Hybrid Genetic Bees Algorithm applied to single machine scheduling with earliness and tardiness penalties

Yuce, B.; Fruggiero, F.; Packianather, M. S.; Pham, D. T.; Mastrocinque, E.; Lambiase, A.; Fera, M.

DOI:
10.1016/j.cie.2017.07.018

Hybrid Genetic Bees Algorithm applied to Single Machine Scheduling with Earliness and Tardiness Penalties

B. Yuce[1*], F. Fruggiero[2], E. Mastrocinque[3], M.S. Packianather[4], D.T. Pham[5], A. Lambiase[6] and M. Fera [7]

[1] *BRE Centre for Sustainable Engineering, School of Engineering, Cardiff University, CF24 3AA Cardiff, UK,*
[2] *School of Engineering, University of Basilicata, Via Ateneo Lucano 10, 85100 Potenza, Italy,*
[3] *Faculty of Engineering, Environment and Computing, Coventry University, Priory Street, CV1 5FB, Coventry, UK*
[4] *High Value Manufacturing Group, School of Engineering, Cardiff University, CF24 3AA Cardiff, UK,*
[5] *School of Mechanical Engineering, University of Birmingham, B15 2TT, Birmingham, UK*
[6] *Department of Industrial Engineering, University of Salerno, Fisciano, Italy,*
[7] *Dept. of Industrial and Information Engineering, Second University of Naples, Via Roma 29, 81031 Aversa, Italy,*
[*]**Correspondent Author email address: yuceb@cardiff.ac.uk**

ABSTRACT This paper presents a hybrid stochastic based solution for the single machine scheduling problem. The proposed hybrid algorithm also addresses the enhancement of the Bees Algorithm on its global search stage. Although the algorithm has several successful implementation on several different types' optimisation problems, it suffered from NP-hard combinatorial problems. To overcome this difficulty, it has been proposed an enhancement on the global search stage of the algorithm. In this work, the algorithm is reinforced with Genetic Algorithm on the global search stage, this new meta-heuristic approach was called Genetic Bees Algorithm (GBA). The combination of the exploration capability by means of a reinforced Global Search and a jumping - mutating function, together with the exploitation strength of the bees approach is implemented. Computational experiments over 280 well-known benchmark problems show the effectiveness of the proposed approach compared to traditional heuristics. The GBA allows to improve scheduling while avoiding getting stuck in local optima. GBA overcomes BA if compared in terms of time to convergence around 60% less and quality of results around 30% better for highly constrained jobs. Although, the performance of the proposed approach is not optimal in lightly constrained tasks, it increases quality of the results over 3 times if compared with the basic Bees Algorithm approach.

KEYWORDS: Swarm-Based Optimisation; Bees Algorithm (BA); Genetic Bees Algorithm (GBA); Single Machine Scheduling Problem (SMSP).

1. INTRODUCTION

Nowadays, firms have to offer a great variety of different products while customers expect orders of goods to be delivered on time. In order to meet these kinds of requirements, Lean Manufacturing, Fit Manufacturing, Just In Time (JIT) production need to be implemented. All these approaches require that jobs are expected to be punctual, since the late as well as early delivery is seen negatively. In JIT production due dates can occur as common due dates, whatever a set of jobs is needed simultaneously for the assembly at the higher stage of the production. Regarding scheduling for optimization in operations management, the identification of bottleneck machine that strongly influences performances of systems can be treated as a single machine scheduling issue where multiple jobs need to be arranged and delivered on time.

This paper addresses the single machine scheduling of a set of jobs with a common due date and the objective of minimizing the job's total earliness and tardiness.

In Single Machine Earliness/Tardiness Problem (SMETP), performance is measured by the minimization of the weighted sum of earliness and tardiness penalties of jobs. In this paper a hybrid Genetic Bees Algorithm (GBA) is proposed in order to solve the SMET problem. The proposed meta-heuristic aims to combine the Genetic Algorithms' exploration performances (Goldberg, 1989) with the exploitation capacity of the Bees Algorithm (Pham *et al.,* 2005). This paper is organized as follows: firstly, a literature review on the single machine scheduling problem and the heuristics used is given; secondly, an overview on the Bees Algorithm followed by the proposed hybrid Genetic Bees Algorithm (GBA) is presented; finally the tuning phase of the GBA and its application on benchmarks of different sizes and complexities are shown. Performances have been measured and compared to the basic Bees approach and other well-known heuristics, by the minimization of the weighted sum of earliness and tardiness penalties. Discussion about the role of algorithm parameters in terms of efficacy (i.e., objective function) and effectiveness (i.e., computational time) is reported.

2. BACKGROUND

The problem of considering optimal due date assignment together with the definition of an optimal scheduling policy was firstly considered by Seidmann *et al.*, (1981) and Panwalkar *et al.*, (1982) using analytical approaches. Moreover, considering the problem of processing jobs within a time window has been showed to be a *NP-hard* (Lee *et al.*, 1991).

Cheng, (1984) demonstrated how, under certain production conditions in which jobs 'completion can be anticipated, an optimal due date assignment can be defined. Single machine scheduling problem considering lateness performance under constrained due date, was firstly discussed by Gupta *et al.*, (1987). Complexity of this problem was studied in the work of Lauff and Werner, (2004) where the aim was to minimize the sum of the absolute deviations of the completion times from the due. Open and job shop systems where compared with two machine flow shop environment in the case of restrictive and non-restrictive due date and it was proven that flow shop environment is *NP-hard* in the strong sense. In the work of Bagchi *et al.*, (1986) was asserted that value of the due date might influence the computational complexity. The optimal objective function value of a certain problem cannot increase by increasing the due date while keeping constant all the others parameters. As a consequence, there is a time period in which the products should finish and then the delivery to customers should be arranged. The restricted common due date problem is generally much more difficult to solve than a non-restricted one (Baker and Scudder, 1990). It is appropriate to consider that in the case of a single machine scheduling, the optimal penalty cost cannot decrease with the increase in the common due date (Webster, 1997). For the case of single machine scheduling under due date penalties, Kanet in 1981 was the first to assume a problem in which penalties occur when a job is completed early or late to restrictive assumptions on the due dates and in penalty functions for jobs.

Tardiness penalties due to delivery after a contractually arranged due date, consider the loss of customers' goodwill and damage reputation as well as delay of payment and shortages. This also entails extra costs including late charges (Fisher and Jaikumar, 1978). On the other hand, completing a job before the due date increases the cost or probability of related cost due to insurance, inventory carrying, holding, theft, perishing and loss of product quality, bounded capital (Webster, 1997).

The due date assignment problem makes practical sense when the company plans delivery to clients.

Moreover, the increasing adoption of the JIT approach in industries has made due date backward assignment an active area of scheduling research (Li *et al.*, 2006). Inventory management such as JIT concepts is mainly dependent by the certainty of production capacity and lead time. In JIT systems, jobs have to be completed neither too early nor too late (Monden, 1983). This leads to the scheduling problems with both earliness and tardiness penalties.

Single machine scheduling problem occurs every time a closed continuous flow is arranged or whatever bottlenecks characterize the overall performances of the considered system. Thus, meeting common due dates has always been one of the most important objectives in scheduling and supply chain management. At the same time, the common due date makes sense whenever it is not required detailed control for jobs or better when all goods and services are comparable in terms of resources allocation (Cheng, 1988).

Common due date helps managers to get economies of scale and facilitate control (Gordon *et al.*, 2002). Moreover, besides the delivery of tardy services, the main issue to be taken into account is the cost discount that can be derived whenever a warehouse does not exist and products do not have obsolescence or extra costs.

The problem of common due date for single machine scheduling definition was firstly analysed in Panwalker *et al.*, (1982). Common due date can be either externally defined and imposed by the market (Baker and Scudder, 1990), or internally defined as a time line manager wants to achieve.

In literature, several pieces of works have been conducted on the solution of the SMETP problem (*e.g.*, Panwalkar, *et al.*, (1982); Cheng, (1984); Janiak, (1991); Cheng *et al*., (2004); Mosheiov and Yovel, (2006); Lin *et al.*, (2007); Nearchou, (2008); Gordon and Strusevich, (2009); Wang and Wang, (*2*010); Li *et al.*, (2011); Nearchou, (2011); and Yang *et al.*, (2014)).

Benchmarks for scheduling with common due date were presented in the paper of Biskup and Feldmann (2001). They generated benchmark data set for SMETP which then became popular among the researchers and solved 280 instances using two dedicated heuristics for identifying the upper bounds on the optimal function values. Instances and values are currently available in order to test performances of newly heuristics. These benchmarks are widely used to test performances in SMETP

(Feldmann and Biskup, (2003); Chen and Sheen, (2007); Nearchou (2008); Lin *et al.*, (2007); and Nearchou, (2011). Further, the benchmark problem generation process for single machine early/tardy scheduling is proposed by Abdul-Razaq and Potts, (1988); Li, (1997); and Liaw, (1999), and widely utilised in the heuristics as stated in Valente and Alves, (2005); Valente *et al.*, (2006); Lin *et al.*, (2007); Valente (2008); Valente and Schaller, (2012); and Sundar and Singh, (2012). The performance of proposed approaches in the last two papers are not presented with exact solutions values. However, there is a relative comparison between heuristics results and upper boundaries.

Due to the complexity of SMETP local search, meta-heuristics approach are mainly introduced as solution method. The total tardiness/earliness problem was first studied by Emmons in late sixties (Emmons, 1969). Up to the early seventies, all the work done in this area were basically practice oriented, aiming at designing fast enumerative algorithm to find an optimal schedule. Pseudo polynomial time algorithm were proposed by Lawler, (1979) in approximation scheme.

