Singh et al., *Cogent Engineering* (2015), 2: 1011929
http://dx.doi.org/10.1080/23311916.2015.1011929

cogent engineering

COMPUTER SCIENCE | SHORT COMMUNICATION

Disk scheduling using a customized discrete firefly algorithm

Amandeep Singh[1*,†], Sidharth Thapar[1,†], Abhishek Bhatia[1,†], Saurabh Singh[1,†] and Rinkaj Goyal[1,†]

Received: 01 November 2014
Accepted: 20 January 2015
Published: 11 March 2015

*Corresponding author: Amandeep Singh, USICT, GGS Indraprastha University, Sector 16C, Dwarka 110078 New Delhi, India
E-mail: ads71993@gmail.com

Reviewing editor:
Jenhui Chen, Chang Gung University, Taiwan

Additional information is available at the end of the article

Abstract: This study empirically investigates the usage of a customized discrete firefly algorithm (DFA) for ordering the disk requests to minimize the total access time. The procedure simulates the movement of each firefly within a population towards others using a variation of edge-based mutation. The algorithm was applied to randomized standard disk sequences with varying length of input disk requests. Owing to the greater impact of seek time in the determination of access time for a disk having sizable number of tracks, this has been taken as the primary performance factor in the scheduling of tasks. The analysis of the results obtained establishes the relative advantage of using the firefly optimization method over the traditional disk scheduling algorithms.

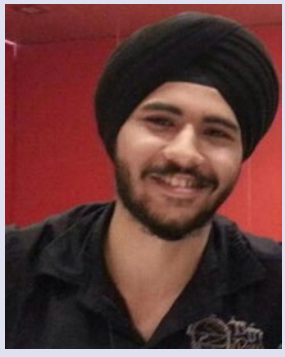
Subjects: Automation; Evolutionary Computing; Simulation & Modeling; Computer Engineering; Computer Science (General); Operating Systems; Systems & Computer Engineering; Systems Architecture

Keywords: disk scheduling; discrete firefly algorithm; seek time

1. Introduction

The file system must be efficiently accessible, particularly with the hard drives since processor speed and disk capacity exhibit manifold increment in a year in comparison to the disk speed (Ruemmler & Wilkes, 1994). The diminished growth in the disk speed technology is considerably related to the difficulty in the advancement of mechanical components. This contrast in the processor speed and disc capacity, and is unlikely to change in the near future (Ökdem & Karaboga, 2006; Rahmani, Arshad, & Moghaddam, 2009). Therefore, development of advanced disk scheduling algorithms is of primary importance.

The disk scheduler processes and schedule disk requests by estimating the seek time and the possible rotational delay of a request (Jacobson & Wilkes, 1991; Seltzer, Chen, & Ousterhout, 1990). Since, disk scheduler knows request processing time, it (greedily) picks the request asking for the



Amandeep Singh

ABOUT THE AUTHORS

Amandeep Singh, Sidharth Thapar, Abhishek Bhatia, and Saurabh Singh are pursuing a Bachelor of Technology degree from University School of Information and Communication Technology, GGS Indraprastha University, Delhi (INDIA). They are the active members of the students' interest group on Bio-inspired computation facilitated by Rinkaj Goyal, who is working as an assistant professor. Major activities of this interest group includes agent-based simulation development, multi-agent System, and Machine learning.

PUBLIC INTEREST STATEMENT

Computer deals with multiple processes requesting for disk access. An efficient handling of these aggregated requests would result in an improved system performance. The operating system uses disk scheduling algorithms to decide which request to fulfil first. In this study, we apply a variant of firefly algorithm for ordering the disk requests to minimize the total seek time. The procedure adopted here simulates the movement process of each firefly within a population, and is similar to other adaptive learning and artificial intelligence techniques.

minimum time. Further, disk has non-uniform access time and this has direct dependency upon seek time (Section 2); therefore, reordering of the disk requests may considerably reduce the access time latency. An efficient exploration of search space will aid in the significant reduction in average access time. Moreover, it is extremely imperative to maintain a steadiness between the starvation of disk requests and fairness in the allocation of scheduler. It is desirable to obtain such a balance for improvement in performance of the disk scheduler. The disk scheduling can be formulated as an instance of a combinatorial optimization problem as follows.

Given a set of n real-time disk tasks $T = T_1, T_2, \ldots, T_n$, where each task, T_i is a quadruple (r_i, d_i, s_i, $size_i$) . Each disk request is defined by its ready time(r_i), deadline time(d_i), sector number(s_i), and data size($size_i$). Ready time is the initial start time of a particular disk. Deadline time is the latest time of the completion of a disk task. To meet the timing requirements, the start time of a task should not be earlier than its ready time. Additionally, its finish time should not be later than the related deadline time (Chen, Yang, & Lee, 1992; Huang, Lu, Chou, & Shib, 2005; Ruemmler & Wilkes, 1994).

Therefore, a feasible permutation schedule for the given set of tasks, $T_z = T_{z(1)} T_{z(2)} \ldots T_{z(n)}$, will be $min_{\forall z} f_{z(n)}$, where $f_{z(n)}$ is defined as the finish time of the newest task in the schedule, and $z(i)$ is the index function which is taken, as a permutation of $\{1, 2, 3, \ldots n\}$ (Huang, Lu, Chou, & Shib, 2005).

Hence, disk scheduling is a combinatorial optimization problem, in which an optimal solution is sought over a discrete search space.

Seek-cost function is a measure of how fast a hard drive can move read/write heads to the desired location (Abbott & Garcia-Molina, 1990). A linear seek-cost disk scheduling problem is an NP-complete problem since this is polynomial time reducible to well-known NP-complete problem viz partition problem(PP) (Huang, Lu, Chou, & Shib, 2005; Huang, Lu, Chou, & Shih, 2005; Ruemmler & Wilkes, 1994). Different techniques like approximation, randomization, restriction, parametrization, heuristic, and metaheuristic have been reported in the literature to approach NP-Complete problems (Blum & Roli, 2003; Yagiura & Ibaraki, 2001).

The firefly algorithm (FA) is a metaheuristic approach inspired by the flashing behavior of fireflies. FA and its variants have been applied to solve optimization and classification problems like feature selection and fault detection (Banati & Bajaj, 2011; Falcon, Almeida, & Nayak, 2011), antenna design (Basu & Mahanti, 2011), structural design (Gandomi, Yang, & Alavi, 2011), semantic web composition (Jati & Suyanto, 2011), clustering (Senthilnath, Omkar, & Mani, 2011), and dynamic problems (Abshouri, Meybodi, & Bakhtiary, 2011). Firefly-based algorithms have been effectively applied for scheduling, task graphs, and job shop scheduling problems, and reported to perform better than other popular metaheuristics approaches (Gandomi et al., 2011). Other variants of FA include binary firefly algorithm and evolutionary discrete FA (Abshouri et al., 2011; Palit, Sinha, Molla, Khanra & Kule, 2011). The performance of FA-based approaches can further be improved by using preferential directions in the firefly movements.

In this study, we use the variation of the recently formulated metaheuristic technique known as new discrete firefly algorithm (NDFA) first proposed by Yang, Cui, Xiao, Gandomi, and Karamanoglu (2013).

Different scheduling policies have been proposed to manage the queue of disk requests efficiently by taking different disk parameters into consideration. The seek time is usually the primary parameter considered in the implementation of the scheduling algorithms. For large disk requests, rotational time is often omitted in most of the proposed algorithms because, it is lower in comparison to seek time. Apart from parameters indicated above, the other relevant parameters are request deadline, request priority, and the request type (Andrews, Bender, & Zhang, 1996).

In the FA-based method to schedule disk requests, firefly parameters have been adjusted to result in an improved performance over traditional strategies. Experimental results are promising, and clearly indicate the superiority of the proposed approach.

This study progresses as follows: Section 2 provides the modelling methodology used in this study. This section provides the brief explanation of the real-time disk scheduling problem along with an induction of firefly algorithm (FA) and its variants. Section 3 illustrates the proposed approach and our experimental results are presented in Section 4 along with the conclusion that is stated in the Section 5.

2. Modelling methodology

A disk comprises a number of cylinders along with disk head. Each cylinder further contains a collection of tracks, where each track has the same distances from the center of the disk. Further, every track has sectors with each sector having 32 bytes.

2.1. Problem statement

Given a set of n real-time disk requests $r = r_1, r_2, r_3 \ldots r_{n-1}, r_n$, the goal is to find a schedule with minimal access time for the given set of requests, where n is the size of the queue which depends on the type of application (Yeh, Kuo, Lei, & Yen, 1998). Assuming that the given disk scheduling sequence has to serve two sequential tasks r_j and r_i. A seek-time cost is incurred when we serve the disk request r_i; that is when the disk-head travels from previous disk task cylinder(r_j)to the requested disk cylinder(r_i) (Abbott & Garcia-Molina, 1990). In addition, a rotational latency, that is the amount of delay in obtaining information from a disk due to the rotation of the disk, is also added to reach the desired sector within r_i. Finally, the transfer time is the time required to transfer requested data from disk to a buffer (Ruemmler & Wilkes, 1994).

Equation 1 gives the $Access_n$(access time) for a given disk request (Abbott & Garcia-Molina, 1990).

$$Access_n = Seek_n + Rotational\,Latency + TransferTime \tag{1}$$

where, n is the track length of a given disk request.

In this study, the transfer time and rotational latency parameters are taken as a single parameter i.e. disk constant. Nonetheless, $Seek_n$ is a nonlinear equation that depends on the seek factor and the associated track number (Equation 2) (Abbott & Garcia-Molina, 1990).

$$Access_n = SeekFactor \times \sqrt{n} + DiskConstant \tag{2}$$

For disks that have a sizeable number of tracks, $Seek_n$ has a greater effect on the equation term and "Disk constant" can be ignored (Equation 2). Therefore, the track number becomes a significant parameter in these scheduling algorithms. Determining these parameters has been subject of intense research for some time now. For example, in Fujitsu M2361A hard disk (FUJITSU, 1984), the seek time with movement distance $D(j, i) = (r_i - r_j)$ is calculated as follows (Equation 3)

$$Seektime(D_{j,i}) = \begin{cases} 4.6\,\text{ms} + 0.87\,\text{ms}\sqrt{(D_{j,i})} & \text{if} D_{j,i} \leq 239 \\ 18\,\text{ms} + 0.028\,\text{ms}(D_{j,i} - 239) & \text{if} D_{j,i} < 239 \end{cases} \tag{3}$$

2.2. Firefly algorithm

The FA is a metaheuristic-based algorithm mimicking the flashing behavior of fireflies. The purpose of a firefly's flash is a signalling mechanism for attracting other fireflies. The execution of FA proceeds with the following two idealized rules (Yang, 2010):

(a) All fireflies are attracted to other fireflies irrespective of their sex, and the measure of the attractiveness of a firefly is in direct proportion to its brightness. The less bright firefly will move toward the brighter one. The attractiveness of a firefly decreases with increasing distance. If a firefly can not find the brighter one nearby than it makes a random movement.

(b) The landscape of an objective function determines the brightness (light intensity) of a firefly.

