List of Electives Courses in CSE

Course Code	Course Name	L	T	P	Credits
CS 570	Model Checking and Software Verification	3	0	0	6
CS 572	Modeling and Simulation of Systems	3	0	0	6
CS 462 / CS 662	Markov chains and their Applications	3	0	0	6
CS 611	Computational Complexity	3	0	0	6
CS 520	Combinatorial Optimization	3	0	0	6
CS 511	Approaches to Software Performance Enhancement	3	0	0	6
CS 521	Foundations of Functional Programming	3	0	0	6
CS 513	Introduction to Information Security	3	0	0	6

Course Name: Model Checking and Software Verification	**Course Code: CS 570**
	Course Type: UG-PG

Description

Arguing for program correctness is a challenging task. But it is non-optional, especially as the software has permeated our lives, in forms that are many times even safety- or business-critical. Formal Verification is an attractive and increasingly appealing alternative to simulation and testing. While the latter only explores a subset of possible behaviors, and leaves open the question of whether the unexplored behaviors may contain a fatal bug, formal verification explores all behaviors exhaustively. This course will focus on model checking as an approach to do formal verification.

Syllabus

Propositional and Predicate Logic (Overview); Modelling systems – Kripke Structure, Concurrent Systems, and NuSMV Model Checker; Linear-time properties – paths, traces, invariants, safety, liveness, fairness; Automata on finite and infinite words - Model checking omega-regular properties.

Temporal logics - Linear temporal logic (LTL), Computation tree logic (CTL)– Introduction to Promela and SPIN; Binary decision diagrams (BDDs) and Symbolic Model Checking.

Program Verification – Hoare Triples and Hoare Logic; Propositional satisfiability - SAT-based Model Checking (BMC, Inductive invariants), Introduction to CBMC; Predicate abstraction and Counterexample-Guided Abstraction Refinement (CEGAR).

References

1. Principles of Model Checking by *Christel Baier* and *Joost-Pieter Katoen*
2. Logic in Computer Science by *Michael Huth* and *Mark Ryan*
3. Model Checking (Second Edition, The Cyber-Physical Systems Series) by *Edmund Clark, Orna Grumberg, Daniel Kroening, Doron Peled,* and *Helmut Veith*

Course Name: Modeling and Simulation of Systems	**Course Code: CS 572**
	Course Type: UG-PG

Description

This course offers an introduction to modeling and simulation of discrete-event systems with examples from several application areas such as computing systems, manufacturing, queues and computer networks. The approach of the course would be to get the student to start modeling and simulating simple representative systems as soon as possible and to explore the system behavior via simulations before delving into the theory to understand the observed behavior. The course will make use of Python's SimPy library.

Syllabus

Introduction to models and simulation, the need for simulation, types of models.
Introduction to Python's SimPy library using examples.
Review of basic probability theory and Markov chains.
Random number generation.
Simulation of queues and their analysis.
Simulation of manufacturing systems.
Simulation of computer networks.
Parallel and distributed simulation: How to incorporate parallelism into the
Simulation execution while still obtaining the same results as a sequential execution.

References

1. Discrete-Event Simulation: A First Course, by Leemis and Park
2. Probability, Markov Chains, Queues, and Simulation, by William J. Stewart
3. Parallel and Distributed Simulation Systems, by Richard Fujimoto
4. SimPy Documentation: https://simpy.readthedocs.io/en/latest/

Course Name: Markov chains and their Applications	**Course Code: CS 462/CS 662**
	Course Type: UG and PG

Description

The course deals with applications of Markov chains techniques in certain areas of computer science and of biology. The unifying aspect in these applications is the role played by mixing time analysis, which is the focus of the course.

Syllabus

In a typical combinatorial optimization problem, each instance *x* is associated with a state space S_x, usually of size exponential in the size of *x*, where each element of S_x has an associated cost (or a value). The problem is to find a state with minimum cost (or with maximum value). In the Markov chains approach to solving such a problem, a Markov chain is associated with each instance with the property that the goal state has the highest probability in the stationary distribution of the chain. Success of this approach crucially depends on the mixing time of the associated chain, that is, how quickly does the chain come close to its stationary distribution.

Markov chains have also been used to model evolution for the finite population case lending themselves to stochastic effects. For a population of size *N* of *m* types, a state of the chain is the labelling of the *N* individuals each into one of the *m* types. The population goes from one state to another through reproduction, mutation, and selection. The specific way in which each of these happens determine the the transition probability matrix of the chain. The result of the evolution modelled by the chain is given by the stationary distribution of the chain. In most cases however, it is not known how to determine the stationary distribution directly, we need to simulate the chain ‘sufficiently long’ and then sample from the resulting distribution to obtain statistical properties of the distribution. We need to determine the mixing time of the chain to find out how long is ‘sufficiently long’. It has been established recently that the expected motion of such evolutionary chains turns out to be a dynamical system, the trajectory of which determines the mixing time properties like rapid mixing.

Approximate outline of the course: Ergodicity theorem of Markov chains. Mixing time of Markov chains. Combinatorial optimization using Markov chains. Metropolis algorithm. Notion of conductance and its relation to rapid mixing. Necessary and sufficient conditions for Markov chains approach to succeed for combinatorial optimization. Coupling of Markov chains. Markov chains modelling of molecular evolution. Quasispecies model. Viral evolution and notion of error threshold. Evolution for the finite population case. Expected motion of evolutionary Markov chains and corresponding dynamical system. Detailed analysis when the population is of two types. A qualitative understanding of the general *m* types case.
Each student will be required to do a term paper.

Textbooks and References

1. Finite Markov Chains and Algorithmic Applications, Olle Haggstrom, Cambridge Univ. Press, 2002.
2. Markov Chains, JR Norris, Cambridge Univ. Press, 1997.
3. Algorithms for random Generation and Counting: A Markov Chain Approach, Alistair Sinclair, Birkhauser, 1993.
4. Markov chains and mixing times, David A. Levin, Yuval Peres, and Elizabeth Wilmer, American Mathematical Society, 2006.
5. Relevant papers.

Course Name: Computational Complexity	**Course Code: CS 611**
	Course Type: PG

Description

In this course, we study various techniques used to classify computational problems according to their inherent complexity. How do we quantify and answer the following questions: “Is problem *A* more difficult than problem *B*? E.g. Is multiplication more difficult than addition?”, “Are there problems that are inherently difficult?”, “Are there problems that can never be solved, irrespective of the resources used?”, “Does randomization help?”

The more difficult a task, the more resources we use to solve it. The resources used by a computing machine are: time, space and randomness. We will study the usage of these resources and their interactions.

Syllabus

The computational model – brushing up algorithms, Big-o notation, Turing machines, P, NP and NP completeness – Reducibility, Cook-Levin theorem, Examples.
Diagonalization technique - Deterministic and non-deterministic time hierarchy, space hierarchy.
Space complexity – Configuration graphs, Savitch’s theorem, L, NL, Reachability, Immerman Szelepcsényi theorem.
Polynomial hierarchy (PH) – TQBF, other ways of defining PH, PH collapse.
Circuits – Equivalence with P/poly, counting argument to show the existence of hard functions, Karp-Lipton theorem.
Randomized Computation – Randomness as a resource, examples, complexity classes that use randomness.
Counting Classes (if time permits)- #P, permanent, Toda’s theorem.
Interactive proofs (if time permits)– Arthur-Merlin games, Interactive proofs, graph non-isomorphism,IP = PSPACE.

Textbooks and References

1. Sanjeev Arora, Boaz Barak: Computational Complexity - A Modern Approach. Cambridge University Press 2009, ISBN 978-0-521-42426-4, pp. I-XXIV, 1-579
2. Christos H. Papadimitriou: Computational complexity. Addison-Wesley 1994, ISBN 978-0-20153082-7, pp. I-XV, 1-523

Course Name: Combinatorial Optimization	**Course Code: CS 520**
Pre-requisites: Algorithms, Linear algebra	**Course Type:** UG/PG

Description

The course introduces optimization and combinatorial techniques used to solve optimization problems. A special focus will be on linear programming.

Introduction to Optimization problem. Example problems like Travelling salesman problem, bipartite matching, max SAT etc.
Mathematics primer - Convex sets and functions, linear algebra

Linear programming introduction - Different forms of linear programming problems, Examples of problems reduced to linear programming.
The Simplex algorithm - Geometry of linear programming. Feasible solutions, basic feasible solution, the algorithm.
Duality - Dual of a linear program, Farkas' lemma, Primal-Dual algorithm schema. Examples will be Ford-Fulkerson max-flow and Dijkstra's shortest path.
Linear programming in Polynomial time - Khachiyan's ellipsoid method and/or Karmarkar's Integer point method.

Textbooks and References

1. Combinatorial optimization by Papadimitriou and Steiglitz
2. Theory of Linear and Integer programming by Schrijver
3. Combinatorial Optimization by Schrijver

Course Name: Approaches to Software Performance	**Course Code: CS 511**
Pre-requisites: Basic programming and algorithms	**Course Type:** UG/PG

Description

The course will introduce the students with some fundamental knowledge of software engineering, some algorithmic tricks to improve the code and some tools to improve performance of softwares

Topics: Following topics will be covered.

- Software development life cycle, process models, module design, testing, modelling
- Choosing variable names, function names, guidelines for creating designer's type, globals, organizing straight line code, choosing most effective ordering of case statements, selecting and exiting loops, length, nesting level of loop, etc, Unusual control structures - goto, recursion, multiple returns from a routine, table driven approach versus inheritance, taming deep nesting, McCabe measure of complexity.
- Different compiler optimizations e.g., code motion, common sub-expression elimination, constant propagation, etc and the optimizations which make the stage for parallelization e.g., loop skewing, interchange, software pipelining , etc which can make the code more efficient
- Algorithm design tricks : using binary search over linear search, tricks to reduce the space (e.g., using unsigned char if the integer is too small), dynamic allocation, verification using assertions, using hand-made quick-sort instead of system qsort or insertion sort, comparing sorting and searching algorithms in terms of time and space and use appropriately
- Code improvement techniques : Debugging, refactoring, caching, etc and tools e.g., coverage tool (gcov), profiling tool (gprof), memory leakage identification tool (valgrind), debugger (gdb)

Textbooks and References

1. Fundamentals of Software Engineering by Rajib Mal
2. Compilers: Principles, techniques and tools by Aho, Lam, Sethi, Ullman
3. Programming Pearls (second edition) by Jon Bentley
4. Code Complete by Steve McConnell

Course Name: Foundations of Functional Programming	**Course Code: CS 521**
	Course Type: UG/PG

Description

The course will introduce students to the functional programming paradigm and its foundations.

Topics covered

- What is functional programming, history and evolution of functional programming languages
- Types, declaring types, lists, operations
- Defining functions, composition
- Currying, lambdas, untyped and simply typed lambda calculus
- Defining functions -- conditionals, guards, pattern matching
- List comprehensions
- Higher order functions — map, filter, folds, Combinatory Logic
- Defining types, type classes
- Functional data structures — lists, trees, bst
- IO, hello world
- Monads and functors

Textbooks and References

Programming in Haskell - Graham Hutton
Learn you a haskell for great good - Miran Lipovaca

Course Name: Introduction to Information Security	**Course Code: CS 513**
	Course Type: UG/PG

Description

Computers and Internet have become an integral part of our lives. On daily basis, we use laptops, mobiles, LAN, Internet, Web and Cloud. As we can access computing and entertainments resources all over the world, cyber criminals have access to our personal data, which they can exploit in many ways. Hence, it is important that we make our systems and network secure.

The course covers basic concepts and principles of Information security and fundamental approaches to secure computers and networks.

The course assumes that students have basic understanding of computer organization, OS, and networks.

Syllabus

Module 1: Basics of Computer Security (threats, attacks, confidentiality, integrity, availability)
Module 2: Software Security – attacks and defences (buffer overflows, viruses, secure software development)
Module 3: Web Security
Module 4: : Internet security, DOS, DDOS, etc
Module 5: Basic concepts of Cryptography
Module 6: Human factors in security
Module 7: Network Security (Internet background, attacks, Firewall, NAT, VPN)
Module 8: Wireless network Security (Wireless network background, security mechanisms)
Module 9: Hacking defined
Module 10: OS and application vulnerabilities
Module 11: Blockchain Security concepts
Module 12: Invited talks from Industry professional
Module 13: Student Project discussions

Textbooks and References

1. *Security in Computing*, Pfleeger & Pfleeger, 4th ed.
2. *Introduction to Computer Security*, Goodrich & Tamassia, 1st ed.