Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.6, No.2, 2015

Optimizing Software Clustering using Hybrid Bee Colony Approach

Kawal Jeet (Corresponding author)
Assistant Professor, Department of Computer science, D.A.V. College Jalandhar, India
E-mail: kawaljeet80@yahoo.com

Abstract
Maintenance of software is the most expensive and complicated phase of the software development lifecycle. It becomes more cumbersome if the architecture of the software system is not available. Search-based optimization is found to be a technique very efficient in recovering the architecture of such a system. In this paper, we propose a technique which is based on a combination of artificial honey bee swarm intelligent algorithm and genetic algorithm to recover this architecture. In this way, it will be very helpful to software maintainers for efficient and effective software maintenance. In order to evaluate the success of this approach, it has been applied to a few real-world module clustering problems. The results we obtained support our claim that this approach produces architecture significantly better than the existing approaches.
Keywords: Artificial bee colony algorithm, Genetic algorithm, Software clustering, Software Modularization.

1. Introduction

The maintenance and evolution of a software system is a most cumbersome, costly and time-consuming task (Schneidewind 1987). This problem is further enhanced if the system is poorly documented or not documented at all (Perry and Wolf 1992; Shaw and Garlan 1996). Sometimes a documented architecture becomes outdated due to regular changes that are made to the system as a consequence of changing customer requirements (Eick et al. 2001). Apparently, the software maintainers need software architecture for efficient and effective maintenance of the software. So, there must be a way to identify this architecture from the source code of the software system if it is not available.

A software system is composed of modules which could be a class, or variables which are related to each other due to procedure calls, inheritance relationships, variable references, etc. The syntactic structure of these systems can be represented as a graph called a Module Dependency Graph (MDG) where the nodes are the modules and edges are the relations between the modules. These MDGs could be retrieved by parsing the source code to determine the modules of the software system and relationship between these modules. Large numbers of source code analysis tools (http://depfind.sourceforge.net, http://source.valtech.com/display/dpm/Dependometer, https://drewnoakes.com/code/dependency-analyser/) are available that could be used to retrieve these MDGs.

In order to identify the architecture of the system, the researchers in the reverse engineering community have been developing clustering tools. Creating appropriate cluster partition of an MDG is NP hard because the number of possible partitions is very large even for a small graph (Mancoridis, 1998). So, automated assistance to partition MDGs is required that would help system maintainers to efficiently work in the absence of original design documentation (Harman, 2007). According to Tzerpos and Holt (Tzerpos and Holt 1998), it is beneficial for the software maintainers to use the clustering techniques that are available rather than re-engineer the software from scratch.

In this paper, we use a technique which is a combination of Artificial Bee Colony (ABC) (Karaboga, 2007 ;Karaboga, 2012; Karaboga, 2011;Yan, 2012) and Genetic Algorithm (GA) (Goldberg, 2006) and is called Genetic Bee Colony algorithm (GBC). It automatically finds a good partition of a system's MDG. This approach treats software partition as a search-based optimization problem in which the aim is to find the best possible partition.

To the best of the authors' knowledge, this is the first time that bee colony algorithm has been applied for software clustering.

2.Related Work

Wiggerts (Wiggerts 1997) introduced clustering techniques quite well that have been successfully applied to system modularization. Similar to the technique followed in this paper, various other clustering techniques like Rigi (Müller et al. 1993) and Arch (Schwanke 1991) work in a bottom-up fashion and produce the architecture of the software system by using its source code only. The main shortcoming of these tools is that they need key involvement of the user.

Various other search-based optimization techniques have been successfully used for partitioning of MDGs. One such remarkable one in this field is the BUNCH tool (Mancoridis, 1999). This tool is based on the optimization of an objective function Modularization Quality (MQ) (Mitchell, 2002). The major goal of MQ is to find a balance between cohesion and coupling. So, the larger the MQ, the better is the partition of the MDG and

the closer to the desired system architecture it is. In other words, it measures the quality of the modularization of the system concerned.

MQ=$\sum MF_k$

Where k is the number of clusters,

MF_k is the modularization factor of cluster k.

$$MF_k = intra\ edges/(intra\ edges + half\ of\ inter\ edges)$$

Here intra edges depict cohesion and inter edges depict coupling.