For a maximization problem, the brightness can simply be proportional to the objective function. Although, other forms of brightness can also be formulated similar to the way, the fitness functions are devised while applying other nature-inspired techniques like genetic algorithms or bacterial foraging algorithm (Gazi & Passino, 2004; Yang, 2010).

Each firefly represents one solution for the disk scheduling problem, which is a permutation representation. Here, light intensity is a value that represents the brightness of a firefly. This value depends on the total seek time of a schedule belonging to a firefly. Since the purpose is to find a schedule with the minimum seek time, a firefly that has lesser seek time will have a greater light intensity (brighter).

2.3. Functional parameters of a firefly

Succeeding sections elaborates different functional parameters utilized in devising the proposed algorithm.

2.3.1. Light intensity and attractiveness

The execution of firefly algorithm depends on the disparity in light intensity and the conceptualization of attractiveness;

Light intensity represents the brightness of a firefly and decreases with distance from its source. Furthermore, due to absorption of light by air, attractiveness also varies with the degree of absorption (Equation 4).

$$I = I_o \exp^{-\gamma r^2} \tag{4}$$

where I_o is the initial light intensity and γ is the absorption coefficient (Arora & Singh, 2013).

The high light intensity represents a more optimum order of requests in the request queue. The attractiveness of a firefly is bounded to its brightness associated with the ciphered fitness function. The original FA defines the attractiveness function β_r for any flat decreasing function as follows (Equation 5) (Arora & Singh, 2013):

$$\beta(r) = \beta_o \exp^{-\gamma r^2} \tag{5}$$

where $\beta(r)$ is the attractiveness of a firefly at a distance r, which is the distance between two fireflies. β_0 is the brightness of a brighter firefly, and γ is a fixed light absorption coefficient. For any two fireflies, firefly i and another brighter firefly j, initially we calculate the distance (r) between firefly i and firefly j by using Equation 6. We then calculate the attractiveness of firefly j observed by firefly i at distance r using Equation 5. If the attractiveness of firefly j is greater than the brightness (light intensity) of firefly i, then firefly i will move toward firefly j, Otherwise, firefly i will move randomly.

2.3.2. Distance

In continuous optimization problems, the distance between two fireflies is calculated using Euclidean distance. For disk scheduling, the distance between firefly i and firefly j can be defined as the number of different edges between them. In Table 1, three edges 120-15, 500-79, and 60-11 in firefly j do not exist in firefly i. Hence, the number of different edges between firefly i and firefly j is 3. Then, the distance between two fireflies is calculated using following equation (Equation 6) (Yang et al., 2013).

Table 1. Distance between two fireflies *i* and *j*

Firefly *i*	140	13	120	79	60	15	500	11	90	10	16	300	2
Firefly	140	13	120	15	500	79	60	11	90	10	16	300	2

$$r = (A/n) * 10 \tag{6}$$

Where A is the total number of different edges between two fireflies, r is the distance between any two fireflies and n is the number of disk requests. This equation scales r in the interval $[0, 10]$ and used for the attractiveness calculation.

2.3.3. Light absorption coefficient

In essence, the light absorption coefficient γ characterizes the variation of the attractiveness value of a firefly. Its value is crucial in determining the speed of convergence and the behavior of FA. Theoretically, $\gamma \in [0, \infty)$, however, in practice, γ depends on the properties of the problem. If $\gamma \to 0$, the attractiveness will be constant and $\beta(r) = \beta_0$. In this case, the attractiveness of a firefly will not decrease when viewed by another. $\gamma \to \infty$, signifies close to zero value of attractiveness of a firefly, when viewed by another firefly. The coefficient γ determines how much light intensity changes the attractiveness of a firefly over distances.

2.3.4. Movement

For each firefly, find the brightest or the most attractive firefly. If there is a brighter firefly, then the less bright firefly will move toward the brighter one and if there is no brighter one, it will move randomly. We use a movement scheme which follows the edge-based movement. This scheme guarantees that after one firefly moves toward a brighter one, the distance between them will decrease (Yang et al., 2013).

2.3.5. Fitness function

In disk scheduling problem, the optimal solution is the best order of disk requests in the request queue. Seek time is used as the fitness function in this study (Equation 3).

3. Proposed solution

Firefly was originally designed to solve continuous optimization problems (Yang, 2009). However, the FA can also be discretized to solve the permutation problems. The discrete firefly algorithm (DFA) has been successfully implemented to solve flow shop scheduling problems amongst others. The DFA outperforms other widely employed metaheuristic algorithms such as ant colony algorithm (Sayadi, Ramezanian, & Ghaffari-Nasab, 2010). Variation of DFA considered in this study includes:

(a) Evolutionary discrete firefly algorithm (EDFA): recently, EDFA has been developed for solving TSP (Jati & Suyanto, 2011). However, in EDFA, firefly has no specific direction to move. Therefore, evolutionary strategies like inversion mutation are adopted to determine the next movement. Each firefly moves for m times using inversion mutation. The initial index of the chromosome is chosen randomly in the beginning. Each firefly will have m new solutions. Therefore, after p moves, $p * m$ new solutions will be generated resulting into $p * m + 1$ total solutions, out of which p best fireflies are chosen as the new population. Total number of solutions are constant after each generation.

(b) New discrete firefly algorithm (NDFA): the inversion mutation scheme which is used in EDFA does not always guarantee the decrease in the distance between the 2 fireflies. Therefore, a new DFA algorithm is introduced by Yang et al. (2013), which uses edge-based movement when a firefly is to move towards another brighter firefly.

