THE UNIVERSITY OF BURDWAN

SYLLABUS OF TWO YEAR POSTGRADUATE PROGRAMME

IN

Computer Science

Under

Curriculum and Credit Framework for Postgraduate Programme (CCFPP), as per N.E.P. 2020

w.e.f. 2026 – 2027

2- year PG Programme

Semesterwise and Coursewise Distribution of Credit & Marks under CCFPP as per NEP, 2020

SEMESTER	Course Type	Code & Level	Name of the Course	Credit	L – T - P	Marks	Marks Dist. Th. – Pr. - IA
I	Major/Core Course	**PG2-COMP-1011 (400-499)**	Research Methodology & IKS	6	3-1-2	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-1012 (400-499)**	Compiler Design	6	3-1-2	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-1013 (400-499)**	Advanced Data Structure & Algorithms	6	3-1-2	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-1014 (400-499)**	Soft Computing	6	3-1-2	75	40 – 20 – 15
	Minor Course	**PG2-COMP-1021 (300-399)**	Database Management Systems or Equivalent Course from SWAYAM* **(For other allied discipline)**	4	2-1-1	75	40 – 20 – 15
	Total			**28**		**375**	

***The SWAYAM course should be approved by the appropriate authority of the University.**

II	Major/Core Course	**PG2-COMP-2011 (400-499)**	Network Security & Cryptography	6	**3-1-2**	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-2012 (400-499)**	Optimization Techniques	4	**2-1-1**	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-2013 (400-499)**	Data Science	4	**2-1-1**	75	40 – 20 – 15
	Major/Core Course	**PG2-COMP-2014 (400-499)**	System Programming using UNIX/Linux	4	**2-1-1**	75	40 – 20 – 15
	Minor Course	**PG2-COMP-2021 (300-399)**	Introduction to Machine Learning or Equivalent Course from SWAYAM* **(For other allied discipline)**	4	**2-1-1**	75	40 – 20 – 15
	Total			**22**		**375**	

***The SWAYAM course should be approved by the appropriate authority of the University.**

OPTION- I: Course-work in the 3rd Semester & Research Thesis/Project/ Patent in the 4th Semester

SEMESTER	Course Type	Code & Level	Name of the Course	Credit	L – T – P	Marks	Marks Dist. Th. – Pr. - IA
III	Major/Core Course	**PG2-COMP-3011 (500-599)**	Advanced Software Engineering	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3012 (500-599)**	Machine Learning	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3013 (500-599)**	Elective-I	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3014 (500-599)**	Technical Seminar	4	**0 – 0 – 4**	50	0 – 40 – 10
	Major/Core Course	**PG2-COMP-3015 (500-599)**	Minor Project	4	**0 – 0 – 4**	50	0 – 40 – 10
	Total			**20**		**250**	
IV	Research Thesis/ Project / Patent	**PG2-COMP-4031 (500-599)**	Research Thesis/ Project / Patent	20	**0- 0- 20**	250	0-250-0 Seminar Presentation, Preparation & Submission of Research Thesis/ Project / Patent Document-150 + Viva-100
	Total			**20**		**250**	

OR

OPTION- II: Only Course- work in 3rd & 4th Semesters

SEMESTER	Course Type	Code & Level	Name of the Course	Credit	L – T - P	Marks	Marks Dist. Th. – Pr. - IA
III	Major/Core Course	**PG2-COMP-3011 (500-599)**	Advanced Software Engineering	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3012 (500-599)**	Machine Learning	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3013 (500-599)**	Elective-I	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-3014 (500-599)**	Technical Seminar	4	**0 – 0 – 4**	50	0 – 40 – 10
	Major/Core Course	**PG2-COMP-3015 (500-599)**	Minor Project	4	**0 – 0 – 4**	50	0 – 40 – 10
	Total			**20**		**250**	
IV	Major/Core Course	**PG2-COMP-4011 (500-599)**	Deep Learning	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-4012 (500-599)**	Distributed Systems	4	**2 – 1 – 1**	50	30 – 10 – 10
	Major/Core Course	**PG2-COMP-4013 (500-599)**	Elective-II	4	**2 – 1 – 1**	50	30 – 10 – 10

	Major/Core Course	**PG2-COMP-4014 (500-599)**	Seminar & Grand Viva	4	**0 – 0 – 4**	50	0 – 40 – 10
	Major/Core Course	**PG2-COMP-4015 (500-599)**	Major Project	4	**0 – 0 – 4**	50	0 – 40 – 10
	Total			**20**		**250**	

OR

<u>OPTION-</u> III: Only Research Thesis/ Project / Patent in 3rd & 4th Semesters

SEMESTER	Course Type	Code & Level	Name of the Course	Credit	L – T - P	Marks	Marks Dist. Th. – Pr. - IA
III	Research Thesis/ Project / Patent	**PG2-COMP-3031 & PG2-COMP-4031 (500-599)**	Research Thesis/ Project / Patent	40	0-0-40	500	0-500-0 Seminar Presentation, Preparation & Submission of Research Thesis/ Project / Patent Document -300 + Viva-200
IV							
	Total			**40**		**500**	

Elective-I:

A. Advanced Algorithm
B. Quantum Computing

Elective-II:

A. Advanced Cryptography and Network Security
B. Block Chain Technology

General Guidelines for Research Project / Dissertation / Research Thesis / Project / Patent for Papers PG2-COMP-2031, PG2-COMP-4031, PG2-COMP-3031, PG2-COMP-4031

The **Research Project / Dissertation** and **Research Thesis / Project / Patent** shall be carried out under the supervision of a faculty member approved by the Department. The research work shall be undertaken in accordance with the academic and research regulations, ethical standards, and quality requirements prescribed by the **University** and the **Department of Computer Science** from time to time.

The topic of research shall be approved by the Department following the prescribed procedure. Students are expected to conduct original work, maintain academic integrity, and adhere to all applicable regulations concerning plagiarism, research ethics, intellectual property rights, and publication practices. The format of the dissertation, thesis, project report, or patent document, together with submission procedures, evaluation methods, seminar presentations, and viva-voce examinations, shall be governed by the guidelines, notifications, and ordinances issued by the University and the Department from time to time.

Where the work leads to an innovative product, process, software, algorithm, or technology with potential intellectual property value, the student may, with the approval of the supervisor and the Department, prepare and submit a **Patent** in place of, or as a component of, the research thesis/project, subject to the University's Intellectual Property Rights (IPR) policy and other applicable regulations.

All matters relating to registration, supervision, evaluation, report preparation, thesis formatting,

patent submission, seminar presentation, viva-voce examination, and award of grades shall be governed by the rules, regulations, and guidelines issued by the University and the Department of Computer Science, as amended from time to time. In the event of any ambiguity, the decision of the University shall be final and binding.

PG2-COMP-1011: Research Methodology & IKS

Course Objectives

The course equips students with scientific research methodology, professional communication skills, technical writing, documentation practices, and digital research tools while introducing selected Indian Knowledge Systems relevant to Computer Science, mathematics, language sciences, and cryptography. Students will learn to conduct independent research, critically evaluate scientific literature, communicate scientific findings effectively, and prepare professional technical documents.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. **Identify** research problems, formulate research objectives, and design appropriate research methodologies.
2. **Conduct** literature surveys, analyze research data using statistical techniques, and interpret research findings.
3. **Prepare** research proposals, technical reports, research papers, and professional presentations following ethical and scholarly practices.
4. **Use** modern research tools, including reference management software, LaTeX, version control systems, AI-assisted research tools, and reproducible research practices.
5. **Demonstrate** awareness of research ethics, publication standards, intellectual property rights, and responsible conduct of research.
6. **Explain** selected contributions of Indian Knowledge Systems to Computer Science, particularly in formal language theory, mathematics, algorithms, and cryptography.

Theory Syllabus **(3 Credits – 45 Hours)**

Module I: Introduction to Research Methodology **(2 Hours)**

Nature and objectives of scientific research; types of research; research process; identification of research problems; literature gaps; formulation of research questions and hypotheses; research design; variables; validity and reliability; qualitative, quantitative and mixed-method research.

Module II: Data Collection and Statistical Analysis **(10 Hours)**

Primary and secondary data; sampling techniques; survey methods; experimental methods; descriptive statistics; probability distributions; correlation and regression; hypothesis testing; ANOVA; effect size; non-parametric statistical tests; interpretation of statistical results.

Module III: Scientific Literature, Research Writing and Publication **(6 Hours)**

Scientific databases and digital libraries; systematic literature review; bibliometric indicators; citation analysis; h-index; impact factor; journal indexing; structure of research papers and dissertations; technical writing; citation and referencing; plagiarism; research integrity; peer review; predatory journals; open-access publishing.

Module IV: Research Tools and Professional Documentation **(15 Hours)**

Reference management software; statistical software; AI tools for research; reproducible research; Git and version control; research documentation; project proposal writing; grant writing latex; preparation of technical reports; professional presentations. Data filtering tools of UNIX. Plotting and data visualization tools. Matplotlib and Pandas package.

Module V: Indian Knowledge Systems for Computer Science (12 Hours)

Overview of Indian Knowledge Systems relevant to scientific inquiry; Paninian grammar and formal language theory; Pingala's binary representation and combinatorics; decimal place-value system, zero and infinity; contributions of Kerala School, Aryabhata, Brahmagupta, Bhaskara II and Srinivasa Ramanujan; Indian number theory in modern cryptography; AKS Primality Testing Algorithm and its significance; Ramanujan graphs(overview); documentation and communication of scientific knowledge in Indian traditions.

Tutorial (1 Credit – 15 Hours)

Research Methodology (10 Hours)

1. Discussion and clarification of research process, research design, and formulation of research problems.
2. Guided study of qualitative, quantitative, and mixed-method research with illustrative examples.
3. Discussion of sampling methods, experimental design, and interpretation of statistical results.
4. Guided reading and critical discussion of selected research papers and review articles.
5. Explanation of research metrics, citation analysis, journal indexing, and publication quality assessment.
6. Discussion of scientific writing, research paper organization, citation styles, plagiarism, research ethics, and publication practices.
7. Demonstration and discussion of modern research tools, including reference management software, LaTeX, version control, AI-assisted research tools, and reproducible research practices.
8. Discussion of research proposal preparation, grant writing, and project planning.

Indian Knowledge Systems (3 Hours)

1. Discussion on Paninian grammar and its relevance.
2. Discussion on Indian mathematics, Ramanujan's contributions, number theory, and the significance of the AKS primality testing algorithm in modern cryptography.
3. Discussion on selected contributions of Indian Knowledge Systems to Computer Science, including Pingala's combinatorics, Indian numeral system, and computational thinking.

Practical (2 Credits – 60 Hours)

Part A: Research Methodology and Professional Skills (45 Hours)

1. Reference management using Zotero/Mendeley.
2. Scientific document preparation using LaTeX.
3. Git and GitHub for research.
4. Statistical analysis Computer.
5. Data visualization.
6. Research paper formatting.
7. Plagiarism detection.
8. Research proposal preparation.

Part B: IKS Applications (15 Hours)

1. Study of Paninian grammar as a rule-based system.

2. Modular exponentiation.
3. Miller–Rabin primality testing.
4. Demonstration/implementation of the AKS Primality Testing Algorithm.
5. Mini report on Ramanujan's contribution to computational mathematics.
6. Case study: Indian number theory in modern cryptography.

