International Journal of Research in Circuits, Devices and Systems

E-ISSN: 2708-454X
P-ISSN: 2708-4531
IJRCDS 2022; 3(1): 38-43
© 2022 IJRCDS
www.circuitsjournal.com
Received: 12-11-2021
Accepted: 20-12-2021

Deepthi M
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Correspondence
Deepthi M
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Analysis and research on computational complexity

Deepthi M

Abstract
Computational complexity is basically the amount time and space (resources) particular algorithm consume while executing. Measurement of this, would help us to predict how fast an algorithm can run, thus helping us to know how fast problem can be solved. The motive of this research paper is to present an outline of Computational complexity. We have mainly focused on defining intrinsic complexity of problem and defining their upper and lower bounds. Problems in computational complexity is based on inherent complexity. This difficulty is dependent on processing time of problem which depends on the steps involved to solve it. So, we begin with processing time, hardware complexity, delay and speed up [1, 2, 3]. We have also discussed Computational classes. They are said to be main part of Complexity theory. It is basically set of problems which uses same amount of resources [4, 5, 6]. As there are many problems in computational complexity and everything can't be covered, definability and named problems will be our focal point. We also go through probabilistic and parallel computation. P and NP problems is heart of computational complexity. P problems are the ones that can be solved pretty quickly [7, 8, 9], while NP problems aren't. So, we finally conclude by considering many problems found in system & classifying them.

Keywords: Computational complexity, Computational classes, intrinsic complexity, inherent complexity, processing time, Parallel computation, Probabilistic computation, P problems, NP problems

Introduction
Green computing is the study in which discarding, recycling and building of computers and electronic items is noticed. Green computing is the environment friendly use of computers and their resources. It is also defined as the study of designing, engineering, manufacturing, using and disposing of computing devices in such a manner that reduces their impact on the environment.
The goals of green computing is same as that of green chemistry, i.e. to minimize the use of harmful materials, maximize energy efficiency during the life span of a product, and to promote the concept of reusability or biodegradability of obsolete products and factory waste. Green computing is important for all classes of systems, ranging from handheld systems to large-scale data centres.
Many IT manufacturers and vendors are investing continuously in manufacturing energy-efficient computing devices which help in reducing the use of dangerous materials and encouraging the recyclability of digital devices. Green computing practices came into existence in 1992, when the Environmental Protection Agency (EPA) launched the Energy Star program. Green computing is also known as green information technology (green IT). Dell, Intel, Google, Microsoft etc. are the companies which use Green Computing Technology.
Green computing is the utilization of computer related resources in an environment friendly manner. This includes the implementation of energy-efficient central processing units, servers and peripherals along with proper disposal of electronic waste. It is also known as the green technology. It ensures that the use and the setup of the IT sectors in the world should produce the minimum amount of carbon foot print.

Regulations and industry initiatives
- Climate savers computing initiative (CSCI) is an effort to reduce the electric power consumption of PCs in active and inactive states. The CSCI provides a record of green products from its member organizations, and information for reducing PC power consumption. It was started on 2007. The WWF is also a member of the Climate savers computing Initiative.

There are 2 types of Complexity theory. They are classified based on what they analyse. Algorithm complexity delas with analysing the algorithm of problem. And the computational complexity deals with complexity of problem.

Asymptotic analysis

It defines the boundary of run-time performance of an algorithm. It is of great in concluding best, average and worst case. It mainly depends on input. If the input is null, then we take it as constant. Others are all not important. Its basically finding run-time of algorithm. If the run time increases linearly with n then its represented as f(n).If it increases exponentially then f(n^2). If n is too small, then there isn't much difference in running time operation. Algorithm may require minimum for execution then its called Best case. It takes lot of time, then its called worst case. Then there is this intermediate case, where time for execution is between maximum and minimum time. This is called Average case.

Problem instances

Let's start with definition of instance. Instance is normally the inputs required to solve a problem. And computational problem is collection of instance and their solution. The problem instance is the word used to refer input string in Computational problem. Abstract is problem for which we need to find solution and instance is solution for that abstract. The problem instance is alphabet which is normally a binary number. So we can call them as bitstrings.

Decision problems are the ones in which have just 2 outputs yes or no, for all possible inputs. An example of this would be "Is x a prime number?". The answer could be either yes or no, even though input could be any real numbers. And the solution for this is computed by checking every possible factor except 1.So there exists an algorithm through which this problem can be solved, In that cases, problems are called decidable. Problems for which we can't create algorithm, i.e, we are not able to decide are called undecidable problems. Example of this would be "The problem of determining if a given set of Wang tiles can tile the plane".

Decision problems are again classified in the basis of how difficult they are to solve. Difficult problems are those which requires more computational resources (time, space, etc). Undecidable problems are classified on basis of Turing degree, which says how non-computable they are.

A. Function problems

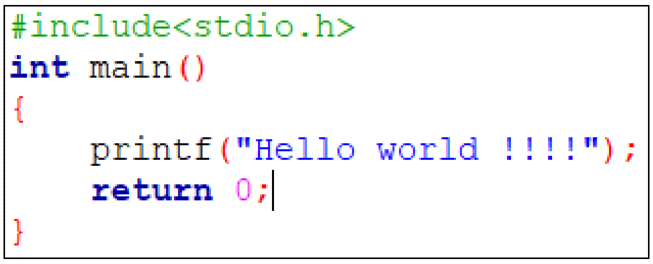
A function problems are the problems which gives single output for every input, but output is not just "yes" or "no". Examples include the traveling salesman problem and the integer factorization problem. This doesn't mean function problems are complicated then decision problems. Sometimes functions can be expressed as decision problems. For example, the addition of 2 integers.

It may feel like function problems are more complicated then decision problems, but that's not always the case. Many of the function problems can be expressed as decision problems.

We can say the difficulty of algorithm by seeing how much time the best algorithm takes to run, and his run time depends on instance/inputs. And the extent of dependency between run time and input size is determined by the problem, algorithm used and computational model we have been working with. We use big-O notation to explain time complexity. Algorithms can have constant run time, linear, Exponential or logarithmic run time.

A. Constant runtime

Constant run time is when regardless of size of input the time taken for computation by that algorithm remains same. Example is given below

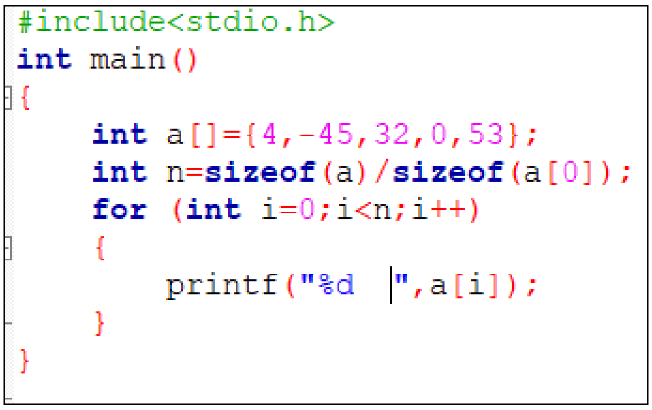


```
#include<stdio.h>
int main()
{
    printf("Hello world !!!!");
    return 0;
}
```

Fig 1: Constant Runtime

B. Linear runtime

It means as the input size grows, the time taken by the output goes proportionally large.



```
#include<stdio.h>
int main()
{
    int a[]={4,-45,32,0,53};
    int n=sizeof(a)/sizeof(a[0]);
    for (int i=0;i<n;i++)
    {
        printf("%d  ",a[i]);
    }
}
```

Fig 2: Linear Runtime

C. Exponential run time

The problems that can be solved in exponential time.

D. Logarithmic run time

The time required for execution logarithmically increases

Machine models and complexity measures

A. Turing machine

Turing machine is one of model used for complexity measure in computing [10, 11]. Its invemted by Alan Turing in 1963.It accepts Recursive enumerable languages. It is a device which has tape of infinite which has infinite cells. Every vall has some content like input symbol or blank (special symbol). Read and write operations can be performed on this tape. There exists Turing machine for all the problems that could be solved by using any algorithms. That is there exists Turing macjine for all the solvable problems using algorithm. This statement is given by founder of Turing machine Turing and church. And Turing machine is so powerful that all the things that can be computed by using other computation models can definitely be computed using Turing machine.

Many types of Turing machines are used to define complexity classes, such as deterministic Turing machines, probabilistic Turing machines, non-deterministic Turing machines, quantum Turing machines, symmetric Turing machines and alternating Turing machines. They are all equally powerful in principle, but when resources (such as time or space) are bounded, some of these may be more powerful than others.

A. Deterministic Turing machine
It is most basic among all the other Turing machines. It is provided with few set of rules to determine future actions.
In this machine, it has transition function for every symbol and state, and it gives information about
- What has to be written in tape
- The direction in which head should proceed
- Next state of control

B. Non-deterministic Turing machine
In this Turing machines, the rules state more than one result for particular action.

C. Probabilistic Turing machine
In this we employee probability distribution. It is non-deterministic distribution which makes a choice between transition states at given point of time. It has 2 options at each step and it decides which move to make

D. Quantum Turing machines
Quantum computer in modelled using Quantum Turing machine. All the quantum alogorithms can be expressed as Quantum Turing machine.

E. Symmetric Turing machine
These Turing machines have graph which proves relationship between graph reachability and complexity classes. These graphs are called configuration graph.

F. Alternating Turing machine
We can generalize some rules after defining NP and Co-NP complexity classes. Alternating Turing machine use these rules for accepting computations.

Upper and lower bounds of a problem
A. Lower bound
It's the minimum time required by an algorithm. Its represented by Big Omega in asymptotic notation.

B. Upper bound
It's the maximum time required by an algorithm. Its represented by Big Oh in asymptotic notation.

Bound theory
A. Lower Bound theory
According to this, for any algorithm, if L(n) is its lower bound, then there exists no other algorithm for same where complexity is less than L(n). Every algorithm that exists for a problem, will have L(n) complexity at the very worst. If lower bound equals upper bound, then it is said to be optimal. In simple words, its time taken by best case input for a program.

B. Upper bound theory
According to this, if U(n) is an upper bound of an algorithm, then maximum time taken any algorithm to solve the problem is U(n). In simple words, its time taken by worst case input for a program.

Complexity classes
It's set of all problems in computation which can be computed using related resource. And we mostly refer to time and space or memory when we say resources. Generally, complexity classes is bound by particular time and space. It mostly deals with decision problems. As we know already decision problems can be solved by Turing machine. For example, we can have a class which has the decision problems that can be solves by non-deterministic Turing machine. We can have many such examples.
There can be cases where one class is subset of other. Most common example would be class problems solvable in deterministic time, is subset of class of problems solvable in non-deterministic polynomial time.
There are lot of relationships between classes. Ans this marks the major field of research in Computer Science.
Time complexity is normally calculated by number of steps involved in solving the problems. More the steps, more is the complexity. It also depends on how quickly we can verify answers. Its main point we consider while deciding whether the algorithm is efficient or not. Mostly the efficiency of algorithm depends on its time complexity and space complexity. So time complexity is of utmost use to anyone who is interested in programming be it the programmer, scientist or normal person who is just curious about programming.
As said earlier space complexity also plays a very important role in deciding the efficiency of program. As we know Turing machine has tape of infinite length. So space in Turing machine refers to number of cells required in the tape.

Space hierarchy
This theorem says that, the number of problems solved by deterministic and non-deterministic machines increases when space is increases. This is almost like a real life situation. The more space we have, the more things we could fill in. Same way, the more space we have the more problems we could solve. To give you an idea space n*log n is greater than space n. So be it the deterministic Turing machine or non-deterministic one, thry solve problems faster in n*logn space because as we said earlier large space, quick solving of problem.
The whole space hierarchy and time hierarchy lies on the hunch that with more and more space and time, problems can be computed efficiently. And one important thig to be mentioned is relaxed bound contains more languages then compared to tighter bounds.
Theorem 1: We can say there is language that can be decided on space O(s(n)) not space o(s(n)), if s is space constructible.

9. Time hierarchy theorem
These are set of statements that deal with time-bounded computation on both deterministic and non-deterministic Turing machine [12, 13]. It suggests that given more time, more problems can be solved. Its again like a real-world situation. Let us consider work A can be done in 10 minutes, B in 2 minutes and C in 20 minutes by person P. So if person P is given 10 minutes he can just complete work A or work B, but not both. Completing C is not at all possible.

Consider other situation. If person P is given 35 minutes, then we can complete work A, work B and work C.
Same way there are problems that can be solved in n^3 but not in n or n^2 amount of time.
Richard and Juris were the first scientists to prove time hierarchy theorem for deterministic multi-tape Turing machine in 1965.Universal Turing machine was improved by Hennie and Richard. There is larger time bound complexity class for every deterministic time bound complexity classes. This implies that there is no complete collapsing of time bounded hierarchy of complexity classes.
So far we have discussed for time hierarchy for deterministic Turing machine. Now let's talk time hierarchy for non-deterministic Turing machine [14, 15].
This theorem i.e, time hierarchy theorem for NTM i.e, non-deterministic Turing machine was poven by Steohen cook. It was in year 1972.And there are many scientists who contributed in improving it to the present form. Here I would like to mention few of them. Joel Seiferas, Albert Meyer and Michael Fischer played a major role in impoving time hierarchy thorem by their complex proof. These people i.e, Joel Seiferas, Albert Meyer and Michael Fischer worked int the year 1978.This played a major role in development of time hierarchy theorem. But the proof was too complicated and difficult. Then there comes Stanislav Zak. Zak achieved exact same result with very simple proof. We use the same proof to teach students till today. Zak did it in 1983.And needless to say its one of the greatest milstone in Theory of Computational complexity.
As mentioned before, complexity is class of problems which can be computed using related resources. So we can say, these classes are useful in organizing in similar type of problems. Like any other topic, there are lot of open problems in this topic too. One of the most important one is when we say 2 classes are equal. It results into various discrepancies. It also leads to conclusion P=NPP (will be explained further) which is not true. If this true, then all algorithm used for security purposes like banks and computers crashes down.

10. P class

It's the class of problems for which there exists a deterministic computer which solves it in quick and efficient matter. The reason for this is time complexity doesn't increase rapidly with input size. Which means input size has less influence on time complexity.
They can be decided in polynomial time. We can decide "yes" or "no" for those problems.
Cobham said "Every problem in P are efficiently solvable" in his thesis called Cobham's Thesis. This is not completely true. There are many problems that are efficiently solvable doesn't belong to P. And there are also examples of problems in P that aren't efficiently solvable.
Exponential time algorithms are not really useful in real life because it consumes lot of resources. So, we try to avoid that as much as possible. Most commonly used one is polynomial time algorithms. Its much more efficient than brute force approach. Because brute force normally takes exponential time. In simple words, for every problem in P class there exists an algorithm that solves an input of size n in $O(n^k)$.
Consider a language L. We say it belongs to P if and only if there exists DTM i.e, deterministic Turing machine N such that

- For all inputs, DTM(deterministic Turing machine) N should run in polynomial time
- M should produce output 1,for all x in Language L
- M should produce output 0,for all x not in Language L

We can also view P in other way. We can take it as family of Boolean circuits. Most importantly, its uniform family. Now let us define when we consider L in P.
Consider a language L. We say it belongs to P if and only if there exists Boolean circuits which are polynomial time uniform C_n, such that

- n which belongs to N, C_n produces output 1 and takes n bits of input.
- $C_x(x)=0$, for all x doesn't belong to L
- $C_x(x)=1$, for all x does belong to L

A. Important problems in P

P has many problems. And most importantly most of them are natural. Examples include linear programming where we have to decide something. Even maximum matching comes under this. And the most important in computation is finding whether given number is prime or not prime. This problem comes in P. This was shown in year 2002.
Let's discuss about polynomial time algorithms. Let us write a polynomial time function and have calls which are of constant time. Now as we said earlier, they are polynomial function. So, they take polynomial amount of time. So whole algorithm requires polynomial amount of time. So, we call this algorithms as closed for some properties like composition. This is the most important reason to call P as not dependent on machine i.e, machine independent class. Let us consider one feature of machine. Be it Random access. It can be simulated in polynomial time.
P is not just closed under composition property. It is closed under many other properties like complementation. Complementation is basically reversing yes to no and no to yes in decision problems. Its closed for intersection. Intersection is basically common elements between 2 sets.
Union is collection of all elements of a sets. P is closed under this too. String Concatenation is basically joining end of one string to beginning to another string. And yet again Polynomial algo is closed under this too.
Let's discuss about of proof of existence of polynomial time algorithm. There exists few problems. And we know we could solve that problems in polynomial time. But again we don't have proper algorithm to compute that Lets take an example. Robertson and Seymor are 2 scientists who contributed immensely in field of computation. They proposed Robertson-Seymour theorem. This ensures that for a hraph to be embedded in torus, there are finite minors. Now lets come to most important of this theorem. It showed determining whether given graph is minor of one graph can be solved by $O(n^3)$ algorithm.
In the last, in [20-29] readers, students, researchers can be found interesting works on emerging topics like Industry 4.0, Society 5.0, etc., and their importance for the smart era.

NP class and Conclusion

It's a complexity class like P. But here it consists of problems that can be solved in Polynomial time not with deterministic Turing machine but with non-deterministic Turing machine [16, 17, 18]. Again it deals with decision problems. It has proofs in polynomial time if the answer is yes for decision problem.

Turing machine has 2 phases. One works deterministic way and other in non-deterministic way. Guessing of solution which occurs in first phase is non-deterministic. In second phase we verify whether our guess is correct or not. That is we check whether the guess is equal to solution. This is done in deterministic way.
Its clear that all the problems in complexity class P (all problems can be computed in polynomial time using deterministic Turing machine) belongs to complexity class NP. So we can say P is subset of NP. Because all the elements of P is present in NP but vise versa in not true. NP contains much larger number of problems.
Its easy to find solution of problems in class P. But that's not the case NP problems. We again have NP-hard, NP-complete problems, NP-easy, NP-equivalent, and NP-intermediate problems.NP-hard and NP-complete are most important ones.

NP-easy
This means they are as hard as NP at maximum, but they don't always have to belong to NP

NP-equivalent
These are problems mainly decision problems which can be classified as both NP-hard and NP-easy. But important point to remember is, they don't always have to belong to NP.

NP-intermediate
These are problems which are intermediate between P and NP complete problems.

NP-hard
NP hard problems are basically those problems which are as hard as NP problems. But yet again they don't have to be in NP class. They don't necessarily have to be decision problem.
We can define NP -hard problems in this way too. Let us consider NP -complete problem. Now let's say its reducible to Y in polynomial time. Then Y is NP-hard problem.
There are lot of examples for NP hard problems:
1. Traveling salesman problem
2. Vertex cover
3. Halting problem

NP-complete
These are the hardest NP problems [19]. Consider a problem N, =. It belongs to NP-complete if
a. It's in NP
b. All the problems in NP should be reducible to N

Examples of NP-complete problems
- Scheduling
- Monochromatic triangle in Graph theory
  Grundy number

References

1. Arora Sanjeev, Computational Complexity. A modern Approcah, Cambridge, ISBN 978-0-521-42426-4,Zbl 1193.68112
2. Garey Michael R, Johnson David. Computers and Intractability's guide to the Theory of NP completeness, W. H. Freeman, 1979. ISBN 0-7167-1045-5.
3. Papdimitriou Christos. Computational Complexity (1st ed.), Addison Wesley, 1994. ISBN 0-201-53082-1
4. Hartmanis J, Stearns RE. (1 May 1965). "On the computational complexity of algorithms". Transactions of the American Mathematical Society. American Mathematical Society. 1994;117:285-306. doi:10.2307/1994208. ISSN 0002-9947. JSTOR 1994208. MR 0170805.
5. Karp Richard. "Reducibility among Combinatorial Problems" (PDF). Complexity of Computer Computations, 1972.
6. Sipser Michael. Introduction to theory of computation (2nd ed.), USA: Thomson Course Technology, 2006. ISBN 0-534-95097-3.
7. Goldreich Oded. Computational Complexity: A Conceptual Perspective, Cambridge University Press Handbook of theoretical computer science (vol. A): algorithms and complexity, 2008.
8. John Wiley, Sons. Theory of Computational Complexity.
9. Hennie FC, Stearns RE. Two-Tape Simulation of Multitape Turing Machines. J. ACM. New York, NY, USA: ACM. 1966;13(4):533-546. doi:10.1145/321356.321362. ISSN 0004-5411.
10. Žák Stanislav. A Turing machine time hierarchy". Theoretical Computer Science. Elsevier Science B.V. 1983;26(3):327-333. Doi:10.1016/0304-3975(83)90015-4.
11. Fortnow L, Santhanam R. "Hierarchy Theorems for Probabilistic Polynomial Time". 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, 316. Doi:10.1109/FOCS.2004.33. ISBN 0-7695-2228-9
12. Luca Trevisan. Notes on Hierarchy Theorems, U.C. Berkeley.
13. Cook Stephen A. "A hierarchy for nondeterministic time complexity". Proceedings of the fourth annual ACM symposium on Theory of computing. STOC '72. Denver, Colorado, United States: ACM, 1972, 187-192. Doi:10.1145/800152.804913.
14. Seiferas Joel I, Fischer Michael J, Meyer Albert R. "Separating Nondeterministic Time Complexity Classes". J. ACM. New York, NY, USA: ACM. January 1978; 25 (1):146-167. Doi:10.1145/322047.322061. ISSN 0004-5411.
15. https://users.dcc.uchile.cl/~clgutier/Time_Complexity_P_vs_NP.pdf
16. Aaronson Scott. "P=? NP" (PDF). Retrieved 13 Apr 2021.
17. Wigderson Avi. "P, NP and mathematics – a computational complexity perspective" (PDF). Retrieved 13 Apr 2021.
18. Garey Michael R, Johnson David S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. 1979. ISBN 0-7167-1045-5.
19. Nair MM, Tyagi AK, Sreenath N. "The Future with Industry 4.0 at the Core of Society 5.0: Open Issues, Future Opportunities and Challenges," 2021 International Conference on Computer Communication and Informatics (ICCCI), 2021, 1-7. Doi: 10.1109/ICCCI50826.2021.9402498.
20. Tyagi AK, Fernandez TF, Mishra S, Kumari S. Intelligent Automation Systems at the Core of Industry 4.0. In: Abraham A., Piuri V., Gandhi N., Siarry P., Kaklauskas A., Madureira A. (eds) Intelligent Systems

Design and Applications. ISDA 2020. Advances in Intelligent Systems and Computing, 2021, 1351. Springer, Cham. https://doi.org/10.1007/978-3-030-71187-0_1

21. Varsha R, Nair SM, Tyagi AK, Aswathy SU, Radha Krishnan R. The Future with Advanced Analytics: A Sequential Analysis of the Disruptive Technology's Scope. In: Abraham A., Hanne T., Castillo O., Gandhi N., Nogueira Rios T., Hong TP. (eds) Hybrid Intelligent Systems. HIS 2020. Advances in Intelligent Systems and Computing, 2021, 1375. Springer, Cham. https://doi.org/10.1007/978-3-030-73050-5_56
22. Madhav AVS, Tyagi AK. The World with Future Technologies (Post-COVID-19): Open Issues, Challenges, and the Road Ahead. In: Tyagi A.K., Abraham A., Kaklauskas A. (eds) Intelligent Interactive Multimedia Systems for e-Healthcare Applications. Springer, Singapore. 2022. https://doi.org/10.1007/978-981-16-6542-4_22
23. Goyal Deepti, Tyagi, Amit. A Look at Top 35 Problems in the Computer Science Field for the Next Decade. 2020. 10.1201/9781003052098-40.
24. Rekha G, Malik S, Tyagi AK, Nair MM. Intrusion Detection in Cyber Security: Role of Machine Learning and Data Mining in Cyber Security, Advances in Science, Technology and Engineering Systems Journal. 2020;5(3):72-81.
25. Mishra S, Tyagi AK. Intrusion Detection in Internet of Things (IoTs) Based Applications using Blockchain Technolgy," 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2019, 123-128, Doi: 10.1109/I-SMAC47947.2019.9032557.
26. Tyagi Amit Kumar, Nair Meghna Manoj, Niladhuri Sreenath, Abraham Ajith. "Security, Privacy Research issues in Various Computing Platforms: A Survey and the Road Ahead", Journal of Information Assurance & Security. 2020;15(1):1-16. 16p.
27. Madhav AVS, Tyagi AK. The World with Future Technologies (Post-COVID-19): Open Issues, Challenges, and the Road Ahead. In: Tyagi A.K., Abraham A., Kaklauskas A. (eds) Intelligent Interactive Multimedia Systems for e-Healthcare Applications. Springer, Singapore. 2022. https://doi.org/10.1007/978-981-16-6542-4_22
28. Amit Kumar Tyagi, Aghila G. A Wide Scale Survey on Botnet", International Journal of Computer Applications (ISSN: 0975-8887). 2011;34(9):9-22.