Adbul-Razaq and Potts, (1988) developed a branch-and-bound algorithm that employs lower bounds by the dynamic programming state space relaxation technique. Satisfactory solution are obtained in large number of jobs (up to 25 jobs) with lower processing times. Moreover, an efficient heuristic based on branch-and-bound algorithm with decomposition of problem into two sub-problems and two efficient multiplier adjustment is proposed in the work of Li, (1997) for up to 50 jobs. Moreover, a combination of priority dispatching rules with local improvements is used for eliminating unpromising nodes in the branch-and-bound algorithm of Liaw, (1999). Valente and Alves, (2005) demonstrated the influence of initial sequence on lower bound as stated in Li, (1997); and Liaw, (1999). A survey regarding algorithms and approaches for SMETP were reported in the works of Crauwels *et al.*, (1997). Hybrid constructive strategies for SMETP are performed in Hino *et al.*, (2005). The role of almost all dispatching rules for the optimal SMTP (earliness is not included) issues was stated in Valente and Schaller, (2012). Heuristics approaches to solve SMETP have been applied by Yeung *et al.*, (2001). In particular, they developed a branch a bound algorithm to minimize, under common due windows, earliness and tardiness penalties. Three meta- heuristics approaches for stable scheduling on a single machine based mainly on Branch & Bound and Genetic

operators are reported in the work of Ballestin and Leus, (2008) when deviation between planned and actual job starting time occurs. Beams search heuristics with recovery procedures is used in the work of Valente in 2008 with optimal performance for small and medium SMETP instances. If pre-evaluation in beam is included based on dispatching rules Excessive computational time is required for medium and large (more than 75 jobs) instances. Filtered beam search method for near optimal sequences of jobs was proposed by Ow and Morton, (1989). Another study using the genetic operators in non-dominated sorting algorithm combined with quantum bit representation are proposed by Liu *et al.,* (2013); and Jolai *et al.,* (2007). A combination of GA with 14 local search and initialization procedures is developed and tested on the randomly generated instances in Valente *et al.,* (2006). They demonstrate that, behind the quality of results, the combination of fitness evaluation and GA is greatly accelerate the convergence, and will reduce number of iterations and computational time at nearby optimal schedule when compared to heuristics based on dispatching and local searches. Hybrid permutation-coded evolutionary approach - confirming the requirement of combining steady state genetic schedules with adjacent pairwise interchange procedure – demonstrates the robustness of genetics and the average gain in computational effort by comparing the fitness evaluation strategies inside GA by Singh, (2010). Another methods used for SMETP is memetic approach, presented by Franca *et al.,* (2001). Greedy Randomized Adaptive Search Procedure (GRASP) is used in Norgueira *et al.,* (2014). Heuristics based on mathematical programming are proposed by Della Croce *et al.,* (2014), to obtain better performances for very large scale problems. A combination of local search heuristics, using dispatching and hill climbing and simulated annealing, with evolutionary algorithm is proposed by M'Hallag, (2007), where it was clear the role of hybridization as to improve the solution quality at a reasonable cost in terms of run time. Another hybrid approach is presented by Sundar and Singh, (2012). They proposed a local search approach combined with Artificial Bees Colony (ABC). The results are reported based on the optimum solutions presented by Valente *et al.,* (2006). The authors demonstrated the superior performances of ABC on quality of solution and convergence rate on the instances with 50, 75 and 100 jobs, compared to GA results. However the convergence performance is slower for the instances greater than 250 jobs. Another approach is based on Tabu Search and Simulated Annealing and Neighbourhood Search, proposed by Almeida and

Centeno, (1998), which is utilised the random generated SMTEP instances. Finally, complete surveys of heuristic methodologies for solving SMETP are reported in the work of Gupta and Sen, (1983); Sen *et al.,* (1996); Chen, (1996); Su and Chang, (1998); Gordon *et al.,* (2002); and Schaller, (2007).

In this paper, it is investigated the performances of a new enhanced hybrid version of the Bees Algorithm, called Genetic Bees Algorithm (GBA), which is enhanced with genetic operators. Since the basic Bees Algorithm may have limitation to converge the optimum solution in the desired time scale by Yuce et al., (2014). The genetic algorithm operators such as crossover and mutation operators are included in order to increase convergence rate by increasing the ability of the global search, the details of the proposed algorithm is defined in section 4.2. The validation and performances of the proposed approach - because of the easily access to the database and optimum solutions- are evaluated in the test data, presented by Biskup and Feldman, (2001). However, there are still other data sets available to be utilised in the literature presented by Valente *et al.,* (2006), Singh (2010); and Sundar and Singh, (2012). Notwithstanding, we will benchmark our results with other meta-heuristics from the major class of pure and hybrid approaches. It has been assumed that restrictive and relaxed common due date exists. For each job, individual earliness and tardiness completion time penalties are given in advance. Validation of the proposed meta-heuristic is presented in terms of computational time, effort and quality of solutions by means of the upper bound as used by Feldman & Biskup, (2003); Hino *et al.,* (2005); and as reported in the GA+ greedy local search and SA + greedy local search of Lin *et al.,* (2007)**.**

3. THE SINGLE MACHINE SCHEDULING PROBLEM

The optimal allocation of scarce resources to certain activities is the objective of the scheduling. Scheduling problems become sequence whenever constraints regarding priorities are not included (Carlier, 1982). A single machine scheduling problem is a well-studied optimisation problem where a set of *n*-jobs with given deterministic processing times T_i and due date, have to be processed on a machine according to some constraints. The goal is to find a schedule for the *n*-jobs which minimizes the sum of all the penalties occurring due to the constraints. This is a challenging optimization problem and therefore it is chosen to test the performance of the proposed GBA.

In the SMETP, resources are commonly referenced as machines M_k that can perform at most one activity - one job J_i i.e., an open or close sequence of tasks *i* with time T_{ijk} (i.e., the time *T* of a task *i* as part of the job *j* which requires the resource *k)* - at any time *t.*

Ubiquity of task is not enabled. All the information that defines a problem instance is known in advance. This characterizes a deterministic scheduling as part of the combinatorial optimisation. In the following we use the 3-parameter classification introduced by Graham *et al.,* (1979). Then, SMETP is formally classified as *n/1//ET* (French, 1982).

Let: $J=\{J_1, J_2, \ldots, J_{j-1}, J_j, J_{j+1}, \ldots, J_n\}$ the set of the *n* jobs existing inside the system to be processed without interruption on a single machine M_k (i.e., *k* here is equal to *1*) that can handle only one job at a time. Each job J_j is available at time zero, requires a positive process time T_{jk} and ideally must be completed exactly on a specific constant due date *D* proportionally to the amount of $C_k = \sum_{j=1}^{n} T_{jk}$ and common for all jobs.

Penalties occur every time the job *j* is completed before or early the fixed due date *D.*

The common due date *D* on machine *k* (i.e., D_k) is calculated by

$$D_k = round\left[\sum_{j=1}^{n} T_{jk} \times h\right] \quad (1)$$

where *round[X]* gives the biggest integer which is smaller than or equal to *X*; parameter h is used to calculate more or less restrictive common due dates.

An early $E_{kj}=max\ (0, D_k-T_{jk})$ or a tardy $R_{kj}=max\ (0, T_{jk}-D_k)$ occurs if the job *j* is not completed exactly on the specific assigned D_k. The possibility to accumulate R_{kj} – whatever its amount is preferable to E_{jk} because of its excessive penalties - in non-restrictive cases is allowed in order to get optimality. The objective is therefore to find a processing order for the *n* jobs that minimises the following objective:

$$OBJ = \sum_{j=1}^{n}\left(\alpha_{jk}E_{kj} + \beta_{jk}R_{kj}\right) \quad (2)$$

Where α_{jk} and β_{jk} are respectively the earliness and tardiness non negative penalties for the job *j* as processed on machine *k* and they constitute the deterministic input for the benchmarks. Thus, an optimal solution to unrestricted SMETP ($h \geq 0.4$) may exist if no idle time in scheduling occurs and the starting time of the first job could not start at time zero (Cheng and Kahlbacher, 1991). Close jobs

is a necessary but a not sufficient condition to the optimization. Here, the complexity is related more to the arbitrary starting date than to the close sequence of jobs. The restrictive form of SMETP is much more complex than the unrestricted one given the NP-hard nature of the problem (i.e., excluding optimum schedule a priori when the n>20) (Du and Leung, 1990).

In order to generate data tests, a set of n jobs with deterministic processing times T_{jk} and a common due date D_k are given. 7 data files were used in this study according to the number of jobs n which are equal to 10, 20, 50, 100, 200, 500, 1000 and with different restricted (h = 0.2 and h = 0.4) and unrestricted (h = 0.6 and h = 0.8) constraints in due date (D_k) on one machine (k=1). Jobs have to be processed on one machine and, for each of the jobs, an individual earliness E_j and tardiness T_j penalties are given, if a job is finished before or after the common due date D, respectively[1].

4. THE ENHANCEMENT OF THE BEES ALGORITHM WITH GENETIC OPERATORS

4.1 THE BEES ALGORITHM

A colony of bees exploits, in multiple directions simultaneously, food sources in the form of antera with plentiful amounts of nectar or pollen. They are able to cover kilometric distances for good foraging (Gould, 1975). Flower paths are covered based on a stigmergic approach – sites with higher nectar content should be visited by more bees (Crina and Ajith, 2006). The foraging strategy starts by scout bees, which represent a percentage of the beehive population. They wave randomly from one patch to another. Returning at the hive, those scout bees deposit their nectar or polled and start a recruiting mechanism called waggle dance (Von Frisch, 2014). Bees, stirring up for discovery, flutter from one to one hundred circuits with a waving and returning phase. The waving phase contains information about direction and distance of flower patches. The waggle dance is used as a guide or a map to evaluate merits of explored different patches and to exploit better solutions. After waggle dancing on the dance floor, the dancer (i.e., the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. A squadron moves forward into the patches. More follower bees are sent to more promising patches, while harvest paths are still explored but not in the long

[1] Common due date scheduling, OR-Library, Available at: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/schinfo.html [Accessed on 2nd April 2014].

term. This behaviour represents a swarm intelligent approach (Yuce *et al.*, 2013), which allows the colony to gather food quickly and efficiently with a recursive recruiting mechanism (Seeley, 2009). The Bees algorithm approach is inspired to such a natural communication mechanism.

The Bees Algorithm (BA) is a type of Swarm Based Optimisation Technique (SBOT) mimicking the foraging behaviour of honey bees (Pham *et al.*, 2005; Fera *et al.*, 2013; and Yuce *et al.*, 2014). Like any other optimization technique it has two basic concepts namely global and local search. The Global Search is conducted by scout bees which fly out from the hive in search of potential flower patches randomly. The returning scout bees communicate the following information to the recruit worker bees by means of the waggle dance. Information includes the direction of the source, the distance of the source from the hive and the quality of the food source (Gould 1975; and Von Frisch, 2014). This is indicated by the orientation of the bee with respect to the sun, the duration of the dance, and the frequency of the waggles in the dance and buzzing respectively (Huang, 2008). This will influence the number of recruited worker bees which will carry out a local search. Over time old patches which have been exploited fully by worker bees will be abandoned and new patches explored by scout bees for further exploitation. This process continues in an iterative manner until a stopping criteria is met. The process will become random if it is dominated by Global Search and, on the other hand, run the risk of getting stuck in a local optimum if the focus is on local or neighbourhood search. Hence a good optimization algorithm must conduct a thorough Local Search while maintaining the Global Search perspective. The BA due to its inherent nature, is expected to get stuck in local optima and in order to overcome this problem the proposed a hybrid Genetic Bees Algorithm (GBA) relies on two extra components to modify or evolve the search similar to that of Genetic Operators. These components are the Reinforced Global Search frame and a jumping function.

The standard Bees Algorithm first developed by Pham *et al.*, in 2005 requires a set of parameters as reported in Table 1: no. of scout bees (*ns*), no. of elite sites selected out of *ns* visited sites (*ne*), no. of best sites out of *ne* selected sites (*nb*), no. of bees recruited for the best *nb* sites (*nrb*), no. of bees recruited for the other *nb*-*ne* selected sites (*nrb*), initial size of patches (*ngh*). According to the flowchart in Fig. 1, the BA has the following steps: the first step is placing the '*ns*' scout bees on the

search space, and then in the next step, fitness values of the visited patches are evaluated. Subsequently the best patches with respect to their fitness value are selected and then split into two groups containing more scout bees to the elite patches '*ne*', and less scout bees to the non-elite best patches '*nb–ne'*. The next step covers the neighbourhood search in the patches given beforehand, and so according to the neighbourhood search, the patches' fitness values are evaluated. Then, the remainder bees, which are created in initial population *'ns-nb'*, will be recruited for the random search to find better random solutions. Finally, the random patches' fitness values are evaluated and this process continues until one of the stopping criteria is met: the solution found is equal to the real optimum value, the number of iterations reaches the pre-set value, if there is no significant improvement in the consecutive solutions found, e.g. stuck in local minima.

Table 1 The initial parameters of the BA.

ns	Scout bees
ne	Elite sites (with *ne<nb)*
nb	Best sites
nre	Bees in elite sites
nrb	Bees in best sites
ngh	Initial size of patches
Itr	Iterations

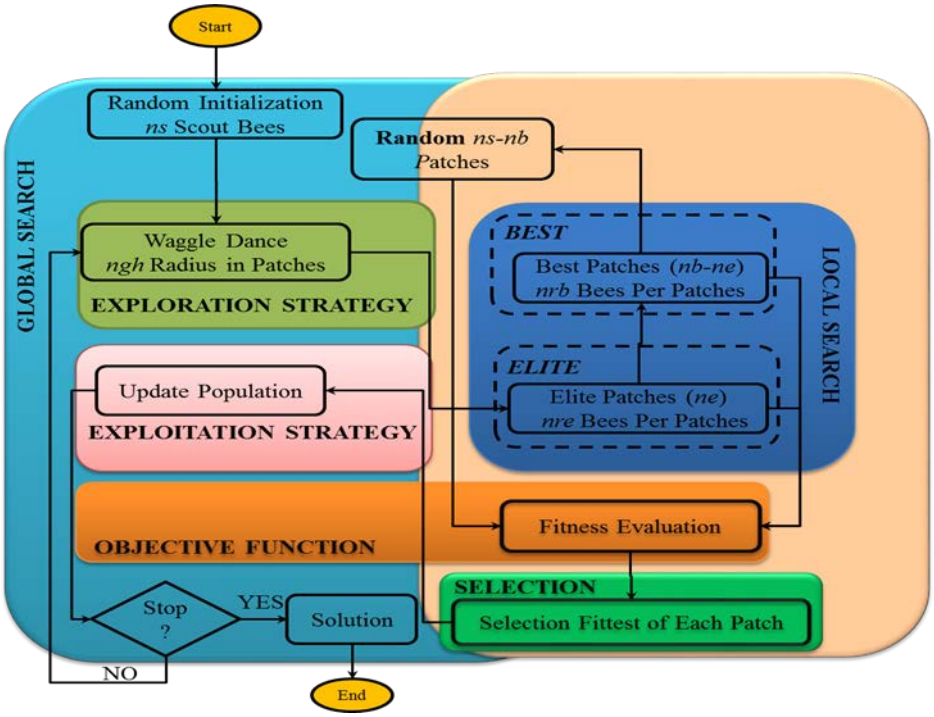


Fig. 1 The flow chart of the basic Bees Algorithm (BA).

4.2 THE BEES ALGORITHM REINFORCED WITH GENETIC OPERATORS

The weakness of the BA is associated with its inability to diversify the global search in order to explore the solutions space when the search algorithm reaches a plateau or local minima. As shown in Fig. 2, the GBA keeps the same structure as BA with the addition of two components namely Reinforced Global Search and Jumping Function. The pseudo-code of hybrid GBA is given in Fig. 3.

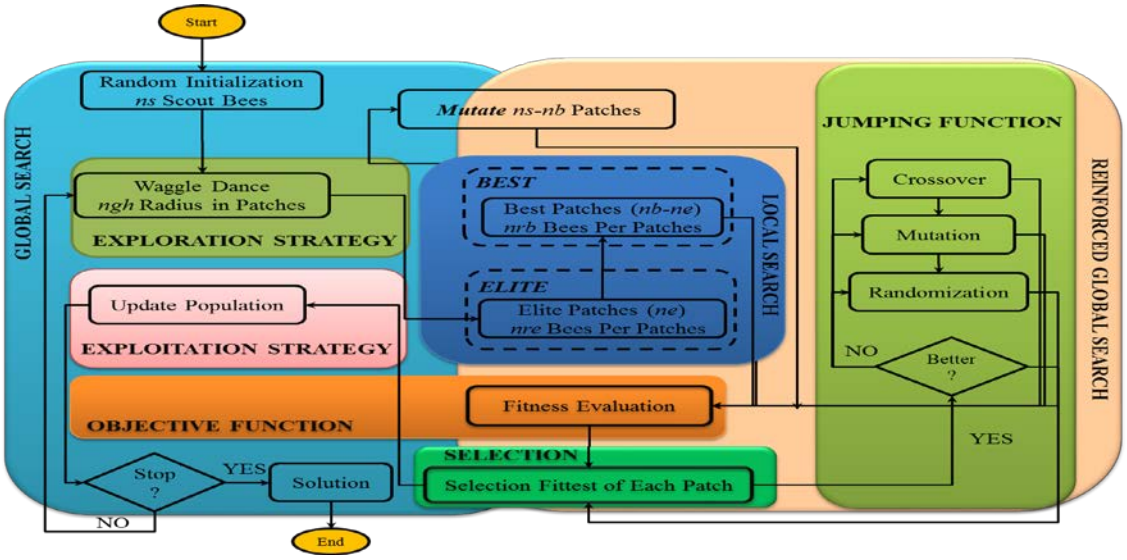


Fig. 2 The flow chart of Genetic Bees Algorithm (GBA).

Reinforced Global Search

In the GBA, the global search is enhanced by introducing a genetic mutation operator. In particular, remaining *ns-nb* bees do not perform a normal random search for the flower patches but, each flower patch where the bees would be sent is a mutation of the best food source found by the algorithm until that point. In other words, considering the analogy between the bee and the flower patch, mutating the best solution is equivalent to mutating the best bee, i.e., creating a ***Superbee***. The aim of the Reinforced Global Search is to create *ns-nb* ***Superbees*** to replace the bees in the initial population.

In the Reinforced Global Search procedure, the length of the mutated string is equal at the most to half the dimension of the vector of the solution. In this way, the mutated solution preserves at least 50% of the original solution. Furthermore, the beginning of the mutation can start from any point of the solution vector. If the algorithm finds a solution close to the optimum, the Reinforced Global Search allows it to converge faster. In other words, the Reinforced Global Search represents the frame upon which the jumping block function is built.

```
Step 1.  1.  Parameters setting: ns,ne,nb,nre,nrb,ngh,itr.
Step 2.  2.  Data set loading: load dataset
Step 3.  3.  Initial bees population generating: X=Xrandom; job(ns, njob);
         4.  Fitness function evaluation:
             F=funObj(ns,C, njob, ptime, ddate, X, data)
         5.  Ascending sorting of the values of F: [Fsorted, Xsorted]=sorting(F, X, ns)
Step 4.  6.  For 1< q <itr
         7.      For 1< i <ne
         8.          Generating, for each solution i, the neighborhood matrix MATRscout
         9.          Randomly allocating of the nre bees to the solutions of MATRscout
        10.          Generating a matrix X1 with the nre solutions related with the bees
        11.          Evaluating X1 → F1 = funObj(…)
        12           Sorting (X1 e F1) →[F1 , X1] = sorting (F1 , X1 , ne)
        13.              If the first element of F1 is minor than the i-th element of Fsorted
        14.                  updating Fsorted and Xsorted with the new found solution
        15.              End
        16.      End
        17.      For (ne+1)< i <nb
        18.          Generating, for each solution i, the neighborhood matrix MATRscout
        19.          Randomly allocating of the nrb bees to the solutions of MATRscout
        20.          Generating a matrix X2 with the nre solutions related with the bees
        21.          Evaluating X2 → F2 = funObj(…)
        22.          Sorting (X2 e F2) → [F2 , X2] = sorting (F2 , X2 , ne)
        23.              If the first element of F2 is minor than the i-th element of Fsorted
        24.              updating Fsorted and Xsorted with the new found solution
        25.              End
        26.      End
Step 5. 27.      For nb< k <ns
        28.          Generating indexes, which contains the indexes of the elements to be mutated
        29.          Mutating the best solution   GX= mutation (Xsorted(:, 1), indexes)
        30.          Evaluating GX using FX=funObj
        31.          If GX is better than Xsorted(:, 1)
        32.              Replacing → Xsorted(:, k)=GX
        33.              Replacing → Fsorted(:, k)=FX
        34.          End
        35.      End
        36.      Sorting the population of Xsorted and the vector Fsorted
Step 6. 37.      If for 10 iterations the best solution Xsorted(:, 1) does not change
        38.          While NEWsol is worse than Xsorted(:, 1) or whilestop is not met
        39.              Employing jumping to Xsorted(:, 1) obtaining NEWsol
        40.              Evaluating NEWsol
        41.          End
        42.          If NEWsol is better than Xsorted(:, 1)
        43.            Replacing Xsorted(:, 1) with NEWsol and updating Fsorted(1, 1)
        44.          End
        45.      End
        46.  End
```

Fig. 3 The pseudo code of the GBA.

Jumping Function

In reality, the global search is not enough to find the optimum of the objective function. The problem related to the Reinforced Global Function is that the *ns-nb* solutions are generated not completely randomly, but keep *alpha*% of the original solution. If the original solution is a local minimum, the algorithm will converge to this solution without considering other possible solutions which could be better. For this reason a jumping function is introduced. The jumping function acts to enhance the global search phase and include crossover (i.e., one-point and multi-point crossover are

implemented), mutation and randomization stages, working consecutively to update solution. It is composed of four steps: one-point crossover, multi-point crossover, mutation operator, and randomization as shown in Fig. 4. The jumping function starts when the algorithm gets stuck in a local minimum for a certain number of iterations. The first step consists of applying the one-point crossover operator to the best solution. If this operation is enough to fix the problem, the algorithm comes back to the main cycle updating the old solution with the new one; if the problem is not solved, the one-point crossover operator works until a certain number of iterations (or until a stopping criteria). When the stopping criteria are met, without finding any better solution, then the multi-point crossover operator starts, working with the same logic as the previous operator. If this fails, then the mutation operator is introduced. If these three operators are not able to find any better solutions, the complete randomization of the solution is used.

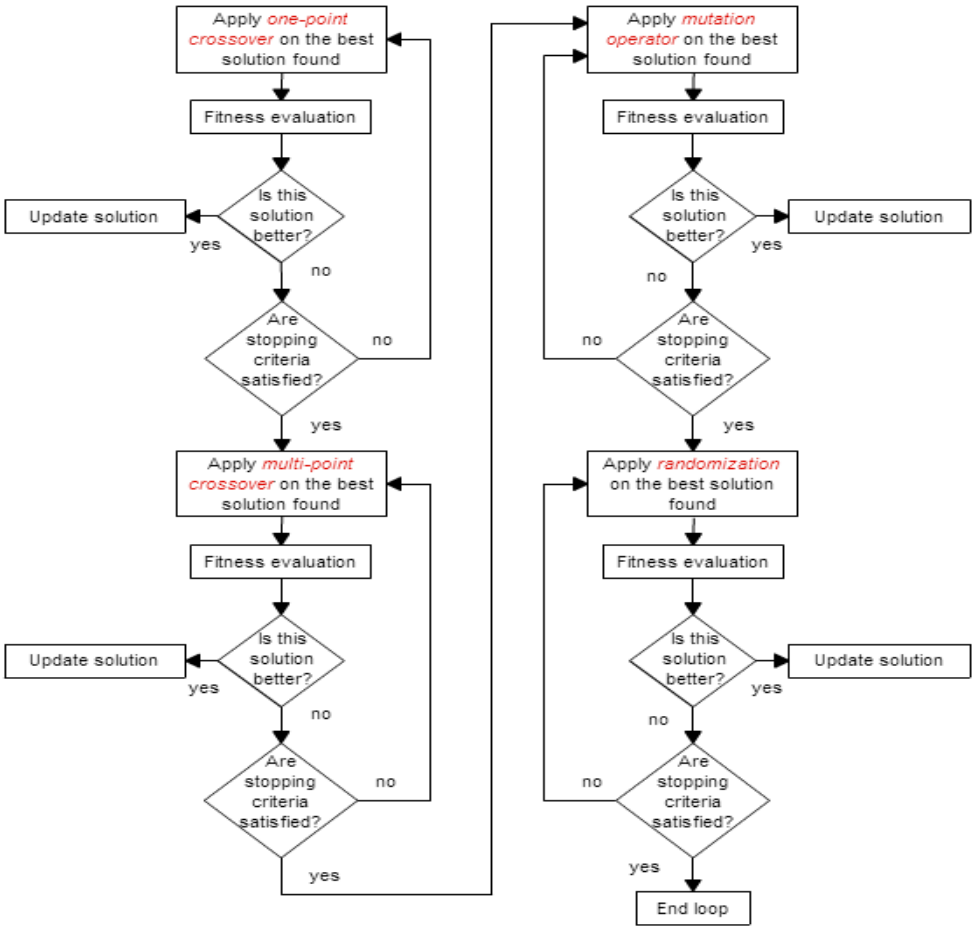


Fig. 4 The flowchart of the jumping Function used in GBA.

Finally, the output of the jumping function can either be the same as the input solution or better than

the input solution (see experiments section).

5. EXPERIMENTAL ANALYSIS AND RESULTS

5.1 TUNING PHASE AND MAIN PERFORMANCES

The numerical experiments used the set of 280 test problems proposed by Biskup and Feldman, (2001), and available on internet at (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/schinfo.html). The problem set is divided by size into 7 groups having n= 10, 20, 50, 100, 200, 500, 1000 jobs, respectively with each category containing 10 instances (*in#* is the instance amount) to be tested. The value of h= 0.2, 0.4, 0.6, 0.8 classifies the problem as less or more restricted against common due date *D*. The proposed GBA was implemented in Intel® Core™ i7 CPU @ 2.93GHz. Since the Bees Algorithm is a stochastic based method, it generally requires to report an average amount while considering %offset over different runs to have a meaningful results.

The advantage of GBA over BA is due to its power to avoid getting stuck in local minima of the objective function values. In other words, GBA performs a *Reinforced Global Search* and employs a *Jumping Function* in order to unblock the search and venture into new space when it gets stuck in local minima. As shown in figure 5 and 6, whenever both algorithms utilise the same initial parameters set for the number of patches, the number of elite and the non-elite best patches, there is a considerable gain in quality of solution at a fixed iteration number. However, the convergence performance of the both algorithms are totally different.

Fig. 5 and 6 show the comparison of performances between GBA and basic BA in a constrained test problem (h=0.2) from Biskup and Feldman, (2001) at the optimum for SMETP. *Reinforcing Global Search* procedure and *jumping mutating function* increase the capability of exploration and exploitation of the basic Bees Algorithm. While the GBA reaches the optimum value after 10 iterations, the BA needs 4000 iterations (on the same test problem under the same algorithm parameters).

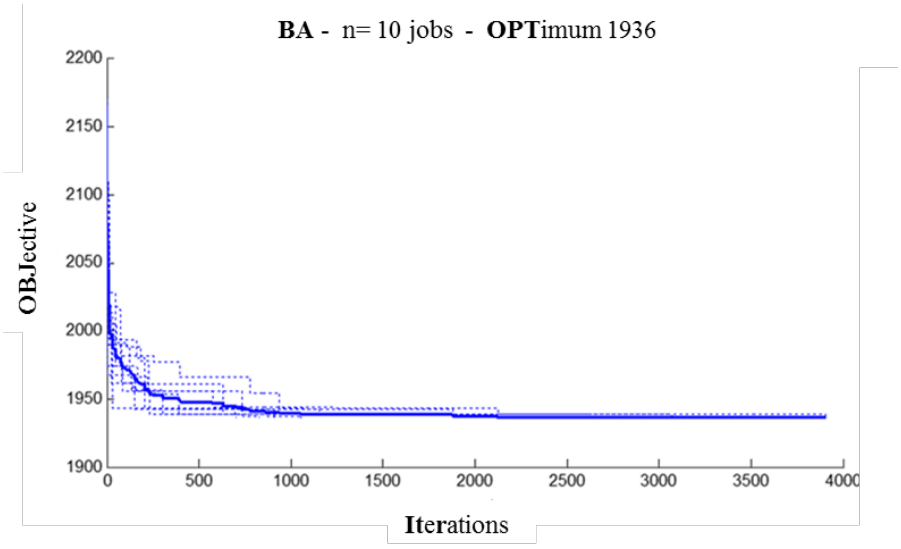


Fig. 5 The performance of the BA for n=10 and h=0.2- optimum value 1936 (tuning under parameters: *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5.

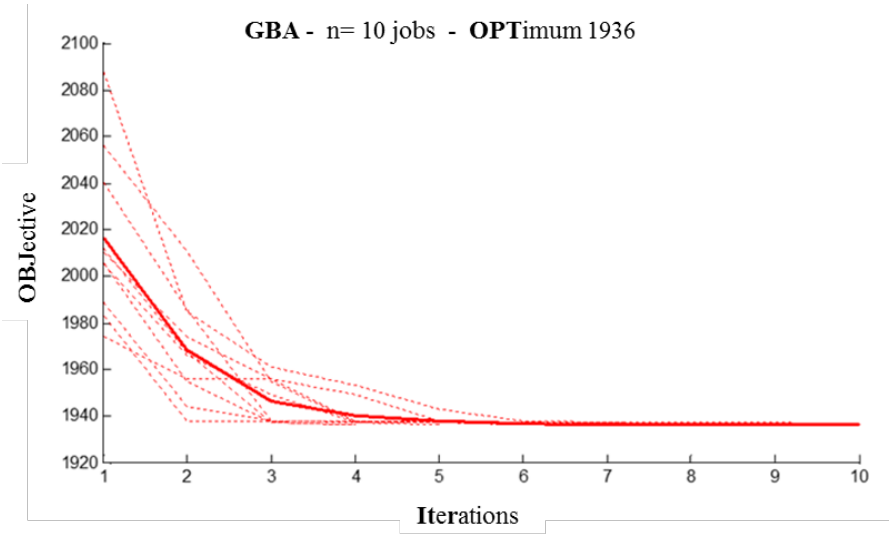


Fig. 6 The performance of the GBA for n=10 and h=0.2- optimum value 1936 (tuning under parameters: *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5).

However, the larger population size influences on exploitation in solution while gaining in exploration of the domain. The larger colony size causes the longer processing time to achieve the optimum solution (following a non-polynomial trend). As a matter of fact, a relation between computational time and number of iteration to an optimum set can be constructed as shown in Fig. 7, evolving linearly with the test data. In condition of highly restricted scheduling instances ($h \leq 0.4$), the GBA

is capable of finding the optimum solution in small size (n<100) instances with low computational effort. Time for a stable solution for benchmark instances is less or around seconds (Feldmann and Biskup, 2003). In almost all the highly restricted instances, the GBA shows superior performances compared to the meta- heuristics proposed by Biskup and Feldman, (2001), and comparable with the Upper Bound of the literature (Lin et al., 2007). However, for value of h>0.4 and number of jobs higher than 200, given the NP-hard nature, the problem required a much more strong computational effort (see Fig. 7).

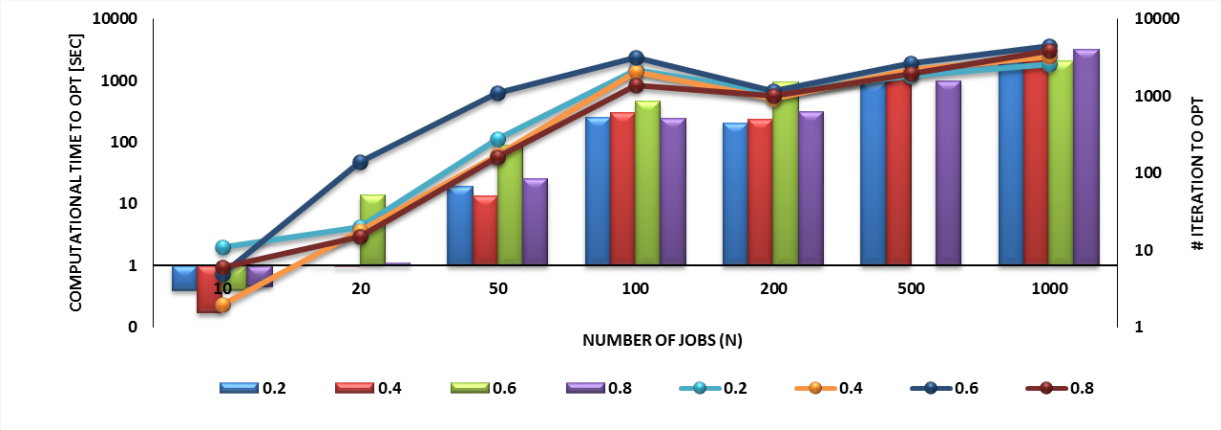


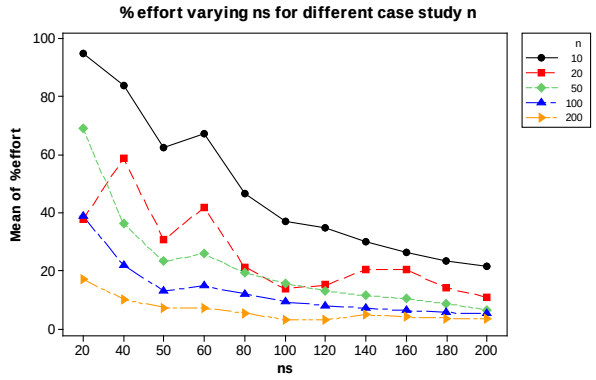
Fig. 4 Computational Time in seconds to OPT and Iteration to OPT for different SMETP instances under varying h-constraint for values as reported in Biskup and Feldman, (2001).

In figure 7, we presented the referral value of calculation time in seconds and number of iterations which the GBA achieved its best solution in the specific test problem of the restricted class (this mean convergence to a fixed optimal amount). In order to tune the GBA, we used an approach based on the analysis of the *% effort= (I_{OPT}/TI) x 100*, where I_{OPT} is the iteration at which the algorithm achieved its best solution over ten runs for a specific test problem and TI is the computation time.

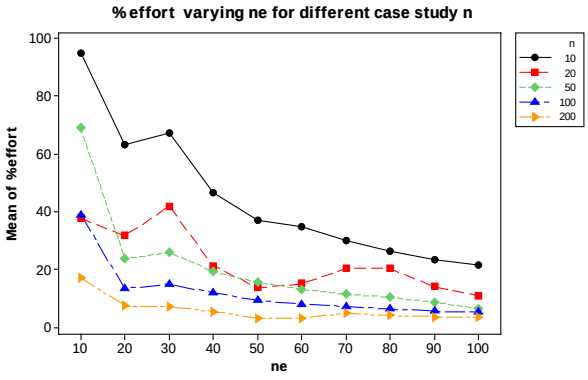
The *%effort* as defined in Nearchou, (2008) is remarkably higher (more than 30%) for the proposed GBA in comparison with the DE approach of Nearchou, (2008). *%effort* is then reported as average value among ten runs according to Bonferroni correction (Fig. 8). Results show that when the problem size increases, the time to reach an optimal solution increases even though the number of iterations remains almost constant (Fig. 7). Moreover, *%effort* seems to decrease when GBA parameters increase. This requires analysis of interactions between parameters under different case study. This

behaviour is almost similar for the basic BA, although the *%effort* is more than two order higher than the GBA as shown - from visual comparison - in Fig. 9 and 10.

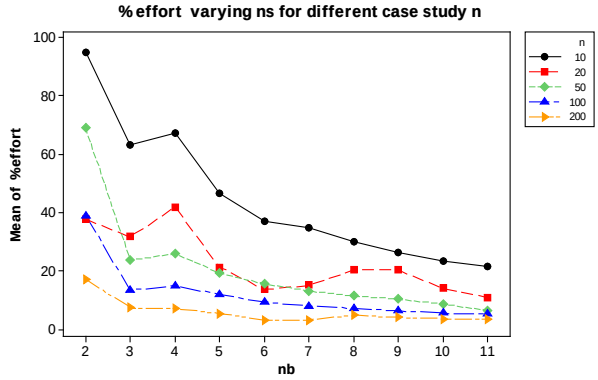
Interaction between parameters was evaluated based on Taguchi orthogonal arrays (Taguchi, 1986) and the corresponding results are shown in reports of Fig. 8. Each line corresponds to a different data test (i.e., n= 10, 20, 50, 100, 200 with h=0.2 - mean across ten instance of each class) and reported the *%efforts* in regards to various configurations. The average value of 10 repetitions is reported in Fig. 8. A larger population size will make the algorithm working more slowly but will eventually achieve a better solution. However, the correct tuning depends on the problems being solved. The run duration requires more *%effort* as the problem increases in size and the amount of parameters increases. There is an evident optimal value of *ngh* around 10 that can be used for all tests, while an optimal value of *nre* around 12 can be used. There is a suggest value of *nrb* around 8. The influence in terms of *%effort* of *nrb* and *nre* is remarkable between tests as its correspondent value increases. Generally, this remarks in *%efforts* is not so evident for *ns* and *nb* and *ne* values of 100, 50 and 8, respectively. These considerations were set in GBA for outputs as per the results section.



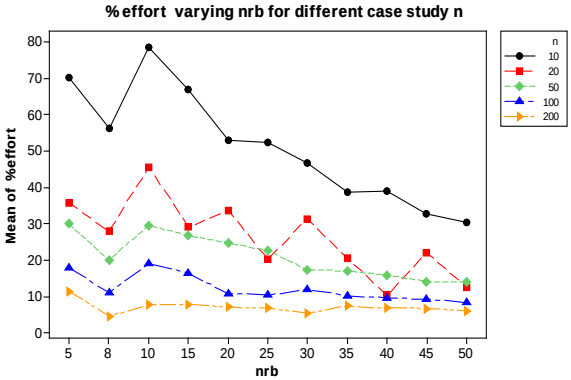
(a)



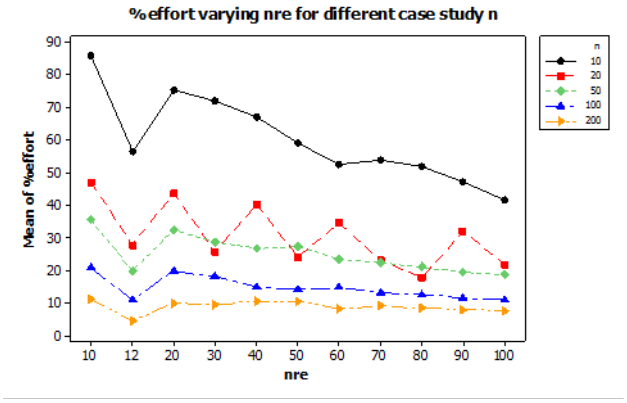
(b)



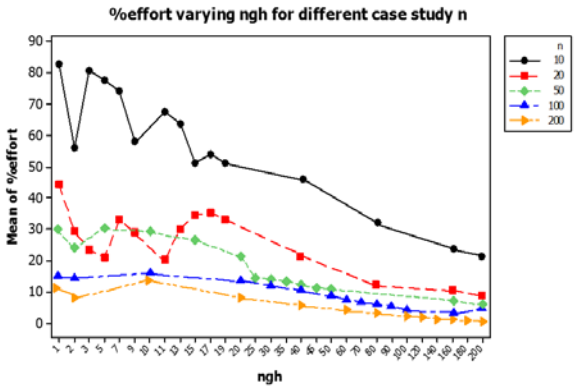
c)



d)



e)



f)

Fig. 8 %effort for different values of *ns* (a), *nb* (b), *ne* (c), *nrb* (d), *nre* (e), *ngh* (f) for different tests data. *h*=0.2.

For the restricted class n/1/ET large test instance, optimal performances could be achieved in a computational time around 45 minutes (mean across $h \leq 0.4$ value for n= 1000) that remains lower compare to the DE approach (Nearchou, 2008). However, the GBA could suffer of getting stuck in local optimum when h>0.4. For this reason, tuning become fundamental for the performances of the GBA, using a *Reinforcing Global* - it needs to be set on a great *ns* amount - *Search* and *Jumping Function* - jump based on limited *Itr* (iteration). The GBA, as explained above, manifests good performance in terms of exploitation of the domain, but suffers in local exploitation, which sometimes requires continuous jumping in new patches. As a matter of fact, *%effort* is high when the GBA parameters assume low values as shown in Fig. 7 and therefore mutation and crossover need to be increased.

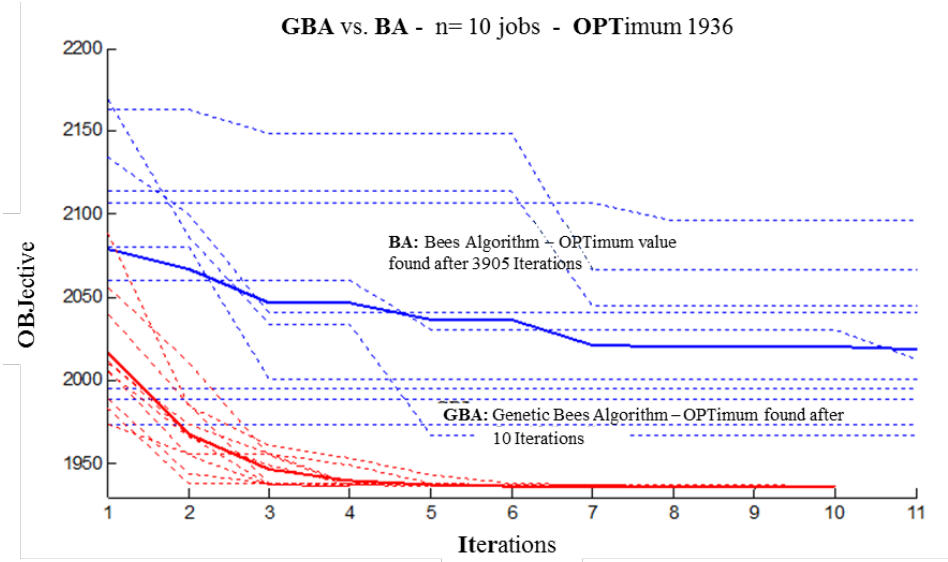


Fig. 9 BA *(blue line)* vs. GBA *(red line)* - comparison in performance under n=10 h=0.2; (tuning under parameters : *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5).

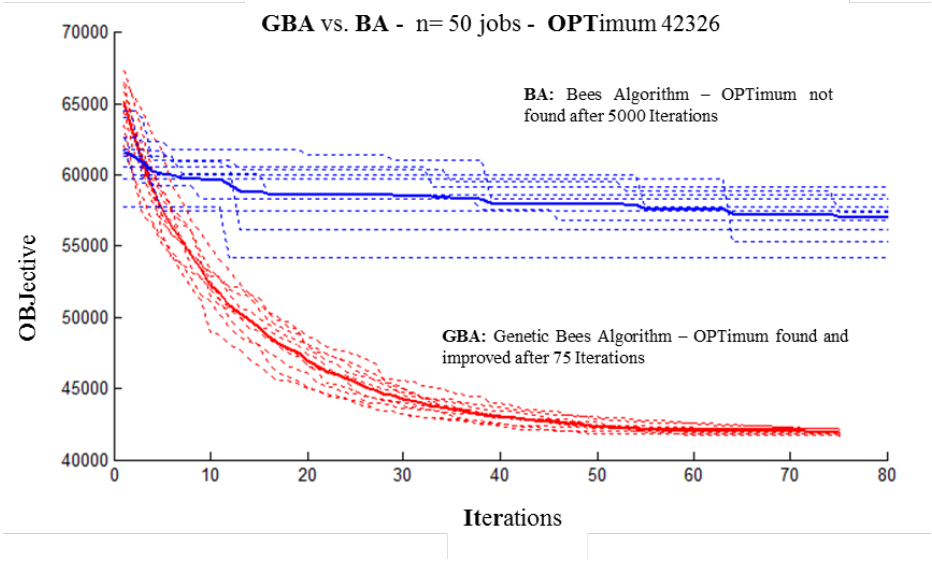


Fig. 10 BA *(blue line)* vs. GBA *(red line)* - comparison in performance under n=t 50 h=0.2;(tuning under parameters : *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5).

5.2 RESULTS

The results obtained by the GBA, applied to the instances proposed by Biskup and Feldmann, (2001), are given in table 2, 3, 4 and 5. In particular, the computation results, concerning small and large size benchmark problems, obtained by the GBA, are compared to: those obtained by the BA (under optimal tuning) and the upper bounds from Feldmann and Biskup, (2003) meta-

heuristics (tables 3 and 4); existing meta-heuristics according to literature review (see table 4 and 5). The comparison between approaches generated using different tunings, ten runs, at fixed *h* and 5000 *iterations* are reported in table 2.

Table 2 The BA vs the GBA: n={ 10 & 20 &50 &100 & 200 & 500 & 1000} - mean of ten runs - *h*=0.2. Registered performance after 5000 iterations (tuning under parameters : *ns*=50*;nb*=5*; ne*=3*;nrb*=5*; nre*=5*; ngh*=5), * as optimum.

n	**Itr**		**OBJ**			**%offset**	
	BA	GBA	UB as F&B	BA	GBA	BA	GBA
10	5000	5000	1936*	1,936	1,936	0,00%	**0,00%**
20	5000	5000	4,431	5,297	4,394	19.54%	**-0.84%**
50	5000	5000	42,363	54,334	40,642	28.26%	**-4.06%**
100	5000	5000	156,103	232,170	146,345	48.73%	**-6.25%**
200	5000	5000	526,666	905,572	498,653	71.94%	**-5.32%**
500	5000	5000	3,113,088	3,480,069	2,954,852	11.79%	**-5.08%**
1000	5000	5000	15,190,371	16,143,289	14,054,930	6.27%	**-7.47%**

Itr is the number iteration, UB as F&B is the upper boundary in Feldmann and Biskup, (2003).

The global performance of the GBA and the BA is reported by calculating the *%offset* under multiple runs. OBJ value of table 2 as reported are based on the difference between the Upper Bound cost function and the best solution found over different runs as described in Law and Kelton, (2000) with a Bonferroni correction due to multiple performance measures used by Quinzi, 2004. The convergence performance is presented in table 2 based on the following transformation: *%offset* = 100 (F_{GBA}-F_{BA})/F_{BA}. The GBA used genetic operator with multi point (two points) crossover proportionally to the size of test instances and related to its steady state in local optimum with uniform mutation (for a fixed-length with upper and lower bound according to the size of test instances and in incremental shape).

Moreover, it is decided to evaluate relative comparison between GBA and BA when they are looking for the optimum. A wider comparison is set to evaluate the performance of GBA if compared with the BA approach

For all the instances, the GBA was settled according to a Taguchi analysis on main and integration effects with: number of scout bees (*ns*) = 200; no. of sites selected out of *ns* visited sites (*nb*) = 100, number of elite sites out of *nb* selected sites (*ne*) = 11 , number of bees recruited for the best *ne* sites (*nre*) = 8, number of bees recruited for the other *nb-ne* selected sites (*nrb*) = 12, initial size of patches

(*ngh*) = 2. Table 3 reports the percentage offset over different problem size *n* over *in#* and 4 restrictive factor *h*. Multiple runs (10 for each issue) have been computed and an the best out of ten runs is used for comparison. This procedure follows as similar to methodology proposed by Hino *et al.*, (2005). The GBA algorithm was implementing using the standard tuning approach. The GBA and the BA used the same set of parameters over iteration (*Itr*). This may influence performance of the approach but it leads to a robust comparison between different test cases. The value of *h* in the header of the tables indicates the constrained shape of the instances. According to the experiments, GBA did not outperform over the BA in all instances. There are some particular instances where the BA gains better outcome - but it manifests a mean global outperformance.

Table 3 The percentage offset (mean among 10 benchmarks under different restrictive factor (*h*) of the GBA and the BA.

h	n							Mean
	10	**20**	**50**	**100**	**200**	**500**	**1000**	
0.2	0.00%	0.00%	-0.09%	-0.29%	-2.42%	-5.56%	-4.74%	-1.87%
0.4	-0.37%	-0.03%	-0.45%	-0.93%	-3.11%	-3.69%	-3.48%	-1.72%
0.6	-1.39%	-1.39%	-1.12%	-2.55%	-5.04%	-3.17%	-2.36%	-2.43%
0.8	-0.31%	-0.31%	-0.56%	-0.31%	0.00%	-4.76%	0.00%	-0.89%
Mean	-0.52%	-0.43%	-0.56%	-1.02%	-2.64%	-4.30%	-2.65%	-1.73%

The results obtained with the proposed algorithm based on the time in minutes and Itr at best - over 10 runs - (as per table 3 and 5) are presented in table 4. Computation time is about an hour for unrestricted large instances. This is longer in relation to what Lin *et al.*, (2007) presented in the Genetic Algorithm and Simulated Annealing with greedy local exploration search. These two approaches will be, hereafter, indicated as *GA/SA + greedy local search*. For the class of 1000 jobs, Lin *et al.*, (2007) obtained an average optimal value after 81.749 seconds, however it is found as 44.42 minutes using GBA. Notwithstanding, GBA- as per the results of table 5 - obtain generally the best performances in terms of quality of results.

As it is highlighted in table 4, that the GBA is quite fast for the instances up to 200 jobs, such as the average CPU time is under 0.46 seconds for small size problem (n=10), and it is found an average CPU time under 37 seconds for problems with n≤50. The corresponding average CPU time reported by Feldman and Biskup (2003) are approximatively 87.3 seconds for n≤50 jobs.

Table 4 Running times (minutes) on Intel® Core™ i7 CPU @ 2.93GHz at the best out ten runs (mean among in#) for GBA.

n	*10*		*20*		*50*		*100*		*200*		*500*		*1000*	
	Time	*Itr*	*Time*	*Itr*	*Time*	*Itr*	*Time*	*Itr*	*Time*	*Itr*	*Time*	*Itr*	*Time*	*Itr*
h=0.2	0.007	11	0.017	20	0.321	281	4.301	2152	3.457	1141	15.649	1831	39.463	2607
h=0.4	0.003	2	0.016	18	0.227	169	5.000	2078	3.936	936	19.717	2159	49.767	3275
h=0.6	0.007	5	0.231	140	1.504	1102	2.936	1554	16.055	1158	25.956	2629	34.264	4420
h=0.8	0.008	6	0.019	15	0.426	162	4.090	1359	5.215	1009	16.463	1962	54.203	3850

For an absolute remarks and the relative comparison with other meta-heuristics, it can be observed from the work of Pham *et al.,* (2011) where the data of table 5 are partially originated. This approach provides to demonstrate the absolute presentation of the proposed solutions.

Each cell of the table 5 represent the average % difference for the 10 instances (*in#*) of the corresponding size *n* and restricted factor *h.* It can be noted that the GBA outperforms the BA in all instances. The mean of difference in quality of computational solution is about -1.73% and remarkable gain is obtained when the size of the problem increases. The GBA is showing an improvement in the solution for all the scenarios and the BA is manifesting not good quality of results whatever unrestricted are implemented (h>0.4). The application of heuristics in this context is justified by the quality of the solution compared with the Feldmann and Biskup, (2003) benchmarks but for the class of problem in issue running time is sometimes an open issue compare to the shortest one (GA/SA+greedy local search).

Table 5 The maximum deviation between heuristics with regard to h - best among ten runs, mean across in# - (Best results so far in the literature are reported in bold).

n	*h 0.2*									*h 0.4*								
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**DE**	**GA/SA + greedy local search**	**BA**	**GBA**	**DPSO**	**TS**	**GA**	**HTG**	**HGT**	**DE**	**GA/SA + greedy local search**	**BA**	**GBA**
10	**0.00**	0.25	0.12	0.12	0.12	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.24	0.19	0.19	0.19	**0.00**	**0.00**	**0.00**	**0.00**
20	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-1.63**	-1.62	-1.62	-1.62	-1.62	**-1.63**	**-1.63**	**-1.63**	**-1.63**
50	-5.70	-5.70	-5.68	-5.70	-5.70	-5.69	-5.70	-5.70	**-5.71**	-4.66	-4.66	-4.60	-4.66	-4.66	-4.66	-4.66	-4.66	**-4.68**
100	-6.19	-6.19	-6.17	-6.19	-6.19	-6.17	-6.19	-6.19	**-6.21**	-4.94	-4.93	-4.91	-4.93	-4.93	-4.89	-4.94	-4.94	**-4.99**
200	-5.78	-5.76	-5.74	-5.76	-5.76	-5.77	-5.78	-5.78	**-5.92**	-3.75	-3.74	-3.75	-3.75	-3.75	-3.72	-3.75	-3.75	**-3.87**
500	-6.42	-6.41	-6.41	-6.41	-6.41	-6.43	-6.43	-6.43	**-6.79**	-3.56	-3.57	-3.58	-3.58	-3.58	-3.57	-3.58	-3.57	**-3.70**
1,000	-6.76	-6.73	-6.75	-6.74	-6.74	-6.72	-6.77	-6.76	**-7.08**	-4.37	-4.39	-4.40	-4.39	-4.39	-4.38	-4.40	-4.35	**-4.50**
Avg.	-4.96	-4.91	-4.92	-4.93	-4.93	-4.95	-4.96	-4.96	**-5.08**	-3.27	-3.24	-3.24	-3.25	-3.25	-3.26	-3.28	-3.27	**-3.34**

n	*h 0.6*									*h 0.8*								
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**DE**	**GA/SA + greedy local search**	**BA**	**GBA**	**DPSO**	**TS**	**GA**	**HTG**	**HGT**	**DE**	**GA/SA + greedy local search**	**BA**	**GBA**
10	**0.00**	0.10	0.03	0.03	0.01	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20	**-0.72**	-0.71	-0.68	-0.71	-0.71	**-0.72**	**-0.72**	**-0.72**	**-0.72**	**-0.41**	**-0.41**	-0.28	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**
50	**-0.34**	-0.32	-0.31	-0.27	-0.31	-0.32	**-0.34**	**-0.34**	**-0.34**	**-0.24**	**-0.24**	-0.19	-0.23	-0.23	**-0.24**	**-0.24**	**-0.24**	**-0.24**
100	**-0.15**	-0.01	-0.12	0.08	0.04	-0.13	**-0.15**	**-0.15**	**-0.15**	**-0.18**	-0.15	-0.12	-0.08	-0.11	-0.17	**-0.18**	**-0.18**	**-0.18**
200	**-0.15**	-0.01	-0.13	0.37	0.07	0.23	**-0.15**	**-0.15**	**-0.15**	**-0.15**	-0.04	-0.14	0.26	0.07	0.20	**-0.15**	**-0.15**	**-0.15**
500	**-0.11**	0.25	**-0.11**	0.73	0.15	1.72	**-0.11**	**-0.11**	**-0.11**	-0.11	0.21	-0.11	0.73	0.13	1.01	-0.11	-0.11	**-0.12**
1,000	**-0.06**	1.01	-0.05	1.28	0.42	1.29	**-0.06**	-0.05	-0.05	**-0.06**	1.13	-0.05	1.28	0.40	2.79	**-0.06**	-0.05	-0.05
Avg.	**-0.22**	0.04	-0.20	0.22	-0.05	0.30	**-0.22**	**-0.22**	**-0.22**	**-0.16**	0.07	-0.13	0.22	-0.02	0.45	**-0.16**	**-0.16**	**-0.16**

* DPSO: Discrete Particle Swarm Optimization; TS: Tabu Search, GA: Genetic Algorithm; HTG: Tabu Search + Genetic Algorithm, HGT: Genetic algorithm + Tabu Search; Differential Evolution; Genetic Algorithm or Simulated Annealing + greedy local search; BA: Bees Algorithm, GBA: Genetic Algorithm + Bees Algorithm.

According to Table 5, the presented results summarize the performances among problems with different *n* values for each *h* value, considering the best (out of 10 runs and mean across in#) performance results as compared in Hino *et al.*, (2005); Pan *et al.*, (2006); Lin *et al.*, (2007); and Nearchou (2008) . Hino *et al.*, (2005), proposed approaches include Tabu Search (TS), Genetic Algorithm (GA), Hybrid of Tabu search and Genetic algorithm (HTG) and Hybrid of Genetic algorithm and Tabu search (HGT). Pan *et al.*, (2006) reported the Discrete Particle Swarm Optimization algorithm (DPSO). Nearchou, (2008) used Differential Evolution (DE) as the optimization heuristic to solve this problem. Note that Feldman and Biskup, (2003) used five heuristics (namely, Evolution Search (ES), Simulating Annealing (SA), Threshold Accepting (TA) and TA with a back step (TAR)) and the best solution among heuristics is presented. Results illustrates performances in constrained instances (i.e., $h \leq 0.4$), there is a good improvement (about -0.012) in gain compare to the best (in average the BA reported, if GBA is not included, the best

among other) among the heuristics in mean - between classes - value. In highly constrained problems (h=0.2), there is generally a great difference (0.024) over the best results so far in the literature. Here GBA outperforms compare to others heuristics, if the large the population sizes (i.e., n≥ 100 test data) are utilised, there is an average of 5.92 gain in performance. When problem size increases, the GBA performances such as exploration of domain and time to convergence worsen (even though comparable with other heuristics except) compare to the best solutions with the differential evolutionary approach presented by Nearchou, (2008). In condition of h>0.4, the GBA generally confirmed its good capability in solving the problem, however the gain is null compare to the bests found by other heuristics.

It is also worth to note that the search process starts with a randomly generated population set, as for the DE approach of Nearchou, (2008), however, the solution obtained by GBA are greatly better in terms of % effort (for the case of 1000 jobs the DE approach obtained optimum in the average among in# of 141 minutes with no comparable performances in terms of average quality of objectives). Moreover, another important properties of GBA is not to have any priority rules to find the optimum solution. Since, there are three types of rules have been presented to achieve the optimum schedule in literature (Feldmann and Biskup, (2003); and Lin et al., (2007)), these rules are as following: 1) the optimum schedule would not have any idle times between consecutive jobs, 2) the optimum schedule would not have an increasing order of ratios (T_j/α_j) for the jobs completed before and starting after due date (D), and 3)an optimum schedule will be achieved with either by starting the first job at time zero or by completing one job at the time D. Most of the heuristics utilises these three priority rues to achieve the optimum schedule, however, there is no need for GBA to find the optimum schedule. Since the random initialisations and additional genetic operators allows the algorithm to search for the optimum schedule. To avoid the randomness of the quality of solution, the best of the 10 runs is considered during the experiments as shown in table 5. Since the usage of single solutions may not provide the robustness of the algorithm, however the average of the multi-results are the robust and trustable results. Based on the results presented in table 5, the performance of the GBA can be observed very table 5 clearly, that the GBA generates very high quality results in average.

Finally, as a matter of fact, when the class of the problem increases, the GBA is surpassed by the DPSO that has the advantage of minimum time convergence.

6. CONCLUSION

In this paper, a novel Genetic Bees Algorithm (GBA) has been introduced. The proposed GBA has been applied to solve the Single Machine Scheduling problem with earliness and tardiness penalties and the results show that its performance is better in most cases when *h* value is low. The Genetic Bees algorithm is a new evolutionary optimization method that has been used in a wide range of applications. In this paper it is evaluated for the optimization of different combinatorial problems under different sets.

The algorithm is conceived without inclusion of idle time between tasks and this mainly affects performances of the approach in lightly constrained ($h>0.4$) jobs. In terms of exploitation and number of iteration, the proposed metaheuristic achieves better performance compare to the upper bound and the Basic Algorithm. The hybrid Genetic Bees Algorithm has proven to be more stable and robust than the basic Bees Approach.

Possible direction for future researches include employing the same GBA approach in the case of multi-machine (m-machine) scheduling problem with general non-linear earliness and tardiness penalities to find an optimal solution in case of real test instances.

Further, an initialization procedure can also be introduced to improve the quality of solution in terms of percentage effort and in particular time for CPU time.

ACKNOWLEDGEMENTS

Authors would like to thank to all participants with their direct and indirect contributions.

REFERENCES

Abdul-Razaq, T. S., and Potts, C. N. (1988). Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society*, 39, 141–152.

Almeida, M. T., and Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers & Operations Research*, 25(7), 625-635.

Bagchi, U., Sullivan, R. S., and Chang, Y. L. (1986). Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly*, 33(2), 227-240.

Baker, K. R., and Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: a review. *Operations research*, 38(1), 22-36.

Ballestín, F., and Leus, R. (2008). Meta-heuristics for stable scheduling on a single machine. *Computers & operations research*, 35(7), 2175-2192.

Biskup, D., and Feldmann, M. (2001). Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research*, 28(8), 787-801.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11(1), 42-47.

Cheng, T. C. E. (1984). Optimal due-date determination and sequencing of n jobs on a single machine. *Journal of the Operational Research Society*, 433-437.

Cheng, T. C. E., and Kahlbacher. H. G. (1991). A proof for the longest-job first policy in one machine scheduling. *Naval Research Logistics*, 38,715-720.

Cheng, T. E. (1988). Optimal common due-date with limited completion time deviation. *Computers & operations research*, 15(2), 91-96.

Chen, Z. L. (1996). Scheduling and common due date assignment with earliness-tardiness penalties and batch delivery costs. *European Journal of Operational Research*, 93(1), 49-60.

Cheng, T. C. E., Kang, L., and Ng, C. T. (2004). Due-date assignment and single machine scheduling with deteriorating jobs. *Journal of the Operational Research Society*, 198-203.

Chen, W. Y., and Sheen, G. J. (2007). Single-machine scheduling with multiple performance measures: Minimizing job-dependent earliness and tardiness subject to the number of tardy jobs. *International Journal of Production Economics*, 109(1), 214-229.

Crauwels, H. A. J., Potts, C. N., and Van Wassenhove, L. N. (1997). Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time. *Annals of Operations Research*, 70, 261-279.

Crina, G., and Ajith, A. (2006). Stigmergic optimization: Inspiration, technologies and perspectives. In *Stigmergic optimization* (pp. 1-24). Springer Berlin Heidelberg.

Della Croce, F., Salassa, F., and T'kindt, V. (2014). A hybrid heuristic approach for single machine scheduling with release times. *Computers & Operations Research*, 45, 7-11.

Du, J., and Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of operations research*, 15(3), 483-495.

Emmons, H. (1969). One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17(4), 701-715.

Feldmann, M., and Biskup, D. (2003). Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering*, 44(2), 307-323.

Fera, M., Fruggiero, F., Lambiase, A., Martino, G., and Nenni, M. E. (2013). *Production scheduling approaches for operations management* - INTECH Open Access Publisher.

Fisher, M. L., and Jaikumar, R. (1978). An algorithm for the space-shuttle scheduling problem. *Operations Research*, 26(1), 166-182.

Franca, P. M., Mendes, A., and Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132(1), 224-242.

French, S. (1982). *Sequencing and scheduling: an introduction to the mathematics of the job-shop* (Vol. 683, p. 684). Chichester: Ellis Horwood.

Goldberg, D. E. (1989). *Genetic algorithms in search optimization and machine learning* (Vol. 412). Reading Menlo Park: Addison-wesley.

Gordon, V., Proth, J. M., and Chu, C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139(1), 1-25.

Gordon, V. S., and Strusevich, V. A. (2009). Single machine scheduling and due date assignment with positionally dependent processing times. *European Journal of Operational Research*, 198(1), 57-62.

Gould, J. L. (1975). Honey bee recruitment: the dance-language controversy. *Science*, 189(4204), 685-693.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5, 287-326.

Gupta, S. K., and Sen, T. (1983). Minimizing a quadratic function of job lateness on a single machine. *Engineering Costs and Production Economics*, 7(3), 187-194.

Gupta, S.K., and Kyparisis, J. 1987. *Single machine scheduling research*, OMEGA 15 207-227.

Hino, C. M., Ronconi, D. P., and Mendes, A. B. (2005). Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. *European Journal of Operational Research*, 160(1), 190-201.

Huang, Z. (2008). Behavioral Communications: The Waggle Dance, Available at: http http://photo.bees.net/biology/ch6/dance2.html [Accessed: 11th February 2012].

Janiak, A. (1991). Single machine scheduling problem with a common deadline and resource dependent release dates. *European Journal of Operational Research*, 53(3), 317-325.

Jolai, F., Rabbani, M., Amalnick, S., Dabaghi, A., Dehghan, M., and Parast, M. Y. (2007). Genetic algorithm for bi-criteria single machine scheduling problem of minimizing maximum earliness and number of tardy jobs. *Applied Mathematics and Computation*, 194(2), 552-560.

Kanet, J. J. (1981). Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28(4), 643-651.

Lauff, V., and Werner, F. (2004). On the complexity and some properties of multi-stage scheduling problems with earliness and tardiness penalties. *Computers & Operations Research*, 31(3), 317-345.

Law, A.M. and Kelton, W.D., 2000. *Simulation Modeling and Analysis*. New York: McGraw-Hill.

Lawler, E. L. (1979). Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4), 339-356.

Lee, C. Y., Danusaputro, S. L., and Lin, C. S. (1991). Minimizing weighted number of tardy jobs and weighted earliness-tardiness penalties about a common due date. *Computers & Operations Research*, 18(4), 379-389.

Li, G. (1997). Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, 96, 546–558.

Li, L., Fonseca, D. J., and Chen, D. S. (2006). Earliness–tardiness production planning for just-in-time manufacturing: A unifying approach by goal programming. *European Journal of Operational Research*, 175(1), 508-515.

Li, S., Ng, C. T., and Yuan, J. (2011). Group scheduling and due date assignment on a single machine. *International Journal of Production Economics*, 130(2), 230-235.

Liaw, C. F. (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research*, 26, 679–693.

Lin, S. W., Chou, S. Y., and Ying, K. C. (2007). A sequential exchange approach for minimizing earliness–tardiness penalties of single-machine scheduling with a common due date. *European Journal of Operational Research*, 177(2), 1294-1301.

Lin, S. W., Chou, S. C., and Chen, S. C. (2007). Meta-heuristic approaches for minimizing total earliness and tardiness penalties of single-machine scheduling with a common due date, *Journal of Heuristics*, 13(2), pp. 151-65.

Liu, F., Wang, J. J., and Yang, D. L. (2013). Solving single machine scheduling under disruption with discounted costs by quantum-inspired hybrid heuristics. *Journal of Manufacturing Systems*, 32(4), 715-723.

M'Hallah, R. (2007). Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers & Operations Research*, 34(10), 3126-3142.

Monden, Y. (1983). Toyota Production Systems Industrial Engineering and Management Press. *Norcross, Ga*.

Mosheiov, G., and Yovel, U. (2006). Minimizing weighted earliness–tardiness and due-date cost with unit processing-time jobs. *European Journal of Operational Research*, 172(2), 528-544.

Nearchou, A. C. (2008). A differential evolution approach for the common due date early/tardy job scheduling problem. *Computers & Operations Research*, 35(4), 1329-1343.

Nearchou, A. C. (2011). An efficient meta-heuristic for the single machine common due date scheduling problem. In *Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006* (p. 431). Elsevier.

Nogueira, J. P. C. M., Arroyo, J. E. C., Villadiego, H. M. M., and Goncalves, L. B. (2014). Hybrid GRASP Heuristics to Solve an Unrelated Parallel Machine Scheduling Problem with Earliness and Tardiness Penalties. *Electronic Notes in Theoretical Computer Science*, 53-72.

Ow, P. S., and Morton, T. E. (1989). The single machine early/tardy problem. *Management Science*, *35*(2), 177-191.

Pan, Q. K., Tasgetiren, M. F., and Liang, Y. C. (2006). A discrete particle swarm optimization algorithm for single machine total earliness and tardiness problem with a common due date. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on* (pp. 3281-3288). IEEE.

Panwalkar, S. S., Smith, M. L., and Seidmann, A. (1982). Common due date assignment to minimize total penalty for the one machine scheduling problem. *Operations research*, 30(2), 391-399.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2005). The bees algorithm. Technical note. *Manufacturing Engineering Centre, Cardiff University, UK*, 1-57.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2011). The Bees Algorithm–A Novel Tool for Complex Optimisation. In *Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006* (p. 454). Elsevier.

Quinzi, A.J., (2004) A Sequential Stopping Rule For Determining The Number Of Replications Necessary When Several Measures Of Effectiveness Are Of Interest. *In proceedings of Tenth U.S. Army Conference on Applied Statistics*.

Schaller, J. (2007). A comparison of lower bounds for the single-machine early/tardy problem. *Computers & operations research*, 34(8), 2279-2292.

Seeley, T. D. (2009). *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press.

Seidmann, A., Panwalkar, S. S., and Smith, M. L. (1981). Optimal assignment of due-dates for a single processor scheduling problem. *The International Journal of Production Research*, 19(4), 393-399.

Sen, T., Dileepan, P., and Lind, M. R. (1996). Minimizing a weighted quadratic function of job lateness in the single machine system. *International journal of production economics*, 42(3), 237-243.

Singh, A. (2010). A hybrid permutation-coded evolutionary algorithm for the early/tardy scheduling problem, *Asia-Pacific Journal of Operational Research*, 27, 713–725.

Su, L. H., and Chang, P. C. (1998). A heuristic to minimize a quadratic function of job lateness on a single machine. *International Journal of Production Economics*, 55(2), 169-175.

Sundar, S., and Singh A., (2012). A swarm intelligence approach to the early/tardy scheduling problem. *Swarm and Evolutionary Computation*, 4, 25-32.

Taguchi, G. (1986). *Introduction to quality engineering: designing quality into products and processes*.

Valente, J. M. S., and Alves, R.A.F.S. (2005). Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research*, 32(3), 557–569.

Valente, J. M. S., Gonçalves, J. F., and Alves, R. A. F. S. (2006). A hybrid genetic algorithm for the early/tardy scheduling problem, *Asia-Pacific Journal of Operational Research*, 23, 393–405.

Valente, J. M. (2008). Beam search heuristics for the single machine early/tardy scheduling problem with no machine idle time. *Computers & Industrial Engineering*, 55(3), 663-675.

Valente, J. M., and Schaller, J. E. (2012). Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. *Computers & Operations Research*, 39(9), 2223-2231.

Von Frisch, K. (2014). *Bees: their vision, chemical senses, and language*. Cornell University Press.

Wang, X. Y., and Wang, M. Z. (2010). Single machine common flow allowance scheduling with a rate-modifying activity. *Computers & Industrial Engineering*, 59(4), 898-902.

Webster, S.T. (1997).The Complexity of Scheduling Job Families about a Common Due Date. *Oper. Res. Lett*. 20, 65 – 74.

Yang, D. L., Lai, C. J., and Yang, S. J. (2014). Scheduling problems with multiple due windows assignment and controllable processing times on a single machine. *International Journal of Production Economics*, 150, 96-103.

Yeung, W. K., Oguz, C., and Cheng, T. E. (2001). Single-machine scheduling with a common due window. *Computers & Operations Research*, 28(2), 157-175.

Yuce, B., Packianather, M. S., Mastrocinque, E., Pham, D. T., and Lambiase, A. (2013). Honey bees inspired optimization method: the Bees Algorithm. Insects, 4(4), 646-662

Yuce, B., Mastrocinque, E., Lambiase, A., Packianather, M. S., and Pham, D. T. (2014). A multi-objective supply chain optimisation using enhanced Bees Algorithm with adaptive neighbourhood search and site abandonment strategy. *Swarm and Evolutionary Computation*, 18, 71-82.