Hybrid Artificial Bee Colony Search Algorithm Based on Disruptive Selection for Examination Timetabling Problems

Malek Alzaqebah and Salwani Abdullah

Data Mining and Optimisation Research Group (DMO),
Center for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
{malek_zaqeba,salwani}@ftsm.ukm.my

Abstract. Artificial Bee Colony (ABC) is a population-based algorithm that employed the natural metaphors, based on foraging behavior of honey bee swarm. In ABC algorithm, there are three categories of bees. Employed bees select a random solution and apply a random neighborhood structure (exploration process), onlooker bees choose a food source depending on a selection strategy (exploitation process), and scout bees involves to search for new food sources (scouting process). In this paper, firstly we introduce a disruptive selection strategy for onlooker bees, to improve the diversity of the population and the premature convergence, and also a local search (i.e. simulated annealing) is introduced, in order to attain a balance between exploration and exploitation processes. Furthermore, a self adaptive strategy for selecting neighborhood structures is added to further enhance the local intensification capability. Experimental results show that the hybrid ABC with disruptive selection strategy outperforms the ABC algorithm alone when tested on examination timetabling problems.

Keywords: Artificial Bee Colony, Simulated Annealing, Examination Timetabling Problems, Disruptive Selection.

1 Introduction

The examination timetabling problem is concerned with allocating a set of examinations into a limited number of timeslots (periods), subject to a set of constraints. The basic challenge of examination timetabling is to schedule examinations over a limited number of timeslots, so as to avoid conflicts (refer to hard constraints) and to satisfy a number of side constraints (refer to soft constraints) [5]. In recent years, a variety of constraints has been addressed in the scientific literature to model the real-world problems and try to close the gap between theory and practice in automated timetabling, as presented by the 2nd International Timetabling Competition (ITC2007) where three tracks of problems are proposed i.e. one for exam timetabling and two for course timetabling.

W. Wang, X. Zhu, and D.-Z. Du (Eds.): COCOA 2011, LNCS 6831, pp. 31–45, 2011.

In the past, a wide variety of approaches for solving the examination timetable problem have been described and discussed in the literature, that basically can be divided into population-based approaches and single solution based approaches. For a recent detailed overview readers should consult [8].

In this paper, we employ a population-based approach that is based on a swarm intelligence algorithm, called artificial bee colony (ABC). It was proposed by Karaboga [1]. ABC algorithm has been successfully developed to solve many optimisation problems [18, 13, and 19]. It mimics the foraging behavior of honey bee swarms. This work concentrates incorporation of the ABC algorithm with a local search (a simulated annealing in this case) to compensate for the insufficiency of using each type of method in isolation. The details of the proposed approach are discussed later.

The paper is organised as follows. Section 2 presents the examination timetabling problem and its formulation. The original Artificial Bee Colony algorithm is presented in Section 3. The proposed approach is discussed in Section 4. Our experimental results are presented in Section 5. This is followed by conclusion and comments in Section 6.

2 Problem Description and Formulation

The problem description in this paper is divided into two parts as discussed below:

- Problem I: This problem is introduced by Carter et al. (1996) [5], which considered as an uncapacitated examination timetabling problem, where a room capacity requirement is not taking into account.
- Problem II: International timetabling competition (ITC2007) datasets which consist of three tracks. In this work, we consider the first track that represents an exam timetabling model which includes a number of real world constraints.

2.1 Problem I

The problem description that is utilised in this paper is adapted from the description presented in Burke et al. (Burke et al., 2004). Examination timetabling problems consist of these inputs as stated below:

- N is the number of exams
- E_i is an exam, $i \in \{1,\ldots,N\}$
- T is the given number of available timeslots
- M is the number of students
- $C = (c_{ij})_{NxN}$ is the conflict matrix where each element denoted by c_{ij}, $i,j \in \{1,\ldots,N\}$ is the number of students taking exams i and j.
- t_k $(1 \leq t_k \leq T)$ specifies the assigned timeslot for exam k $(k \in \{1,\ldots,N\})$

We formulate an objective function which tries to space out students' exams throughout the exam period (Expression (1)) that can then be formulated as the minimisation of:

$$\frac{\sum_{i=1}^{N-1} F_1(i)}{M} \tag{1}$$

where

$$F_1(i) = \sum_{j=i+1}^{N} c_{ij} \cdot proximity \quad (t_i, t_j) \tag{2}$$

and

$$proximity \quad (t_i, t_j) = \begin{cases} 2^5 / 2^{|t_i - t_j|} & if \quad 1 \le |t_i - t_j| \le 5 \\ 0 & otherwise \end{cases} \tag{3}$$

subject to:

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} \cdot \lambda(t_i, t_j) = 0$$

where

$$\lambda(t_i, t_j) = \begin{cases} 1 & if \quad t_i = t_j \\ 0 & otherwise \end{cases} \tag{4}$$

Equation (2) presents the cost for an exam *i* which is given by the proximity value multiplied by the number of students in conflict. Equation (3) represents a proximity value between two exams [5]. Equation (4) represents a clash-free requirement so that no student is asked to sit two exams at the same time. The clash-free requirement is considered to be a hard constraint.

2.2 Problem II

International timetabling competition (ITC2007) introduces three tracks of problems i.e. examination timetabling, curriculum-based course timetabling problem and post-enrolment course timetabling problems. In this paper, we concentrate on the first track i.e. examination timetabling problems that include a number of real world constraints [8]. A set of hard and soft constraints are listed in Table 1 and Table 2, respectively.

Table 1. Hard Constraints

Hard Constraints	Explanation
H1	There cannot be any students sitting for more than one exam at the same time.
H2	The total number of students assigned to each room cannot exceed the room capacity.
H3	The length of exams assigned to each timeslot should not violate the timeslot length.
H4	Some sequences of exams have to be respected. e.g. Exam_A must be schedule after Exam_B.
H5	Room related hard constraints must be satisfied e.g. Exam_A must be scheduled in Room 80.

Table 2. Soft Constraints

Soft Constraints	Mathematical Symbol	Explanation
S1	C_S^{2R}	Two exams in a row: Minimise the number of consecutive exams in a row for a student.
S2	C_S^{2D}	Two exams in a day: student should not be assigned to sit more than two exams in a day. Of course, this constraint only becomes important when there are more than two examination periods in the same day.
S3	C_S^{PS}	Periods spread: all students should have a fair distribution of exams over their timetable.
S4	C_S^{2NMD}	Mixed durations: The numbers of exams with different durations that are scheduled into the same room has to be minimised as much as possible.
S5	C^{FL}	Larger examinations appearing later in the timetable: Minimise the number of examinations of large class size that appear later in the examination timetable (to facilitate the assessment process)
S6	C^{P}	Period Penalty: some periods have an associated penalty, minimise the number of exams scheduled in penalised periods.
S7	C^{R}	Room Penalty: some rooms have an associated penalty, minimise the number of exams scheduled in penalised rooms.

A feasible timetable is one in which all examinations have been assigned to a period and room, and there is no violation of hard constraints. The objective function is to minimise the violation of soft constraints as given in expression (5) [8].

$$\min \sum_{s \in S} (w^{2R} C_S^{2R} + w^{2D} C_S^{2D} + w^{PS} C_S^{PS}) + w^{NMD} C_s^{2NMD} + w^{FL} C^{FL} + w^{p} C^{P} + w^{R} C^{R} \quad (5)$$

Each dataset has its own weight as shown in Table 3 [8].

Table 3. The Associate Weight of ITC2007 Collection of Examination Datasets

Datasets	w^{2}	w^{2}	$w^{P.}$	w^{N}	w^{FL}	w^{F}	w^{R}
Exam_1	5	7	5	10	100	30	5
Exam_2	5	15	1	25	250	30	5
Exam_3	10	15	4	20	200	20	10
Exam_4	5	9	2	10	50	10	5
Exam_5	15	40	5	0	250	30	10
Exam_6	5	20	20	25	25	30	15
Exam_7	5	25	10	15	250	30	10
Exam_8	0	150	15	25	250	30	5

3 Artificial Bee Colony Algorithm (ABC)

3.1 Basic Artificial Bee Colony (ABC) Algorithm

Artificial Bee Colony (ABC) algorithm is a global optimisation algorithm that replicates the real behavior of honey bees, introduced by Karaboga [1]. The algorithm classifies the bees in the hive into three groups i.e. employed bees, onlooker bees and scouts bees. In this algorithm, employed bees fly around the search space to collect the information of food sources, and share the information with onlooker bees through a wiggle dance. Then onlooker bees choose their food sources based on this information. During the search process, the employed bees whose food source has been abandoned become scout bees and start to search for new food sources randomly without any previous information. In ABC algorithm, the position of a food source represents a possible solution, and the nectar amount of a food source corresponds to the quality (fitness value) of the associated solution. The number of the employed bees is tied to the number of solutions in the population. Figure 1 shows the pseudo code of the ABC algorithm [1].

```
Initial food sources are produced for all employed bees
REPEAT
   Each employed bee fly's to a food source in her memory and
       determines a neighbour source, then evaluates its nectar
       amount and dances in the hive
   Each onlooker watches the dance of employed bees and chooses one
       of their sources depending on the dances, and then goes to that
       source. After choosing a neighbour around that, she evaluates
       its nectar amount.
   Abandoned food sources are determined and are replaced with the
       new food sources discovered by scouts.
   The best food source found so far is registered.
UNTIL (requirements are met)
```

Fig. 1. Original artificial bee colony search algorithm

As shown in Figure 1, at the first step, a constructive heuristic algorithm is applied to initialise the population (food source positions). After the initialization, an employed bee produces an adjustment on the source position in her memory and discovers a new food source position. If the nectar amount of the new food source is higher than the previous food source, the bee memorises the new source position and forgets the old one. Otherwise, she keeps the position of the one in her memory. After all the employed bees complete the search process, they share the information of the sources with the onlookers on the dance area. Each onlooker bee evaluates the nectar information taken from all employed bees and then chooses a food source

depending on the nectar amounts. Finally scout bees find out the abandoned sources and replace it by randomly produced sources.

3.2 Onlooker Bees Selection Process

Onlooker bees choose the solution by a stochastic selection strategy, which is summarised as below:

1. Calculates the fitness value (fit_i) by using the fitness function as follow:

$$fit_i = \begin{cases} \dfrac{1}{1 + f_i} & f_i \geq 0 \\ 1 + abs\ (f_i) & f_i < 0 \end{cases} \tag{6}$$

   where f_i is fitness function.

2. Calculate the probability value by using the following expression:

$$p_i = \frac{f_i}{\sum_{i=1}^{SN} f_i} \tag{7}$$

   where SN is the number of food sources, f_i is the fitness function of the i^{th} food source.
3. Finally, chose a candidate solution based on the selection probability by "roulette wheel selection" method.

As stated in [10], there are two problems of using basic ABC selection strategy i.e. (i) A "super- individual" being too often selected the whole population tends to converge towards his position. The diversity of the population is then too reduced to allow the algorithm to progress; (ii) with the progression of the algorithm, the differences between fitness are reduced. The best ones then get quite the same selection probability as the others and the algorithm stops progressing." Thus, this selection strategy is hard to keep the diversity and to avoid the premature convergence. In order to alleviate these problems, this paper employed a disruptive selection strategy to improve the performance of the ABC algorithm.

3.3 Disruptive Selection Strategy

Disruptive selection gives more chances for higher and lower individuals to be selected by changing the definition of the fitness function as in Equation (8) [13].

$$fit_i = | f_i - \overline{f_t} | \qquad P_i \frac{fit_i}{\sum_{i=0}^{n} fit_i} \tag{8}$$

Where f_i is the fitness function, $\overline{f_t}$ is the average value of the fitness function f_i of the individuals in the population. By using a disruptive selection, it has a tendency to maintain the diversity slightly longer, because both higher and lower quality of the solutions is more preferable.

4 The Proposed Algorithm

4.1 Neighborhood Search Operations

In this paper, four neighborhood search operations are employed in order to enhance the performance of searching algorithms i.e. [2]:

Nbs1: Select 2 exams at random and swap timeslots.
Nbs2: Select a single exam at random and move to a new random feasible timeslots.
Nbs3: Select 4 exams randomly and swap the timeslots between them feasibly.
Nbs4: Select 2 exams at random and move to a new random feasible timeslots.

4.2 Self-adaptive Method for Neighbouring Search

To find neighbouring food sources, both employee and onlooker bees apply a self-adaptive strategy that is explained as follows [21]:

1. At the beginning, the neighbour list (*NL*) with a specified length is generated by filling the list randomly from four neighbourhood search operations as explained in Section 4.1.
2. During the evolution process, one neighbourhood is taken from *NL* and is used to generate a new food source for an employed bee or onlooker bee.
3. If the new food source is better than the current one, this approach will put the employed neighbourhood search operation into a new list, called a winning neighboring list (*WNL*).
4. When the *NL* became empty, it will be refilled as follow: 75% is refilled from the *WNL* list, and 25% is refilled randomly from four neighbourhood search operations, and also *WNL* is reset to zero to keep away of any accumulation effects.
5. If the *WNL* is empty (this perhaps happen when the search perform near an optimal with negligible population variety) the most recent *NL* is used again.

By using this method, the suitable neighboring search operation can be learned based on the current search process and the solution state. The length of *NL* is set to 200 as stated in [21].

4.3 A Local Search Algorithm (Simulated Annealing)

A simple local search (i.e. simulated annealing) is embedded to the basic ABC algorithm in order to enhance the utilisation capability of the algorithm. This is due to the basic ABC that uses a greedy acceptance criterion i.e. only accepts an improved solution and eliminate the worse.

Simulated annealing has been proposed by Kirkpatrick et al. [12]. It mimics the annealing process of metals molten that is heated and then slowly cooled. A simulated annealing algorithm works on a single solution and tries to improve it by finding nearby solutions, and slowly amending a parameter called temperature. The algorithm always accepts a better solution. In SA, a worse solution is accepted with the a certain probability between [0,1] if it less than $e^{-\delta/Temp}$ where δ is the difference between the penalty cost of the new and current solutions (i.e. $\delta = f(Sol^*)-f(Sol)$). The process is repeated until the temperature *Temp* is less than the final temperature T_f , as shown in Figure 2. In this paper, the parameter used for the simulated annealing algorithm are set as follow (adapted from Abdullah and Burke [20]): the initial temperature (T_0) is set to 5000, final temperature (T_f) is set to 0.05, and the number of iterations, $NumOfIte_{SA}$ is set to 200000.

4.4 Constructive Heuristic

In this paper, we use a graph colouring approach (i.e. largest degree heuristic) to generate the initial solution, where the examination with the largest number of conflicts are scheduled first. For more details about graph colouring applications to timetabling see Burke et al. [3]

4.5 Improvement Algorithm

Figure 2 illustrates the pseudo-code that represents our approach, and also a SA pseudo-code. The algorithm starts with feasible initial solutions which are generated by a largest degree heuristic.

The algorithm starts with initial population that is generated using a graph colouring approach. The employed bees work on random solutions and apply a neighborhood structure based on self-adaptive method (as explained in Section 4.2) on each solution. The solutions are arranged based on the profitability. Onlooker bees calculate the selection probability based on disruptive selection as in Equation (8) and then she applies a local search (SA) (as explained in Section 4.3) on the highest probability solution. Finally, scout bees determine the abandoned food source and replace it with the new food source.

```
Initialisation:
Initialise the initial population and evaluate the fitness;
Calculate the initial fitness value, f(Sol);
Set best solution, Solbest ← Sol;
Set maximum number of iteration, NumOfIte;
 Set the population size;
 //where population size = OnlookerBee = EmployeedBee;

iteration ←   0;
Improvement:
do while (iteration < NumOfIte)
Exploration process
    for i=1: EmployeedBee
        Select a random solution and apply neighborhood
          structure based on Self-Adaptive;
    end for

    for i=1: OnlookerBee
     Calculate the selection probability P_i, based on
     disruptive selection as in Equation (8).
     Sol*  ← select the solution depending on P_i;
    Start local search(SA) on Sol*;
      Set initial temperature T_0;
      Set final temperature T_f;
      Set number of iteration NumOfIte_SA;
      Set decreasing temperature rate as α
      where α = (log(T_0) - log(T_f))/NumOfIte_SA;
      Set Temp ← T_0;
      Set Solbest_SA ← Sol*;
      Set Sol_SA ← Sol*;
      do while (Temp > T_f)
       Sol_SA* ← Apply neighbourhood structure on Sol_SA;
       Calculate cost function f(Sol_SA*);
       if (f(Sol_SA*) < f(Solbest_SA))
         Sol_SA ← Sol_SA*;
         Solbest_SA ← Sol_SA*;
       else
         Generate a random number called RandNum;
         if (RandNum ≤ e^(-δ/Temp)) where δ = f(Sol_SA*)-f(Sol_SA)
           Sol_SA ← Sol_SA*;
           Temp = Temp-Temp*α;
      end while
    end local search
Exploitation process
     if (f(Solbest_SA) < f(Sol*))
       Sol* ← Solbest_SA;
     end if
    end for
      Solbest ← best solution found so far;
Scouting process
   Scoutbee determines the abandoned food source and replace
   it with the new food source.
  iteration++;
end do
```

Fig. 2. The pseudo code for the artificial bee colony search algorithm

5 Simulation Results

In our experimental results we employed three different modifications of ABC algorithm, called ABC algorithm based on disruptive selection (coded as DABC), DABC algorithm with a local search (i.e. Simulated Annealing) (coded $DABC_{SA}$) and $DABC_{SA}$ algorithm with a self-adaptive method for neighbouring search (coded as self-adaptive $DABC_{SA}$). We compare the performance of these modifications with the basic ABC in order to show the effects of employing different modification on basic ABC algorithm. The parameter settings used in this work are shown in Table 4.

Table 4. Parameters setting

Parameter	Value
Iteration	500
population size	50
Scout Bee	1

5.1 Problem I

Table 5 provides the comparison of basic ABC, DABC, $DABC_{SA}$ and self-adaptive $DABC_{SA}$ results, and compared with the best known results in the literature. The purpose here is to compare the performance among the three versions of ABC algorithms when tested in Problem I which represents the uncapacitated examination timetabling problem.

Table 5. Results Comparison on Uncapacitated Problems

Instance	Basic ABC	DABC	$DABC_{SA}$	Self adaptive $DABC_{SA}$	Best known	Authors for best known
car91	5.86	5.42	5.33	5.19	**4.50**	Yang and Petrovic (2004)
car92	4.92	4.84	4.39	4.36	**3.98**	Yang and Petrovic (2004)
ear83 I	38.34	37.54	35.17	32.26	**29.3**	Caramia et al. (2001)
hec92 I	11.51	11.21	11.19	10.89	**9.2**	Caramia et al. (2001)
kfu93	16.04	15.13	14.07	13.73	**13.0**	Burke et al. (2010)
lse92	12.42	12.06	11.89	11.15	**9.6**	Caramia et al. (2001)
sta83 I	158.12	157.52	157.39	157.23	**156.9**	Burke et al. (2010)
tre92	9.58	9.23	9.41	9.22	**7.9**	Burke et al. (2010)
uta92 I	3.99	3.94	3.89	3.83	**3.14**	Yang and Petrovic (2004)
ute92	27.80	27.57	27.11	26.73	**24.8**	Burke et al. (2010)
yor83 I	41.44	40.94	40.76	40.63	**34.9**	Burke et al. (2010)

The comparison between basic ABC, DABC, $DABC_{SA}$ and the self-adaptive $DABC_{SA}$ shows, that the three modified version of ABC perform much better than the basic ABC. From Table 5, we can say that the disruptive selection strategy outperform the basic ABC, and then after applying the local search on DABC ($DABC_{SA}$) the algorithm is able to produce better solutions. The comparison between the $DABC_{SA}$ and the self-adaptive $DABC_{SA}$ shows that, by employing a self-adaptive method for neighbouring search helps the algorithm to perform better than with the local search alone (i.e. that select the neighbourhood search operations at random).

Overall comparison with the best known results shows that even we are unable to beat any of the best known results in the literature, we are still able to produce good enough solutions.

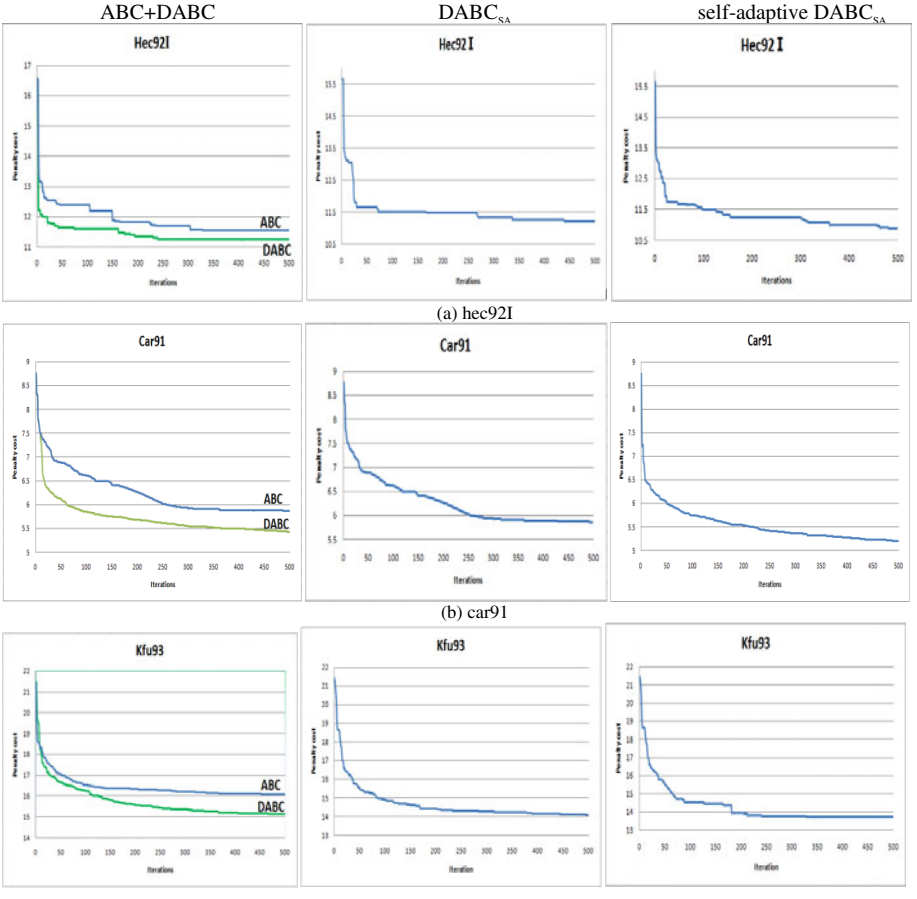
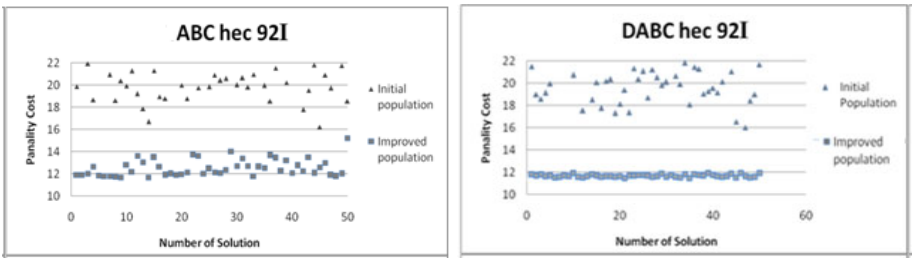


Fig. 3. Convergence graph for hec92I, car91 and kfu93

Figure 3 shows the behavior of the algorithm over three datasets. The *x*-axis represents the number of iterations, while the *y*-axis represents the penalty cost. The first column is the basic ABC convergence, the middle is the DABC$_{SA}$, and the right column is the self-adaptive DABC$_{SA}$. These graphs show how our algorithm explores the search space in which we believe that the way the algorithm behaves has a correlation with the complexity of the datasets (represented by the conflict density value). Note that the details of the conflict density values can be found in Qu et al. (2009). The higher conflict density signifies that more exams are conflicting with each other. The conflict density value for hec92I is 0.42, car91 is 0.13 and kfu93 is 0.06. The behavior of the algorithm works similar at the beginning of the iterations

where the improvement of the solution can easily be obtained. Later it becomes steady and hard to be improved. However, for the kfu93 dataset (where the conflict density value is low compared to hec92I and car91 datasets), the algorithm is able to slowly improve the quality of the solution until it get stuck in the local optimum when the number of iteration almost reaches the maximum number of iteration used in this experiment.



(a) (b)

Fig. 4. The effects of using the disruptive selection strategy

Figure 4 shows the effects of using the disruptive selection strategy (as explained in Section 3.3). The x-axis represents the number of solution, while the *y*-axis represents the penalty cost. The graph shows that DABC can explore the search space better than basic ABC. This is due to the behavior of the selection strategy i.e. in the basic ABC, a random selection is used to select the solution (as explained in Section 3.2) where the solution with highest fitness value will be the most selected during the improvement process. However, the disruptive selection strategy concentrates on both worse and high fitness solutions, and trying to keep the population diversity by improving the worse fitness solutions in concurrency with the high fitness solutions. This can be seen in Figure 4 (b) where all the solutions (improved population) are converged together after the improvement step are executed.

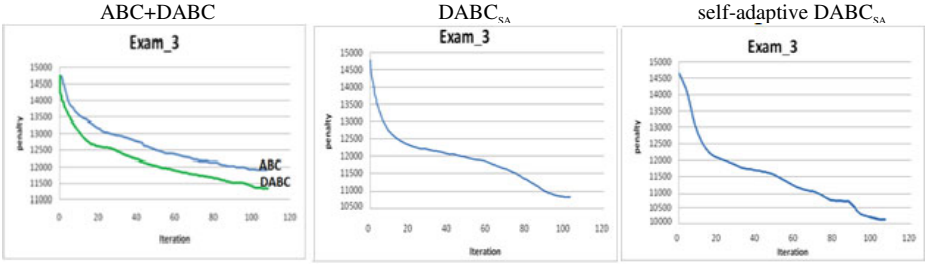
5.2 Problem II

The three modified version of ABC algorithm are also tested on Problem II which represents the (ITC2007). The results are shown in Table 6, which provides the comparison between the basic ABC, DABC, $DABC_{SA}$ and self-adaptive $DABC_{SA}$, and with other available results in the literature.

As shown in Table 6, the results for the three modified version of ABC (DABC, $DABC_{SA}$ and the self-adaptive $DABC_{SA}$) perform better than the basic ABC. The efficiency of the algorithm increases starting with applying a disruptive selection (DABC) and then the hybridization with a local search ($DABC_{SA}$). Form Table 6, we can conclude that the self-adaptive $DABC_{SA}$ shows better result in comparison with the three version of ABC, and they are comparable with some of other results in the literature.

Table 6. Results Comparison on ITC2007 Datasets

Datasets	Müller (2009)	Atsuta et al. (2007)	Pillay (2007)	Gogos et al. (2009)	ABC	DABC	$DABC_{SA}$	Self-adaptive $DABC_{SA}$
Exam_1	4370	8006	12035	4699	6582	6552	6361	6354
Exam_2	400	3470	3074	385	1517	1455	1377	1352
Exam_3	10049	18622	15917	8500	11912	11441	10867	10146
Exam_4	18141	22559	23582	14879	19657	19591	18929	18214
Exam_5	2988	4714	6860	2795	17659	17104	16942	16124
Exam_6	26950	29155	32250	25410	26905	23309	21309	22309
Exam_7	4213	10473	17666	3884	6840	6747	6692	6317
Exam_8	7861	14317	16184	7440	11464	10948	10857	10293



(a) (b) (c)

Fig. 5. Convergence graph for Exam_3

Figure 5 shows the behavior of the algorithm when tested on Problem II where the conflict density for Exam_3 is equal to 2.62 [15]. The *x*-axis represents the number of iterations, while the *y*-axis represents the penalty cost. As in Figure 5 (a, b and c), all of three modified algorithms are better than the basic ABC (based on the results obtained). Figure 5(a) shows that the DABC algorithm with disruptive selection strategy able to better explore the search space and brings all solutions to converge together. By incorporating the simulated annealing algorithm with DABC algorithm create a balance of exploration and exploitation. This is evidenced from the obtained results where the local search helps to further improve the obtained solutions. Introducing a self-adaptive strategy (compare to random) in selecting neighbourhood structures is managed to enhance the local intensification capability as shown in Figure 5(c). The results from Table 6 also show that the self-adaptive $DABC_{SA}$ is able to obtain solutions that are better than other proposed approaches here on almost of the tested datasets. Note that, the experiment carried out here is terminated when the time reach to 600 seconds (as set in the ITC2007 computation rules).

6 Conclusion and Future Work

The primary aim of this paper is to enhance the performance of the basic ABC algorithm by using a disruptive selection strategy (DABC), and later incorporate with a local search algorithm (i.e. a simulated annealing in this case) ($DABC_{SA}$). Adding a self adaptive strategy (self-adaptive $DABC_{SA}$) for selecting a neighborhood structure helps to further enhance the performance of $DABC_{SA}$. Experimental results show that the three modified version of ABC algorithm outperforms the ABC algorithm alone and are comparable with state-of-the-art when tested on examination timetabling problems. As a future work, we will investigate the effect of adaptively incorporating a different local search algorithm and tested its performance on broader timetabling problems.

References

1. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)
2. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. OR Spectrum 29(2), 351–372 (2007)
3. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A time-predefined local search approach to exam timetabling problem. IIE Transactions 36(6), 509–528 (2004)
4. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New algorithms for examination timetabling. In: Näher, S., Wagner, D. (eds.) WAE 2000. LNCS, vol. 1982, pp. 230–241. Springer, Heidelberg (2001)
5. Carter, M.W., Laporte, G.: Examination Timetabling: Algorithmic Strategies and Applications. Journal of the Operational Research Society 47, 373–383 (1996)
6. Carter, M.W.: A survey of practical applications of examination timetabling algorithms. Operations Research 34(2), 193–202 (1986)
7. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum 30(1), 167–190 (2008)
8. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12, 55–89 (2009)
9. Abdullah, S., Turabeih, H., McCollum, B.: A hybridization of electromagnetic like mechanism and great deluge for examination timetabling problems. In: Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (eds.) HM 2009. LNCS, vol. 5818, pp. 60–72. Springer, Heidelberg (2009)
10. Bao, L., Zeng, J.: Comparison and Analysis of the Selection Mechanism in the Artificial Bee Colony Algorithm. HIS 1, 411–416 (2009)
11. Abdullah, S., Burke, E.K., McCollum, B.: Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling. In Metaheuristics: Progress in complex systems optimization (Operations Research / Computer Science Interfaces Series). Ch. 8. Springer, Heidelberg (2007) ISBN:978-0-387-71919-1

12. Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G.: Bonabeau.Self-Organization in Biological Systems. Princeton University Press, Princeton (2003)
13. Karaboga, N.: A new design method based on artificial bee colony algorithm for digital IIR filters. Journal of the Franklin Institute 346(4), 328–348 (2009)
14. Karaboga, N., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8, 687–697 (2008)
15. Karaboga, D., Akay, B.: A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation (2009) doi:10.1016/j.amc.2009.03.90
16. Karaboga, D.: An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey (2005)
17. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39, 459–471 (2007)
18. Kang, F., Li, J., Xu, Q.: Structural inverse analysis by hybrid simplex artificial bee colony algorithms. Computers and Structures 87, 861–870 (2009)
19. Singh, A.: An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. Applied Soft Computing 9, 625–631 (2009)
20. Abdullah, S., Burke, E.K.: A Multi-start large neighbourhood search approach with local search methods for examination timetabling. In: Long, D., Smith, S.F., Borrajo, D., McCluskey, L. (eds.) The International Conference on Automated Planning and Scheduling (ICAPS 2006), Cumbria, UK, June 6-10, pp. 334–337 (2006)
21. Pan, Tasgetiren, Q.-K., Suganthan, M.F., N., P., Chua, T.J.: A Discrete Artificial Bee Colony Algorithm for the Lot-streaming Flow Shop Scheduling Problem, Information Sciences. Elsevier, Netherlands (2010)
22. Burke, Eckersley, E.K., J. A., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. European Journal of Operation Research 206(1), 46–53 (2010)
23. Yang, Y., Petrovic, S.: A novel similarity measure for heuristic selection in examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 247–269. Springer, Heidelberg (2005)