This tool is based on heuristic approaches such as hill climbing, GA and simulated annealing which are used for software re-modularization (Doval, 1999; Mancoridis, 1999; Mancoridis, 1998; Mitchell, 2002; Mitchell, 2002). Hill climbing algorithm is a local optimization technique and so it may get struck at local minima. Evolutionary algorithms such as GA do not suffer as much from this effect. But by performing repeated experiments, it is observed that the hill climbing algorithm performs better in terms of quality and execution time than other algorithms in software clustering. Due to this reason, the proposed algorithm is compared to hill climbing algorithm.

Authors of this paper are using a concatenation of two evolutionary algorithms: Artificial Bee Colony algorithm (ABC) (Yan, 2012) and Genetic Algorithm (GA) (Goldberg, 2006), which are compensating for the shortcomings of each other and hence results in efficient modularization. ABC has been applied to a wide variety of applications (Huang, 2013;Karaboga, 2012;Öztürk, 2012)ext

3. Genetic Bee Colony Optimization Approach

GBC is a combination of ABC and GA. ABC is an algorithm that simulates the intelligent foraging behavior of honey bee swarms. According to ABC algorithm, the colony of artificial bees contains three groups of bees: employed bees, onlookers and scouts (Karaboga and Basturk 2007).

Employee bee: A bee that is going to the food source visited by it before it is an employed bee.

Onlooker bee: A bee waiting in the dance area for making a decision to choose a food source.

Scout bee: This bee goes on a random search to discover new sources.

So, ABC has three phases: employee phase, onlooker phase and scout phase.

The positions of food sources represent a possible modularization of the software system, and the nectar amount of a food source corresponds to the quality or fitness of this associated solution. This quality could be evaluated by using an objective function MQ. It is observed that this function works very well in evaluating the quality of software clustering.

It is a very simple, robust and population-based optimization algorithm. In another approach, the authors have used selection and crossover operators of genetic algorithms before the scout phase for adding the advantage of a global search to this algorithm (Yan, 2012). We further enhanced this algorithm by adding a mutation operator and observed an increase in quality in terms of better software system re-modularization. It is assured by the increased value of MQ.

The main steps of the algorithm are given below:

Step 1: Initialization

Set the control parameter of ABC to values shown in Table 1.

Make the first half of the colony includes the employee bees and the second half include the onlooker bees.

Randomly generate a modularization as a candidate solution using equation (1).

$$x_j^i = x_{min}^j + rand(0,1)(x_{max}^j - x_{min}^j) \qquad (1)$$

Where i=1,2,,,,,N_s (Number of food sources as shown in Table 1).

j=1,2,,,,,,N (Number of parameters or modules to be clustered as shown in Table 1).

$x_{min}^j = Lower\ bound\ for\ jth\ parameter\ (LB) = 1$ (as shown in Table 1)

$x_{max}^j = Upper\ bound\ for\ jth\ parameter\ (UB) = Number\ of\ nodes\ in\ MDG$
(as shown in Table 1)

x_j^i indicates jth parameter of i^{th} food source which means cluster to which j^{th} node (module) has been allocated in i^{th} modularization or candidate solution.

Evaluate fitness of each candidate in the population using objective function MQ.

Set the current scout number s = 0.

Set number of trials for each bee = 0.

Set cycle to 1

Step 2: Employee bee

For each employee bee produce a new neighboring solution by using equation (2).

$$v_i^j = x_i^j + \emptyset_i^j(x_i^j - x_k^j) \qquad (2)$$

Where $\emptyset$ is a random number between [-1,1].

i and k=1,2,,,,N_s (Number of food sources as shown in Table 1).

k is determined randomly and should be different from i.
Calculate the fitness of each newly created solution by using MQ and apply greedy selection process to keep the employed bees with greater value of MQ (greater fitness) between newly created and already available employee bee.
If the new solution is better than the existing solution, replace existing solution with new and reset trial counter=1, else if the current solution can't be improved further increase its trial counter.
Calculate probability of a modularization solution to be selected as shown in equation (3).

$$p_{i=\frac{Fitness_i}{\sum_{j=1}^{N_s} Fitness_j}} \quad (3)$$

Where $Fitness_i$ is the fitness of solution i in the population (calculated by objective function MQ). So, P_i is the ratio of fitness of i^{th} solution and sum of fitness of all employee bee solutions.

Step 3: Onlookers phase

Similarly, each onlooker bee of the second half of the population produces new solutions from the current food sources. Apply greedy selection process to keep the onlooker bees with greater value of MQ (greater fitness) between newly created and already available onlooker bees.
In order to improve ABC, add genetic phase at this stage.

Step 4: Genetic phase

A genetic algorithm is applied for one generation. The current position of food sources acts as the population of GA. Value for other parameters found appropriate for software clustering is given in Table 2.
It is observed that the convergence speed of the ABC algorithm will decrease as the dimension of the problem increases. To overcome this, a genetic phase could be added which improves the optimization ability by involving crossover and mutation operators of genetic algorithms. By applying genetic algorithm, the information exchange of the algorithm is enhanced. The modularizations with higher fitness are fully utilized too.

Step 5: Scout phase

If a modularization can't be improved further during a pre-specified number of cycles called limit (indicated in Table 1), then that modularization (food source) is replaced by a new partition (food source) created by using equation 1.
This newly created solution is compared to existing solutions and best solution achieved so far is memorized.
Next iteration is started (cycle = cycle + 1) until stopping criteria is met which is 1000 for software modularization (as shown in Table 1). The main steps of this approach are summarized in the flowchart shown in Figure 1.

4. Software Clustering Using Genetic Bee Colony Optimization Approach

Experiments have been conducted on hill climbing, GA, ABC and GBC. In order to compare our work fairly with other remarkable works, we are using MDGs which are used in some other research (Doval, 1999;Mancoridis, 1999;Mancoridis, 1998;Mitchell, 2002;Mitchell, 2002). The details of these MDGs are presented in Table 3. Each algorithm is executed on each MDG independently and is repeated 30 times.

The detail of mean and standard deviation of 30 independent runs for GBC and hill climbing algorithm are shown in Table 4. It also shows the result of paired two tailed student's t-test between these two approaches at 58 degree of freedom. The values in bold show significant increase in quality due to this new approach.

Similarly, Tables 5 and 6 compare the result of 30 independent runs of GBC to that of GA and ABC respectively. It is observed that we have obtained a remarkably greater optimization or quality of clustering by using GBC.

In order to compare the performance of these algorithms, the number of times the objective function is evaluated (Function Evaluation) is calculated. If we compare the number of times objective function MQ has been evaluated, we observe that the proposed algorithm takes much more effort especially when compared to hill climbing (shown in Table 7). So, we obtained superior results, but at the cost of additional efforts in evaluating objective function. It could be justified, as re-modularization is an occasional task which is done only when software engineers observe the requirement and so to get better results, they can wait.

5. Conclusion

Although efforts have been made to use various techniques for software modularization, to the best of the authors' knowledge, this is the first time a honey bee-based swarm intelligent system has been applied to this field. In this paper, we presented an algorithm based on a combination of two evolutionary algorithms: artificial bee colony optimization and genetic algorithm. It has been tested on various software systems and compared to other remarkable works in the field. It has been observed that the combination of these evolutionary algorithms resulted in better solutions with an additional drawback of a much higher number of function evaluations.

Encoding of population has an impact on the quality of any evolutionary algorithm, so this work could be further enhanced by modifying the encoding of the population. It is observed that in certain cases, in order to increase MQ, modularization is a result where a single module is kept in a separate cluster. Some technique could be added to avoid such modularization.

References

Chen, Y.-F., Gansner, E. R., & Koutsofios, E. (1998). A C++ data model supporting reachability analysis and dead code detection. *Software Engineering, IEEE Transactions on, 24*(9), 682-694.

Doval, D., Mancoridis, S., & Mitchell, B. S. (1999). *Automatic clustering of software systems using a genetic algorithm.* Paper presented at the Software Technology and Engineering Practice, 1999. STEP'99. Proceedings.