Figure 1 elucidates the edge-based movement mechanism used in NDFA. Two adjacent nodes x and y collectively form an $edge(x - y)$ or $(y - x)$. In the edge-based movement between firefly i and brightest firefly j, a unique (which does not exist in firefly i) $edge(x - y)$ in firefly j is selected at

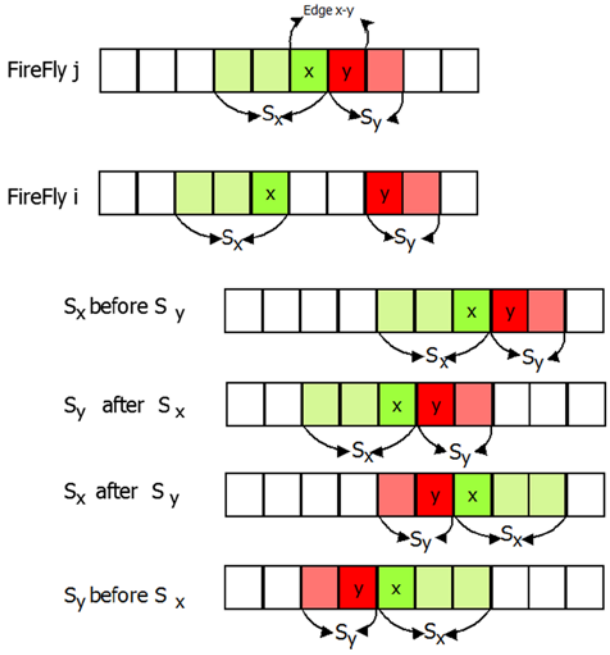


Figure 1. Illustration of edge-based movement scheme.

random. After determining the positions of nodes x and y in the firefly i, segment around each node is created comprising of the node itself. For each node (x and y), segments are extended towards left and right to include those edges to the segment that are the part of firefly j. This extension of the segments continues until no common edge can be found between firefly i and firefly j. Subsequently, these two segments are merged such that nodes x and y appear adjacent to each other in firefly i also. Thereby, referring to the addition of an $edge x-y$ to the firefly i. Henceforth, a segment built starting from node n is referred as S_n. There are four ways to accomplish this merger of segments resulting into four new solutions (fireflies)

(1) Inserting S_x before S_y
(2) Inserting S_x after S_y
(3) Inserting S_y before S_x
(4) Inserting S_y after S_x inversion of some these segments might be required to ensure the presence of $edge(x-y)$ in firefly i.

This new scheme follows an inclusion of a new edge which exists in the brighter firefly and is not a part of this firefly. This decrease in the number of edges that are unique to the brighter firefly guarantees the decrease in the distance between the two fireflies. Each firefly will have 2 cases:

Case 1 If it finds a brighter firefly, it uses new edge based mutation.
Case 2 If no brighter firefly is found, firefly will move randomly (inverse mutation).

When edge-based movement is used, four new solutions are found after the mutation. One of the four solutions is chosen at random and added to the population. After p fireflies move using this scheme, p best fireflies are chosen according to the objective function. A random movement of firefly generates a new solution using inversion mutation in m different positions, on a chromosome.

In this paper, we adopt NDFA, which employs the new edge-based movement as illustrated through the following pseudocode (Algorithm 1). Initially, each firefly generates an initial solution randomly. For each firefly, the brightest or most attractive firefly is identified. Similar to NDFA, less bright firefly

movement towards the brighter one will yield four new solutions. In the case of edge-based movement, instead of taking randomly one of the four solutions generated (as the case with NDFA), we have taken the best among the four mutations and the original firefly that replaces the original firefly in the population. For the next iteration, the set of better fireflies will be determined on the basis of the defined objective function viz seek time. This condition will continue until the maximum iteration is reached. In metaheuristic algorithms, the evaluation of the objective function is the most expensive computation. Owing to the two nested for loops involving population sizes i.e. p (one *for* loop is implicit in the calculation of most attractive firefly) and an outer loop for the iteration t, the complexity of the algorithm is $O(p^2t)$. Since, p is small in comparison to the number of iterations (t), the complexity of the algorithm is linear in terms of t. Furthermore, we observe an early convergence (Section 4.3) in the execution of the algorithm, which leads to a linear complexity comparable to other algorithms considered in this study (Yang et al., 2013).

Algorithm 1: The new Edge based DFA Algorithm (Yang et al., 2013)

```
Define objective function f(x)
Define population size p
Define light absorption coefficient γ
Define updating index m
List <Firefly> temp=new List <Firefly>
Firefly[] x=new Firefly[p];
for i → p do
    x[i]=Generate_initial_solution
while true do
    for i = 1 → p do
        Firefly f=Get_most_attractive_firefly(x[i])
        if f! = null then
            Firefly[ ] new_Solutions=new Firefly[4]
            new_Solutions=Adapted_Edge-based_movement(x[i])
            x[i]=Max_of(new_Solution,x[i])
        else
            for j = 1 → m do
                x[i]=Move_random(x[i])
```

4. Simulation experiment and results

In this section, we confer the experimental results obtained by applying the proposed approach. A comparison of the results obtained with previous popular techniques establishes the suitability of proposed approach. For all implementations, we use a personal Computer with 2.10 GHZ of CPU and 4GB of RAM in the java environment. Equation 3 provides the specifications of the simulation built to model the disk requests with a single data (Table 2) (FUJITSU, 1984; Seltzer et al., 1990).

Figure 2 demonstrates the execution steps of proposed algorithm. The values of following parameters need to be determined to apply the proposed algorithm;

(a) Queue size: Maximum number of disk requests that the queue can accommodate.

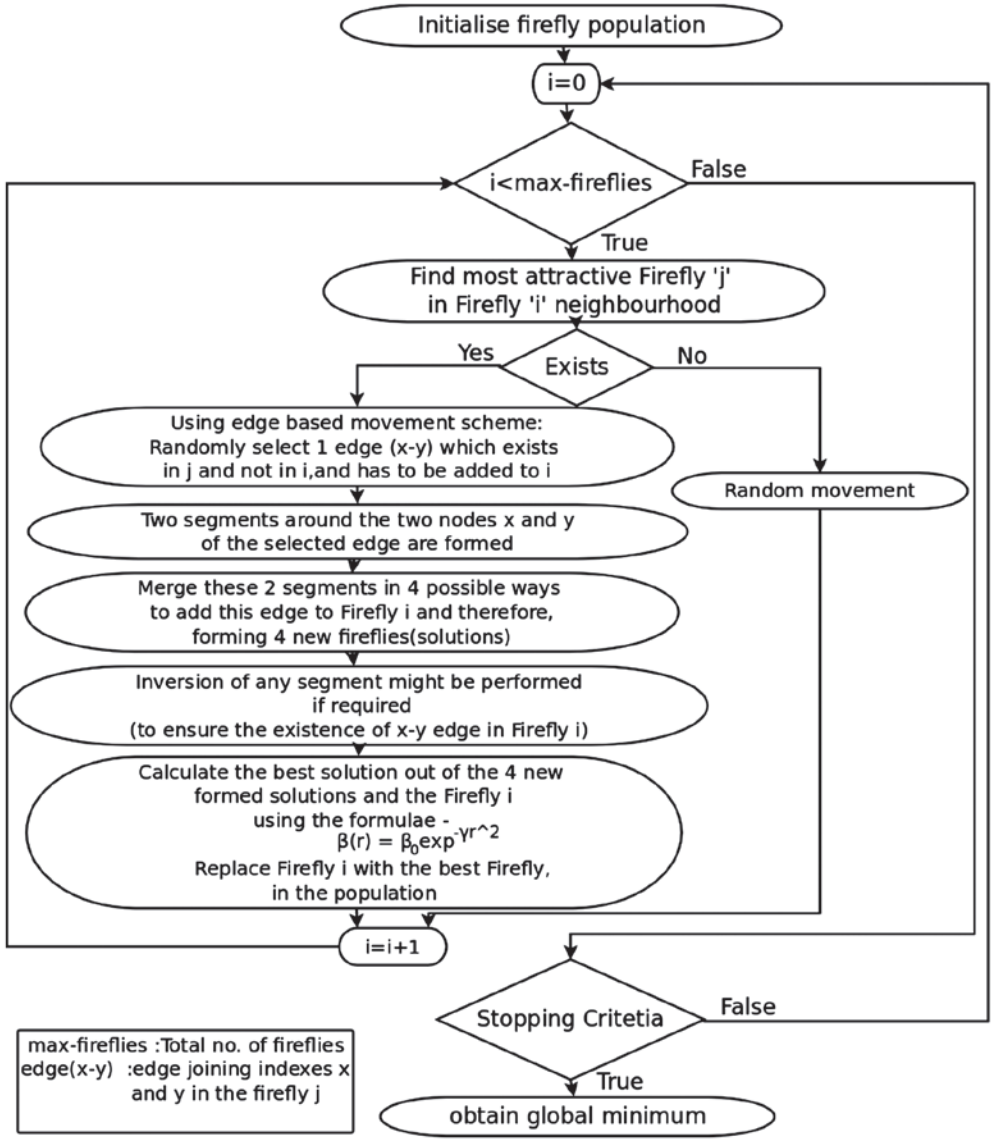
(b) The maximum track number of the disk.

(c) Values of the control parameters

Table 2. Disk parameters pertaining to the disk model of Fujitsu M2361A

Cylinders per disk	842
Data transfer rate (MB/sec)	2.458
Seek time function (ms)	$\begin{cases} 4.6 + 0.87\times\sqrt{(D_{j,i})} & \text{if} D_{j,i} \le 239 \\ 18 + (D_{j,i} - 239)\times 0.028 & \text{if} D_{j,i} < 239 \end{cases}$
Rotational speed (RPM)	3,600 + 2%

Figure 2. Flow chart of the working scheme.



4.1. Performance for different instances disk sequences at given queue size

The performance of traditional scheduling algorithms depends on the number and type of disk I/O requests. For example, in the case of heavy load on the disk, SCAN, or C-SCAN algorithms are preferred. Since, in these algorithms, head movement switches at the end of the disk. Otherwise, SSTF or LOOK are the commonly applied traditional scheduling algorithms. But, these techniques are not

adaptive in nature and only use track information to optimize seek time. On the other hand, Bio-inspired algorithm efficiently explore the search space in different scenarios and provide at least equally good solution, if not better.

We simulated the traditional disk scheduling algorithms, namely LOOK, CSCAN, FCFS, SCAN, SSTF (Teorey & Pinkerton, 1972), Silberschatz, Galvin, & Gagne, 2013) and compared their performance with the proposed approach. The execution of algorithms took different random input disk requests of queue size of 15.

Figure 3 illustrates the relative performance of different algorithms by plotting a graph between different instances of randomly generated sequences of disk requests with the queue size, and the value of total seek time in milliseconds.

We observed that FCFS show much higher values of total disk seek time in comparison to the other traditional disk scheduling algorithms (LOOK, CSCAN, SCAN, and SSTF). Whereas SSTF and CSCAN showed consistently low total seek time in contrast to the other known scheduling algorithms.

The proposed firefly algorithm prominently resulted in the lowest seek time in comparison to all other algorithms over all the randomly generated disk sequences of queue size of 15 (Figure 3).

4.2. Variation in the total seek time with increasing queue size

The performance of different algorithms was tested with 10 randomly generated disk requests against the queue size of 10, 15, 20, 25, and 30, respectively. To include more possibilities, we have plotted the values by averaging over 10 instances. Each task has been assigned an equal priority. The comparison of the proposed algorithm with existing algorithms (Figure 4) makes it clear that the algorithm results in the lowest total seek time, but deteriorates in its performance in contrast to others with increasing queue size. The seek time of the proposed algorithm increases with an increase in the queue size, but the increment is almost linear.

4.3. Average Fitness of the Individuals from each successive generation

The overall fitness of individuals from each successive generation till the algorithm terminates has been displayed to observe the working of the applied algorithm. That indicates the extent to which a generation has learned from the previous one. The improvement in the subsequent generation is clearly evident, while not getting trapped in local optimums. The graph shows (Figure 5) total seek time plotted for the number of generations and the population evolved for a significant time over 500 generations. It shows a slow trend downward, with the biggest jump occurring in the first 50 generations. After the 70 generations, individuals maintained the stumbling behavior but prone towards the target. We observe a steep falling slope for low numbers of generations followed by a

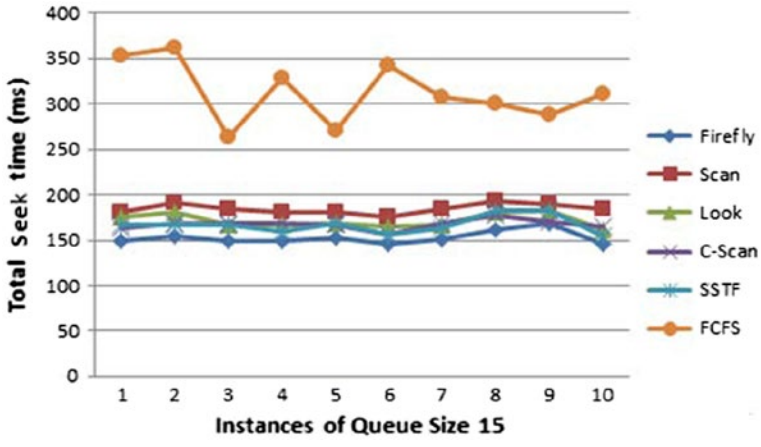


Figure 3. Comparison of total seek time for the scheduling algorithms at a queue size of 15.

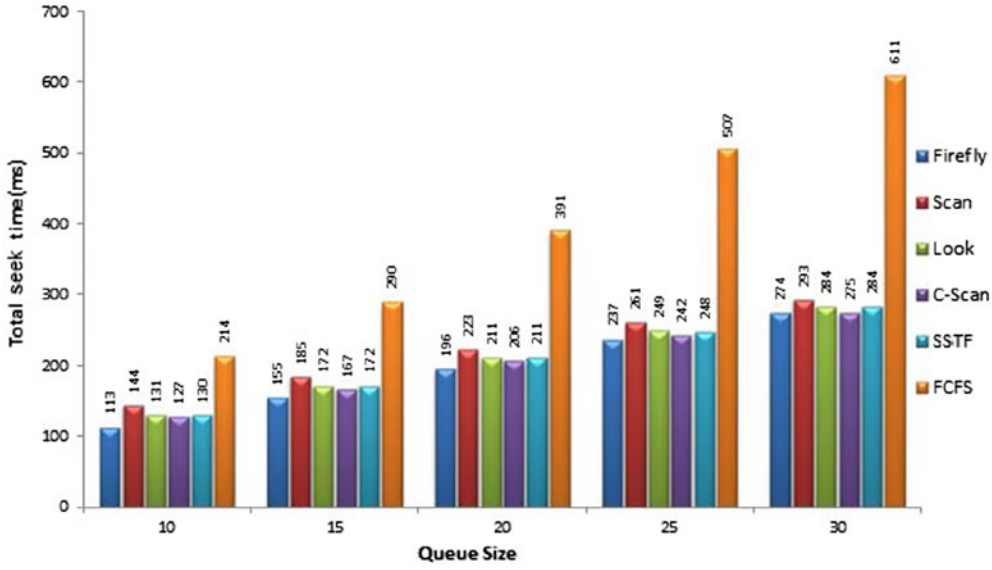


Figure 4. Performance for different instances disk sequences at given queue size.

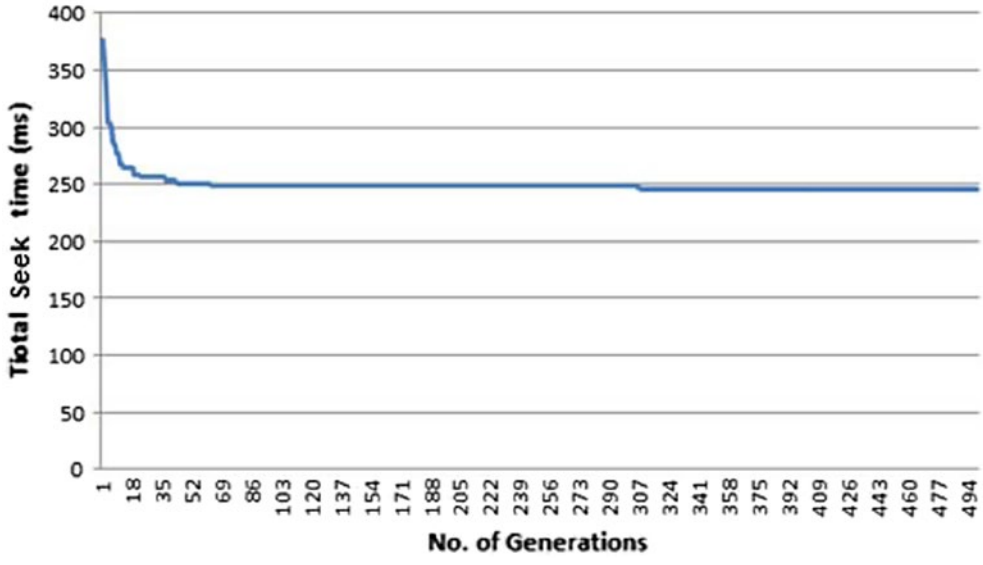


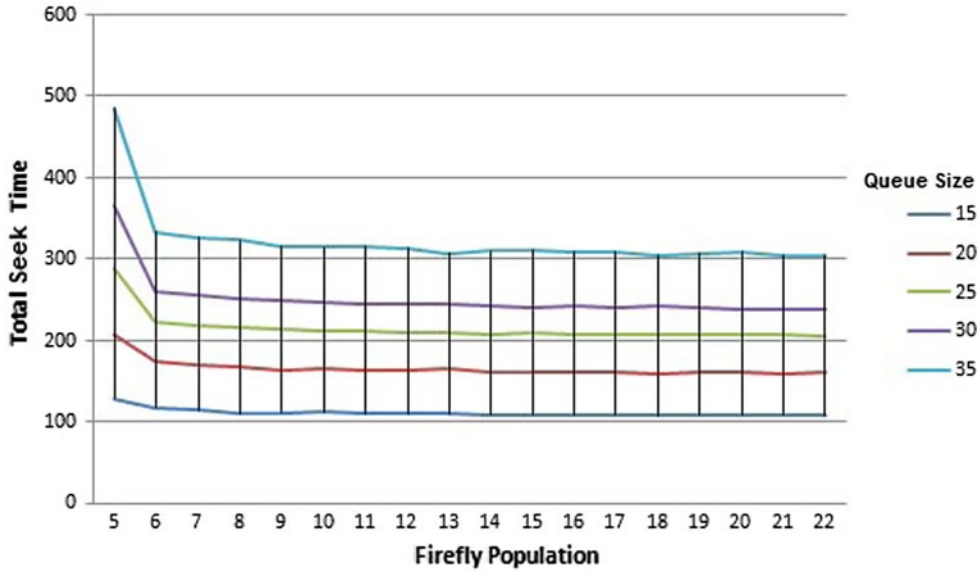
Figure 5. The variation of total seek time (fitness) with the proceeding generations.

leveling off. Thereby, suggesting little or no improvement beyond 100 generations. We observe that after 100 generations, execution of the algorithm results in sufficiently stagnant fitness value and stabilizes.

4.4. Variation of population size

The Figure 6 shows the total seek time plotted by varying the population size, i.e. the number of fireflies, carried out for various values of queue size of a firefly. On observing, we see that the initial fall in the seek time becomes steeper at higher queue size i.e. the absolute of slope increases with increasing queue size. It is evident from Figure 6 that for a given population size, an increase in the fitness value may not be deserving the added computation cost beyond a certain point. The results affirm the belief that decreasing population size causes premature convergence that slows the optimization speed down.

Figure 6. Total seek time vs. firefly population for different lengths.



5. Conclusion and future work

This study recommends an approach based on the customized firefly optimization algorithm to address the disk scheduling problem. The total seek time is the measurement of the performance of algorithms here, with the criterion of a single parameter track number. Rotational latency is neglected because of its lower value in comparison to the seek time. The comparison of the performances of the proposed and traditional approaches verifies the superiority of the proposed approach.

Proposed customized firefly algorithm results in an efficient disk scheduling problem with lesser number of disk requests. However, with the increasing number of disk requests, the performance of FA is not optimal enough. Future work of this study may include the comparison of these algorithms for parameters like respective throughput and missed ratios amongst others.

Acknowledgements
Corresponding author would like to take this opportunity to acknowledge Prof C.S. Rai, Professor, USICT and Mr. Amrit Pal Singh. Assistant Professor, GTBIT, GGSIPU, New-Delhi to provide detailed insight of the functioning of firefly algorithm.

Funding
The authors received no direct funding for this research.

Author details
Amandeep Singh[1]
E-mail: ads71993@gmail.com
Sidharth Thapar[1]
E-mail: sidharthapar@gmail.com
ORCID ID: http://orcid.org/0000-0002-3948-3156
Abhishek Bhatia[1]
E-mail: abhigenie92@gmail.com
Saurabh Singh[1]
E-mail: saurabhsingh.911@gmail.com
Rinkaj Goyal[1]
E-mail: rinkajgoyal@gmail.com
ORCID ID: http://orcid.org/0000-0002-4380-4012
[1] USICT, GGS Indraprastha University, Sector 16C, Dwarka 110078 New Delhi, India
[†] All authors contributed equally in this paper.

Citation information
Cite this article as: Disk scheduling using a customized discrete firefly algorithm, Amandeep Singh, Sidharth Thapar, Abhishek Bhatia, Saurabh Singh & Rinkaj Goyal, *Cogent Engineering* (2015), 2: 1011929.

References

Abbott, R. K., & Garcia-Molina, H. (1990). Scheduling I/O requests with deadlines: A performance evaluation. In *Proceedings, 11th real-time systems symposium, 1990* (pp. 113–124). Lake Buena Vista, FL: IEEE.

Abshouri, A. A., Meybodi, M. R., & Bakhtiary, A. (2011). New firefly algorithm based on multi swarm & learning automata in dynamic environments. *IEEE Proceedings, 13*, 989–993.

Andrews, M., Bender, M. A., & Zhang, L. (1996). New algorithms for the disk scheduling problem. In *Proceedings of 37th conference on foundations of computer science*. Burlington, VT: IEEE Computer Society Press. doi:10.1109/sfcs.1996.548514

Arora, S., & Singh, S. (2013). The firefly optimization algorithm: Convergence analysis and parameter selection. *International Journal of Computer Applications, 69*, 48–52.

Banati, H., & Bajaj, M. (2011). Firefly based feature selection approach. *International Journal of Computer Science Issues, 8*, 473–480.

Basu, B., & Mahanti, G. K. (2011). Firefly and artificial bees colony algorithm for synthesis of scanned and broadside

linear array antenna. *Progress in Electromagnetics Research B, 2*, 169–190.
Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization. *Canadian Society for Unconventional Resources, 35*, 268–308. doi:10.1145/937503.937505
Chen, T.-S., Yang, W.-P., & Lee, R. C. T. (1992). Amortized analysis of some disk scheduling algorithms: SSTF, SCAN, and N-Step SCAN. *BIT Numerical Mathematics, 32*, 546–558.
Falcon, R., Almeida, M., & Nayak, A. (2011). Fault identification with binary adaptive fireflies in parallel and distributed systems. In *2011 IEEE Congress of Evolutionary Computation (CEC)*. New Orleans, LA: IEEE. doi:10.1109/cec.2011.5949774
FUJITSU (1984). M236l a mini-disk drive customer engineering manual. Retrieved from http://chiclassiccomp.org/docs/content/computing/Fujitsu/B03P-4825-0002A02A_M2361A_CE_Manual_Jan87.pdf
Gandomi, A. H., Yang, X.-S., & Alavi, A. H. (2011). Mixed variable structural optimization using firefly Algorithm. *Computers & Structures, 89*, 2325–2336. doi:10.1016/j.compstruc.2011.08.002
Gazi, V. & Passino, K. M. (2004). Stability analysis of social foraging swarms. *IEEE Transactions on Systems Man and Cybernetics, Part B (Cybernetics), 34*, 539–557. doi:10.1109/tsmcb.2003.817077
Huang, P. C., Lu, W. C., Chou, C. N., & Shib, W. K. (2005). The NP hardness and the algorithm for real time disk scheduling in a multimedia system. In *Proceedings, 11th IEEE International conference on embedded and real-time computing systems and applications, 2005* (260–265). Hong Kong: IEEE.
Huang, P. C., Lu, W. C., Chou, C. N., & Shih, W. K. (2005). The NP-hardness and the algorithm for real-time disk-scheduling in a multimedia system. In *11th IEEE International conference on embedded and real-time computing systems and applications (RTCSA'05)*. Hong Kong: IEEE. doi:10.1109/rtcsa.2005.98
Jacobson, D. M., & Wilkes, J. (1991). *Disk scheduling algorithms based on rotational position*. Palo Alto, CA: Citeseer.
Jati, G. K., & Suyanto, S. (2011). Evolutionary discrete firefly algorithm for travelling salesman problem. In A. Bouchachia (Ed.), *Adaptive and Intelligent Systems* (pp. 393–403). Klagenfurt: Springer. doi:10.1007/978-3-642-23857-4_38.
Ökdem, S., & Karaboga, D. (2006). Optimal disk scheduling based on ant colony optimization algorithm. *Erciyes University Journal Institute of Science and Technology, 22*, 11–19.
Palit, S., Sinha, S. N., Molla, M. A., Khanra, A., & Kule, M. (2011). A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm. In *2011 2nd International conference on computer and communication technology (ICCCT-2011)*. Allahabad: IEEE. doi:10.1109/iccct.2011.6075143
Rahmani, H., Arshad, S., & Moghaddam, M. E. (2009). *A disk scheduling algorithm based on ant colony optimization*. In *ISCA PDCCS* (pp. 37–42). Louisville, KY.
Ruemmler, C., & Wilkes, J. (1994). An introduction to disk drive modeling. *Computer, 27*, 17–28. doi:10.1109/2.268881
Sayadi, M. K., Ramezanian, R., & Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations, 1*(1), 1–10. doi:10.5267/j.ijiec.2010.01.001
Seltzer, M., Chen, P., & Ousterhout, J. (1990). Disk scheduling revisited. In *Proceedings of the winter 1990 USENIX technical conference* (pp. 313–323). Washington, DC.
Senthilnath, J., Omkar, S. N., & Mani, V. (2011). Clustering using firefly algorithm: Performance study. *Swarm and Evolutionary Computation, 1*, 164–171. doi:10.1016/j.swevo.2011.06.003
Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating system concepts* (Vol. 8). New York, NY: Wiley.
Teorey, T. J., & Pinkerton, T. B. (1972). A comparative analysis of disk scheduling policies. *Communications of the ACM, 15*, 177–184.
Yagiura, M., & Ibaraki, T. (2001). On metaheuristic algorithms for combinatorial optimization problems. *System and Computers in Japan, 32*, 33–55. doi:10.1002/1520-684X(200103)32:3<33::AID-SCJ4>3.0.CO;2-P
Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In O. Watanabe & T. Zeugmann (Eds.), *Stochastic algorithms: Foundations and applications* (pp. 169–178). Sapporo: Springer.
Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-inspired Computation, 2*, 78–84.
Yang, X.-S., Cui, Z., Xiao, R., Gandomi, A. H., & Karamanoglu, M. (2013). *Swarm intelligence and bio-inspired computation: Theory and applications*. Amsterdam: Newnes.
Yeh, T.-H., Kuo, C.-M., Lei, C.-L., & Yen, H.-C. (1998). Competitive analysis of on-line disk scheduling. *Theory of Computing Systems, 31*, 491–506.