---

1. C. R. Kothari and Gaurav Garg, *Research Methodology: Methods and Techniques*, New Age International Publishers.
2. John W. Creswell and J. David Creswell, *Research Design: Qualitative, Quantitative and Mixed Methods Approaches*, Sage Publications.
3. Ranjit Kumar, *Research Methodology: A Step-by-Step Guide for Beginners*, Sage Publications.
4. Wayne C. Booth, Gregory G. Colomb, Joseph M. Williams, Joseph Bizup, and William T. FitzGerald, *The Craft of Research*, University of Chicago Press.
5. Robert A. Day and Barbara Gastel, *How to Write and Publish a Scientific Paper*, Cambridge University Press.
6. Kate L. Turabian, *A Manual for Writers of Research Papers, Theses, and Dissertations*, University of Chicago Press.
7. M. Ashraf Rizvi, *Effective Technical Communication*, McGraw-Hill Education.
8. Meenakshi Raman and Sangeeta Sharma, *Technical Communication: Principles and Practice*, Oxford University Press.
9. Kapil Kapoor and Avadesh Kumar Singh (Eds.), *Indian Knowledge Systems*, Volumes I & II, Indian Institute of Advanced Study (IIAS), Shimla.
10. B. Mahadevan, Vinayak Rajat Bhat, and Nagendra Pavana R. N., *Introduction to Indian Knowledge System: Concepts and Applications*, PHI Learning.
11. Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley.
12. Scott Chacon and Ben Straub, *Pro Git*, Apress.

PG2-COMP-1012: Complier Design

Course Objectives

The course aims to provide students with a comprehensive understanding of the fundamental principles of compiler design and the different phases involved in the compilation process. It introduces formal languages, regular expressions, finite-state automata, lexical analysis, syntax analysis, intermediate-code generation, code optimization and target-code generation. The course also develops an understanding of the algorithms, data structures and automated tools used for designing and implementing modern compilers.

The lecture component develops the theoretical foundations required to understand compiler architecture and compilation techniques. The tutorial component strengthens conceptual understanding through analytical exercises, grammar transformations, automata construction, parsing-table preparation and intermediate-code-generation problems. The practical component provides hands-on experience in implementing lexical analyzers, parsers, symbol tables, intermediate-code generators and code-optimization techniques using suitable programming languages and compiler-construction tools.

Course Learning Outcomes

On successful completion of the course, the students will be able to achieve the following outcomes.

- Describe the structure of a compiler and explain the functions of its different phases and passes.
- Apply formal-language concepts, regular expressions, regular grammars and finite-state automata in compiler design.
- Explain the role and working principles of lexical analyzers and recognize tokens, patterns and lexemes.
- Construct parse trees and apply top-down and bottom-up parsing techniques for analysing programming-language syntax.
- Generate intermediate code using syntax-directed translation, syntax trees, dependency graphs and three-address-code representations.
- Apply type checking, symbol-table management, control-flow translation and back-patching

techniques.

- Explain runtime-storage management and generate target code using appropriate code-generation techniques.
- Apply peephole optimization and other suitable optimization techniques to improve intermediate and target code.
- Use modern compiler-writing tools and technologies for designing and implementing compiler components.

Lecture

Lecture Credit: 3

Total Lecture Hours: 45 Hours

Unit 1: Overview of Compiler

This unit introduces language-processing systems and the role of a compiler. It covers the structure of a compiler, analysis and synthesis phases, front-end and back-end organization, phases and passes of a compiler, single-pass and multi-pass compilers, symbol-table management and error handling. The unit also discusses the major problems associated with compiler design, applications of compiler-design technology and the interdependencies between compiler design and computer architecture. (3 Hrs)

Unit 2: Formal Languages and Finite Automata

This unit introduces the fundamental elements of formal-language theory, including alphabets, strings, languages, grammars and operations on languages. It covers regular languages, regular grammars, right-linear and left-linear grammars, regular expressions and their algebraic properties. Deterministic finite automata, non-deterministic finite automata and finite automata with epsilon transitions are discussed. The unit also covers the conversion of regular expressions into finite automata, conversion of non-deterministic finite automata into deterministic finite automata, conversion of finite automata into regular expressions and conversion between regular grammars and finite automata. Equivalence among regular expressions, regular grammars and finite automata is examined. Methods for identifying equivalent states and minimizing deterministic finite automata are also included. (8 Hrs)

Unit 3: Lexical Analysis

This unit discusses the purpose and functions of lexical analysis and distinguishes lexical analysis from syntax analysis. It introduces tokens, patterns, lexemes and token attributes. Specification of tokens through regular expressions and recognition of tokens through finite automata are covered. The unit explains input buffering, buffer pairs, sentinels, look-ahead operations and the recognition of identifiers, keywords, numbers, operators and delimiters. It also addresses the handling of white spaces, comments and lexical errors. The interaction of the lexical analyzer with the parser and symbol table is discussed. The unit concludes with the design of hand-written lexical analyzers and an introduction to lexical-analysis tools such as Lex and Flex. (6 Hrs)

Unit 4: Syntax Analysis and Parsing

Context-Free Grammars and Grammar Transformation

This subsection introduces context-free grammars, terminals, non-terminals, productions, start symbols, sentential forms, leftmost derivations, rightmost derivations and parse trees. It explains ambiguity in grammars, ambiguous derivations, operator associativity and operator precedence. Techniques for eliminating immediate and indirect left recursion and performing left factoring are discussed.

Top-Down Parsing

This subsection covers the principles of top-down parsing, recursive-descent parsing, backtracking and non-backtracking parsing. It introduces predictive parsing, calculation of FIRST and FOLLOW sets, construction of predictive parsing tables and the properties of LL(1) grammars. Table-driven predictive parsing and error-recovery techniques for predictive parsers are also included.

Bottom-Up Parsing

This subsection explains handles, handle pruning and viable prefixes. It introduces shift-reduce parsing and the shift, reduce, accept and error actions of a bottom-up parser. Shift-reduce conflicts and reduce-reduce conflicts are discussed. Operator-precedence grammars, operator-precedence relations and operator-precedence parsing are also covered.

LR Parsing

This subsection introduces the structure and working principles of LR parsers. It covers augmented

grammars, LR items, closure and GOTO operations and the construction of canonical collections of LR items. Simple LR parsing, canonical LR parsing and look-ahead LR parsing are studied. Construction of SLR, CLR and LALR parsing tables is discussed along with a comparison of their power, complexity and applications. Error detection in LR parsing and the use of parser-generation tools such as Yacc and Bison are also introduced. (14 Hrs)

Unit 5: Intermediate-Code Generation

Syntax-Directed Translation

This subsection introduces syntax-directed definitions and syntax-directed translation schemes. It covers synthesized attributes, inherited attributes, S-attributed definitions and L-attributed definitions. Dependency graphs and the order of evaluation of attributes are also discussed.

Intermediate Representations

This subsection explains syntax trees, abstract syntax trees, postfix notation and Directed Acyclic Graph representations of expressions. Three-address code and its common statement forms are introduced. Representations of three-address code using quadruples, triples and indirect triples are covered.

Type Checking and Symbol Tables

This subsection introduces type expressions, type systems, type checking, type conversion and type coercion. It explains the organization and functions of a symbol table, including insertion, lookup, update and scope management. Common symbol-table data structures are also discussed.

Translation of Statements

This subsection covers the translation of arithmetic expressions, Boolean expressions, assignment statements, conditional statements and iterative statements. Short-circuit evaluation of Boolean expressions is explained. Back-patching techniques for the translation of control-flow statements are also included. (9 Hrs)

Unit 6: Runtime Environment, Code Generation and Optimization

This unit introduces runtime-storage organization, static allocation, stack allocation and heap allocation. It discusses activation records, access links, control links, parameter passing and access to non-local data. The unit examines the major issues involved in code generation, including the input to the code generator, target-program representation, instruction selection, register allocation, register assignment and order of evaluation. Basic blocks, flow graphs and Directed Acyclic Graphs for basic blocks are introduced.

The unit also covers machine-independent and machine-dependent optimization techniques. Common subexpression elimination, constant folding, constant propagation, copy propagation, dead-code elimination, algebraic simplification and strength reduction are discussed. Peephole optimization and basic loop-optimization techniques are included. The unit concludes with an overview of compiler-writing tools and modern compiler infrastructures such as Lex, Flex, Yacc, Bison, ANTLR and LLVM. (5 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Compiler Structure and Compilation Phases

This tutorial unit includes exercises on identifying the phases of a compiler, tracing the transformation of a source-language statement through different compiler phases, distinguishing phases from passes and comparing compilers, interpreters, assemblers, linkers and loaders. Students will also analyse the relationship between compiler design and target-machine architecture. (2 Hrs)

Tutorial Unit 2: Regular Expressions, Grammars and Finite Automata

This tutorial unit includes the construction of regular expressions for specified languages, development of regular grammars and conversion between regular expressions and finite automata. Students will solve problems involving epsilon-NFA construction, NFA-to-DFA conversion, regular-grammar conversion and DFA state minimization. Equivalence of regular expressions, grammars and finite automata will be demonstrated through worked examples. (3 Hrs)

Tutorial Unit 3: Lexical Analysis

This tutorial unit includes the identification of tokens, patterns and lexemes from sample source programs. Students will formulate regular expressions for identifiers, constants, operators, comments and other programming-language constructs. Exercises on input buffering, token recognition, transition diagrams and lexical-error handling will also be undertaken. (2 Hrs)

Tutorial Unit 4: Top-Down and Bottom-Up Parsing
This tutorial unit includes the construction of parse trees and leftmost and rightmost derivations. Students will identify ambiguous grammars and perform elimination of left recursion and left factoring. Exercises will include calculation of FIRST and FOLLOW sets, verification of LL(1) conditions and construction of predictive parsing tables.
Bottom-up parsing exercises will include identification of handles, preparation of shift-reduce parsing sequences and analysis of shift-reduce and reduce-reduce conflicts. Students will construct LR item sets and prepare SLR, CLR and LALR parsing tables for suitable grammars. (4 Hrs)
Tutorial Unit 5: Syntax-Directed Translation and Intermediate Code
This tutorial unit includes exercises on synthesized and inherited attributes, dependency-graph construction and evaluation order. Students will construct syntax trees and Directed Acyclic Graphs for arithmetic expressions. Problems involving three-address code, quadruples, triples and indirect triples will be solved. Type-checking, symbol-table organization, Boolean-expression translation and back-patching exercises will also be undertaken. (3 Hrs)
Tutorial Unit 6: Code Generation and Optimization
This tutorial unit includes problems on activation records, runtime-storage allocation, basic-block identification and flow-graph construction. Students will perform instruction selection for simple intermediate-code sequences and apply common subexpression elimination, constant folding, constant propagation, copy propagation, dead-code elimination and peephole optimization. (1 Hrs)

Practical
Practical Credit: 2
Hours per Practical Credit: 30 Hours
Total Practical Hours: 60 Hours
The practical component may be conducted using C, C++, Java or Python along with suitable compiler-writing tools such as Lex, Flex, Yacc, Bison, ANTLR or equivalent technologies.
Practical 1: Recognition of Tokens in a Source Program
Students will develop a program to read a source-code fragment and identify keywords, identifiers, constants, operators, delimiters and special symbols. The program will display the token type and corresponding lexeme. (4 Hrs)
Practical 2: Implementation of Regular-Expression Matching
Students will implement a program for recognizing strings belonging to selected regular languages. Regular expressions for identifiers, integer constants, floating-point constants, operators and comments will be designed and tested. (4 Hrs)

Practical 3: Conversion of NFA to DFA
Students will implement the subset-construction algorithm for converting a non-deterministic finite automaton into an equivalent deterministic finite automaton. The program will generate DFA states and corresponding transitions. (4 Hrs)
Practical 4: Minimization of a Deterministic Finite Automaton
Students will implement an algorithm for minimizing a deterministic finite automaton by identifying equivalent states and constructing the minimized transition table. (4 Hrs)
Practical 5: Design of a Lexical Analyzer
Students will design and implement a lexical analyzer for a selected subset of a programming language. The analyzer will recognize identifiers, keywords, constants, operators and delimiters and will report invalid tokens. (5 Hrs)
Practical 6: Lexical Analyzer Using Lex or Flex
Students will prepare a Lex or Flex specification for recognizing common programming-language tokens. The generated scanner will be compiled and tested using valid and invalid source-code inputs. (5 Hrs)
Practical 7: Elimination of Left Recursion and Left Factoring
Students will implement a program to eliminate immediate left recursion from a context-free grammar. The program will also perform left factoring for suitable grammar productions. (4 Hrs)
Practical 8: FIRST and FOLLOW Set Generation
Students will implement a program to calculate the FIRST and FOLLOW sets of non-terminals in a context-free grammar. The generated sets will be used to determine whether the grammar satisfies LL(1)

conditions. (4 Hrs)

Practical 9: Predictive Parser Implementation

Students will construct an LL(1) parsing table and implement a table-driven predictive parser. The parser will accept or reject input strings and display the sequence of parsing actions. (5 Hrs)

Practical 10: Shift-Reduce or Operator-Precedence Parser

Students will implement a shift-reduce parser or operator-precedence parser for a suitable expression grammar. The implementation will demonstrate shift, reduce, accept and error actions. (5 Hrs)

Practical 11: Parser Generation Using Yacc or Bison

Students will use Yacc or Bison to generate a parser for arithmetic expressions or a selected subset of a programming language. The parser will be integrated with a Lex or Flex scanner and will perform syntax-error reporting. (5 Hrs)

Practical 12: Symbol-Table Implementation

Students will design and implement a symbol table that supports insertion, lookup, modification and deletion operations. The symbol table will store attributes such as identifier name, data type, scope, memory location and value. (4 Hrs)

Practical 13: Intermediate-Code Generation

Students will implement a program to generate three-address code for arithmetic and assignment expressions. The generated code will also be represented using quadruples, triples or indirect triples. (4 Hrs)

Practical 14: Back-Patching and Control-Flow Translation

Students will generate intermediate code for conditional statements and iterative statements using back-patching. The implementation will maintain and merge appropriate true, false and next lists. (4 Hrs)

Practical 15: Basic Code Optimization

Students will implement selected optimization techniques such as constant folding, constant propagation, algebraic simplification, common subexpression elimination and dead-code elimination. The program will display the intermediate code before and after optimization. (4 Hrs)

Reference Books

Primary Reference

1. Aho, A. V., Sethi, R. and Ullman, J. D., *Compilers: Principles, Techniques and Tools*, Addison-Wesley.

Additional References

1. Holub, A. I., *Compiler Design in C*, Prentice Hall of India.
2. Dhamdhere, D. M., *Compiler Construction: Principles and Practice*.
3. Aho, A. V. and Ullman, J. D., *Principles of Compiler Design*, Narosa Publishing House.
4. Cooper, K. D. and Torczon, L., *Engineering a Compiler*, Morgan Kaufmann.
5. Appel, A. W., *Modern Compiler Implementation*, Cambridge University Press.

PG2-COMP-1013: Advanced Data Structure & Algorithms

Course Objectives

The objectives of this course are to enable students to:

- Understand advanced data structures and their applications in efficient problem solving.
- Analyze the correctness and computational complexity of algorithms.
- Apply fundamental algorithm design paradigms for solving computational problems.
- Design efficient algorithms for searching, sorting, graph, and string processing problems.
- Develop programming skills for implementing data structures and algorithms.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Select appropriate data structures for different computational problems.
2. Analyze the correctness and complexity of algorithms using asymptotic techniques.
3. Design efficient algorithms using divide-and-conquer, dynamic programming, and greedy strategies.
4. Implement graph and string algorithms for practical applications.
5. Compare different algorithmic approaches based on efficiency and suitability.
6. Develop optimized algorithmic solutions using modern programming techniques.

Theory Syllabus (45 Hours)

Unit I: Advanced Data Structures (10 Hours)

Review of linear and non-linear data structures. Binary Search Trees (BST), insertion, deletion and searching and query. Balanced trees, B-Trees and B+ Trees. Splay Trees. Disjoint Set (Union-Find) data structures. K-D Trees. Introduction to Object-Oriented Data Structures. Removal of recursion and recursive data structures.

Unit II: Algorithm Analysis (7 Hours)

Problem-solving methodology. Algorithm design and development. Correctness of algorithms. Time and space complexity. Best, average and worst-case analysis. Asymptotic notations. Upper and lower bounds. Recurrence relations. Master Theorem. Amortized analysis. Experimental analysis of algorithms. Complexity comparison of searching and sorting algorithms.

Unit III: Divide-and-Conquer and Dynamic Programming (10 Hours)

Divide-and-Conquer paradigm. Top-down approach. Binary Search. Merge Sort. Quick Sort. Recurrence analysis. Dynamic Programming. Principle of Optimality. Memoization and Tabulation. Overlapping subproblems and optimal substructure. Matrix Chain Multiplication. 0/1 Knapsack. Optimal Binary Search Tree. Correctness and complexity analysis.

Unit IV: Greedy Algorithms and Sorting Techniques (6 Hours)

Greedy strategy. Greedy choice property. Comparison between Greedy and Dynamic Programming. Activity Selection. Fractional Knapsack. Introduction to Matroids. Merge Sort. Quick Sort. Insertion Sort, Selection Sort Counting Sort. Radix Sort. Bucket Sort. Medians and Order Statistics. Lower bounds

on comparison-based sorting.

Unit V: Graph Algorithms (6 Hours)

Graph representations. Graph traversal using DFS and BFS. Connected components. Strongly connected components. Bi-connectivity. Topological Sorting. Minimum Spanning Trees (Prim's and Kruskal's algorithms). Shortest Path. Introduction to Network Flow.

Unit VI: String Algorithms (6 Hours)

Introduction to string processing. Naïve String Matching. Rabin-Karp Algorithm. Finite Automata based Pattern Matching. Knuth-Morris-Pratt (KMP) Algorithm. Boyer-Moore Algorithm. Longest Common Subsequence. Introduction to String Compression. Sequence Alignment.

Total Theory Hours: 45

Tutorial Syllabus (15 Hours)

1. Discussion on abstract data types, balanced trees, B-Trees, and disjoint-set data structures.
2. Complexity analysis using asymptotic notations, recurrence relations, and Master Theorem.
3. Guided problem solving using Divide-and-Conquer and Dynamic Programming techniques.
4. Comparative analysis of Greedy and Dynamic Programming approaches for optimization problems.
5. Discussion and analysis of graph traversal, shortest path, and minimum spanning tree algorithms.
6. Discussion on string matching algorithms.
7. Analysis of algorithm correctness, efficiency, and suitable data structure selection through case studies.

Practical Syllabus (60 Hours)

1. Implementation of Binary Search Tree operations.
2. Implementation of AVL Tree.
3. Implementation of Red-Black Tree.
4. Implementation of B-Tree/B+ Tree.
5. Implementation of Disjoint Set (Union-Find).
6. Implementation of K-D Tree.
7. Comparative study of Merge Sort and Quick Sort.
8. Implementation of Counting Sort, Radix Sort, and Bucket Sort.
9. Binary Search and Order Statistics.

10. Dynamic Programming: Matrix Chain Multiplication.
11. Dynamic Programming: 0/1 Knapsack Problem.
12. Greedy Algorithms: Activity Selection and Huffman Coding.
13. Minimum Spanning Tree using Prim's and Kruskal's algorithms.
14. Shortest Path using Dijkstra's and Bellman-Ford algorithms.
15. Graph traversal using BFS and DFS.
16. String Matching using Rabin-Karp and KMP algorithms.
17. Pattern matching using Boyer-Moore algorithm.
18. Comparative performance analysis of searching, sorting, graph, and string algorithms.

Total Practical Hours: 60

Text / Reference Books

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 4th Edition, MIT Press.
2. Mark Allen Weiss, *Data Structures and Algorithm Analysis in C++*, Pearson.
3. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *Data Structures and Algorithms*, Addison-Wesley.
4. Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta, *Fundamentals of Data Structures in C++*, Universities Press.
5. Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th Edition, Addison-Wesley.
6. Steven S. Skiena, *The Algorithm Design Manual*, 3rd Edition, Springer.
7. Gilles Brassard and Paul Bratley, *Fundamentals of Algorithmics*, Prentice Hall.
8. Donald E. Knuth, *The Art of Computer Programming*, Volumes 1–3, Addison-Wesley.

PG2-COMP-1014: Soft Computing

Course Objectives (COB)

After successful completion of this course, students will be able to:

1. **Understand the fundamentals of Artificial Intelligence and Soft Computing**, including intelligent problem-solving techniques, state-space search, knowledge representation, and reasoning methods.
2. **Develop the ability to model and solve computational problems** using uninformed and informed search algorithms, heuristic methods, and logical inference techniques.
3. **Understand the concepts of fuzzy sets, fuzzy logic, and fuzzy inference systems**, and apply them to represent and solve problems involving uncertainty and imprecision.
4. **Learn the principles of Artificial Neural Networks (ANNs)**, including different learning paradigms, neural network architectures, and their applications in pattern recognition and data analysis.
5. **Understand the concepts and applications of Genetic Algorithms (GAs)** for solving optimization and search problems using evolutionary computation techniques.
6. **Acquire practical skills in implementing soft computing techniques** using programming languages such as C, Java, or Python for solving real-world intelligent computing problems.

Course Outcomes (CO)

Upon successful completion of the course, students will be able to:

- Explain the fundamental concepts of Artificial Intelligence, formulate state-space representations, and apply search and reasoning techniques to solve AI problems.
- Analyze and compare uninformed and informed search algorithms, evaluate heuristic functions, and select appropriate search strategies for different problem domains.
- Model uncertainty using fuzzy sets and fuzzy logic, design membership functions, perform fuzzification and defuzzification, and develop basic fuzzy inference systems.
- Apply neural network learning algorithms, including perceptron learning, backpropagation, Hebbian learning, SOM, Hopfield, and BAM networks, to pattern recognition and classification problems.
- Apply Genetic Algorithms to optimization problems by implementing chromosome representation, selection, crossover, mutation, and evaluating evolutionary search performance.
- Design, implement, and evaluate soft computing solutions using appropriate programming tools (C, Java, or Python) for real-world applications involving intelligent decision-making, classification, optimization, and pattern recognition.

Theory Syllabus
Module-I: Fundamentals of Artificial Intelligence (10 Hrs)
Introduction:

Introduction to Artificial Intelligence, History and Applications of AI

Problem solving using AI approach:

AI as a search problem, State space representation, Water-jug problem, missionary carnival problem, 8-Puzzel problem, vacuum-cleaner problem, travelling-salesman problem, N-Queen problem, Wampus world problem.

Predicate Logic in AI:

First Order Predicate Logic and its use in knowledge representation, Resolution Principle, Use of Resolution in reasoning and question answering, Inference Rules.

Search Techniques:

Uninformed Search Techniques (Breadth First Search (BFS), Depth First Search (DFS), Uniform Cost Search (UCS)); Informed Search Techniques (Heuristic function, Greedy Best-First Search, A* Search Algorithm, Evaluation functions, Admissible heuristic).

Module-II: Fuzzy Logic (15 Hrs)
Introduction:

Introduction to fuzzy sets and fuzzy logic systems.

Classical Sets and Fuzzy Sets and Fuzzy relations :

Operations on Classical sets, properties of classical sets, Fuzzy set operations, properties of fuzzy sets, cardinality, operations, and properties of fuzzy relations.

Membership functions:

Features of membership functions, standard forms and boundaries, different fuzzification methods.

Fuzzy to Crisp conversions:

Lambda Cuts for fuzzy sets, fuzzy Relations, Defuzzification methods.

Classical Logic and Fuzzy Logic:

Classical predicate logic, Fuzzy Logic, Approximate reasoning and Fuzzy Implication.

Fuzzy Rule based Systems:

Linguistic Hedges, Fuzzy Rule based system – Aggregation of fuzzy Rules, Fuzzy Inference System, Mamdani Fuzzy Models – Takagi-Sugeno Fuzzy Models.

Module-III: Neural Networks (15 Hrs)
Introduction to Neural Networks:

Advent of Modern Neuroscience, Classical AI and Neural Networks, Biological Neurons and Artificial neural network; model of artificial neuron.

Learning Methods:

Hebbian, competitive, Boltzman.

Neural Network models:

Perceptron, Adaline and Madaline networks; single layer network; Delta rule and back-propagation and multi layer networks.

Competitive learning networks:

Kohonen self organizing networks (SOM),Hebbian learning; Hopfield Networks, Bidirectional Associative Memory(BAM).

Module-IV: Genetic Algorithms (5 Hrs)
Introduction:

Introduction to Genetic Algorithm.

Genetic Algorithms:

Simple GA, crossover and mutation, Multi-objective Genetic Algorithm (MOGA).

Text Books:

1. Fuzzy logic with engineering applications, Timothy J. Ross, John Wiley and Sons.
2. S. Rajasekaranand G.A.V.Pai,"Neural Networks, Fuzzy Logic and Genetic Algorithms", PHI
3. Principles of Soft Computing , SN Sivanandam, S. Sumathi, John Wiley & Sons
4. Genetic Algorithms in search, Optimization& Machine Learning by David E. Goldberg
5. Neuro-Fuzzy and Soft computing, Jang, Sun, Mizutani, PHI
6. Neural Networks: A ClassroomApproach,1/eby Kumar Satish, TMH,
7. Genetic Algorithms in search, Optimization & Machine Learning by David E. Goldberg, Pearson/PHI
8. A beginners approach to Soft Computing, Samir Roy &Udit Chakraborty,
9. Soft Computing: Fundamentals and Applications, by D.K. Pratihar.

Soft Computing (Tutorials)
Credit: 1 **15 Hrs**
Syllabus
Module-I Tutorials: Fundamentals of Artificial Intelligence

- Identify AI applications in different domains (healthcare, finance, education, agriculture, transportation).
- Solve state-space search problems (Water Jug Problem, Missionaries and Cannibals, 8-Puzzle).
- Compare BFS, DFS, Uniform Cost Search, Greedy Best-First Search, and A* Search using example search trees.
- Represent real-world knowledge using predicate logic.

Module-II Tutorials: Fuzzy Logic

- Solve numerical problems on fuzzy sets and fuzzy operations.
- Design membership functions for real-life variables.
- Perform fuzzification and defuzzification.
- Construct simple fuzzy inference systems.
- Compare classical logic and fuzzy logic using examples.

Module-III Tutorials: Neural Networks

- Solve numerical problems on perceptron learning.
- Demonstrate Hebbian learning.
- Train a simple single-layer neural network manually.
- Compare Perceptron, Adaline, and Madaline.
- Analyze applications of SOM, Hopfield, and BAM networks.

Module-IV Tutorials: Genetic Algorithms

- Simulate chromosome encoding.
- Perform selection, crossover, and mutation manually.
- Solve a simple optimization problem using GA.
- Compare GA with traditional optimization methods.

Soft Computing (Practical)
Credit: 2 **60 Hrs**
Syllabus
All programs should be implemented using C / Java / Python.
Practical exercises based on concepts listed in theory, such as:

1. Applying an ANN to traina model
2. Apply a Kohonen SOM
3. Apply a Hopfield network to memorize and rectify patterns
4. Apply BAM to memorize
5. Apply Genetic Algorithms to solve an optimization problem
6. Applying ANN for pattern recognition tasks

PG2-COMP-1021: Database Management Systems

Course Objective:

A Database Management System (DBMS) course aims to provide a comprehensive understanding of how to design, implement, and manage databases effectively. Core objectives include mastering relational models, constructing SQL queries, applying normalization to reduce redundancy, and managing transactions and concurrency control for data integrity.

Course Learning Outcomes:

After undergoing the course, Students will be able to:

- Differentiate between traditional file systems and the database approach, explaining how the three-schema architecture provides data independence.
- Construct Entity-Relationship (ER) models to capture complex business requirements, defining entity sets, attributes, and relationship constraints.
- Execute complex data retrieval and manipulation tasks using Relational Algebra and industry-standard SQL (Structured Query Language).
- Transform ER models into relational schemas and apply Normalization (up to BCNF) to eliminate redundancy and ensure data integrity through functional dependency analysis.

Credit: 03 45
Hours

1.Introduction (6 Hrs)

Characteristics of database approach, data models, database system architecture and data independence.

2.Entity Relationship (ER) Modeling (8 Hrs)

Entity types, relationships, constraints.

3.Relation data model (20 Hrs)

Relational model concepts, relational constraints, relational algebra, SQL queries

4. Database design (15 Hrs)

Mapping ER/EER model to relational database, functional dependencies, Lossless decomposition, Normal forms (upto BCNF).

5. Transaction Processing (3 Hrs)

ACID properties, concurrency control & recovery

6. File Structure and Indexing (8 Hrs)

Operations on files, File of Unordered and ordered records, overview of File organizations, Indexing structures for files(Primary index, secondary index, clustering index), Multilevel indexing using B and B+ trees.

References Book :

1. R. Elmasri, S.B. Navathe, Fundamentals of Database Systems 6th Edition, Pearson Education, 2010.
2. R. Ramakrishanan, J. Gehrke, Database Management Systems 3rd Edition, McGraw-Hill, 2002.
3. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts 6th Edition, McGrawHill, 2010.
4. R. Elmasri, S.B. Navathe Database Systems Models, Languages, Design and application Programming, 6th Edition, Pearson Education, 2013.

Database Management System (Practical)

Credit: 01 30 Hours

Create and use the following database schema to answer the given queries.

EMPLOYEE Schema

Field	Type	NULL	DEFAULT
Eno	Char(3)	NO	NIL
Ename	Varchar(50)	NO	NIL
Job_type	Varchar(50)	NO	NIL
Manager	Char(3)	Yes	NIL
Hire_date	Date	NO	NIL
Dno	Integer	YES	NIL
Commission	Decimal(10,2)	YES	NIL
Salary	Decimal(7,2)	NO	NIL

DEPARTMENT Schema

Field	Type	NULL	KEY	DEFAULT
Dno	Integer	No	PRI	NULL
Dname	Varchar(50)	Yes		NULL
Location	Varchar(50)	Yes		New Delhi

Query List

5. Query to display Employee Name, Job, Hire Date, Employee Number; for each employee with the Employee Number appearing first.
6. Query to display unique Jobs from the Employee Table.
7. Query to display the Employee Name concatenated by a Job separated by a comma.
8. Query to display all the data from the Employee Table. Separate each Column by a comma and name the said column as THE_OUTPUT.
9. Query to display the Employee Name and Salary of all the employees earning more than $2850.
10. Query to display Employee Name and Department Number for the Employee No= 7900.
11. Query to display Employee Name and Salary for all employees whose salary is not in the range of $1500 and $2850.
12. Query to display Employee Name and Department No. of all the employees in Dept 10 and Dept 30 in the alphabetical order by name.
13. Query to display Name and Hire Date of every Employee who was hired in 1981.
14. Query to display Name and Job of all employees who don,,t have a current Manager.
15. Query to display the Name, Salary and Commission for all the employees who earn commission.
16. Sort the data in descending order of Salary and Commission.
17. Query to display Name of all the employees where the third letter of their name is _A,,.
18. Query to display Name of all employees either have two _R,,s or have two _A,,s in their name and are either in Dept No = 30 or their Manger,,s Employee No = 7788.
19. Query to display Name, Salary and Commission for all employees whose Commission Amount is 14 greater than their Salary increased by 5%.
20. Query to display the Current Date.
21. Query to display Name, Hire Date and Salary Review Date which is the 1st Monday after six months of employment.
22. Query to display Name and calculate the number of months between today and the date each employee was hired.
23. Query to display the following for each employee <E-Name> earns < Salary> monthly but wants < 3 * Current Salary >. Label the Column as Dream Salary.
24. Query to display Name with the 1st letter capitalized and all other letter lower case and length of their name of all the employees whose name starts with _J,,, ,,A,, and _M,,.

25. Query to display Name, Hire Date and Day of the week on which the employee started.
26. Query to display Name, Department Name and Department No for all the employees.
27. Query to display Unique Listing of all Jobs that are in Department # 30.
28. Query to display Name, Dept Name of all employees who have an _A,, in their name.
29. Query to display Name, Job, Department No. And Department Name for all the employees working at the Dallas location.
30. Query to display Name and Employee no. Along with their Mangers Name and the Managers employee no; along with the Employees,, Name who do not have a Manager.
31. Query to display Name, Dept No. And Salary of any employee whose department No. and salary matches both the department no. And the salary of any employee who earns a commission.
32. Query to display Name and Salaries represented by asterisks, where each asterisk (*) signifies $100.
33. Query to display the Highest, Lowest, Sum and Average Salaries of all the employees
34. Query to display the number of employees performing the same Job type functions.
35. Query to display the no. of managers without listing their names.
36. Query to display the Department Name, Location Name, No. of Employees and the average salary for all employees in that department.
37. Query to display Name and Hire Date for all employees in the same dept. as Blake.
38. Query to display the Employee No. And Name for all employees who earn more than the average salary.
39. Query to display Employee Number and Name for all employees who work in a department with any employee whose name contains a "T"
40. Query to display the names and salaries of all employees who report to King.
41. Query to display the department no, name and job for all employees in the Sales department

PG2-COMP-2011: Network Security & Cryptography

Course Objectives

The core objectives of a Network Security & Cryptography course are to provide a foundational understanding of data protection, enabling students to understand, analyze, and implement cryptographic algorithms, security protocols (SSL/TLS, IPSec), and defense mechanisms against network attacks. The course focuses on achieving confidentiality, integrity, availability, and non-repudiation.

Course Learning Outcomes

After undergoing the course, students will be able to:

- Understand the fundamentals of Cryptography and Network Security.
- Secure information transmitted over insecure communication channels using appropriate cryptographic techniques.
- Apply cryptographic algorithms to ensure confidentiality, integrity, authenticity, and availability of data.
- Analyze and implement public-key cryptographic techniques, digital signatures, and key management mechanisms.
- Understand and apply network security protocols to protect computer networks against various security threats.
- Implement and evaluate practical cryptographic algorithms and network security mechanisms using appropriate software tools.

Theory Syllabus (45 Hours)

Unit I: Introduction to Security (5 Hours)

Introduction to security, brief history of cryptography, understanding attacks, services, mechanisms, security attacks, security services, model for network security, internet standards, basic principles of good cryptosystems, modular arithmetic and GF(2), substitution and transposition ciphers, stream ciphers, pseudo-random numbers.

Unit II: Symmetric Key Cryptography (5 Hours)

Symmetric block ciphers, DES, AES, IDEA principles of S-Box design, block cipher design principles, other block ciphers.

Unit III: Public Key Cryptography (9 Hours)

Introduction to public key cryptography, number theoretic foundations, public key cryptography principles, RSA encryption system, primality testing, number theoretic algorithms.

Unit IV: Discrete Logarithm and Digital Signatures (8 Hours)

Discrete logarithm and Diffie-Hellman key exchange, ElGamal cryptosystem, digital signatures, RSA-based digital signatures, DSA, attacks on digital signatures.

Unit V: Hash Functions and Key Management (5 Hours)

Secure hash functions, understanding collisions, Secure Hash Algorithms (SHA), HMAC, key

establishment and key management, digital certificates.

Unit VI: Advanced Cryptography (8 Hours)

Distribution of primes, lattice-based cryptography, introduction to elliptic curves, introduction to elliptic curve cryptography.

Unit VII: Network Security Protocols (3 Hours)

Authentication, e-mail security, Kerberos, X.509 directory authentication services, PGP, S/MIME, overview of IP security, IP security architecture, authentication header, encapsulating security payload, combining security associations.

Unit VIII: Web and System Security (2 Hours)

Introduction to web security, web security requirements, SSL/TLS, Secure Electronic Transaction (SET), network management security, basics of system security, intruders, virus-related threats, firewalls, firewall design principles, trusted systems.

Text / Reference Books

1. William Stallings, *Cryptography and Network Security: Principles and Practice*, Pearson Education.
2. William Stallings, *Network Security Essentials: Applications and Standards*, Pearson Education.
3. Charlie Kaufman, Radia Perlman, and Mike Speciner, *Network Security: Private Communication in a Public World*, Pearson Education.
4. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman, *An Introduction to Mathematical Cryptography*, 2nd Edition, Springer.
5. Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, 3rd Edition, CRC Press.
6. Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons.
7. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press.
8. Christof Paar and Jan Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer.
9. Neal Koblitz, *A Course in Number Theory and Cryptography*, 2nd Edition, Springer.
10. Alfred Menezes, *Elliptic Curve Public Key Cryptosystems*, Springer.

PG2-COMP-2012: Optimization Techniques

Total Credits: 4
Total Contact Hours: 75 Hours
Course Objectives
The course aims to develop an understanding of the fundamental concepts, mathematical models and solution techniques used in operations research and optimization. It introduces linear programming, transportation and assignment models, network analysis, project scheduling, inventory control, decision analysis, game theory and queuing theory.
The lecture component provides the theoretical and mathematical foundations of operations-research methods. The tutorial component develops problem-formulation and analytical skills through structured numerical exercises. The practical component provides hands-on experience in implementing optimization algorithms and solving real-life decision-making problems using suitable programming languages, spreadsheets and optimization software.
The course enables students to translate practical managerial, industrial, engineering and computational problems into mathematical models and apply appropriate optimization techniques to obtain efficient and logically justified solutions.
Course Learning Outcomes
On successful completion of the course, the students will be able to achieve the following outcomes.

- Formulate real-life decision-making problems as linear-programming models by identifying decision variables, objective functions and constraints.
- Solve linear-programming problems using graphical, simplex and Big-M methods.
- Explain and apply the concepts of feasible solutions, basic feasible solutions, basic and non-basic variables, degeneracy, convex sets and duality.
- Formulate and solve integer-programming, transportation and assignment problems using appropriate techniques.
- Apply shortest-path and maximum-flow algorithms to network-optimization problems.
- Use PERT and CPM techniques for project planning, scheduling and network analysis.
- Apply inventory-control and decision-analysis techniques to managerial and operational problems.
- Analyse two-person zero-sum games using saddle-point, dominance, graphical, maximin and minimax methods.
- Explain the basic concepts of queuing systems and solve standard Poisson queue models.
- Select, implement and evaluate suitable operations-research techniques for solving optimization and decision-making problems.

Lecture
Lecture Credit: 2
Total Lecture Hours: 30 Hours
Unit 1: Linear Programming and Related Optimization Models
Introduction and Mathematical Formulation
This subsection introduces operations research, its scope, characteristics, methodology and applications in business, industry, engineering, computing and management. It explains the basic concepts of optimization and the components of a linear-programming problem. The formulation of linear-programming models through decision variables, objective functions, constraints and non-negativity restrictions is covered with suitable examples.
Graphical Solution of Linear-Programming Problems
This subsection explains the graphical solution of linear-programming problems involving two decision variables. It covers the representation of constraints, identification of feasible regions, corner-point evaluation and determination of optimal solutions. Special cases involving multiple optimal solutions, unbounded solutions, infeasible solutions and redundant constraints are also discussed.
Fundamental Concepts of Linear Programming
This subsection introduces feasible solutions, basic solutions, basic feasible solutions, basic and non-basic variables, slack variables, surplus variables and artificial variables. Degenerate and non-degenerate solutions are explained. Convex sets, convex combinations, extreme points and their significance in linear programming are discussed with suitable examples.

Simplex and Big-M Methods
This subsection covers the standard and canonical forms of linear-programming problems. The simplex algorithm, simplex table, entering variable, leaving variable, pivot element, optimality condition and iteration procedure are discussed. Charnes' Big-M method is introduced for solving problems involving equality constraints and greater-than-or-equal-to constraints. Cases involving degeneracy, unboundedness and infeasibility are also considered.
Duality Theory
This subsection introduces the concept of duality and the formulation of the dual of a given primal problem. It covers the relationships between primal and dual variables, constraints and objective functions. Weak duality, strong duality and complementary slackness are introduced. The economic interpretation of dual variables and their use in decision-making are also discussed.
Integer Programming
This subsection introduces pure and mixed-integer programming problems and their applications. The need for integer restrictions is explained. The basic concepts of the branch-and-bound method and cutting-plane approaches are introduced for solving simple integer-programming problems.
Transportation Problems
This subsection introduces the mathematical formulation of transportation problems. Balanced and unbalanced transportation models, transportation tables and feasible allocations are discussed. Initial basic feasible solutions using the North-West Corner Rule, Least-Cost Method and Vogel's Approximation Method are covered. Optimality testing and improvement using the MODI method or stepping-stone method are explained. Degeneracy, prohibited routes and maximization variants are also introduced.
Assignment Problems
This subsection explains the formulation of assignment problems as special transportation problems. The Hungarian method is discussed for minimization and maximization problems. Unbalanced assignment problems, restricted assignments and travelling-salesperson-type variations are introduced. (13 Hrs)
Unit 2: Network Analysis, Project Scheduling, Inventory Control and Decision Analysis
Network-Optimization Problems
This subsection introduces networks, nodes, arcs, directed networks, undirected networks, paths, cycles, capacities and costs. Mathematical representations of network problems and their applications in transportation, communication, routing and resource distribution are discussed.
Shortest-Path Problems
This subsection covers single-source and all-pairs shortest-path problems. The principles and implementation steps of Dijkstra's algorithm for single-source shortest paths and Floyd's algorithm for all-pairs shortest paths are explained. Suitable numerical examples and network interpretations are included.
Maximum-Flow Problems
This subsection introduces flow networks, source nodes, sink nodes, arc capacities, feasible flows and flow conservation. The maximal-flow and maximum-flow concepts are explained. The Ford-Fulkerson algorithm, augmenting paths, residual capacities and termination conditions are discussed through suitable problems.
PERT and CPM
This subsection introduces project-network analysis, activities, events, precedence relationships and network construction. Activity-on-arrow and activity-on-node representations are discussed. Forward-pass and backward-pass calculations, earliest and latest event times, activity floats and critical-path identification are covered.
The Programme Evaluation and Review Technique is discussed using optimistic, most likely and pessimistic time estimates. Expected activity time, variance, project completion time and completion probability are included. Cost analysis, project crashing and resource allocation are excluded from the syllabus.
Fundamentals of Inventory Control
This subsection introduces inventory systems, inventory costs and the need for inventory control. Ordering cost, carrying cost, shortage cost and purchase cost are discussed. The Economic Order Quantity model without shortages, reorder level and lead time are covered. Basic deterministic inventory models are introduced.
Decision Analysis
This subsection introduces decision-making under certainty, risk and uncertainty. Payoff tables, opportunity-loss tables, expected monetary value, expected opportunity loss and expected value of

perfect information are discussed. Decision criteria including maximax, maximin, minimax regret, Hurwicz and Laplace criteria are introduced. (7 Hrs)

Unit 3: Game Theory

Basic Concepts of Game Theory

This subsection introduces competitive decision-making, players, strategies, pure strategies, mixed strategies, payoff matrices and the value of a game. The assumptions and applications of two-person zero-sum games are discussed.

Saddle Point and Pure-Strategy Games

This subsection explains row minima, column maxima, lower value, upper value and saddle points. The maximin and minimax principles are discussed. Statements and interpretations of the minimax and maximin theorems are included. Games possessing saddle points are solved using pure strategies.

Games without Saddle Points

This subsection introduces mixed strategies for games without saddle points. The formulation and solution of two-by-two games using algebraic methods are explained. The determination of optimal strategy probabilities and the value of the game is covered.

Principle of Dominance

This subsection explains row dominance, column dominance and modified dominance. The principle of dominance is applied to reduce larger payoff matrices into simpler equivalent games. The solution of reduced games is illustrated through suitable numerical problems.

Graphical Solution of Games

This subsection covers the graphical solution of two-by-(n) and (m)-by-two games. The construction of payoff lines, identification of the lower envelope or upper envelope and determination of optimal mixed strategies are explained.

Linear-Programming Formulation of Games

This subsection introduces the representation of two-person zero-sum games as linear-programming problems. The relationship between the game formulations of the two players and the duality principle is discussed. (7 Hrs)

Unit 4: Queuing Theory

Fundamentals of Queuing Systems

This subsection introduces queuing theory and its applications. It covers customers, servers, arrival processes, service processes, queue discipline, system capacity, calling population and service channels. Kendall's notation and the classification of queuing models are introduced.

Arrival and Departure Processes

This subsection discusses stochastic arrival and departure processes. The assumptions underlying Poisson arrivals and exponential service times are explained. The axiomatic derivation and fundamental properties of the Poisson process are introduced.

Infinite-Capacity Single-Server Model

This subsection covers the (M/M/1:(\infty/FIFO)) queuing model. It explains the steady-state probability distribution, probability of an empty system, average number of customers in the system, average queue length, average waiting time in the system, average waiting time in the queue and server utilization. Little's formula and related numerical problems are included.

Finite-Capacity Single-Server Model

This subsection introduces the (M/M/1:(N/FIFO)) model. Steady-state probabilities, effective arrival rate, average number of customers, expected waiting times and blocking probabilities are discussed. Related numerical problems are solved. (3 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Linear-Programming Formulation and Graphical Solution

This tutorial unit includes the identification of decision variables, objective functions and constraints from verbal problem statements. Students will formulate production-planning, product-mix, resource-allocation, diet, blending and scheduling problems as linear-programming models. Graphical solutions involving feasible regions, corner points, optimal solutions, infeasible solutions, unbounded solutions and alternative optimal solutions will be practised. (2 Hrs)

Tutorial Unit 2: Simplex Method, Big-M Method and Duality
This tutorial unit includes the conversion of linear-programming problems into standard form, introduction of slack, surplus and artificial variables and preparation of the initial simplex table. Students will perform simplex iterations and solve maximization and minimization problems. Problems requiring Charnes' Big-M method will be solved. Students will formulate dual problems, establish primal-dual relationships and verify complementary slackness for suitable cases. (3 Hrs)
Tutorial Unit 3: Integer Programming, Transportation and Assignment Problems
This tutorial unit includes simple branch-and-bound exercises for integer-programming problems. Transportation problems will be solved using the North-West Corner Rule, Least-Cost Method and Vogel's Approximation Method. Optimality will be examined using the MODI method. Balanced, unbalanced, degenerate, prohibited-route and maximization transportation problems will be practised.
Assignment problems will be solved using the Hungarian method. Exercises will include minimization, maximization, unbalanced assignments, restricted assignments and simple travelling-salesperson variations. (3 Hrs)
Tutorial Unit 4: Network Analysis and Project Scheduling
This tutorial unit includes the representation of practical problems as networks. Students will solve single-source and all-pairs shortest-path problems using suitable algorithms. Maximum-flow problems will be solved using augmenting paths and the Ford-Fulkerson method.
Students will construct project networks from precedence data and calculate earliest event times, latest event times, total float, free float and independent float. Critical paths and expected project-completion times will be determined. PERT exercises involving expected duration, activity variance and probability of project completion will also be undertaken. (3 Hrs)
Tutorial Unit 5: Inventory Control and Decision Analysis
This tutorial unit includes numerical problems on ordering cost, carrying cost, total inventory cost, Economic Order Quantity, reorder level and lead time. Students will prepare payoff and opportunity-loss tables and apply expected monetary value, expected opportunity loss and expected value of perfect information. Decision-making problems under uncertainty will be solved using maximax, maximin, minimax regret, Hurwicz and Laplace criteria. (1 Hrs)
Tutorial Unit 6: Game Theory
This tutorial unit includes the identification of saddle points and the solution of pure-strategy games using maximin and minimax principles. Students will solve two-by-two games without saddle points using algebraic methods. Larger games will be simplified using dominance. Two-by-(n) and (m)-by-two games will be solved using the graphical method. Suitable games will also be formulated as linear-programming problems. (2 Hrs)
Tutorial Unit 7: Queuing Models
This tutorial unit includes the identification of arrival rate, service rate, utilization factor and steady-state conditions. Numerical problems and models will be solved. Students will calculate queue length, system length, waiting time, utilization, idle probability and blocking probability. (1 Hrs)

Practical
Practical Credit: 1
Hours per Practical Credit: 30 Hours
Total Practical Hours: 30 Hours
The practical component may be conducted using Python, R, MATLAB, GNU Octave, Microsoft Excel Solver, LibreOffice Calc Solver, LINGO, LINDO, MATLAB Optimization Toolbox or any other suitable programming and optimization environment. Students should interpret the results produced by the software rather than treating the software only as a computational tool.
Practical 1: Formulation and Graphical Solution of Linear-Programming Problems
Students will formulate a real-life production, allocation or product-mix problem as a linear-programming model. A program or spreadsheet model will be developed to plot the constraints, identify the feasible region, evaluate corner points and determine the optimal solution. (3 Hrs)
Practical 2: Implementation of the Simplex Method
Students will implement the simplex method or use a suitable optimization library to solve a linear-programming problem. The program will display the initial table, pivot operations, successive iterations, values of basic variables and the optimal objective value. (3 Hrs)
Practical 3: Big-M Method and Duality Analysis
Students will solve linear-programming problems involving greater-than-or-equal-to and equality constraints using the Big-M method. The corresponding dual problem will be formulated and solved.

Primal and dual objective values and complementary-slackness conditions will be compared. (3 Hrs)

Practical 4: Integer-Programming Model

Students will formulate and solve an integer-programming problem such as capital budgeting, facility selection, project selection or workforce scheduling. A branch-and-bound-based solver or suitable optimization library will be used, and the integer solution will be compared with the relaxed linear-programming solution. (3 Hrs)

Practical 5: Transportation and Assignment Problems

Students will implement or use suitable software to solve transportation problems using initial-allocation methods and optimality testing. A separate assignment model will be solved using the Hungarian method. Balanced, unbalanced and restricted cases will be examined. (3 Hrs)

Practical 6: Shortest-Path Algorithms

Students will implement a single-source shortest-path algorithm and Floyd's all-pairs shortest-path algorithm. The program will accept a weighted network as input and generate the shortest distances and corresponding paths. (3 Hrs)

Practical 7: Maximum-Flow Algorithm

Students will implement the Ford-Fulkerson algorithm or an equivalent augmenting-path algorithm for a flow network. The implementation will display residual capacities, augmenting paths and the final maximum flow. (3 Hrs)

Practical 8: PERT and CPM Network Analysis

Students will develop a program or spreadsheet model for project-network analysis. The model will calculate earliest start, earliest finish, latest start, latest finish, activity floats and the critical path. For PERT data, expected completion time, variance and probability of completion within a specified duration will be calculated. (3 Hrs)

Practical 9: Inventory and Decision-Analysis Models

Students will develop computational models for Economic Order Quantity, reorder level and total inventory cost. A decision-analysis model will also be prepared to evaluate alternatives using expected monetary value, expected opportunity loss and expected value of perfect information. (3 Hrs)

Practical 10: Game-Theory and Queuing Models

Students will develop a program or spreadsheet model to determine saddle points and solve simple two-person zero-sum games. Dominance or mixed-strategy calculations will be applied where required. (3 Hrs)

Reference Books

Primary References

1. Hadley, G., *Linear Programming*, Narosa Publishing House, Reprint, 2002.
2. Taha, H. A., *Operations Research: An Introduction*, 9th Edition, Prentice Hall, 2010.

Additional References

1. Hadley, G., *Linear Algebra*, Narosa Publishing House, Reprint, 2002.
2. Ravindran, A., Phillips, D. T. and Solberg, J. J., *Operations Research: Principles and Practice*, John Wiley & Sons, 2005.
3. Hillier, F. S. and Lieberman, G. J., *Introduction to Operations Research: Concepts and Cases*, 9th Edition, Tata McGraw-Hill, 2010.

PG2-COMP-2013: Date Science

Total Credits: 4

Total Contact Hours: 75 Hours

Course Objectives

The course aims to provide students with a comprehensive understanding of the fundamental concepts, processes, tools and techniques used in data science. It introduces data collection, preprocessing, statistical analysis, probability, regression, hypothesis testing, feature engineering, data visualization, machine learning, clustering, database systems, big-data technologies and deep learning.

The lecture component develops the conceptual and theoretical foundations of data science. The tutorial component strengthens students' analytical and problem-solving abilities through numerical exercises, statistical interpretation, algorithmic analysis and case-based discussions. The practical component provides hands-on experience in collecting, cleaning, analysing, visualizing and modelling data using suitable programming languages, database systems and data-science tools.

The course also enables students to formulate and solve real-life data-analysis problems in areas such as bioinformatics, computer vision, image processing, natural-language processing and social-network analysis.

Course Learning Outcomes

On successful completion of the course, the students will be able to achieve the following outcomes.

- Explain the fundamental concepts, processes and life cycle of a data-science project.
- Collect, clean, preprocess, transform, store and organize data using appropriate methods and tools.
- Calculate and interpret descriptive statistical measures, including frequency, central tendency, dispersion, percentiles and quartiles.
- Apply probability concepts, probability distributions, Bayes' theorem and regression models to data-analysis problems.
- Perform hypothesis testing using appropriate parametric and non-parametric statistical methods.
- Apply feature-extraction, feature-selection, normalization and dimensionality-reduction techniques.
- Create and interpret appropriate graphs and charts for effective data visualization.
- Apply supervised and unsupervised machine-learning algorithms to classification and clustering problems.
- Evaluate classification models using suitable performance measures, including Receiver Operating Characteristic curves and Area Under the Curve.
- Explain database-management systems, SQL, data warehousing and big-data technologies such as Hadoop, MapReduce, NoSQL and Apache Spark.
- Describe the fundamental concepts of artificial neural networks, deep learning, Convolutional Neural Networks and Recurrent Neural Networks.
- Develop data-driven solutions for real-life applications using suitable statistical, machine-learning and deep-learning techniques.

Lecture

Lecture Credit: 2

Total Lecture Hours: 30 Hours

Unit 1: Introduction to Data Science and Descriptive Statistics

Fundamentals of Data Science

This subsection introduces the definition, scope and applications of data science. It explains the relationship among data science, statistics, data analytics, machine learning, artificial intelligence, knowledge discovery and data mining. Different types of data, including structured, semi-structured and unstructured data, are discussed. Qualitative, quantitative, discrete, continuous, nominal, ordinal, interval and ratio data are also introduced.

Data Collection and Preparation

This subsection covers primary and secondary data collection, observational data, experimental data, survey data, transactional data and sensor-generated data. Data cleaning techniques for handling missing values, duplicate records, inconsistent values, noise and outliers are discussed. Data preprocessing, transformation, integration, reduction, encoding and normalization are introduced.

Data Engineering and Project Life Cycle

This subsection explains Extract, Transform and Load techniques, data storage, data warehousing and data wrangling. The stages of a data-science project life cycle, including problem definition, data collection, data preparation, exploratory analysis, modelling, evaluation, deployment and monitoring, are discussed.

Descriptive Statistics

This subsection covers frequency distributions, counts, percentages and tabular summaries. Measures of central tendency, including arithmetic mean, median and mode, are explained. Measures of dispersion, including range, variance, standard deviation and interquartile range, are discussed. Measures of position, including percentile ranks, quartile ranks and z-scores, are also introduced. (5 Hrs)

Unit 2: Probability and Regression Models

Fundamentals of Probability

This subsection introduces random experiments, sample spaces, events, mutually exclusive events, exhaustive events and independent events. The addition and multiplication laws of probability are

discussed. Joint probability, marginal probability and conditional probability are explained with suitable examples.

Bayes' Theorem and Probability Distributions

This subsection introduces Bayes' theorem and its applications in classification and decision-making. Random variables and probability distributions are discussed. Basic discrete and continuous probability distributions, including Bernoulli, binomial, Poisson, uniform and normal distributions, are introduced.

Regression Models

This subsection explains the purpose and applications of regression analysis. Simple linear regression, dependent and independent variables, regression coefficients, least-squares estimation, residuals and goodness of fit are discussed. The interpretation of regression models and their role in prediction are included. A brief introduction to multiple regression and logistic regression is also provided. (3 Hrs)

Unit 3: Hypothesis Testing and Statistical Analysis

Fundamentals of Hypothesis Testing

This subsection introduces population, sample, parameter, statistic, sampling distribution and standard error. Null and alternative hypotheses, levels of significance, confidence levels and p-values are explained. One-tailed and two-tailed tests are discussed.

Statistical Tests

This subsection covers the t-test for suitable one-sample, independent-sample and paired-sample situations. Analysis of Variance is introduced for comparing the means of multiple groups. The Wilcoxon signed-rank test is discussed as a non-parametric alternative for paired observations. Spearman's rank-order correlation is introduced for measuring monotonic relationships between variables.

Statistical Errors and Sampling

This subsection explains Type I and Type II errors, statistical power and confidence intervals. Probability and non-probability sampling methods are discussed. Simple random, systematic, stratified and cluster sampling techniques are introduced. Data normalization and the interpretation and reporting of statistical results are also covered. (3 Hrs)

Unit 4: Feature Engineering and Data Visualization

Feature Engineering

This subsection introduces features and their role in data-science and machine-learning models. Feature extraction, feature construction, feature transformation and feature selection are discussed. Filter, wrapper and embedded feature-selection methods are introduced. Techniques for handling categorical variables, missing values and outliers are also included.

Data Scaling and Dimensionality Reduction

This subsection covers min-max normalization, standardization and robust scaling. The need for dimensionality reduction is explained. Principal Component Analysis is introduced through covariance, eigenvalues, eigenvectors, principal components and explained variance. The benefits and limitations of dimensionality reduction are discussed.

Data Visualization

This subsection introduces the principles of effective data visualization. Bar charts, line charts, pie charts, histograms, box plots, scatter plots, heat maps and correlation matrices are covered. Selection of suitable visual representations for categorical, numerical, temporal and multivariate data is discussed. Visual clarity, labelling, scale selection and avoidance of misleading representations are emphasized. (4 Hrs)

Lecture Hours: 4 Hours

Unit 5: Machine Learning and Classification

Introduction to Machine Learning

This subsection introduces machine learning and its relationship with data science. Supervised, unsupervised and semi-supervised learning are discussed. The concepts of training data, validation data, test data, features, labels, model fitting, prediction and generalization are explained. Overfitting, underfitting, bias and variance are introduced.

Classification Concepts

This subsection explains binary and multiclass classification. The confusion matrix, accuracy, precision, recall, specificity, sensitivity and F1-score are discussed. Receiver Operating Characteristic curves, decision thresholds and Area Under the Curve are introduced for model evaluation. Cross-validation and train-test splitting are also covered.

K-Nearest Neighbours

This subsection explains the working principle of the K-Nearest Neighbours algorithm. Distance measures, selection of the value of (K), feature scaling, classification procedure, advantages and limitations are discussed.

Decision Trees and Random Forests
This subsection introduces decision-tree structure, root nodes, internal nodes, branches and leaf nodes. Attribute-selection measures such as entropy, information gain and Gini impurity are discussed. Tree pruning and overfitting are introduced. Random Forests, bootstrap sampling, random feature selection and ensemble prediction are explained.
Bayes Classifier
This subsection introduces probabilistic classification using Bayes' theorem. The Naive Bayes classifier, conditional independence assumption and classification of categorical and continuous data are discussed.
Support Vector Machines
This subsection introduces separating hyperplanes, margins, support vectors and maximum-margin classification. Linear and non-linear Support Vector Machines and the basic concept of kernel functions are covered. Applications, strengths and limitations of Support Vector Machines are discussed. (6 Hrs)
Unit 6: Clustering Techniques
Fundamentals of Clustering
This subsection introduces clustering as an unsupervised-learning technique. Similarity, dissimilarity, distance measures and the objectives of clustering are discussed. Partition-based, hierarchical, density-based and probabilistic clustering approaches are introduced.
K-Means and Fuzzy C-Means Clustering
This subsection explains the K-Means clustering algorithm, centroid initialization, cluster assignment, centroid updating and convergence. Selection of the number of clusters and the within-cluster sum of squares are discussed. Fuzzy C-Means clustering is introduced with the concepts of membership values and fuzzy cluster assignment.
Density-Based and Probabilistic Clustering
This subsection covers the DBSCAN algorithm, including neighbourhood radius, minimum points, core points, border points and noise points. The Expectation-Maximization algorithm is introduced as an iterative approach for probabilistic parameter estimation and clustering.
Hierarchical Clustering
This subsection introduces Agglomerative Hierarchical Clustering, linkage criteria and dendrogram interpretation. The advantages, limitations and applications of different clustering techniques are compared. (3 Hrs)
Unit 7: Database Systems and Big-Data Technologies
Database-Management Systems and SQL
This subsection introduces database-management systems, relational databases, tables, rows, columns, keys and relationships. The basic concepts of database design and data integrity are discussed. Structured Query Language operations for data definition, data insertion, selection, filtering, grouping, joining, updating and deletion are introduced.
Data Warehousing
This subsection explains the purpose and architecture of data warehouses. Operational databases and analytical databases are distinguished. Data marts, fact tables, dimension tables, star schemas and basic Online Analytical Processing concepts are introduced.
Big-Data Technologies
This subsection introduces big data and its major characteristics, including volume, velocity, variety, veracity and value. Hadoop architecture, the Hadoop Distributed File System and MapReduce programming concepts are discussed. NoSQL database categories, including key-value, document, column-family and graph databases, are introduced. Apache Spark, resilient distributed datasets, distributed data processing and basic Spark applications are also discussed. (3 Hrs)
Unit 8: Deep Learning and Data-Science Applications
Artificial Neural Networks and Deep Learning
This subsection introduces biological and artificial neurons, weights, biases, activation functions, layers and feed-forward computation. The concepts of loss functions, training, gradient descent and backpropagation are briefly discussed. The differences between conventional machine learning and deep learning are explained.
Convolutional Neural Networks
This subsection introduces convolution operations, filters, feature maps, pooling layers, fully connected layers and image-classification applications. The importance of Convolutional Neural Networks in image processing and computer vision is discussed.
Recurrent Neural Networks
This subsection introduces sequential data, hidden states and recurrent connections. Basic Recurrent

Neural Networks and their applications in text, speech and time-series analysis are discussed. The limitations of traditional recurrent networks and the basic motivation for Long Short-Term Memory networks are introduced.

Data-Science Case Studies

This subsection examines representative data-science applications in bioinformatics, medical-data analysis, image processing, computer vision, natural-language processing and social-network analysis. The stages of problem formulation, data collection, preprocessing, modelling, evaluation and interpretation are discussed through suitable case studies. (3 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Data Preparation and Descriptive Statistics

This tutorial unit includes the classification of data according to type and measurement scale. Students will solve problems involving frequency tables, percentages, mean, median, mode, range, variance, standard deviation, percentiles, quartiles, interquartile range and z-scores. Exercises on missing-value treatment, duplicate removal, inconsistent data and outlier identification will also be undertaken. (2 Hrs)

Tutorial Unit 2: Probability and Regression

This tutorial unit includes numerical exercises on joint probability, conditional probability, independent events and Bayes' theorem. Students will identify and interpret common probability distributions. Simple linear-regression problems involving regression coefficients, predicted values, residuals and goodness of fit will also be solved. (2 Hrs)

Tutorial Unit 3: Hypothesis Testing and Sampling

This tutorial unit includes the formulation of null and alternative hypotheses, selection of significance levels and interpretation of p-values. Students will solve problems involving t-tests, Analysis of Variance, the Wilcoxon signed-rank test and Spearman's rank correlation. Exercises on Type I and Type II errors, confidence intervals and sampling methods will also be included. (2 Hrs)

Tutorial Unit 4: Feature Engineering and Data Visualization

This tutorial unit includes exercises on feature encoding, data normalization, standardization, feature selection and dimensionality reduction. Students will perform small calculations related to covariance, principal components and explained variance. Suitable graphs and charts will be selected for different types of data, and incorrect or misleading visualizations will be critically examined. (2 Hrs)

Tutorial Unit 5: Classification Algorithms and Model Evaluation

This tutorial unit includes manual exercises on K-Nearest Neighbours, decision-tree splitting, entropy, information gain, Gini impurity and Naive Bayes classification. Students will construct confusion matrices and calculate accuracy, precision, recall, specificity, F1-score and related performance measures. Receiver Operating Characteristic curves and Area Under the Curve will be interpreted for sample classification results. (3 Hrs)

Tutorial Unit 6: Clustering Techniques

This tutorial unit includes manual iterations of the K-Means algorithm, calculation of distances and reassignment of observations to clusters. Students will analyse fuzzy membership values, identify DBSCAN core and border points and interpret hierarchical-clustering dendrograms. The suitability of different clustering algorithms for different datasets will be discussed. (2 Hrs)

Tutorial Unit 7: Database and Big-Data Concepts

This tutorial unit includes relational-schema interpretation and Structured Query Language exercises involving selection, filtering, aggregation and joins. Students will analyse simple Hadoop, MapReduce, NoSQL and Spark use cases and identify appropriate storage and processing technologies for different data volumes and structures. (1 Hrs)

Tutorial Unit 8: Deep-Learning and Application Case Studies

This tutorial unit includes exercises on artificial-neuron computations, activation functions, convolution output dimensions and sequence-processing concepts. Students will analyse case studies from computer vision, natural-language processing, bioinformatics or social-network analysis and identify suitable preprocessing, modelling and evaluation techniques. (1 Hrs)

Part C: Practical Syllabus

Practical Credit: 1

Hours per Practical Credit: 30 Hours

Total Practical Hours: 30 Hours

The practical component may be conducted using Python, R or another suitable programming environment. Appropriate libraries and tools may include NumPy, Pandas, Matplotlib, scikit-learn, SciPy, Statsmodels, TensorFlow, Keras, PyTorch, SQL database systems, Apache Spark or equivalent technologies.

Practical 1: Data Collection, Loading and Inspection

Students will collect or obtain a structured dataset from a suitable source and load it into a data-analysis environment. The dataset will be inspected using its shape, dimensions, attributes, data types, summary information and sample records. Structured, semi-structured and unstructured data formats will be examined. (3 Hrs)

Practical 2: Data Cleaning and Preprocessing

Students will identify and handle missing values, duplicate records, inconsistent formats and outliers. Data-type conversion, categorical-variable encoding, normalization, standardization and dataset transformation will be performed. The cleaned dataset will be compared with the original dataset. (3 Hrs)

Practical 3: Descriptive Statistics and Exploratory Data Analysis

Students will calculate frequency, percentage, mean, median, mode, range, variance, standard deviation, percentiles, quartiles and interquartile range. Exploratory analysis will be performed to identify distributions, patterns, relationships and unusual observations. (3 Hrs)

Practical 4: Probability, Regression and Statistical Testing

Students will conduct probability calculations and fit a simple linear-regression model to an appropriate dataset. Regression coefficients, predictions, residuals and goodness-of-fit measures will be interpreted. Suitable statistical tests, such as a t-test, Analysis of Variance, Wilcoxon signed-rank test or Spearman's correlation test, will also be performed. (3 Hrs)

Practical 5: Feature Engineering and Dimensionality Reduction

Students will perform feature encoding, transformation, scaling and selection. Principal Component Analysis will be applied to a multivariate dataset. The explained-variance ratio and the effect of dimensionality reduction on the dataset will be analysed. (3 Hrs)

Practical 6: Data Visualization

Students will create bar charts, line charts, histograms, box plots, scatter plots, heat maps and correlation matrices. Appropriate titles, labels, legends and scales will be used. The visualizations will be interpreted to identify distributions, trends, correlations, group differences and outliers. (3 Hrs)

Practical 7: Classification Models and Evaluation

Students will implement suitable classification algorithms such as K-Nearest Neighbours, Decision Trees, Random Forests, Naive Bayes and Support Vector Machines. The dataset will be divided into training and testing sets. Models will be evaluated using confusion matrices, accuracy, precision, recall, F1-score, Receiver Operating Characteristic curves and Area Under the Curve. (3 Hrs)

Practical 8: Clustering Analysis

Students will apply K-Means, Fuzzy C-Means, DBSCAN or Agglomerative Hierarchical Clustering to an appropriate dataset. Cluster assignments, centroids, membership values, density-based groupings or dendrograms will be interpreted. The results of at least two clustering techniques will be compared. (3 Hrs)

Practical 9: Database and Big-Data Processing

Students will create a simple relational database and perform Structured Query Language operations involving table creation, data insertion, selection, filtering, grouping, aggregation and joins. A basic large-scale data-processing task may also be performed using Apache Spark, MapReduce simulation or a suitable NoSQL database environment. (3 Hrs)

Practical 10: Deep-Learning-Based Data-Science Case Study

Students will develop a basic artificial neural network, Convolutional Neural Network or Recurrent Neural Network for a suitable classification or prediction problem. The practical may involve image classification, text classification, time-series analysis or another real-life application. Data preparation, model architecture, training, evaluation and interpretation of results will be documented. (3 Hrs)

Reference Books

Primary References

1. Cady, F., *The Data Science Handbook*, John Wiley & Sons, 2017.
2. Bruce, P. and Bruce, A., *Practical Statistics for Data Scientists*, O'Reilly Media, 2017.
3. VanderPlas, J., *Python Data Science Handbook*, O'Reilly Media, 2016.

Additional References

1. Pierson, L., *Data Science for Dummies*, John Wiley & Sons, 2015.
2. Moreira, J., Carvalho, A. and Horvath, T., *A General Introduction to Data Analytics*, John Wiley & Sons, 2018.
3. Mitchell, T. M., *Machine Learning*, McGraw-Hill, 1997.
4. Alpaydin, E., *Introduction to Machine Learning*, 3rd Edition, MIT Press, 2014.
5. Braun, W. J. and Murdoch, D. J., *A First Course in Statistical Programming with R*, Cambridge University Press, 2007.
6. Lesmeister, C., *Mastering Machine Learning with R*.
7. Bruce, P., Bruce, A. and Gedeck, P., *Practical Statistics for Data Scientists*, O'Reilly Media.
8. Grus, J., *Data Science from Scratch*, O'Reilly Media, 2015.

PG2-COMP-2014: System Programming using UNIX/Linux

Course Objective

The primary objective of this course is to provide students with a deep understanding of the internal mechanisms of UNIX/Linux operating systems and the skills required to develop system-level software that interacts directly with the operating system through system calls, libraries, and kernel services. The course emphasizes shell programming, process management, inter-process communication, concurrent programming, and system-level programming in C.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Demonstrate proficiency in the UNIX/Linux operating environment and command-line utilities.
2. Develop shell scripts for system administration and automation tasks.
3. Apply operating system concepts using UNIX/Linux system calls and APIs.
4. Design and implement process management and inter-process communication mechanisms.
5. Develop concurrent programs using POSIX threads and synchronization techniques.
6. Utilize UNIX/Linux programming tools for system-level software development and debugging.

Theory Syllabus (30 Hours)

Unit I: Introduction to UNIX/Linux (3 Hours)

History and philosophy of UNIX, GNU and Linux evolution, UNIX architecture (kernel, shell, system utilities), user space and kernel space, system call interface, Linux file system hierarchy, basic UNIX commands (ls, cp, mv, rm, cat, grep, wc, awk, etc.), I/O redirection and pipes, file permissions and ownership, environment variables, types of shells, shell initialization files.

Unit II: UNIX Shell Programming (6 Hours)

Introduction to shell scripting, structure of shell scripts, execution permissions, variables and environment variables, command substitution, arithmetic operations (expr, let, bc), conditional statements (if, if-else, case), looping constructs (for, while, until), break and continue, positional parameters, shift command, functions, arrays, string operations, test command and file testing, input/output in scripts, trap and signal handling, automation scripts.

Unit III: C Programming for System Programming (4 Hours)

Compilation process, GCC compiler options, structure of C programs in Linux, memory layout of C programs, pointers and dynamic memory allocation, file handling in C, low-level and high-level I/O, static and shared libraries, Makefile basics, debugging using gdb, memory debugging using valgrind.

Unit IV: System Calls and File Management (4 Hours)

Concept of system calls, kernel mode switching, file descriptors, open(), close(), read(), write(), lseek(), stat(), fstat(), dup(), dup2(), directory handling, file and directory permissions, chmod(), chown(), umask().

Unit V: Process Management (3 Hours)

Concept of process and process states, process control block, fork(), exec() family, wait(), waitpid(), exit(), _exit(), process scheduling basics, process identifiers.

Unit VI: Inter-Process Communication (4 Hours)

Need for IPC, unnamed pipes, named pipes (FIFO), message queues (System V and POSIX), shared memory, semaphores, synchronization between processes, introduction to sockets, client-server model.

Unit VII: Threads and Memory Management (3 Hours)

Processes versus threads, POSIX threads, OpenMP, mutexes, condition variables, race conditions, deadlocks, virtual memory, paging, segmentation, dynamic memory allocation, memory leaks, buffer overflow.

Unit VIII: File Systems, Security and Advanced Topics (3 Hours)

File system structure, inodes, hard and symbolic links, ext4 overview, mounting and unmounting file systems, file security and access control.

Total Theory Hours: 30

Tutorial Syllabus (15 Hours)

1. Discussion on UNIX/Linux architecture, shell environment, command-line utilities, file system hierarchy, and file permissions.
2. Guided problem solving on shell scripting concepts, variables, control structures, functions, arrays, and script debugging.
3. Discussion on C programming tools, compilation process, memory organization, Makefiles, gdb, and valgrind.
4. Explanation and worked examples on UNIX system calls, file descriptors, file management functions, and directory operations.
5. Discussion on process creation, process states, process scheduling, fork(), exec(), wait(), and process synchronization.
6. Guided discussion on inter-process communication mechanisms, including pipes, FIFOs, message queues, shared memory, semaphores, and sockets.
7. Discussion on POSIX threads, synchronization mechanisms, virtual memory, file systems, security concepts, and advanced UNIX/Linux programming topics.

Practical Syllabus (30 Hours)

1. Write a shell script to check whether a given number is prime.
2. Write a shell script to modify the cal command to display the calendar of a specified month.
3. Write a shell script to display calendars for a specified range of months.
4. Write a shell script to validate a login name.
5. Write a shell script to display the current date in mm/dd/yy format.
6. Write a shell script to display the sorted output of the who command along with the total number of logged-in users.
7. Write a shell script to compare two files and delete duplicate files after user confirmation.
8. Write a shell script to compute the sum of digits of a given number.
9. Write a shell script to merge three files, sort the contents, and display the output page by page.
10. Develop UNIX shell scripts using variables, control structures, and functions for automation tasks.
11. Write C programs using system calls for file handling and low-level I/O operations.
12. Implement process management using fork(), exec(), wait(), and signal handling.
13. Design concurrent and IPC-based programs using pipes, shared memory, semaphores, and POSIX threads.

Text / Reference Books

1. W. Richard Stevens and Stephen A. Rago, *Advanced Programming in the UNIX® Environment*, Addison-Wesley.
2. W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff, *UNIX Network Programming, Volume 1: The Sockets Networking API*, Addison-Wesley.
3. Sumitabha Das, *UNIX: Concepts and Applications*, Tata McGraw-Hill.
4. Evi Nemeth, Garth Snyder, Trent R. Hein, and Ben Whaley, *UNIX and Linux System Administration Handbook*, Pearson Education.
5. Michael Jang, *RHCSA/RHCE Red Hat Linux Certification Study Guide (EX200 & EX300)*, McGraw-Hill.
6. Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice Hall.
7. Robert Love, *Linux System Programming*, O'Reilly Media.
8. Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment*, Prentice Hall.
9. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Pearson.
10. Michael Kerrisk, *The Linux Programming Interface*, No Starch Press.

PG2-COMP-2021: Introduction to machine Learning (for other allied discipline)

Credit:04 **75 Hrs**

Course Objectives:

Machine learning courses aim to provide a foundational understanding of algorithms that allow computers to learn from data, identify patterns, and make predictions without explicit programming. Key objectives include mastering supervised/unsupervised learning, implementing models using Python and applying these techniques to real-world problems.

Course Learning Outcomes:

(i) Define and explain machine learning concepts, types, and basic metrics.

(ii) Implement and apply supervised learning techniques (e.g., KNN, Linear Regression, Logistic Regression).

(iii) Apply unsupervised learning methods (e.g., K-Means, Hierarchical Clustering, Association Rules).

(iv): Develop and evaluate simple machine learning models (e.g., Perceptron, single-layer,neural networks).

(v) Analyze and apply appropriate machine learning algorithms depending on the problems with some real-world data.

(vi) Implement and evaluate supervised learning techniques, including K-Nearest Neighbours, linear regression, and logistic regression, and measure model performance using accuracy, precision, recall, and F1 score.

Syllabus

(Theory 2 + Tutorial 1)

UNIT I: Introduction to Machine Learning(20 Hrs)

Introduction: Definition, History and Application of Machine Learning,

Types of Machine Learning: Supervised, Unsupervised, Semi-Supervised, and Reinforcement Learning.

Labelled and Unlabelled Dataset.
Supervised Learning Tasks: Regression vs. Classification,
Learning Framework: Training, Validation and Testing of ML models. *Performance*
Evaluation Parameters: Confusion matrix, Accuracy, Precision, Recall, F1 Score, and AUC.

UNIT II: Supervised Learning and Unsupervised Learning(25 Hrs)
Regression: Linear and non-linear Regression, Logistic Regression.
Classification: Naïve Bayes, K-Nearest Neighbors, Decision Trees.
Models: Introduction to Artificial Neural Networks, Perceptron Learning Algorithm, Single Layer Perceptron, Introduction to Support
Vector Machine for linearly separable data.
Clustering: K-Means, Hierarchical Clustering,
DBSCAN, Clustering Validation Measures.
ML Applications: Ethical Considerations in Machine Learning, Case study and Real-world Applications.

Machine Learning Laboratory (Practical)

Credit: 1
Teaching Hours: 30 Hours

Programming Language: Python (preferred) / R / MATLAB or any suitable Machine Learning software.

Practical Exercises

1. Study and configure the machine learning programming environment and libraries.
2. Perform data acquisition, preprocessing, cleaning, normalization, and feature encoding on a given dataset.
3. Perform exploratory data analysis and visualize the characteristics of a dataset.
4. Split a dataset into training, validation, and testing sets.
5. Implement and analyze a linear regression model for prediction.
6. Implement a logistic regression model for binary classification.
7. Develop and evaluate a Naïve Bayes classifier.
8. Implement the K-Nearest Neighbour (KNN) algorithm for classification.
9. Construct and analyze a Decision Tree classifier.
10. Implement a Single Layer Perceptron for a classification problem.
11. Apply a Support Vector Machine (SVM) for linearly separable datasets.
12. Implement K-Means clustering on an unlabeled dataset.
13. Apply Hierarchical Clustering and interpret the resulting dendrogram.
14. Implement DBSCAN clustering and compare its performance with K-Means.
15. Evaluate machine learning models using appropriate performance measures such as Accuracy, Precision, Recall, F1-score, Confusion Matrix, ROC-AUC, and clustering validation metrics.
16. Compare the performance of multiple supervised learning algorithms on the same dataset.
17. Compare different clustering algorithms using suitable validation measures.
18. Conduct a case study by selecting an appropriate machine learning algorithm for a real-world problem and interpret the results.
19. Prepare and present a laboratory report summarizing the implementation, observations, and

performance analysis of the developed models.

Reference Books:

1. Rajiv Chopra (2024), Machine Learning and Machine Intelligence, Khanna Publishing House.
2. Jeeva Jose (2023), Introduction to Machine Learning, Khanna Publishing House.
3. Mitchell T. (1997). Machine Learning, First Edition, McGraw-Hill.
4. Kalita, J. K., Bhattacharyya, D. K., & Roy, S. (2023). Fundamentals of Data Science: Theory and Practice. Elsevier. ISBN9780323917780
5. Flach, P. A. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press. ISBN: 9781107422223, 2012.
6.. Duda, R. O., Hart, P. E., Stork, D (2007). Pattern classification (2Ed), John Wiley & Sons, ISBN-13: 978-8126511167.
7. Haykin S. (2009). Neural Networks and Learning Machines, Third Edition, PHI Learning.
8. Chollet, F. (2018). Deep Learning with Python. Manning Publications.

PG2-COMP-3011: Advanced Software Engineering

Total Credits: 4
Total Contact Hours: 75 Hours

Course Objectives

The course aims to provide students with an advanced understanding of software-engineering principles, processes, models and practices used throughout the software-development life cycle. It introduces software-process models, project planning and estimation, requirements engineering, software modelling, architectural and component-level design, software quality assurance, testing, debugging, software reuse and Computer-Aided Software Engineering tools.

The lecture component develops the theoretical and methodological foundations required for analysing, designing, developing, evaluating and managing complex software systems. The tutorial component strengthens analytical and problem-solving abilities through project-estimation exercises, requirements modelling, design problems, quality measurement, test-case preparation and case-based discussions. The practical component provides hands-on experience in requirements documentation, Unified Modeling Language modelling, project planning, software design, testing, version control, quality analysis and the use of modern CASE and software-development tools.

The course also develops the ability to apply systematic, disciplined, measurable and cost-effective engineering approaches to software development and maintenance.

Course Learning Outcomes

On successful completion of the course, the students will be able to achieve the following outcomes.

- Explain software characteristics, components, applications, processes and major software-development models.
- Apply software-project management concepts, metrics, measurement techniques and estimation models.
- Perform software-project planning, requirements elicitation, requirements analysis, specification and software-system modelling.
- Develop data, functional, behavioural and information-flow models for software systems.
- Design modular, architectural, procedural, object-oriented and user-interface components.
- Evaluate software quality using quality metrics, technical reviews, reliability measures, formal assurance techniques and international standards.
- Design and execute suitable white-box, black-box, unit, integration, validation, system, alpha

and beta testing strategies.

- Apply debugging, defect-tracking and systematic fault-identification techniques during software development.
- Explain software-reuse processes and evaluate the technical and economic benefits of reusable software components.
- Use Computer-Aided Software Engineering tools and integrated development environments for efficient software planning, modelling, development, testing, documentation and management.

Lecture

Lecture Credit: 2

Total Lecture Hours: 30 Hours

Unit 1: Software-Engineering Processes, Models and Project Management

Fundamentals of Software Engineering

This subsection introduces software and its characteristics, including complexity, conformity, changeability and invisibility. Software components, software applications and different categories of software systems are discussed. The evolving nature of software and the major challenges associated with software development are examined. Software engineering is introduced as a systematic, disciplined and measurable approach to software development, operation and maintenance. Software engineering as a layered technology is discussed through its quality focus, process, methods and tools. The importance of ethical and professional responsibilities in software engineering is also introduced.

Software Processes and Process Models

This subsection introduces software processes and the major framework activities of communication, planning, modelling, construction and deployment. Supporting activities such as project tracking, risk management, software-quality assurance, configuration management, technical reviews and documentation are discussed. The linear sequential or waterfall model, prototyping model, Rapid Application Development model, incremental model, evolutionary process models and spiral model are covered. The characteristics, activities, advantages, limitations and application areas of each model are discussed. Model selection based on requirement stability, project size, risk, delivery schedule and customer involvement is examined. An introduction to agile and iterative development principles may also be included to relate traditional process models to contemporary software-development practices.

Fundamentals of Software-Project Management

This subsection introduces software-project management and its importance in the successful development of software systems. The people, product, process and project dimensions are discussed. Project stakeholders, team structures, communication, coordination, leadership and responsibility allocation are examined. Project initiation, feasibility analysis, planning, scheduling, monitoring, control and closure are introduced. