Int. J. Advance. Soft Comput. Appl., Vol. 5, No. 2, July 2013
ISSN 2074-8523; Copyright©SCRG Publication, 2013
www.home.ijasca.com

Sieve Algorithm - A New Method for Optimization Problems

Hemmak Allaoua, Bouderah Brahim

Department Of Informatics, University Of M'sila, Algeria
e-mail: hem_all@yahoo.fr
Department Of Informatics, University Of M'sila, Algeria
e-mail: bouderah_b@yahoo.fr

Abstract

In this paper, we propose a new approach for solving combinatorial optimization problems as scheduling problems, traveling salesman problem, transport problems, and images segmentation which have exponential complexity and known as NP-hard problems. This approach, known as Sieve approach is based on the sieving operation idea used to sift grains by translating it on an algorithmic tool. This approach generates randomly and iteratively a great number of feasible solutions by batches. The bad items are removed, while the good items are assembled in a smaller central set according to an appropriated fitness function. The best solution is computed from this small set according to the problem objective. The fitness function may be easy to compute but it may simulate the objective. We provide a mathematical formulation for representing the problem environment. The implementation is done on a single machine scheduling problem and the results are compared to show the approach efficiency. The findings show that our proposed method can give better solutions than existing meta heuristics like genetic algorithms, ant colonies, and simulated annealing.

Keywords**:** ***Meta heuristics, Optimization, Combinatorial problems, Sieve method, Scheduling.***

1 Introduction

As it is known, the existing meta-heuristics have two main disadvantages: the quality of solution and the eventual premature convergence. In the one hand, we cannot always proof that the solution obtained is near to the optimal for biggest problem size. On the other hand, sometimes they converge prematurely to a local optimum; often it is very costly in time to avoid this situation. So, by using the proposed approach, we try to fill these gaps and give an alternative to treat combinatorial problems [4]. As other meta-heuristics such as genetic algorithms [2], ant colonies [1] [30], and simulated annealing [5] [29] are inspired from natural phenomena, our approach is also inspired from the grains sieve phenomenal where, in order to sift them, the grains are added by handfuls periodically into the sieve. For each handful, they are sift: it implies that the little grains fall through the holes of the sieve and the big grains move towards the center of the sieve by proportional speeds to their volumes. These two steps are repeated for a great number of times. In this paper we propose mathematical model and formulation to this approach. Thus, the structure of the paper is given as follows: In section 2, we provide an overview of Sieve Algorithm and this covers the Sieve Principal, Sieve Properties, Sieve General Algorithm and The Modeling of Sieve Approach. Section 3 gives an insight of Sieve Algorithm for a Single Machine Scheduling, while Section 4 provides Results and Analysis of Sieve Algorithm. Finally, Section 5 draws some remarks and discussions. Note that our approach is different to sieve method used in primary numbers searches [3] [32].

2 Overview of Sieve Algorithm

In this section, we explain the main object of our work in details. It starts with the method principles, showing its parameters, and then, follows by the formulation of the general algorithm and its implementation.

2.1 Sieve Principles

By analogy to the sieving operation, from which is inspired the approach, we have to satisfy following principles:

- *The sieve is used to sift grains for getting the best ones;*
- *Grains to be sift are added into the sieve periodically by handfuls;*
- *The initial sieve is chosen according a sample of the grains to be sift;*
- *The grains can be divided on a great number of handfuls;*
- *For each added handful, the grains are sift many times;*
- *When sieving, the little grains fall out the sieve from holes and the biggest ones move slowly toward the sieve center; if all grains fall, we must change sieve by another one having smaller holes.*
- *The movements of grains are assumed linear and decelerated but their speeds are proportional to their volumes, bad items may never reach the center;*
- *If the operation is not well accomplished, we can also change the sieve by another one having larger holes;*
- *At least, the best grains will stabilize in a small circular area near the sieve center.*

Now, we have to translate these principles on a general algorithm to accomplish the sieve task, with some parameters to perfectly simulate the environment of the task. Since, a good projection of these principles on the problem environment may provide good results. However, this projection will impose the intervention of several parameters which could also affect the solution quality.

2.2 Sieve Parameters

Before using this method, we need to carefully define following such data of the problem, some parameters to be used by the principles of our approach (fig. 1):

- *sieve S, set of grains initially empty, it is the solutions pool;*
- *the radius r of the sieve ($r \in R^+$), taken large enough to hold a significant number of grains;*
- *the radius b of the critical area A ($b \in R^+$, b = small fraction of r, $b \approx r/10$, r/20,…); this critical area is used to accelerate the approach, so, no need to search the solution in S, but just in the area A;*
- *the diameter h of the hole sieve ($h \in R^+$, $h << r$, $h \approx$ lower bound, h can be increased by $h=\alpha*h$ ($\alpha >1$) changing sieve to improve results); a big value of h can produce divergence of the algorithm in which case we could reduce h by $h=\beta*h$ ($0<\beta <1$). So, the initial value of h must be chosen according the problem properties.*
- *the quantity n of grains in each handful ($n \in N$, $n \approx 100,200,\ldots$);*
- *the grains gi, i = 1,2,…,n (gi is a solution of the problem, well codified as vector, set, array, string, object, …);*
- *the respective distances di, i = 1,2,…,n of grains gi, to the sieve center ($di \in R^+$, $di < r$), these distances will decrease during the movement of grains (when sieving): the grains approach to the center, initially we have $b< di < r$ to avoid making bad grains in zone A;*
- *a fitness function f to evaluate grains ($f(gi) \in R+$, f(gi) = diameter of gi = function simulating the problem objective);*
- *the speed vi of grain movement (such as $d_i^{t+1}= vi*d_i^t$, vi function of f, $0 \leq vi \leq 1$);*
- *the number Nb of iterations (Nb = number of added handfuls, generally taken large enough as $Nb \approx 1000,10000,\ldots$);*
- *the number N_s of sifts (N_s = number of sieving times for each added handful, $Ns \approx 10,20,\ldots$), also it is the number of grains movement;*

The values of these parameters are chosen according to the treated problem. However, the adjustment of these values can considerably improve the results.

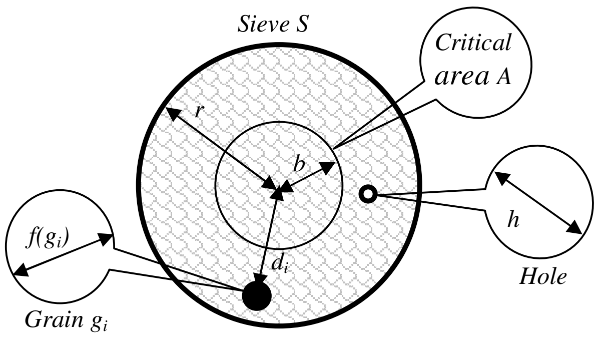

Fig 1: Sieve Parameters

2.3 General Sieve Algorithm

As it was described above, the general algorithm of our approach is composed of two main steps which are repeated for a great number of iterations and a third step that consists just to get the best solution from a small central area of the sieve:

```
Sieve Algorithm
        Choose initial sieve hole;
        For a great number of iterations
                Add handful grains into the sieve;
                If all grains fall change sieve;
                Sift grain;
                If no improvement change sieve;
        End for
        Get best grain;
End.
```

Before starting sieving, we need to choose suitable sieve; that means select good holes diameter. In terms of combinatorial problems, this initial value of holes diameter will represent the lower bound of problem solutions. So it depends of the problem and it must be carefully chosen to accelerate the approach and avoid premature convergence of the algorithm. While evolving in the approach, this value will be adjusted to the problem environment. Note that we can use the specific properties of the problem to find the initial *h*.

The first step consists to generate randomly a lot (handful) of grains (feasible solutions for the proposed problem). The size n of the handful, since the number of grains added at each time is depending of the problem. So, the grains g_i $(i = 1,2,\ldots,n)$ and their respective distances d_i from the center are generated randomly by using uniform law to avoid favoring certain solutions.

The second step represents the sieving operation itself. It consists to move grains by applying the sieve principles, so:

- *When sieving, the grains g_i such as $f(g_i) \leq h$ will fall out the sieve, and the grains g_i such as $f(g_i) > h$ will move slowly toward the sieve center with appropriated speed v_i;*
- *$f(g_i)$ must imperatively simulate the problem objective to be optimized; it may be easy to compute because it concerns a large set of feasible solutions (g_i), but when computing the best solutions (in the area A), we will use the objective itself; that means: the grains are moving according a fitness function f (their diameters), but the best ones are chosen according their real objectives (their real weight); so some bad grains can arrive to the area A but they are not selected (false items); because f simulate the objective ($f \approx Obj$);*
- *v_i must be proportional to $f(g_i)$, that means: the grains which have a great diameter $f(g_i)$ will move more quickly toward the center and vice versa; that will favorite the best solutions to be selected and otherwise;*
- *because of congestion around the center, the speed v_i of grain must be a decreasing function of time: more grains approach the center, the choice is more rigorous;*
- *after N_s sieving iterations, if there are no improvement of the solutions, we can change the sieve, that means increasing the hole diameter by replacing h by α*h ($\alpha>1$) as $\alpha=1.1,1.2,1.3,\ldots$. This will bring down more grains having higher diameters, since that will increase the speed of remaining seeds. Longer solution candidates will enter the critical zone A;*
- *in the other hand, to avoid algorithm divergence, since when we obtain S* $= \varnothing$, we reduce h by replacing it by $\beta*h$ $(0 < \beta < 1)$ as $\beta = 0.9,0.8,0.7,\ldots.$;
- *this second step will be repeated for N_s times of sieving, ;*
- *the two steps are also repeated Nb times of adding handfuls;*

The last step allows finding the best solution from the central critical area A which have radius b, that will accelerate the algorithm because we will just explore this small area instead of sieve entire contents;

2.4 The Modeling of Sieve Approach

Firstly, as in any other method, we have to carefully define a system codification of the sieve contents S and the solutions (grains) g_i. This codification has a great importance and a significant influence on the approach efficiency. For example, g_i

can be integer vector, binary array, string of characters, set of items, and complex object.

Examples:

- in single machine scheduling problems, g_i can be an integer sequence of jobs $(j_1,j_2,\ldots,j_k)$;
- in job shop scheduling problems, g_i can be an integer matrix of jobs

$$\begin{array}{l} j_{11},\ j_{12},\ \ldots,j_{1k}, \\ j_{21},\ j_{22},\ \ldots,j_{2k}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \end{array}$$

- in salesman travelling problems, g_i can be an integer sequence of cities $(j_1,j_2,\ldots,j_k)$;
- in images segmentation problems, g_i can be a binary array of digits 1 and 0;
- in search engine problems, g_i can be a string of characters like URL web sites;
- in affectation problems, g_i can be a set of couples as $\{(p_i,s_j)\}$;….

Then, we must establish the rule of the fitness function f to be used to evaluate the grains g_i by simulating the problem objective to be optimized and the rule of the speed $v_i(g_i)$ of grain g_i by using fitness function $f(g_i)$.

Finally, we must choose suitable values for the parameters defined above $Nb, Ns, n, r, b, \alpha, \beta$.

Note that, in this approach, we have always to maximize f. So, if we have to minimize the problem objective, we must adjust rules to satisfy these constraints, we can take for example:

$$f(g_i) = 1 - \ Objective'(g_i)/\sum_S Objective'(g_i)$$

$$v_i(g_i) = (1 - \ d_i/r)(\ 1 - f(g_i)/\sum_S f(g_k)$$

Where *objective'* is a sample function that simulating the objective, it must be easy to compute to accelerate the approach, however it must be proportional to the objective. It is clear that all grains have the same function f but they have different speeds v_i. This speed v_i may be chosen such as bad items may move too slowly toward the center or they may never reach it. Bad items means g_i such $f(g_i) > h$, *i.e.* $f(g_i) = h + \varepsilon \approx h$.

The first rule shows that $f(g_i)$ is depending of all both grains g_i, it means that the choice of a solution is depending of the others ones. The second rule shows that $v_i(g_i)$ is proportional against both $f(g_i)$ and d_i. That means: in one hand, the big grains will arrive quickly to the critical area before the little ones, in the other hand, when we increase h, some g_i will fall, so the sum $\sum f(g_k)$ will decrease, it imply that $v_i(g_i)$ will increase. That means, when some grains fall, that will allow the other grains to move more quickly.

Since we have initially $b < d_i < r$, we will assume the set S - A at the set S. It is clear that no need to sift or generate grains in zone A. In the other hand, the algorithm must not terminate by empty set A. so the number of iterations Nb must be enough great to avoid this bug.

In fact, we can take other rules for $f(g_i)$ and $v_i(g_i)$ but we must always respect the method principles to obtain good results. The translation of the above algorithm on a programming code produces the following program:

```
Program sieve;
S = ∅; A = ∅;  // initializations
Compute initial h; // lower bound according problem properties
For (k=1;k≤Nb;k++) // method iterations
  {
    For(i=1;i≤n;i++)  // add handful of n grains into sieve S
        { generate $g_i$, .$d_i$; // randomly by using uniform law
         S = S ∪{$g_i$;} // adding grain to S
        }
       for (t=1;t≤Ns;t++) // sieving
          {   for each $g_i$ of  S
              {  if (f($g_i$) ≤ h)  S = S - {$g_i$;} // removing grains from S
                 if (S = ∅) { h = β*h; loop; } // reducing h
                 if ($d_i$ ≤ b ) {S = S - {$g_i$;}; A = A ∪{$g_i$;}}
                                                    // making grain in critical area A
                 $d_i$ = $v_i$*$d_i$;   // decreasing distances
              }
          }   // end sieving
       if (A = ∅) h = α*h; // increasing h
   } // end method iterations
    min = ∞      // computing best solution
  for each $g_i$ of   A
       if (Objective($g_i$) < min) { min = Objective($g_i$) ; solution = $g_i$ ;}
  print solution, Objective(solution);
end.
```

To well complete this implementation, we must add the code for three functions:

- *Objective* : according the treated problem;
- *f*: according the objective function;
- v_i: according fitness function *f*;

Then, we need to write the code of "*generate procedure*" which is following used codification of grains and parameters values. Also, it is clear that this implementation is independent of specified problems. Since, it gives a formal generic model which can be applied to any optimization problem. We just need to well choose the right parameters and well establish above functions.

3 Sieve Algorithm For a Single Machine Scheduling

To show our approach efficiency, we will applied it on a single machine scheduling problem of independent jobs where the objective consists to minimize the sum of earliness and tardiness penalties against common due date. This problem was treated in many search subjects where was proofed as NP-hard problem and was applied in JIT (Just-In-Time) philosophy as in manufactures, commerce, and transport. In the recent years, this type of problems has received significant attention and become important with the advent Just-In-Time (JIT) concept, where early and tardy deliveries are highly discouraged. For example, the just-in-time-principle states that the right amount of goods should be produced or delivered at exactly the right time. According to our search, we found few research related to this problem: Kanet (1981) [11] [13], Lee and Kim (1995) [6], James (1997), Gordon et al. (2002) [9], Feldmann et Biskup (2003) [6] [7] [8], Hino, Ronconi et Mendes (2005) [12], Lin, Chou, Ying (2005), Biskup and Cheng (1999) [14], Hall and Posner (1991) [10]. On the other hand, some work have been done on solving this problem using exact methods as in [22] [23] [25]. While other researchers have treated heuristics methods as in [15] [16] [18] [21] [24] [27]. Some works used genetic algorithm as meta heuristic as in [19] [26] [28]. We note that the problem has treated in many options: single machine [6] [20] [25] [27], two machines [15] and multi machines [16] [17] [31].

For our propose method, we provide notations of this problem as below:

I : a set of n jobs: $I = \{ 1 , 2 , ... , n \}$;

d : common due date of all the n jobs ;

C_i : complete time of job i ;

p_i : processing time of job i ;

$E_i = \max\{d-C_i , 0\}$ (Earliness of job i)

T_i *= max {*C_i*-d , 0} (Tardiness of job i)*

α_i *: penalty per unit time of earliness for job i ;*

β_i *: penalty per unit time of tardiness for job i ;*

*h : parameter of common due date : d = h * T ; where:* $T = \sum_{k=1}^{n} p_i$ *;*

h $\in$ *{ 0.2 , 0.4,0.6,0.8 }.*

The definitions for a single machine scheduling problem with the objective to minimize $\sum_{k=1}^{n}(\alpha_i E_i + \beta_i T_i)$ is given below,

where n is independent jobs to be processed on a single machine without interruption with common due date d ; each job is available at time 0; each job must be processed just once. For each job i, the processing time p_i, the cost per unit time of earliness α_i, the cost per unit time of tardiness β_i are given and we assume as integer values;

3.1 Problem Properties

For this problem, an optimal solution must satisfy three optimality properties. To obtain the objective value more efficiently, these three properties are integrated in our meta-heuristic.

Property 1. An optimal schedule does not contain any idle time between any consecutive jobs.

Property 2. An optimal schedule is V-shaped around the common due date: the jobs complete before or on the common due date are sorted in decreasing order of the ratios p_i / α_i, and the jobs starting on or after the common due date are sorted in increasing order of the ratios p_i / β_i.

Property 3. In the optimal schedule, either the first job starts at time zero or the completion time of one job coincides with the common due date. These properties can be established using proof by contradiction.

3.2 Sieve Algorithm for a Single Machine Scheduling

In this implementation, we develop the above general algorithm on a computer program to be tested with real cases. Since, this program could be immediately implemented once, we have carefully defined the necessary elements of the problem as the objective function and some parameters values. The main operations of this program are given below and follows by explaination.

- *Initializing variables;*
- *Generating handful;*
- *Removing little grains;*
- *Sieving grains;*
- *Changing sieve;*
- *Getting best grain;*
-

Initializing variables

$S = \varnothing$: *S* is a set of grains, the contents of the sieve, initially empty. At any step of the algorithm we have: $S = \{g_i / b < d_i < r\}$;

$A = \varnothing$: *when sieving we will obtain* $A = \{g_i / d_i \geq b\}$; *Note that* $S \cap A = \varnothing$.

Generating handful

"*Generate*" is a procedure used to randomly generate *n* grains and their respective distances to the center. it is recommended to use uniform law to cover all space feasible solutions. We need also choose good structures for data representation.

Removing little grains

if $(f(g_i) \leq h)$ $S = S - \{g_i\}$: this operation consists to remove little grains, which having: $f(gi) \leq h$. All solutions of g_i such as $f(g_i) \leq h$ *are eliminated from S.*

Sieving grains

if $(d_i \leq b)$ $\{S = S - \{g_i;\}; A = A \cup \{g_i;\}\}$:
Note that, at least, we need the associated objective.
solution= g_i: we store the best solution associated to the best fitness.
improve=true: we note that there are improvement of the solution : no need to change sieve at this step.

$d_i = v_i * d_i$: this operation allows us to change all grains positions by updating (decreasing) their distances to the center. Here, we consider that the movements of grains are linear and decelerated. In mathematical terms, this operation can be written: $d_i^{t+1} = v_i * d_i^t$. We have $d_i \leq r$ and $0 < v_i < 1$, so: $d_i^{t+1} < d_i^t$, it implies the grain gradually approach from the center.

Changing sieve

There are two cases where we need change sieve:

i) if $(S = \varnothing)$ $h = \beta * h$: In mathematical terms, this operation can be written: $h_{k+1} = \beta * h_k$. We have $0 < \beta < 1$ (such as $\beta = 0.9, 0.8, \ldots$) to decrease the hole diameter of sieve. Note that we must carefully choose the first value of h to avoid the divergence of the algorithm (that means when h is too great, all grains will fall out. So, we need to select a suitable value for h and we increase it gradually.

ii) if $(A = \varnothing)$ $h = \alpha * h$: In mathematical terms, this operation can be written: $h_{k+1} = \alpha * h_k$. We have $\alpha > 1$ (such as $\alpha = 1.1, 1.2, \ldots$) to increase the hole diameter of sieve. This causes not only the fall of larger grains but also the acceleration of the grains remaining, which leads to the acceleration of the method.

Getting best grain

```
   min = ∞      // computing best solution
   foreach g_i of  A
        if (Objective(g_i) < min) { min = Objective(g_i) ; solution = g_i ;}
   print solution, Objective(solution);
print solution, Objective(solution)
```

The last operation consists to compute the best solution found and its objective.

4 Results and Analysis of Sieve Algorithm

The set of problems tested was selected from Biskup and Feldmann (2001,2003), with seven different instance sizes with n = 10; 20; 50;100; 200; 500;1000 jobs and 4 restrictive factors h = 0.2; 0.4; 0.6; and 0.8. They are used to determine the common due date d , by multiplying the total sum of all processing times, as follows: $d = h * T$ .There are 10 instances for each problem size for our computational results.

Firstly, we have computed the exact solutions for smaller sizes such as n = 5; 10 ; 20 ; 50 by using dynamic programming method then by using the proposed approach. Thus, we have obtained two solutions for each instance: Exact Solution (ES) and the Near Solution (NS). The deviation, $\Delta = NS - NE$ is inversely proportional to h as shown in fig. 2.

This variation of Δ means that when h is small; there are only few choices to schedule jobs because of their time processing p_i since, some jobs can not scheduled before common due date d . Inversely, when h is larger, there are many choices to schedule jobs against d , so we can conclude that the near solution is best when h is larger. Even when the size n of the problem is larger; this conclusion stay satisfied because the sequence can begin at any time near d .

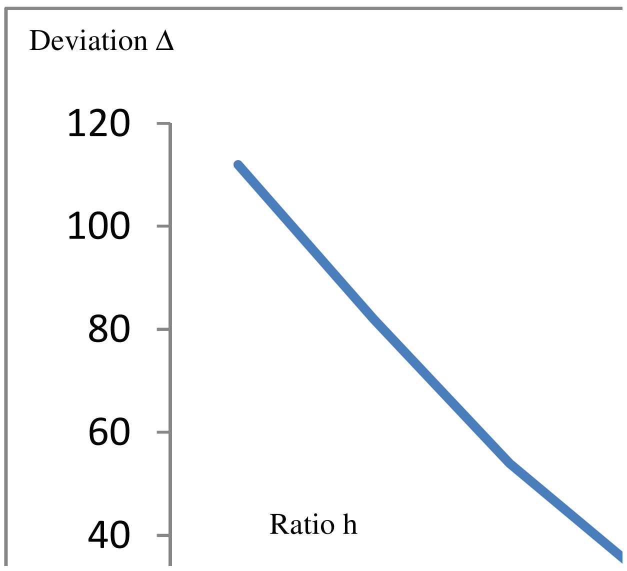

Fig. 2 : Variation of the deviation Δ as a function of the ratio h.

Then, we have chosen the most restrictive set of 140 instances, i.e. with h = 0.2 and h = 0.4. In order to measure the effectiveness of the results obtained, we computed the Ratio Percentage Deviation (RPD) of the average of our solution values (SM) (Sieve Method) of every 10 instances over the average of the corresponding benchmark values provided by Biskup and Feldman Approach (BFA) as follows: RPD = {(SM – BFA)/BFA }* 100.

Our results were obtained with following number of iterations: 500 iterations for n= 10, 20, 50 jobs; 1000 iterations for n= 100, 200 jobs, and 10000 iterations for n = 500 to terminate the algorithm. Table 1 illustrates the findings of our proposed method using various number of jobs.

Table 1: Percentage deviation of the average solutions

$h=0.2$	$n=10$	$n=20$	$n=50$	$n=100$	$n=200$	$n=500$	$n=1000$
BFA	1674.4	6429.2	37583.7	141143.3	543591.2	3348405.6	13293514.6
SM	1674.4	6204.7	35505.1	133021.7	509500.7	3147715.8	9208717.2
RPD%	0	-3.62	-5.85	-6.11	-6.69	-6.38	-44.36
$h=0.4$	$n=10$	$n=20$	$n=50$	$n=100$	$n=200$	$n=500$	$n=1000$
h=0.4	n=10	n=20	n=50	n=100	n=200	n=500	n=1000
BFA	973.1	3703.7	21419.5	82120.2	315312.9	1917425.8	7651046.5
SM	973.1	3651.1	20519	79051.7	307061.2	1787201.8	7320456.9
RPD%	0	-1.44	-4.39	-3.88	-2.69	-7.29	-4.52

The computational results reported in the Table 1 above show that our algorithm improves the average of all instances. The overall average improvement for h = 0.2 is around -10.43% while it is –6.94% for h = 0.4.

5 Conclusion

The paper proposes a new approach, " Sieve Approach" for solving optimization problem. Sieve method is inspired from the sieving operation to give a new alternative for existing approaches as genetic algorithm, and ant colonies. This approach consists random and iterative batches of solutions in a great set. It eliminates bad items and makes good items in a small set by moving them according to appropriate speed. We provide a mathematical formulation for this approach as a general algorithm. Since the goal of the proposed method is to find the best solutions in reasonable time especially for great problems sizes, we have elaborated the different components by proposing the principles, formulating the algorithm, defining different parameters and finally producing the associated program. The proposed approach has been tested on the single machine scheduling problem with proper parameters values for better results. We hope that our proposed method will open more rooms for improvement in solving the optimization problems of large scale datasets.

References

[1] Dorigo, M. Di Caro G. (1999) The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo, F. Glover (eds) New Ideas in Optimization, McGraw-Hill, London, UK, pp 11-32.

[2] Halberstam, H.; Richert, H.-E., Sieve Methods, Academic Press, 1974.

[3] Holland J.H. (1975) Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992).

[4] Ibaraki, T. Enumerative Approaches to Combinatorial Optimization", Annals of Operations Research vol. 10, 11, J.C.Baltzer 1987.

[5] Kirkpatrick, S. Gelatt C. D.; M. P. Optimization by Simulated Annealing, Vecchi. Science, New Series, Vol. 220, No. 4598. (May 13, 1983), pp. 671-680.

[6] M. Feldman, and D. Biskup, "Single-machine scheduling for minimizing earliness and tardiness penalties by meat-heuristic approaches", Computer & Industrial Engineering, 44 (2003) 307-323.

[7] M. Feldman, and D. Biskup, "Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates", Computer & Industrial Engineering, 28 (2001) 787-801.

[8] M. Feldman, and D. Biskup, "Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates", Computer & Industrial Engineering, 28 (2001) 787-801.

[9] R. Hassin, and M. Shani, "Machine scheduling with earliness and tardiness and non-execution penalties", Computer & Industrial Engineering, 32 (2005) 683-705.

[10] G. Li, "Single machine earliness and tardiness scheduling", European Journal of Operational Research, 96 (1997) 546-558.

[11] S. W. Lin, S. Y. Chou and K. C. Ying, "A sequential exchange approach for minimizing earliness-tardiness penalties of single machine scheduling with a common due date", European Journal of Operational Research, xx Volume 177, Issue 2, Pages 1294–1301.

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

[13] K. R. Baker, G. D. Scudder, "Scheduling with earliness and tardiness penalties: a review", European Journal of Operational Research, 160 (2005) 190-201.

[14] T. Vallée, and M. Yiltizogli, "Présentation des algorithmes génétiques et leurs applications en économie", Mai 2004, V .5.

[15] C.S. Sung, J.I. Min. "Scheduling in a two-machine flow shop with batch processing machine(s) for earliness/tardiness measure under a common due date". European J. Oper. Res., 131 (2001), pp. 95–106

[16] J. Bank, F. Werner. "Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties". Mathl. Comput. Modelling, 33 (4/5) (2001), pp. 363–383.

[17] H. Sun, G. Wang. "Parallel machine earliness and tardiness scheduling with proportional weights". Comput. Oper. Res., 30 (2003), pp. 801–808.

[18] Ching-Jong Liao, Che-Ching Cheng. "A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date". Computers & Industrial Engineering. Volume 52, Issue 4, May 2007, Pages 404–413.

[19] Liu Min, Wu Cheng. "Genetic algorithms for the optimal common due date assignment and the optimal scheduling policy in parallel machine earliness/tardiness scheduling problems". Robotics and Computer-Integrated Manufacturing. Volume 22, Issue 4, August 2006, Pages 279–287

[20] Bülbül, K., Kaminsky, P., Yano, C.: “Preemption in single machine earliness/tardiness scheduling”. Journal of Scheduling 10, 271–292 (2007).

[21] Detienne, B., Pinson, E., Rivreau., D.: “Lagrangian domain reductions for the single machine earliness-tardiness problem with release dates”, European Journal of Operational Research 201, 45–54 (2010).

[22] Sourd, F., Kedad-Sidhoum, S., “A faster branch-and- bound algorithm for the earliness-tardiness scheduling problem”, Journal of Scheduling 11, 49–58 (2008).

[23] Sourd, F., “New exact algorithms for one-machine earliness-tardiness scheduling”. INFORMS Journal on Computing 21, 167–175 (2009).

[24] Yau, H., Pan, Y., Shi, L., “New solution approaches to the general single machine earliness tardiness problem”. IEEE Transactions on Automation Science and Engineering 5, 349–360 (2008).

[25] Débora P. Ronconi; Márcio S. Kawamura , "The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm”, Comput. Appl. Math. vol.29 no.2 São Carlos June 2010.

[26] S. Webster, D. Jog & A. Gupta, “A genetic algorithm for scheduling job families on a single machine with arbitrary earliness/tardiness penalties and an unrestricted common due date”, International Journal of Production Research. Volume 36, Issue 9, pp. 2543-2551, 1998.

[27] Shih-Hsin Chen, Min-Chih Chen, Pei-Chann Chang, and Yuh-Min Chen, “EA/G-GA for Single Machine Scheduling Problems with Earliness/Tardiness Costs", Entropy 2011, 13, 1152-1169; doi:10.3390/e13061152.

[28] Chang, P.C.; Chen, S.H.; Liu, C.H. “Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems”. Expert Syst. Appl. 2007, 33, 762–771.

[29] Chaudhuri, A. and De, K. Fuzzy Genetic Heuristic for University Course Timetable Problem. International Journal of Advances in Soft Computing and its Application, 2,1 (2010), 1-24.

[30] Premalatha, K and Natarajan, A.M. Combined Heuristic Optimization Techniques for Global Minimization. *International Journal of Advances in Soft Computing and its Application,* 2,1 (2010), 1-15.

[31] Premalatha, K. and Natarajan, A.M. Hybrid PSO and GA Models for Document Clustering. *International Journal of Advances in Soft Computing and its Application,* 2,3 (2010), 1-20.

[32] Bahesti, Z. and Shamsuddin, S.M. A Review of Population Based Meta-Heuristic Algorithm. International Journal of Advances in Soft Computing and its Application, 5, 1(2013), 1-35.