and also S.D of Beam-ABC algorithm is lesser than ABC algorithm.
* RPE means relative percent error with respect to the best known solution, which is calculated from the equation below

Job number *=* × 100

* The size of these problem instances range from 4 to 20 jobs and 4 to 15 machines.
  1. *Analysis*

Now I am going for the analysis of the results which have been shown in table 3 for various benchmark of OSSP provided by Taillard. I have taken 50 instances in which 10 instances for tai4x4\_\*, 10 instances for tai5x5\_\*, 10 instances for tai7x7\_\*, 10 instances for tai10x10\_\*, and 10 instances for tai15x15\_\*.

On the small problem instances (tai4x4\_\*) RPE is almost ‘0.0’ for both Beam-ABC and ABC algorithm. It means both algorithm give almost 100% accuracy for tai4x4\_\* instances. After that both algorithm give different–different RPE for each instance. Best results of Beam-ABC for 13 instances from 50 instances have been matched with best known solutions whereas only 6 instances for ABC. Accuracy for tai4x4\* instances Beam-ABC has 100% while ABC has 99.5-100%, for tai5x5\_\* instances Beam-ABC has 97-100% and ABC has 95.4-100%, for tai7x7\_\* instances Beam-ABC has 94.5-98.25% and ABC had 94-97.18%, for ta10x10\_\* instances Beam-ABC has 94-97% and ABC has 93-95%, for tai15x15\_\* instances Beam-ABC has 92-94% and ABC has 92-93%

Comparison chart of the solutions of Beam-ABC and ABC for 15 runs of each algorithm:

Fig.13. For tai4x4\_2 instance

Fig.12. For tai4x4\_1 instance

Fig.15. For tai5x5\_2 instance

Fig.14. For tai5x5\_1 instance

Fig.17. For tai7x7\_2 instance

Fig.16. For tai7x7\_1 instance

Fig.19. For tai10x10\_2 instance

Fig.18. For tai10x10\_1 instance

Fig.20. For tai15x15\_1 instance

From comparison charts (Fig.12 – Fig.20) we can observe that ABC algorithm gives more times higher value of solutions compare than Beam-ABC algorithm whereas we try to minimize the value. Horizontal axis represent iteration number and vertical axis represent makespan time.

Fig.16. Result of Beam-ABC and ABC algorithm for each instance

From Fig. 16 we can observe that Beam-ABC algorithm provides better solution compare than ABC algorithm. Solution is fully depended on randomization so sometimes solution of ABC algorithm is better than Beam-ABC algorithm. But most of time Beam-ABC algorithm provides better solution.

Comparison of Beam-ABC, SA-PSO and Tabu-ACO for solving JSSP are shown in table 4.

Table 4: Result of Beam-ABC, SA-PSO and Tabu-ACO for solving the various JSSP benchmarks [18]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instance | Size  (nxm) | Best Known | Beam-ABC  Best Average RPE | SA-PSO  Best Average RPE | Tabu-ACO  Best Average RPE |
| abz05 | 10x10 | 1234 | 1340 1384.50 8.42 |  | 1345 1391.11 8.99 |
| abz06 | 10x10 | 943 | 1010 1048.45 7.21 |  | 1005 1036.71 6.57 |
| ft06 | 6x6 | 55 | 55 56.73 0.00 | 55 55 0.00 | 55 57.11 0.00 |
| ft10 | 10x10 | 930 | 1017 1079.60 9.35 | 1035 1055.55 11.29 | 1029 1088.81 10.64 |
| ft20 | 20x5 | 1165 | 1203 1266.00 3.26 | 1165 1190.10 0.00 | 1178 1198.81 1.15 |
| la01 | 10x5 | 666 | 666 689.27 0.00 | 666 691.18 0.00 | 666 687.18 0.00 |
| la02 | 10x5 | 655 | 678 704.27 3.35 | 655 678.18 0.00 | 678 711.11 3 .35 |
| la03 | 10x5 | 597 | 631 657.93 5.69 | 597 615.13 0.00 | 635 665.56 6.36 |
| la04 | 10x5 | 590 | 626 642.33 6.10 | 590 618.38 0.00 | 620 641.14 5.08 |
| la05 | 10x5 | 593 | 593 593.00 0.00 | 593 594.11 0.00 | 598 601.11 0.84 |
| la06 | 15x5 | 926 | 935 955.20 0.91 | 930 945.81 0 .42 | 932 947.71 0.64 |
| la07 | 15x5 | 890 | 933 963.86 3.82 | 906 939.17 1.79 | 937 968.82 5.28 |
| la08 | 15x5 | 863 | 912 950.33 5.46 | 872 891.11 1.04 | 910 951.19 5.44 |
| la09 | 15x5 | 951 | 966 1002.67 1.57 | 956 966.16 0.52 | 972 1019.44 2.21 |
| la10 | 15x5 | 958 | 958 967.53 0.00 | 966 978.23 0.83 | 958 970.10 0.00 |
| la12 | 20x5 | 1039 | 1076 1117.53 3.56 | 1069 1098.81 2.89 | 1075 1105.55 3.54 |
| la13 | 20x5 | 1150 | 1194 1230.93 3.82 | 1175 1188.18 2.17 | 1195 1234.44 3.91 |
| la14 | 20x5 | 1292 | 1292 1301.40 0.00 | 1301 1334.41 0.70 | 1292 1319.10 0.70 |
| la15 | 20x5 | 1297 | 1346 1379.53 3.77 | 1360 1381.18 4.85 | 1346 1378.54 3.77 |
| la16 | 10x10 | 945 | 995 1031.87 5.29 | 1013 1031.11 7.19 | 995 1037.81 5.29 |
| la17 | 10x10 | 784 | 825 860.73 3.95 | 825 851.41 3.95 | 841 887.74 7.27 |
| la18 | 10x10 | 848 | 897 954.40 5.77 | 925 965.51 9.08 | 895 943.39 5.54 |
| la19 | 10x10 | 842 | 917 946.40 8.91 | 930 961.31 10.45 | 921 946.61 9.26 |
| la20 | 10x10 | 902 | 974 1000.53 7.98 | 947 984.51 4.76 | 965 984.41 6.98 |
| orb01 | 10x10 | 1059 | 1160 1212.60 9.53 | 1136 1173.73 7.27 | 1151 1191.49 8.68 |
| orb02 | 10x10 | 888 | 961 1013.73 8.22 | 973 1011.19 9.57 | 970 1019.84 9.34 |
| orb03 | 10x10 | 1005 | 1096 1162.73 9.05 | 1113 1161.11 10.74 | 1096 1157.81 9.05 |
| orb04 | 10x10 | 1005 | 1101 1141.40 9.35 | 1097 1135.16 9.15 | 1110 1131.17 10.44 |
| orb05 | 10x10 | 887 | 954 993.00 7.55 | 962 991.18 8.45 | 968 1001.00 9.13 |

In Table 4, I compared the solutions of Beam-ABC algorithm for job shop scheduling problem (JSSP), from SA-PSO algorithm (which is done by Mr. Vinay Kumar – IIT2009168) and Tabu-ACO algorithm (which is done by Mr. Vinay Saini – IIT2009074). SA-PSO algorithm is hybridization of Simulated Annealing with Particle Swarm Optimization and Tabu-ACO algorithm is hybridization of Tabu search and Ant Colony Optimization. The problems have been taken from the Operation Research Library (OR-Library) [18].

* 3 problems from Fisher and Thompson: referred as ft06, ft10, and ft20.
* 20 problems from Lawrence: referred as la01–la20.
* 2 problems from Adams et al.: referred as abz05-abz06
* 5 problems from Applegate and Cook: referred as orb01-orb10.

1. **Conclusions**

In this thesis, we have hybridized the solution construction mechanism of ABC algorithm with beam search. This approach, which we called Beam-ABC, is general and can in principles be applied to any combinatorial optimization problem. I proposed a Beam-ABC approach for the application to open shop scheduling problem and job shop scheduling problem. The results obtained from my proposed method show that Beam-ABC can find the best known solution more effectively than ABC algorithm. The Beam-ABC algorithm gives a better average makespan value. The results also show that Beam-ABC is more robust than the ABC algorithm as shown by the value of SD in Table 3. Thus, we can conclude that that Beam-ABC is effective than ABC algorithm from both perspective of solution quality and algorithm robustness.

1. **Recommendations and Future Work**

We should increase maximum cycle number (MCN) (see section 5.1) to improve the solutions but it needs high configuration computer for large size problem. High configuration means system should do computation faster than normal computer otherwise system will take time to give the results. We can use other heuristic search method instead of beam search method like A\* search method, tabu search, best first search, simulated annealing method to improve the solutions.

In the future work, JSSP and OSSP can be considered in a case where “Job interrupt” is permitted. In that case, ABC algorithm may split a job into a number of smaller sub-jobs. This will allow each job to be preempted by other jobs with higher priorities. The preempted job can later be resumed when a machine is free. After applying this improvement, ABC will be more practical in most of the shop scheduling problems. We can apply Beam-ABC approaches to other NP-Complete problems.

**Appendix - Explanation of the Source Code**

I made a source code for this application in Java programming language. So files extensions are \*.java. The control flow of the source code for this application goes like this.

First source file which has hybridized ABC algorithm with beam search will be called to run this application. This file name is BEAM\_ABC\_OSSP.java.

Main work of this class shown below:

*public class BEAM\_ABC\_OSSP {*

*static ABC\_OSSP abc = new ABC\_OSSP();*

*public BEAM\_ABC\_OSSP() {*

*for(int run=0; run<abc.runtime; run++ ) {*

*abc.initial();*

*abc.MemorizeBestSource();*

*for(int iter=0; iter<abc.maxCycle; iter++ ) {*

*abc.SendEmployedBees(); //calling employee bees*

*abc.CalculateProbabilities(); //calculating probability of each food*

*abc.SendOnlookerBees(); //calling onlooker bees to select best food*

*abc.MemorizeBestSource(); //memorizing best food source*

*abc.SendScoutBees(); //calling scout bees*

*abc.BeamSearch(); //Apply beam search on best food source*

*}*

*for(int j=0; j<abc.D; j++ ) {*

*bestSeq[run][j] = abc.GlobalParams[j];*

*}*

*}*

*}*

In this class ABC\_OSSP (AO) class has been called as object ‘abc’ which contains implementation of ABC algorithm and beam search algorithm. In this class Beam-ABC algorithm runs 15 times (abc.runtime which set by user) and result of each run stored in bestSeq[][] array. Runtime and maxCycle is data member of ABC\_OSSP class.

abc.initial() call initial() method which implement in ABC\_OSSP.java file. This is initialization phase described in section 4.1. It initializes all food sources randomly. Source code for initial() methods:

*void initial() {*

*int i,j;*

*for( i=0; i<SN; i++ ) {*

*init(i);*

*for( j=0; j<D; j++ )*

*Food[i][j] = Foods[i][j];*

*}*

*GlobalMin = f[0];*

*for( i=0; i<D; i++ )*

*GlobalParams[i] = Foods[0][i];*

*}*

SN refer for number food sources and init(i) function generate random food source of D dimension. Where D = number of job \* number of machine and calculate fitness value of each food source.

abc.MemorizedBestSource() call MemorizeBestSource() method of class ABC\_OSSP which memorize best source at the time of calling according fitness value of food sources. A food source has lowest fitness value that will memorize as best food source (best solution) globally. Source code for MemorizeBestSource() mehod:

*void MemorizeBestSource() {*

*int i,j;*

*for( i=0; i<SN; i++ ) {*

*if( f[i]<GlobalMin ) {*

*GlobalMin = f[i];*

*for( j=0; j<D; j++ ) {*

*GlobalParams[j] = Foods[i][j];*

*}*

*}*

*}*

*}*

GlobalMin store lowest fitness value among all food source and and GlobalParams[] array store corresponding operation scheduling list.

SendEmployedBees() is method of ABC\_OSSP class. This is Employed bees phase described in section 4.2. In this method one food source will assign to each employed bee. After assigning food source to employed bees, employed bees update their food source from neighbor’s food source using PBX method and counting trial of the food source. Trial is *limit* parameter for each food source. If any food source is not updated limit number of times in continuous iteration than that food source will be abandoned by employed bee and employed bee become scout bee corresponding the abandoned food source. Source code for SendEmployedBees() method:

*void SendEmployedBees() {*

*int i,j;*

*for( i=0; i<SN; i++ ) {*

*int randSelecJ[] = new int[D];*

*boolean randSelecD[] = new boolean[D];*

*int nj[] = new int[noj];*

*for( j=0; j<noj; j++)*

*nj[j] = nom;*

*// Exchanging information with a neighbouring food source based on PBX Method*

*for( j=0; j<D; j++ ) {*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*if( pr>=0.55 ) {*

*randSelecJ[Food[i][j]]++;*

*randSelecD[j] = true;*

*}*

*}*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*r1 = (int)(pr\*SN);*

*while( r1==i ) {*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*r1 = (int)(pr\*SN);*

*}*

*neighbour = r1;*

*int k = 0;*

*for( j=0; j<D; j++ ) {*

*if( randSelecD[j] ) {*

*while( randSelecJ[Food[neighbour][k]]==0 )*

*k++;*

*randSelecJ[Food[neighbour][k]]--;*

*Food[i][j] = Food[neighbour][k];*

*k++;*

*}*

*}*

*for( j=0; j<D; j++ ) {*

*solution[j] = Food[i][j];*

*}*

*ObjValSol = CalculateFunction(solution);*

*FitnessSol = CalculateFitness(ObjValSol);*

*if( FitnessSol>fitness[i] ) {*

*trial[i] = 0;*

*for( j=0; j<D; j++ ) {*

*Foods[i][j] = solution[j];*

*}*

*f[i] = ObjValSol;*

*fitness[i] = FitnessSol;*

*}*

*else {*

*trial[i] += 1;*

*for( j=0; j<D; j++ )*

*Food[i][j] = Foods[i][j];*

*}*

*}*

*}*

Math.random() method generate random number. It is predefined method in java. CalculateFunction() method is calculating makespan operation scheduling list. CalculateFitness() calculate fitness value of scheduling list of operations. randSelecJ[] array count how many operation of each job has been selected for creating new food source according to randomly selected neighbor’s food source.

CalculateProbabilities() method calculate probability of each food source. Source code for CalculateProbabilities() method:

*void CalculateProbabilities() {*

*double totfit = fitness[0];*

*for(int i=1; i<SN; i++ )*

*totfit += fitness[i];*

*for(int i=0; i<SN; i++ )*

*prob[i] = fitness[i]/totfit;*

*}*

totfit is summation of fitness value of all food sources. prob[] array store probability of each job.

SendOnlookerBees() method implemented in ABC\_OSSP class. This is onlooker bees phase described in section 4.3. Source code:

*void SendOnlookerBees() {*

*int i=0,j,t=0;*

*while( t<SN ) {*

*pr = ( (double)Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*if( pr<=prob[i] ) {*

*t++;*

*int randSelecJ[] = new int[D];*

*boolean randSelecD[] = new boolean[D];*

*int nj[] = new int[noj];*

*for( j=0; j<noj; j++)*

*nj[j] = nom;*

*// Exchanging information with a neighbouring food source based on PBX Method*

*for( j=0; j<D; j++ ) {*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*if( pr>=0.55 ) {*

*randSelecJ[Food[i][j]]++;*

*randSelecD[j] = true;*

*}*

*}*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*r1 = (int)(pr\*SN);*

*while( r1==i ) {*

*pr = ((double) Math.random()\*32767 / ((double)(32767)+(double)(1)) );*

*r1 = (int)(pr\*SN);*

*}*

*neighbour = r1;*

*int k = 0;*

*for( j=0; j<D; j++ ) {*

*if( randSelecD[j] ) {*

*while( randSelecJ[Food[neighbour][k]] == 0 )*

*k++;*

*randSelecJ[Food[neighbour][k]]--;*

*Food[i][j] = Food[neighbour][k];*

*k++;*

*}*

*}*

*for( j=0; j<D; j++ ) {*

*solution[j] = Food[i][j];*

*}*

*ObjValSol = CalculateFunction(solution);*

*FitnessSol = CalculateFitness(ObjValSol);*

*if( FitnessSol>fitness[i] ) {*

*trial[i] = 0;*

*for( j=0; j<D; j++ ) {*

*Foods[i][j] = solution[j];*

*}*

*f[i] = ObjValSol;*

*fitness[i] = FitnessSol;*

*}*

*else {*

*trial[i] += 1;*

*for( j=0; j<D; j++ )*

*Food[i][j] = Foods[i][j];*

*}*

*}*

*i++;*

*if( i==SN ) i = 0;*

*}*

*}*

SendScoutBees() is a method of ABC\_OSSP class. This is scout bees phase described in section 4.4. Source code:

*void SendScoutBees() {*

*int i,maxtrialindex = 0;*

*for( i=1; i<SN; i++ ) {*

*if( trial[i]>trial[maxtrialindex] )*

*maxtrialindex = i;*

*if( trial[maxtrialindex]>=limit ) // Abandon food source F[maxtrialindex]*

*init(maxtrialindex);*

*}*

*}*

It check abandoned food source and generate new random food source by calling init(maxtrialindex)method.

BeamSearch() is method of ABC\_OSSP class. This is beam search phase described in section 4.5. BeamSearch() method call BEAM\_SEARCH\_OSSP (BSO) class where beam search is implemented. After getting sequence from beam BSO class BeamSearch() method improve the solution by using PBX method on best source and sequence provided by BSO class.

Makespan of sequences calculated by calling OSSP class, which is implemented in OSSP.java and stored start time and end time of each operation.

After getting best sequence, AO class called GANTT\_CHART\_OSSP class, which is implemented in GANTT\_CHART\_OSSP.java file and shows the Gantt chart for best sequence of operations with their end time.

**References**

1. B.Basturk, Dervis Karaboga, “A powerful and efficient algoithm for numerical function optimization: artificial bee colony (ABC) algorithm,” Journal of global optimization (2007) 39, Springer: 459-471.
2. Manish Gupta, Govind Sharma, “An Efficient Modified Artificial Bee Colony Algorithm for Job Shop Scheduling Problem,” International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307, Volume-1, Issue-6, January 2012.
3. Ling Wang, Gang Zhou, Ye Xu, Shengyo Wang, Min Liu, “An Effective Artificial Bee Colony Algorithm for the Flexible Job-Shop Scheduling Problem,” International Journal of Advanced Manufacturing Technology (2012) 60:303-315.
4. D. Karaboga, “An Idea Based On Honey Bee Swarm for Numerical Optimization,” Technical Report-TR06,Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
5. Dervis Karaboga, Bahriye Akay, “A comperative study of Artificial Bee colony algorithm,” Applied Mathematics and Computation 214, Elsevier, 2009, 108-132.
6. Eugeniusz Nowicki, Czeslaw Smutnicki, “An Advanced Tabu Search Algorithm For The Job Shop Problem,” Journal of Scheduling 8, Springer: 145-159, 2005.
7. Ali Hadidi, Sina Kazemzadeh Azad, Saeid Kazemzadeh Azad, “Structural optimization using artificial bee colony algorithm, 2nd International Conference on Engineering Optimization,” 2010, September 6 – 9, Lisbon, Portugal.
8. Y. Zhang and L. Wu, “Optimal multi-level Thresholding based on Maximum Tsallis Entropy via an Artificial Bee Colony Approach,” Entropy, vol. 13, no. 4, (2011), pp. 841-859.
9. Anan Banharnsakun, Booncharoen Sirinaovakul, Tiranee Achalakul, “Job Shop Scheduling with the Best-so-far ABC,” Engineering Application of Artificial Intelligence 25, Elsevier, 2012, 583-593.
10. Yueh-Min Huang, Jin-Chen Lin, “A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems,” Expert Systems with Applications 38 (2011), Elsevier, 5438-5447.
11. Christian Blum, “Beam-ACO – hybridizing ant colony optimization with beam search: an application to open shop scheduling,” Computers & Operations Research 32 (2005), Elsevier, 1565-1591.
12. Wong, Li-Pe et al., “Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems,” WSC 2008, IEEE, pp. 2050-2058.
13. Liaw, Ching-Fang, “A hybrid genetic algorithm for open shop scheduling problems,” European Journal of Operational Research, 124 (2000), Elsevier, 28-42.
14. Prins, Christian, “Competitive genetic algorithms for open shop scheduling problems,” Dissertation, University of Technology of Troyes, Department of GSI, 2000
15. I. Sabuncuoglu, M. Bayiz, “Job shop scheduling with beam search,” European Journal of Operational Research, 118 (1999), Elsevier, 390-412.
16. Julia A. Bennell, Xiang Song, “A beam search implementation for the irregular shape packing problem,” Journal of Heuristic (2010) 16, Springer: 167-188.
17. Éric Taillard's, Scheduling instances [Online],

Available: http://mistic.heig-vd.ch/taillard/problems.dir/ordonnancement.dir/ordonnancement.html.

1. Dirk C. Mattfeld and Rob J.M Vaessens, jobshop1.txt [Online],

Available: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt