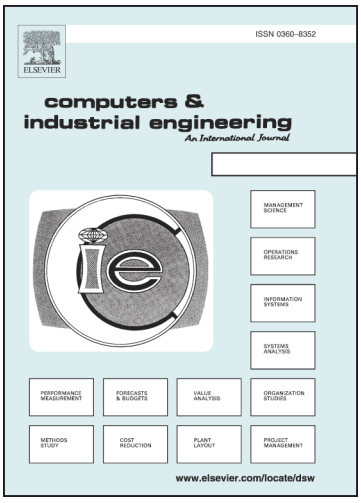
Hybrid Genetic Bees Algorithm applied to Single Machine Scheduling with Earliness and Tardiness Penalties

Yuce, B, Fruggiero, F, Packianather, MS, Pham, DT, Mastrocinque, E, Lambiase, A & Fera, M

Original citation & hyperlink:
Yuce, B, Fruggiero, F, Packianather, MS, Pham, DT, Mastrocinque, E, Lambiase, A & Fera, M 2017, 'Hybrid Genetic Bees Algorithm applied to Single Machine Scheduling with Earliness and Tardiness Penalties' *Computers & Industrial Engineering*, vol 113 pp. 842-858
https://dx.doi.org/10.1016/j.cie.2017.07.018

DOI 10.1016/j.cie.2017.07.018
ISSN 0360-8352

Publisher: Elsevier



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

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

PII: S0360-8352(17)30320-0
DOI: http://dx.doi.org/10.1016/j.cie.2017.07.018
Reference: CAIE 4825

To appear in: *Computers & Industrial Engineering*

Received Date: 2 June 2016
Revised Date: 14 July 2017
Accepted Date: 15 July 2017

Please cite this article as: Yuce, B., Fruggiero, F., Packianather, M.S., Pham, D.T., Mastrocinque, E., Lambiase, A., Fera, M., Hybrid Genetic Bees Algorithm applied to Single Machine Scheduling with Earliness and Tardiness Penalties, *Computers & Industrial Engineering* (2017), doi: http://dx.doi.org/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], M.S. Packianather[3], D.T. Pham[4], E. Mastrocinque[5] A. Lambiase[6] and M. Fera [7]*

[1] *Innovation Centre, College of Engineering, Mathematics and Physical Sciences, Streatham Campus, University of Exeter, EX4 4QJ, Exeter, UK,*

[2] *School of Engineering, University of Basilicata, Via Ateneo Lucano 10, 85100 Potenza, Italy,*

[3] *High Value Manufacturing Group, School of Engineering, Cardiff University, CF24 3AA Cardiff, UK,*

[4] *School of Mechanical Engineering, University of Birmingham, B15 2TT, Birmingham, UK*

[5] *Faculty of Engineering, Environment and Computing, Coventry University, Priory Street, CV1 5FB, Coventry, 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: b.yuce@exeter.ac.uk**

ABSTRACT This paper presents a hybrid Genetic-Bees Algorithm based optimised solution for the single machine scheduling problem. The enhancement of the Bees Algorithm (BA) is conducted using the Genetic Algorithm's (GA's) operators during the global search stage. The proposed enhancement aims to increase the global search capability of the BA gradually with new additions. Although the BA has very successful implementations on various type of optimisation problems, it has found that the algorithm suffers from weak global search ability which increases the computational complexities on NP-hard type optimisation problems e.g. combinatorial/permutational type optimisation problems. This weakness occurs due to using a simple global random search operation during the search process. To reinforce the global search process in the BA, the proposed enhancement is utilised to increase exploration capability by expanding the number of fittest solutions through the genetical variations of promising solutions. The hybridisation process is realised by including two strategies into the basic BA, named as "*reinforced global search*" and "*jumping function*" strategies. The *reinforced global search* strategy is the first stage of the hybridisation process and contains the mutation operator of the GA. The second strategy, *jumping function* strategy, consists of four GA operators as single point crossover, multipoint crossover, mutation and randomisation. To demonstrate the strength of the

proposed solution, several experiments were carried out on 280 well-known single machine benchmark instances, and the results are presented by comparing to other well-known heuristic algorithms. According to the experiments, the proposed enhancements provides better capability to basic BA to jump from local minima, and GBA performed better compared to BA in terms of convergence and the quality of results. The convergence time reduced about 60% with about 30% better results for highly constrained jobs.

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

1. INTRODUCTION

The recent technological developments in production and manufacturing systems have created a substantial opportunity to increase the production and manufacturing capability of the organisations in order to meet customers' expectations like the desired product quality and the delivery date. In literature, the most popular production and manufacturing systems are found as lean manufacturing, flexible manufacturing system, fit manufacturing and just in time (JIT) (Belgran *et al.,* 2010). Although these manufacturing systems comprise with robust and well-defined philosophies, in practice, unexpected problems like the early or late product delivery times generates a negative influence on the organisations and their total manufacturing costs. Therefore, several strategies have been proposed in literature to increase the productivity in manufacturing systems including technology strategy, location strategy, strategy of validity integration and capacity strategy (Dombrowski et al., 2016).

A well-defined production planning methodology provides a great opportunity to utilise effective manufacturing strategies, which involves the information about machining operations, their sequence, tools, setup requirements and technological parameters (Yanushevsky, 1992). One of the main difficulty to utilise an effective production planning is to determine the uncertainty of the demand, hence, an effective master production schedule (MPS) can be achieved (Gessner, 1986). However, the MPS can be realised with optimised schedules, but the scheduling optimisations are classified as the NP-hard type problems (Pinedo, 2012), which aims to optimise the operations' schedules, where

these schedules mostly suffer from the bottlenecks due to machine capacity and jobs' completion times. In literature, the machine scheduling and the determination of the optimum jobs' orders have been tackled since 1980s.

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 literature, Genetic Algorithm is utilised individually to tackle this problem, however, random nature of the algorithm is one of the drawbacks to determine the optimum solution for the scheduling problem (2012). Hence, they have suggested to increase the efficiency of the algorithm by combining other algorithms, especially to increase the local search property. In addition, Pham *et al., (*2007) presented a single machine scheduling solution using the Bees Algorithm. However, the proposed solution suffered from being stuck in nonpromising patches which required some perturbations to avoid this weakness. Hence, this paper presents an enhancement of the Bees algorithm by combining genetic operators in order to solve the SMET problem.

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. 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: section two presents the detailed background information and literature on the single machine scheduling problem and applied heuristics solutions; section three illustrates modelling of the single machine scheduling problem. In section four, the Bees algorithm and its enhancements with the genetic operators are presented, section five presents the experimental analysis and results about the proposed technique including the tuning phase of the GBA and its application on benchmarks problems. Performances were evaluated 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. Finally, the conclusion and detailed discussions are included in section six.

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. Jaegyun, (1999) stated that an ideal schedule is the one where the all jobs finish on their assigned due date. However, it is not an easy task to determine the correct order to assign the jobs due the complex nature of the problem. Considering the job scheduling problem within a completion time frame, the scheduling problems classified as one of the *NP-hard* type problems (Lee *et al.,* 1991). Moreover, there is also no a certain algorithm to address all types of scheduling problem however researchers are trying to address different types of problems with different search algorithms to find an optimum solution and increase the search quality such as, Cheng, (1984) demonstrated the anticipation for the jobs' completion under certain production conditions and definition of an optimal due date assignments for each job. Single machine scheduling problem, considering lateness performance under constrained due date, was firstly discussed by Gupta and Kyparisis, (1987). Complexity of this problem was studied in the work of Lauff and Werner, (2004), and, 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 machines flow shop environment in the case of restrictive and non-restrictive due date and it is proven that flow shop environment is *NP-hard* in the strong sense. In addition, Bagchi *et al.,* (1986) was asserted that the 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 other parameters as constant. Therefore, there is a time-period in which the products should be completed on time and then the customers' deliveries are arranged. In general, the restricted common due date types problems are much more difficult to solve compared to non-restricted types (Baker and Scudder, 1990). Further, the optimal penalty cost cannot decrease with the increase in the common due date in the single machine scheduling (Webster, 1997). For the case of single machine scheduling under due date penalties, Kanet, (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.

In addition to the above, 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 which 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). 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). In addition, common due date allows managers to reduce the production cost and control the organisation financial status by assigning the most appropriate jobs in the manufacturing operations (Gordon *et al.,* 2002). However, the selection of the appropriate task is highly complicated process which requires advanced solution to be implemented. 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 by 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.,* (2007a); Nearchou, (2008); Gordon and Strusevich, (2009); Wang and Wang, (2010); 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.,* (2007b); 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.*, (2007a); 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-heuristic approaches are mainly introduced as the solution methods. The total tardiness/earliness problem was first studied by Emmons in late sixties (Emmons, 1969). Until the early 70s, most of the studies presented in this field were mainly practice oriented, and 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 results were obtained in a 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 adjustments are 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 on Branch & Bound and Genetic operators were reported in the work of Ballestin and Leus, (2008) based on the start time deviation between planned time and actual time. Beams search heuristics with recovery procedures is used in the work of Valente, (2008) with optimal performance for small and medium SMETP instances. According to experiments of Valente, (2008), an excessive computational time was required for medium and large (more than 75 jobs) instances, when the pre-evaluation in beam was included based on dispatching rules. A 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 was proposed by Liu *et al.,* (2013); and Jolai *et al.,* (2007). A combination of GA with 14 local search and initialization procedures were developed and tested on the randomly generated instances in Valente *et al.,* (2006). They demonstrated that, behind the quality of results, the combination of fitness evaluation and GA was greatly accelerating the convergence, and reduced number of iterations and computational time at nearby optimal schedule 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 method used for SMETP is memetic approach, presented by Franca *et al.,* (2001). Greedy Randomized Adaptive Search Procedure (GRASP) was used in Norgueira *et al.,* (2014). Heuristics based on mathematical programming was proposed by Della Croce *et al.,* (2014), to obtain better performances for the large-scale problems. A combination of local search heuristics, using dispatching and hill climbing and simulated annealing, with evolutionary algorithm was proposed by M'Hallah, (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 was 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 was 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 was utilised the random generated SMTEP instances. Finally, complete survey of heuristic methodologies for solving SMETP was 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).

This paper demonstrates the performances of an enhanced hybrid version of the Bees Algorithm, called Genetic Bees Algorithm (GBA). Since the basic Bees Algorithm may have limitation to converge the optimum solution in the desired time scale by Yuce *et al.,* (2014a). The genetic algorithm operators like 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 are defined in section 4.2. Similar approach is proposed by Ming *et al.,* (2011) to increase the efficiency of the Bee Colony Algorithm, they have utilised GA to increase the local search capability of the Bee colony algorithm. However, the main weakness of the BA is the global search stage, therefore, this paper focuses on the enhancements of the global search stage. 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, results will be benchmarked 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.,* (2007a).

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 optimisation 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 characterises a deterministic scheduling as part of the combinatorial optimisation. In the following part, it is presented 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 the time zero (Cheng and Kahlbacher, 1991). Close jobs are necessary but not a sufficient condition for the optimisation. Here, the complexity is related more to the arbitrary starting date than to the close sequence of jobs. The restrictive form of SMETP is also much more complex than the unrestricted form, given the NP-hard nature of the problem (i.e., excluding the 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. In this study, seven benchmark data files are utilised with different job numbers, n, which are equal to 10, 20, 50, 100, 200, 500, 1000 jobs, and under different restricted (h = 0.2 and h = 0.4) and unrestricted (h = 0.6 and h = 0.8) constraints with due date (D_k) on one machine (k=1). Each job should be processed on one machine, further, the individual earliness E_j and tardiness T_j penalties are given for each job, which will be included in the objective function, if a job is finished before or after the common due date D[1].

4. THE ENHANCEMENT OF THE BEES ALGORITHM WITH GENETIC OPERATORS

4.1 THE BEES ALGORITHM

A colony of honey bees exploit, in multiple directions simultaneously, food sources in the form of

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

antera with plentiful amounts of nectar or pollen. They can conduct this foraging exploitation up to 11 kilometres far from their hives on multiple directions (Gould, 1975). Flower patches are marked based on a virtual stigmergic approach – sites with higher nectar content should be visited by more bees (Crina and Ajith, 2006). The foraging strategy starts with 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 are 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 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.,* 2014a). The algorithm conducts a global and a local search process to determine the global optimum solution. According to literature, most of the optimisation algorithms are not capable to utilise both of these search process, they utilise either a local search or a global search individually. Hence, the Bees Algorithm can search the entire solution space randomly and focuses on the promising regions. *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 optimisation 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: number of scout bees (*ns*), number of elite sites selected out of *ns* visited sites (*ne*), number of best sites out of *ne* selected sites (*nb*), number of bees recruited for the best *nb* sites (*nrb*), number 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
nb	Best sites
ne	Elite sites (with *ne<nb)*
nre	Bees in elite sites
nrb	Bees in best sites
ngh	Initial size of patches
Itr	Iterations

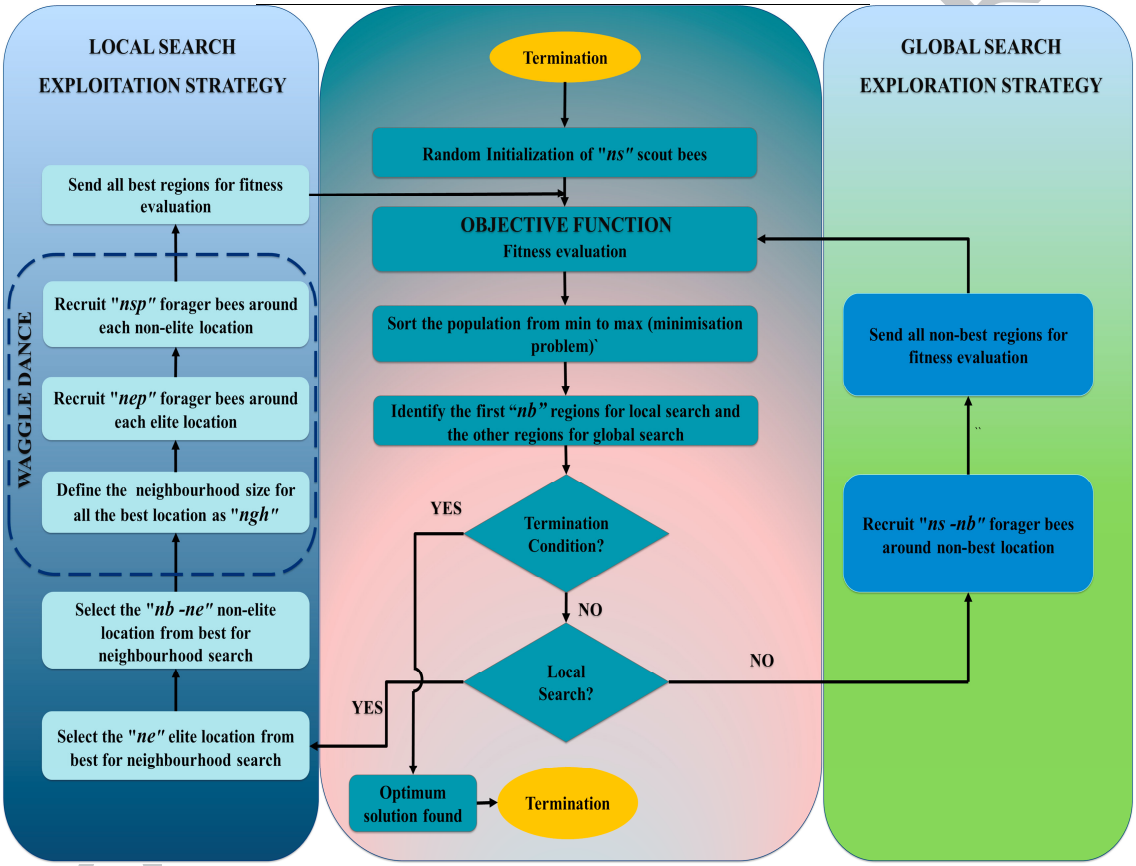


Fig. 1 The flow chart of the basic Bees Algorithm (BA).

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 reinforced global search and

jumping function strategies. The reinforced search utilises a genetically mutated approach if there is no optimum solution with the basic BA. In addition, the jumping function utilises single point crossover, multipoint crossover, mutation and randomisation operators step by step if there is no improvement with previous operators and strategies. The pseudo-code of hybrid GBA is given in Fig. 3.

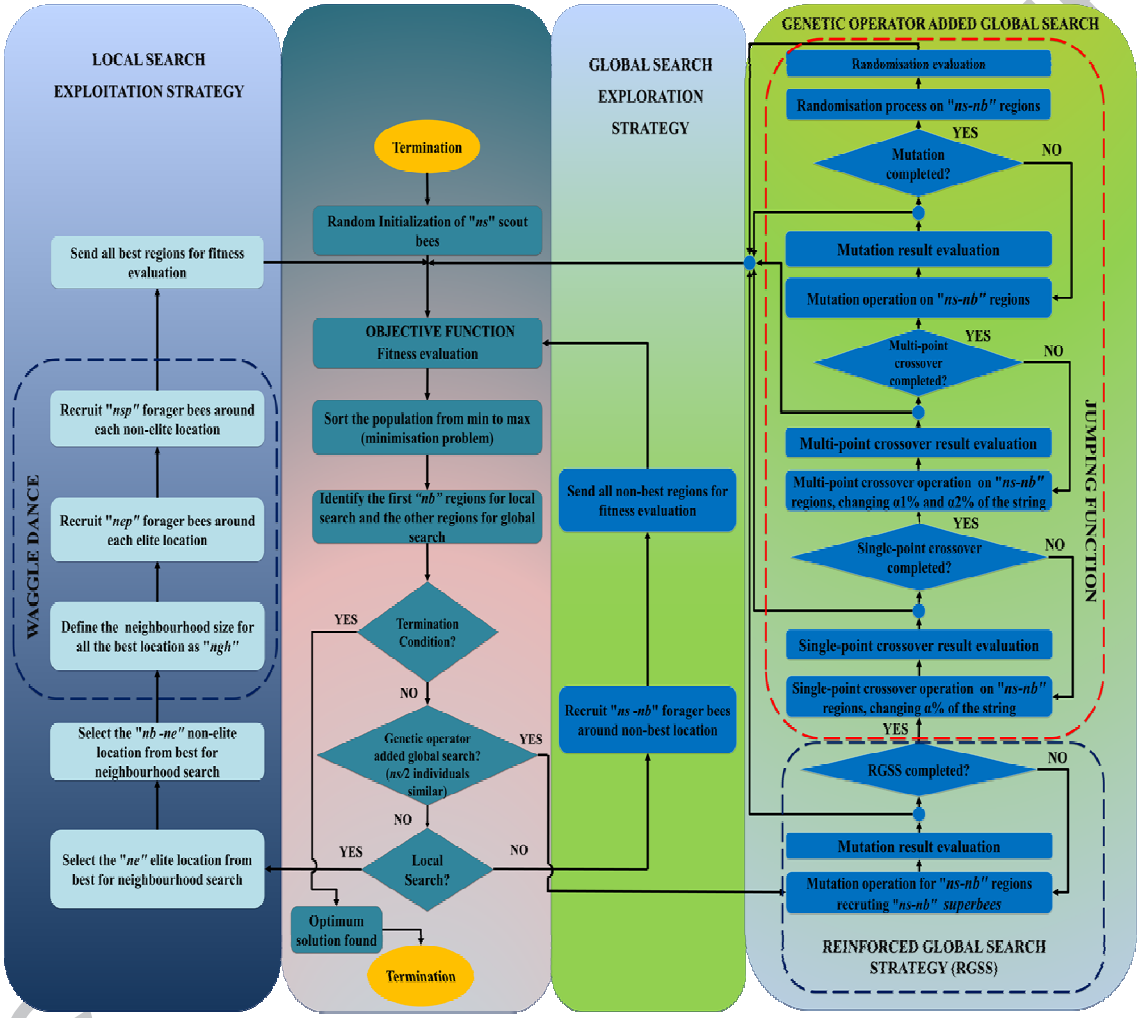


Fig. 2 The flow chart of Genetic Bees Algorithm (GBA).

4.2.1. Reinforced global search strategy

The *reinforced global search strategy* is one of the proposed strategy in this work which utilises the mutation operator of the genetic algorithm to enhance the BA. In the proposed GBA, the global search is enhanced by introducing a genetic mutation operator. This operator is activated if the 50% of the initial population remains the same using the basic BA. The proposed strategy

implements the mutation operator on the locations which are the non-best *ns-nb* patches. The main idea is to generate a best solution from inefficient solutions through the mutations. Considering the analogy between the bee and the flower patch, generating a best solution is equivalent to generating the best bee, which is called a ***Superbee***. The aim of the *reinforced global search* strategy is to create *ns-nb* ***Superbees*** to replace the bees in the initial population. In the *reinforced global search* strategy starts, the length of the mutated string contains the string value of at least half the dimension of the original solution vector. Thus, 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, randomly. If the algorithm finds a solution close to the optimum, the solution will increase the convergence rate of the algorithm, However, the operator of the *reinforced global search* strategy is conceived in order to generate offspring from the current population starting from elitism but then applying combination between crossover and mutation operators (Gen and Cheng, 1997). In this procedure, the best individuals are obtained to the date that are preserved, so that the algorithm can report, after applying *reinforced global search,* the best value is identified. This is, as per literature, a commonly used approach (Lin *et al.,* 2007a). The quality of each solution is measured by the fitness and the search space proceeds until the termination condition or enhancement is met.

However, if there is no improvement with this strategy, then the second strategy function, jumping function strategy, will be activated until an optimised solution is found. In the following section, the details about the jumping function strategy is presented.

4.2.2. Jumping function strategy

In complex NP-hard type functions, the elite global search may not be enough to find the optimum solution, this is due to the lack of complete randomness, as stated previously. Moreover, if the optimum solution is not in the vicinity of one of the existing locations, the BA is not able to converge to this solution without exploring other promising solutions. Hence, another function, called, *jumping function* strategy is proposed in this section using the GA's crossover (single-point and multi-point), mutation and randomisation operators which is anticipated to increase the strength and randomness of the existing solution if there is no solution with the reinforced global search strategy. The jumping function

aims to enhance the global search of the BA and includes initially a crossover operation as stated in Fig. 2 (i.e., single-point and multi-point crossover are implemented), then a mutation operation and finally a randomization operation with consecutive evaluations. The main steps of the proposed strategy are one-point crossover, multi-point crossover, mutation operator, and randomization as shown in Fig. 2. The main assumption of the jumping function is that if promising solutions are found, the global optimum can be achieved faster solution with these solutions. To explain steps of the jumping function, an example is selected as shown in Fig. 4. The selected example is presented for the test instance of 20 jobs under highly restricted condition (common due date = 43).

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 X_1 with the nre solutions related with the bees
11. Evaluating $X_1 \rightarrow F_1 = funObj(\ldots)$
12 Sorting $(X_1 \text{ e } F_1) \rightarrow [F_1, X_1] = sorting\ (F_1, X_1, ne)$
13. **If** the first element of F_1 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 X_2 with the nre solutions related with the bees
21. Evaluating $X_2 \rightarrow F_2 = funObj(\ldots)$
22. Sorting $(X_2 \text{ e } F_2) \rightarrow [F_2, X_2] = sorting\ (F_2, X_2, ne)$
23. **If** the first element of F_2 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** *NEW*sol is worse than *Xsorted(:, 1)* or *whilestop* is not met
39. Employing *jumping* to *Xsorted(:, 1)* obtaining *NEW*sol
40. Evaluating *NEW*sol
41. **End**
42. **If** *NEW*sol is better than *Xsorted(:, 1)*
43. Replacing *Xsorted(:, 1)* with *NEW*sol and updating *Fsorted(1, 1)*
44. **End**
45. **End**
46. **End**

Fig. 3 The pseudo code of the GBA.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
tj	3	11	5	6	7	1	5	7	19	18	19	2	17	14	5	18	18	18	12	14
αj	3	7	1	5	8	1	4	6	7	6	9	9	4	9	8	9	2	9	9	9
βj	9	15	8	3	12	6	4	8	1	12	12	10	4	12	4	7	1	7	4	2
dj	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43

Fig. 4 The selected example to demonstrate the jumping function strategy.

According to the Fig. 4, the row starting with j denotes the job number, t_j is the process time for the job j, the row starting with α_j symbolizes the earliness, the row starting with β_j denotes the tardiness, and the row start with d_j denotes the common due date of the job j.

A) Single-point crossover operation

This operator is the first stage of the jumping function strategy. The operator activates this stage if there is no solution with the reinforced global search strategy which utilises only the mutation operation. This operation starts with the random selection of two parent job string from the solution space, and the random crossover distance which can be at most the half length of the job string (α% exchange< 50%). The process will continue to generate two children chromosomes from their parents by the job string exchange between the parent chromosomes. The child chromosome will be accepted if the fitness results are better than any parent chromosome and other child chromosome. To illustrate this process, an example is presented in Fig. 5. According to this figure, two parent chromosomes are selected as S and T with 4401 and 4498 fitness values, and a random crossover distance is found as 30% (from job 1 to job 6). After the gene exchanges, the child chromosomes are found as ST1 and ST2 with the fitness values of 4494 and 4406, respectively. Based on this figure, the fitness value of ST2 job string is found better than parent T and child ST1, hence, the job string of the parent T is replaced with the job string of ST2 in the gene pool. Hence, the further computation will be conducted with the chromosomes T and ST2.

Parent Chromosomes

S																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Objective Value
3	10	2	5	12	6	1	8	15	7	14	11	4	18	16	19	13	20	17	9	4402

T																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Objective Value
3	10	2	6	5	12	7	15	1	14	8	11	4	18	16	19	13	20	17	9	4498

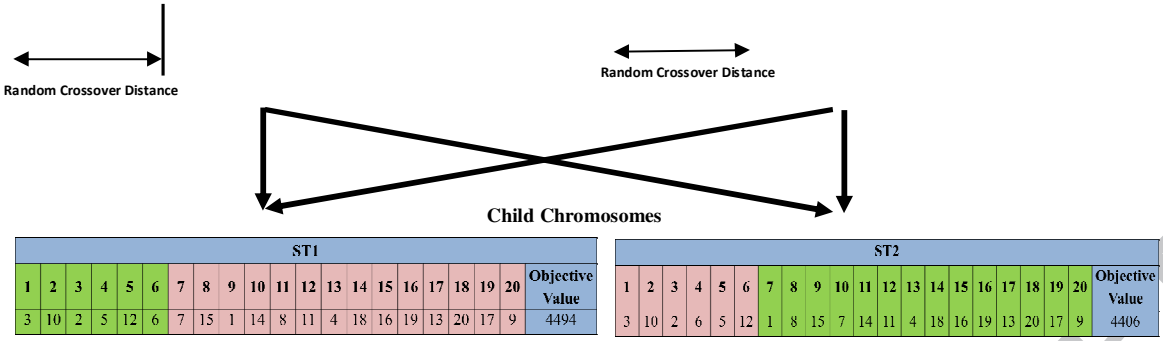
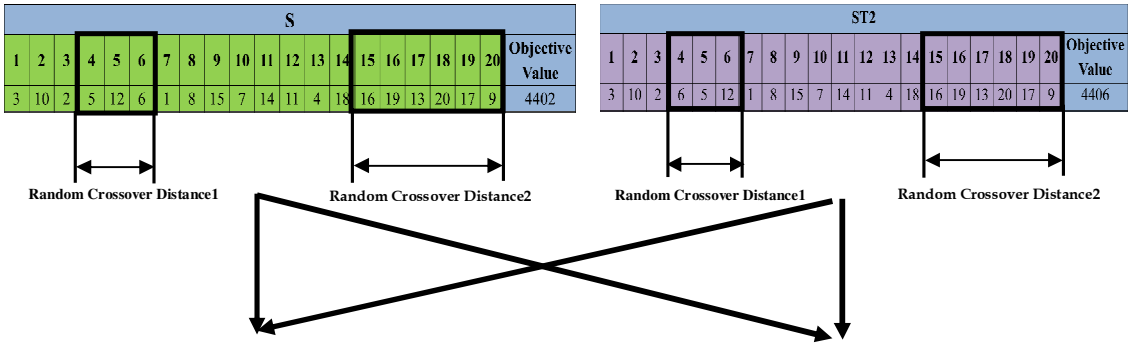


Fig. 5 An example to demonstrate the single point crossover operation.

B) Multi-point crossover operation

This operator is the second GA operators utilised in the jumping function strategy which performs if the optimum solution or the improvements in both parent chromosomes are not achieved at the previous step. The strategy implemented on the parent chromosomes. It is proposed two points crossover operation approach. The process initiates with the parent chromosome selection. The next stage is to determine the random crossover distances string (α1% and α2% exchanges, and α1% + α2% < 50%). Based on these two information, the child chromosomes can be generated. To demonstrate the proposed multi-point crossover operation, the previous example will be utilised. As stated in the previous stage that one of the parent chromosome was updated. Hence, the process will be implemented on the updated parents which are chromosomes S and ST2. The second step is to determine the random crossover distances. The first random crossover distance is found to be started from forth string with 15% ending gene point, (sixth gene) and second random crossover distance is found as fifteenth gene in the string with 30% ending gene point (twentieth gene) as shown in Fig. 6.

According to Fig. 6, the strings of the child chromosomes are found as identical with the parents hence there is no update in the solution pool. Hence the next operation will be utilised.



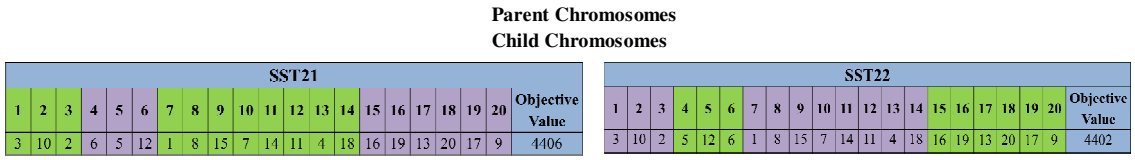


Fig. 6 An example to demonstrate the multi point crossover operation.

C) Mutation operation

The mutation operation is another operation to find the optimum solution with the genetically mutated genes. The process starts with the selection of the original chromosome. Following to the chromosome selection, the mutation rate and the selection of the gene strings which need to be mutated need to be found. The process is completed with the gene updates and the updated chromosomes evaluation stages. To demonstrate the mutation operation, one of the parent chromosome, which performed the worst in the crossover operation, will be utilised, which is the chromosome *ST2*. The mutation rate is found as 15% which is equal to three genes in the chromosome. These three genes are found as genes 7, 8 and 10. A random mutation exchange is determined between these genes as following: the gene 7 becomes gene 8, the gene 8 becomes gene 10 and the gene 10 becomes gene 7 with the random allocations. After the gene exchange the fitness value of the new chromosome is found as 4501, as shown in Fig. 7, which is higher than the original gene, hence, this chromosome is ignored.

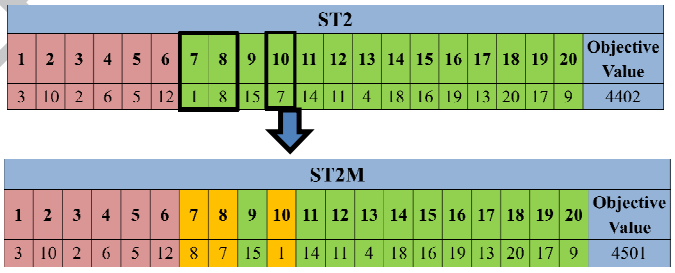


Fig. 7 An example to demonstrate the mutation operation.

D) Randomisation operation

The final operators in the jumping function strategy is the randomisation operator which aims to find a better or an optimum solution by generation a new chromosome. According to the fitness

evaluation stage, the newly generated chromosome either will be kept or deleted compare to the weaker parent chromosome. To demonstrate the process, a newly generated chromosome is found as ST2R as shown in Fig. 8. Based on the fitness evaluation the fitness (objective) value is found as 4394 (it is an optimum solution). Hence, the chromosome ST2 will be replaced to the chromosome ST2R. This process will continue along side with the basic BA until the optimum solution is found.

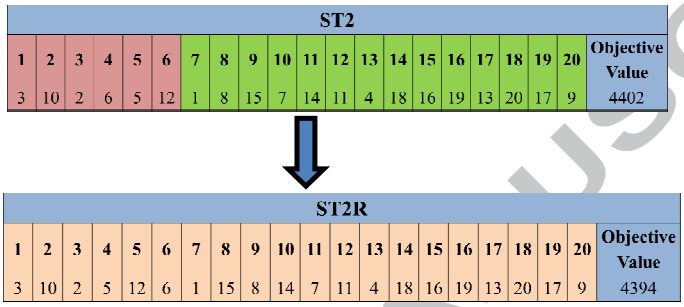


Fig. 8 An example to demonstrate the randomisation operation.

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 seven groups having n= 10, 20, 50, 100, 200, 500, 1000 jobs, respectively with each category containing ten instances (*in#* is the instance amount) to be tested. The values of h= 0.2, 0.4, 0.6, 0.8 classify 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 reporting an average amount while considering percentage offset (*% Offset*) over different runs to have a meaningful result.

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 *Reinforced Global Search* and *Jumping*

Function strategies, in order to unblock the search and venture into new space when it gets stuck in local minima. As shown in Fig. 9 and 10, 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. 9 and 10 demonstrate the comparison of performances between the GBA and the basic BA in a constrained test problem (h=0.2) from Biskup and Feldman, (2001) at the optimum for SMETP. The proposed *Reinforcing Global Search* and *jumping function* strategies increase the capability of exploration and exploitation of the basic BA. While the GBA reaches the optimum value after 10 iterations, the basic BA needs 4000 iterations (on the same test problem under the same algorithm parameters).

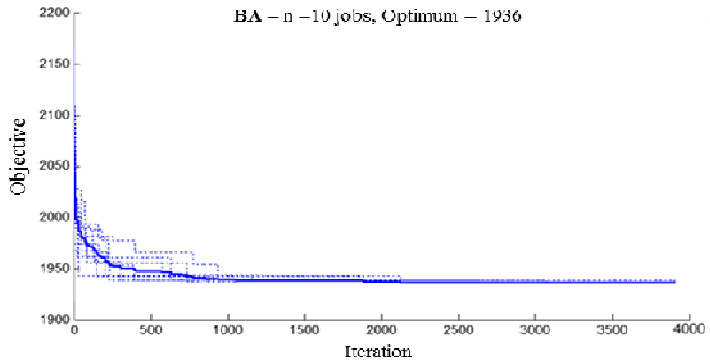


Fig. 9 The performance of the BA for n=10, h=0.2 and the optimum value of 1936 (tuning under parameters: *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5.

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

The grater population size influences on exploitation in solution while gaining in exploration of the domain. However, 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. 11, 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) of instances with the lower 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.,* 2007a). However, for value of h>0.4 and number of jobs higher than 200, given the NP-hard nature, the problem required a higher computational effort (see Fig. 11).

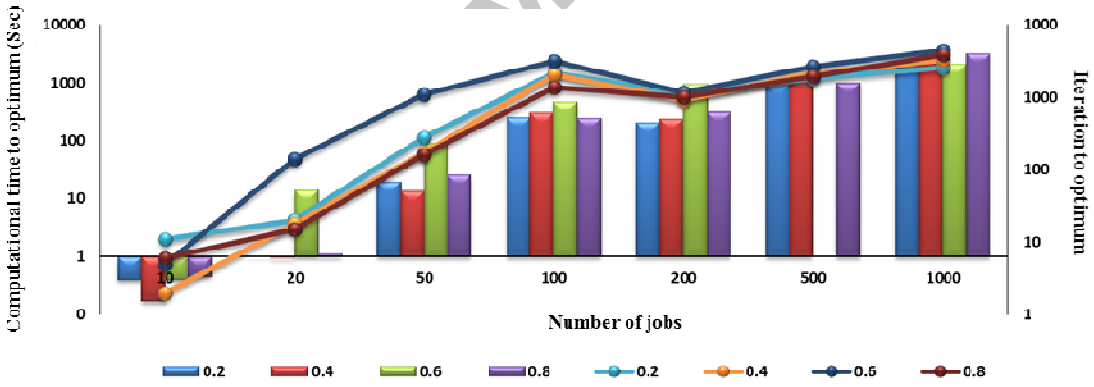


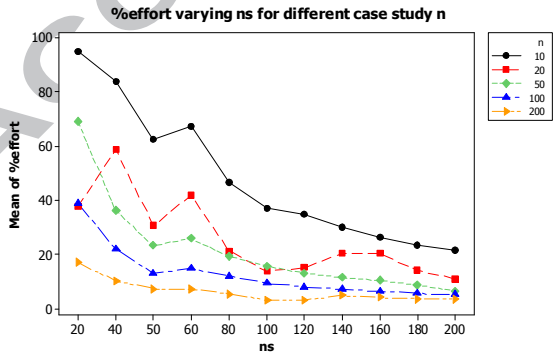
Fig. 11 Computational time in seconds and Iteration to optimum for different SMETP instances under varying h-constraint for values as reported in Biskup and Feldman, (2001).

In Fig. 11, the referral value of calculation time is presented 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, an approach based on the analysis of the *% effort= ($I_{optimum}$/TI) x 100* is utilised, where $I_{optimum}$ 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.

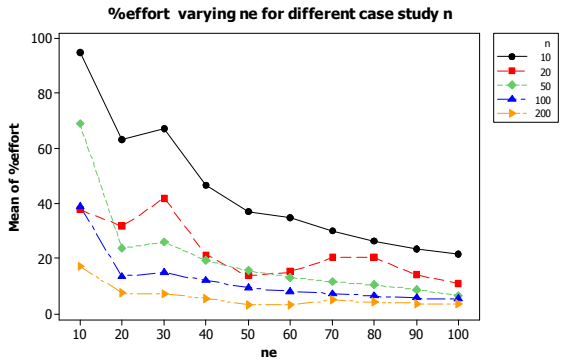
The percentage effort *(%effort)* as defined in Nearchou, (2008) is remarkably higher (more than

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. 12). 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. 11). 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.

Interaction between parameters was evaluated based on Taguchi orthogonal arrays (Taguchi, 1986; Lambiase and Miranda 2003; Yuce *et al.,* 2014b) and the corresponding results are shown in reports of Fig. 12. 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 as percentage efforts (*%efforts*) according to various configurations. The average values of ten repetitions are reported in Fig. 13 and 14. A larger population size makes the algorithm working more slowly, but a better solution will eventually be achieved. However, the correct tuning depends on the problems being solved. The optimisation computation time requires higher *%effort* as the problem increases in size and the values of parameters increase. 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. In general, the remarks in *%efforts* don't demonstrate the evidence for *ns* and *nb* and *ne* values of 100, 50 and 8, respectively. These considerations are set in GBA for outputs as per the results section.



(a)



(b)

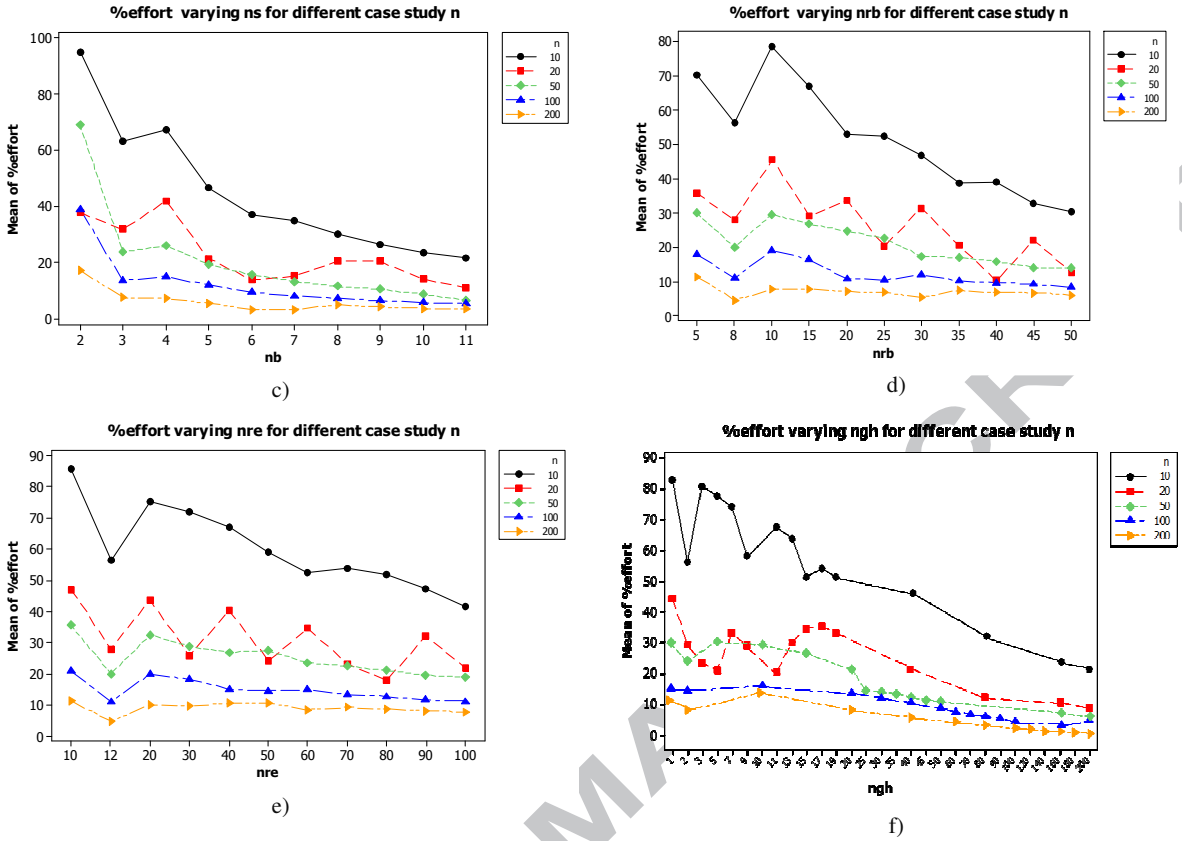


Fig. 12 *Percentage effort* for different values of *ns* (a), *nb* (b), *ne* (c), *nrb* (d), *nre* (e), *ngh* (f) for various test data. *h*=0.2.

For the restricted class n/1/ET large test instance, optimal performances can 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 Search* strategy - it needs to be set on a great *ns* amount - and *Jumping Function* strategy which is based on limited *Iteration* (iteration). The GBA, as explained above, manifests better 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. 11 and therefore mutation and crossover need to be increased.

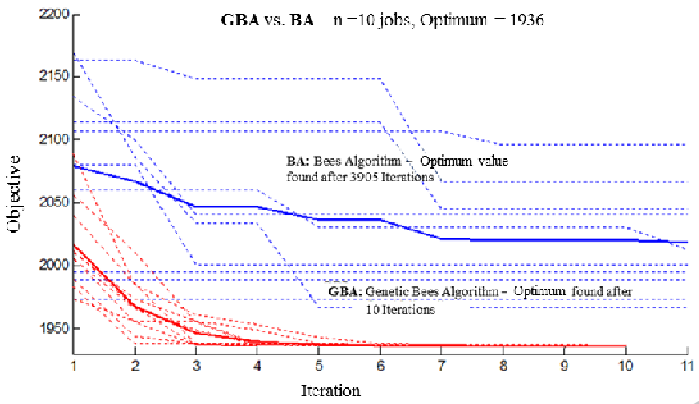


Fig. 13 The performance comparison of the BA *(blue line)* and the GBA *(red line)* under n=10 h=0.2; (tuning under parameters : *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5).

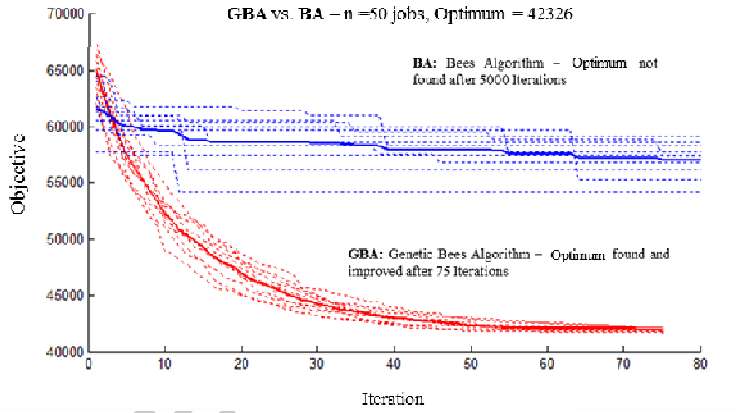


Fig. 14 The performance comparison of the BA *(blue line)* vs. GBA *(red line)* - under n= 50 h=0.2;(tuning under parameters : *ns*=50; *nb*=5; *ne*=3; *nrb*=5; *nre*=5; *ngh*=5).

5.2 RESULTS

The results obtained by the GBA, which are applied on the instances proposed by Biskup and Feldmann, (2001), are presented in Table 2, Table 3, Table 4 and Table 5. In particular, the computation results, concerning small and large size benchmark problems, obtained by the GBA, are compared to those obtained from the BA (under optimal tuning), the upper bounds from Feldmann and Biskup, (2003) meta-heuristics (Table 3, Table 4) and the existing meta-heuristics according to literature review (Table 4, Table 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 for n= {10 & 20 &50 &100 & 200 & 500 & 1000} – average results of ten runs for *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	Iteration		Fitness Value			% 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%**

Iteration 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 *Percentage Offset (% Offset)* under multiple runs. The fitness value presented in Table 2 is computed based on the difference between the Upper Bound cost function and the best solution found over different runs as described in Law and Kelton's, (2000) with a Bonferroni correction due to multiple performance measures used by Quinzi, (2004). The relative convergence performance is presented in Table 2 based on the following transformation: *% Offset* = 100 $(F_{GBA}-F_{BA})/F_{BA}$ (the relative convergence between GBA and BA). The GBA used genetic operator with single-point and multi-point (two points) crossover operators 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 it is seeking for the global 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; number 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 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 (*Iteration*). 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							Average
	10	20	50	100	200	500	1000	(Row)
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%
Average (Column)	**-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 *Iteration* at best - over 10 runs - (as per Table 3 and Table 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.,* (2007a) 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.,* (2007a) 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 - obtains generally the best performance 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 that the average CPU time is 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.

h	n													
	10		20		50		100		200		500		1000	
	Time	Iteration	Time	Iteration	Time	Iteration	Time	Iteration	Time	Iteration	Time	Iteration	Time	Iteration
0.2	0.007	11	0.017	20	0.321	281	4.301	2152	3.457	1141	15.649	1831	39.463	2607
0.4	0.003	2	0.016	18	0.227	169	5.000	2078	3.936	936	19.717	2159	49.767	3275
0.6	0.007	5	0.231	140	1.504	1102	2.936	1554	16.055	1158	25.956	2629	34.264	4420
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 remark and the relative comparison with other meta-heuristics, it can be observed in the work presented by Pham *et al.,* (2011) where the data of Table 5 is partially originated. This approach enables to demonstrate the absolute presentation of the proposed solutions. Each cell of the Table 5 represents the average percentage differences 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 the 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 according to each *h* value- best among ten runs, mean across in# - (Best results so far in the literature are reported in bold).

n	h = 0.2								
	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**	**-0.03**
20	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**
50	-5.70	-5.70	-5.68	-5.70	-5.70	-5.69	-5.70	-5.70	**-5.70**
100	-6.19	-6.19	-6.17	-6.19	-6.19	-6.17	-6.19	-6.19	**-6.19**
200	-5.78	-5.76	-5.74	-5.76	-5.76	-5.77	-5.78	-5.78	**-5.78**
500	-6.42	-6.41	-6.41	-6.41	-6.41	-6.43	-6.43	-6.43	**-6.43**
1,000	-6.76	-6.73	-6.75	-6.74	-6.74	-6.72	-6.77	-6.76	**-6.76**
Average	-4.96	-4.91	-4.92	-4.93	-4.93	-4.95	-4.96	-4.96	**-4.96**

n	h = 0.4								
	DPSO	TS	GA	HTG	HGT	DE	GA/SA & Greedy Local Search	BA	GBA
10	**0.00**	0.24	0.19	0.19	0.19	**0.00**	**0.00**	**0.00**	**0.00**
20	**-1.63**	-1.62	-1.62	-1.62	-1.62	**-1.63**	**-1.63**	**-1.63**	**-1.63**
50	-4.66	-4.66	-4.60	-4.66	-4.66	-4.66	-4.66	-4.66	**-4.66**
100	-4.94	-4.93	-4.91	-4.93	-4.93	-4.89	-4.94	-4.94	**-4.94**
200	-3.75	-3.74	-3.75	-3.75	-3.75	-3.72	-3.75	-3.75	**-3.75**
500	-3.56	-3.57	-3.58	-3.58	-3.58	-3.57	-3.58	-3.57	**-3.57**
1,000	-4.37	-4.39	-4.40	-4.39	-4.39	-4.38	-4.40	-4.35	**-4.35**
Average	-3.27	-3.24	-3.24	-3.25	-3.25	-3.26	-3.28	-3.27	**-3.27**

n	h = 0.8								
	DPSO	TS	GA	HTG	HGT	DE	GA/SA & Greedy Local Search	BA	GBA
10	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20	**-0.41**	**-0.41**	-0.28	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**
50	**-0.24**	**-0.24**	-0.19	-0.23	-0.23	**-0.24**	**-0.24**	**-0.24**	**-0.24**
100	**-0.18**	-0.15	-0.12	-0.08	-0.11	-0.17	**-0.18**	**-0.18**	**-0.18**
200	**-0.15**	-0.04	-0.14	0.26	0.07	0.20	**-0.15**	**-0.15**	**-0.15**
500	-0.11	0.21	-0.11	0.73	0.13	1.01	-0.11	-0.11	**-0.11**
1,000	**-0.06**	1.13	-0.05	1.28	0.40	2.79	**-0.06**	-0.05	-0.05
Average	-0.16	0.07	-0.13	0.22	-0.02	0.45	-0.16	-0.16	**-0.16**

n	h = 0.6								
	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**
20	**-0.72**	-0.71	-0.68	-0.71	-0.71	**-0.72**	**-0.72**	**-0.72**	**-0.72**
50	**-0.34**	-0.32	-0.31	-0.27	-0.31	-0.32	**-0.34**	**-0.34**	**-0.34**
100	**-0.15**	-0.01	-0.12	0.08	0.04	-0.13	**-0.15**	**-0.15**	**-0.15**
200	**-0.15**	-0.01	-0.13	0.37	0.07	0.23	**-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**
1,000	**-0.06**	1.01	-0.05	1.28	0.42	1.29	**-0.06**	-0.05	-0.05
Average	-0.22	0.04	-0.20	0.22	-0.05	0.30	-0.22	-0.22	**-0.22**

* DPSO: Discrete Particle Swarm Optimisation; 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 summarise 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.,* (2007b); 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 Optimisation algorithm (DPSO). Nearchou, (2008) used Differential Evolution (DE) as the optimisation 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 other heuristics, if the large the population sizes (i.e., $n \geq 100$ 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). For the case 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, results 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 property 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.*, (2007a)), 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 utilise these three priority rules to achieve the optimum schedule, however, there is no requirement for the GBA to find the optimum schedule. Since the random initialisations and additional genetic operators allow the algorithm to seek for the optimum schedule. To avoid the randomness of the quality of solution, the best result 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 in Table 5 clearly, that the GBA provides more accurate 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) is introduced. The proposed GBA is applied to solve the Single Machine Scheduling problem with earliness and tardiness penalties and the results illustrate that its performance is better in most cases when h value is lower. The GBA is a new evolutionary optimisation method that is used in a wide range of engineering applications. In this paper, it is utilised for the optimisation of single machine scheduling problem which is classified as the combinatorial optimisation problems.

The algorithm is developed 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 meta-heuristic achieves better performance compare to the upper boundary and the basic Bees Algorithm. The hybrid Genetic Bees Algorithm has proven to be more stable and robust than the basic Bees Algorithm.

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 penalties to find an optimal solution in case of real test instances.

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

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.

Bellgran, M. and Säfsten, K. (2010). *Manufacturing Development: Design and Operation of Manufacturing Systems*. London: Springer-Verlag.

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.

Dombrowski, U., Intra, C., Zahn, T. and Krenkel, P. (2016). Manufacturing Strategy – A Neglected Success Factor for Improving Competitiveness. *Procedia CIRP*, 41, 9-14.

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.*

Chichester: Ellis Horwood.

Gen, M. and R. Cheng. (1997). *Genetic Algorithms and Engineering Design*. New York: John Wiley and Sons.

Gessner, R. A. A. (1986). Master Production Schedule Planning. New York: John Wiley and Sons.

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. and 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.

Jaegyun, K. (1999). A hybrid genetic approach for single machine scheduling with distinct due dates and release times. *In the proc. of the Third Russian-Korean International Symposium on Science and Technology, KORUS '99*, 1, 245-248.

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.

Lambiase, A. and Miranda, S. (2003) Performance parameters optimization of a pneumatic programmable palletizer using Taguchi method. *Robotics and Computer-Integrated Manufacturing*, 19(1-2), 147-155.

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

Lin, S. W., Chou, S. Y. and Ying, K. C. (2007b). 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.

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.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

Ming, H., Baohui, J. and Xu, L. (2011). An Improved Bee Algorithm-Genetic Algorithm, In: J. Watada, G. Phillips-Wren, L. C. Jain and R. J. Howlett (eds.). *Intelligent Decision Technologies, vol. 10 of the Series Smart Innovation, Systems and Technologies*. Berlin: Springer-Verlag, 683-689.

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*, 431.

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).

Pinedo, M. L. (2012). Scheduling: Theory, Algorithms, and Systems, London: Springer.

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., Koc, E., Lee, J. Y. and Phrueksanant, J. (2007). Using the Bees Algorithm to schedule jobs for a machine. *In proc. of the 8th Int. Conf. and Exhibition on Laser Metrology, Machine Tool, CMM & Robotic Performance, LAMDAMAP*, Cardiff, UK.

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*, 454.

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.

Sioud, A., Gravel, M. and Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 39(10), 2415-2424.

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.

Yanushevsky, R.T. (1992). Optimal strategic planning problems in manufacturing based on the input-output models. *Applied Mathematical Modelling*, 16(4), 208-213.

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. (2014a). 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.

Yuce, B., Mastrocinque, E., Packianather, M. S., Lambiase, A. and Fruggiero, F. (2014b) Neural network design and feature selection using principal component analysis and Taguchi method for identifying wood veneer defects. *Production and Manufacturing Research,* 2 (1), 291-308.

HIGHLIGHTS

- Development of a robust hybrid stochastic optimisation algorithm using The Bees Algorithm and Genetic algorithm,
- Genetic operator implementation in the global search side of the Bees Algorithm with a reinforced global search approach,
- Implementation of the proposed hybrid optimisation algorithm on the Single Machine scheduling problem,
- Evaluation of the proposed algorithm by comparing the results to the results of the other well-known stochastic optimisation algorithms.