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., & Mockus, A. (2001). Does code decay? assessing the evidence from change management data. *Software Engineering, IEEE Transactions on, 27*(1), 1-12.

Goldberg, D. E. (2006). *Genetic algorithms*: Pearson Education India.

Harman, M. (2007). *The current state and future of search based software engineering.* Paper presented at the 2007 Future of Software Engineering.

http://depfind.sourceforge.net/, accessed December 2013.

http://source.valtech.com/display/dpm/Dependometer, accessed December 2013.

https://drewnoakes.com/code/dependency-analyser/, accessed February 2013.

Huang, S.-J., & Liu, X.-Z. (2013). Application of artificial bee colony-based optimization for fault section estimation in power systems. *International Journal of Electrical Power & Energy Systems, 44*(1), 210-218.

Kang, F., Li, J., & Ma, Z. (2013). An artificial bee colony algorithm for locating the critical slip surface in slope stability analysis. *Engineering Optimization, 45*(2), 207-223.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization, 39*(3), 459-471.

Karaboga, D., Okdem, S., & Ozturk, C. (2012). Cluster based wireless sensor network routing using artificial bee colony algorithm. *Wireless Networks, 18*(7), 847-860.

Karaboga, D., & Ozturk, C. (2011). A novel clustering approach: Artificial Bee Colony (ABC) algorithm. *Applied Soft Computing, 11*(1), 652-657.

Mancoridis, S., Mitchell, B. S., Chen, Y., & Gansner, E. R. (1999). *Bunch: A clustering tool for the recovery and maintenance of software system structures.* Paper presented at the Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on.

Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y., & Gansner, E. R. (1998). *Using automatic clustering to produce high-level system organizations of source code.* Paper presented at the International Conference on Program Comprehension.

Mitchell, B. S. (2002). *A heuristic search approach to solving the software clustering problem.* Drexel University.

Mitchell, B. S., & Mancoridis, S. (2002). *Using Heuristic Search Techniques To Extract Design Abstractions From Source Code.* Paper presented at the GECCO.

Müller, H. A., Orgun, M. A., Tilley, S. R., & Uhl, J. S. (1993). A reverse ‐ engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice, 5*(4), 181-204.

Öztürk, C., Karaboğa, D., & GÖRKEMLİ, B. (2012). Artificial bee colony algorithm for dynamic deployment of wireless sensor networks. *Turkish Journal of Electrical Engineering & Computer Sciences, 20*(2), 255-262.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes, 17*(4), 40-52.

Schneidewind, N. F. (1987). The state of software maintenance. *IEEE Trans. Software Eng., 13*(3), 303-310.

Schwanke, R. W. (1991). *An intelligent tool for re-engineering software modularity.* Paper presented at the Software Engineering, 1991. Proceedings., 13th International Conference on.

Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline* (Vol. 1): Prentice Hall Englewood Cliffs.

Tzerpos, V., & Holt, R. C. (1998). *Software botryology. automatic clustering of software systems.* Paper presented at the Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on.

Wiggerts, T. A. (1997). *Using clustering algorithms in legacy systems remodularization.* Paper presented at the Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on.

Yan, X., Zhu, Y., Zou, W., & Wang, L. (2012). A new approach for data clustering using hybrid artificial bee colony algorithm. *Neurocomputing, 97*, 241-250.

Biographical Notes. Kawal Jeet is an Assistant Professor in Post-Graduate Department of Computer Science, D.A.V. College, Jalandhar, India. She received her Master's of Technology in Computer Science from Dr. B.R. Ambedkar National Institute of Technology, Jalandhar, India in 2012. Currently she is pursuing Ph.D from this institute. Her current research interest focuses on nature inspired computation, software modularization, Bayesian networks, software quality. She has published her research work in more than 15 international journals and conference proceedings. She is member of the IRED, UACEE and ACM India.

Table 1.: Control parameters for ABC

Parameter	Value	Comment
Size of colony	10*Number of parameters to be optimized	Employed bees and Onlooker bees together
Number of food sources (Ns)	Size of colony/2	Half of the colony size
Trials limit	100	Abandoned a food source which could not be improved trial limit
Cycles for foraging	1000	a stopping criteria
Objective function	Modularization Quality (MQ)[23]	It is a cost function to be optimized. The goal of MQ is to limit excessive coupling but not to eliminate coupling altogether. The best thing is to find a balance between coupling and cohesion by combining them into a single measurement.
Number of variables to be optimized (N)	Number of modules to be clustered.	Number of nodes in MDG.
Lower bound of parameters	1	No Coupling. It means a single cluster with all the nodes in it.
Upper bound of parameters	Number of nodes in MDG.	No Cohesion. It means every node is in a separate cluster.

Table 2.: Control parameters for GA

Parameter	Value	Comment
Selection Algorithm	Tournament	Candidate modularisations Parent1 and Parent2 are selected for crossover using this method
Crossover function	Arithmetic	Child=R1 XParent1+ R2 XParent2 Where R1,R2 are random numbers between 0 and 1 and are independent
Mutation function	Uniform	Rate 0.02
Population size	Number of food sources	Ns of table 1
Population	Food sources of ABC	Modified by employee phase and onlooker phase of ABC
Crossover fraction	0.6	
Generations	1	
Objective function	MQ	

Table 3: Description of the software systems to be used in case study

Software System	Modules in MDG	Edges in MDG	System Description
Compiler	13	12	A Small compiler
Nos	16	52	C++ program that implements file system service
Mini-Tunis	20	57	A simple operating system
Ispell	24	103	An open source spell checker
Rcs	29	163	Open source version control tool.
Star	36	89	Source code analysis tool to produce the MDG
Bison	37	179	Parser Generator
Grappa	74	112	Graph Visualization and Drawing Tool
Incl	174	360	Subsystem from a Source Code Analysis system

Table 4: Comparison of proposed GBC and Hill Climbing

Software System	GBC		Hill Climbing		Student's t-test at 58 degree of freedom
	Mean	Standard Deviation	Mean	Standard Deviation	
Compiler	1.6667	0	1.44953	0.049396	24.0809
Nos	1.66571	0.026246	1.63038	0.051536	0.7493
M-Tunis	2.72356	0.066064	2.255394	0.054769	20.6235
Ispell	2.34712	0.04091	2.340724	0.0255	0.7268
Rcs	2.24701	0.028897	2.21347	0.020	5.4291
Star	3.79903	0.0012	3.777057	0.057313	3.2820
Bison	2.68599	0.046547	2.639	0.041	1.2315
Grappa	17.7858	0.3231	12.676	0.017	86.4371
Incl	11.2669	0.276649	13.568	0.035	-44.8879
Boxer	3.1011	0	3.101	0	0
Modulizer	2.71699	0.062747	2.685987	0.064958	1.8803

Table 5: Comparison of proposed GBC and GA

Software System	GBC		GA		Student's t-test at 58 degree of freedom
	Mean	Standard Deviation	Mean	Standard Deviation	
Compiler	1.6667	0	1.6667	0	0
Nos	1.66571	0.026246	1.60072	0.054506	3.4018
M-Tunis	2.72356	0.066064	2.57605	0.119806	6.3108
Ispell	2.34712	0.04091	2.18869	0.095947	8.3193
Rcs	2.24701	0.028897	2.06682	0.103749	9.1641
Star	3.79903	0.0012	3.29476	0.21403	12.9047
Bison	2.68599	0.046547	2.33062	0.10614	16.7942
Grappa	17.7858	0.3231	12.0577	0.787977	36.8390
Incl	11.2669	0.276649	-5.96656	0.367613	63.1008
Boxer	3.1011	0	2.97925	0.114837	5.8117
Modulizer	2.71699	0.062747	2.34317	0.144475	12.9989

Table 6: Comparison of proposed GBC and ABC

Software System	GBC		ABC		Student's t-test at 58 degree of freedom
	Mean	Standard Deviation	Mean	Standard Deviation	
Compiler	1.6667	0	1.6667	0	0
Nos	1.66571	0.026246	1.63829	0.014005	5.0487
M-Tunis	2.72356	0.066064	2.731	0.05157	1.0982
Ispell	2.34712	0.04091	2.28618	0.026491	6.8486
Rcs	2.24701	0.028897	2.16131	0.037775	9.8701
Star	3.79903	0.0012	3.45343	0.06691	27.7645
Bison	2.68599	0.046547	2.39958	0.072407	18.2240
Grappa	17.7858	0.3231	9.71646	0.759839	53.5285
Incl	11.2669	0.276649	3.53938	0.459941	84.4936
Boxer	3.1011	0	3.08335	0.022512	4.3191
Modulizer	2.71699	0.062747	2.4396	0.077777	15.2034

Table 7: Comparison of number of function evaluations in proposed GBC, ABC, GA and Hill Climbing algorithms

Software System	Hill	GBC	ABC	GA
Compiler	446.7333	200887s	100557	5200
Nos	860	200792.3	100280	5536.667
M-Tunis	1004.267	200791.8	100268.4	5566.667
Ispell	1887.7	200695.8	100196	6243.333
Rcs	3495	200629	100162.7	6693.333
Star	4917	200563.5	360543.9	7996.667
Bison	5957	200546.8	100139.6	7996.667
Grappa	79586.6	200311.8	100163.9	10100
Incl	155020.2	200134.8	100197.7	10090
Boxer	1144.167	360849.8	100269.1	5723.333
Modulizer	2227.067	200692.7	100205.7	6240

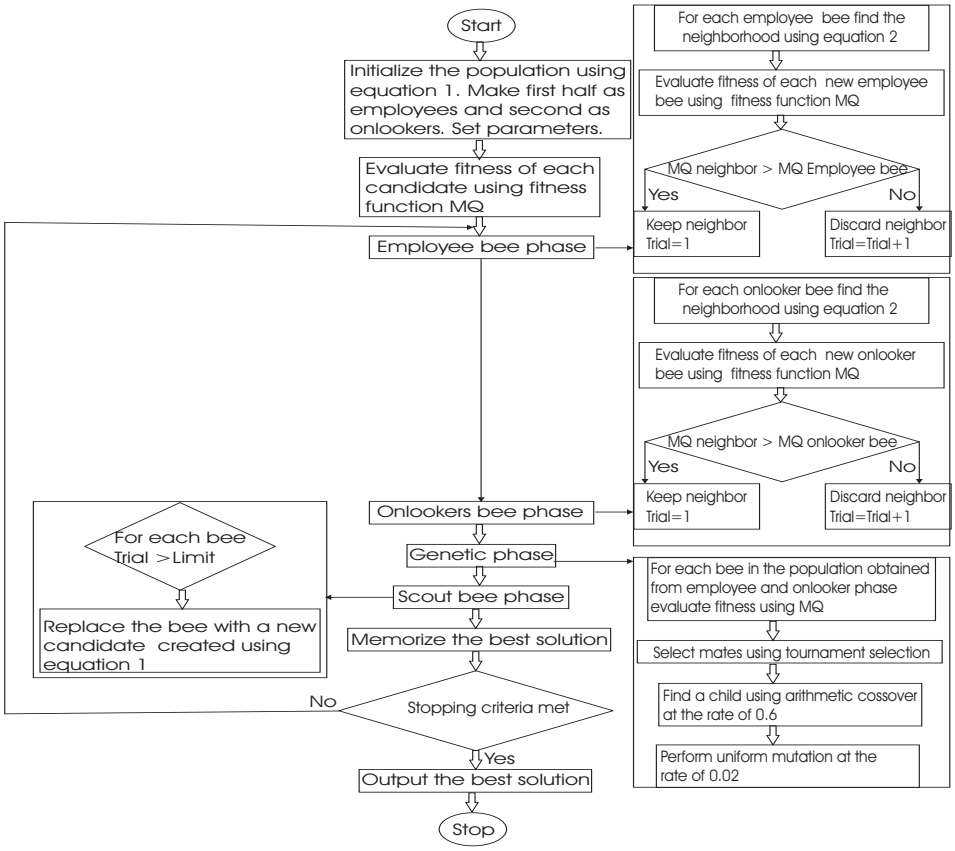

Figure 1. Flowchart of Genetic Bee Colony Algorithm

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar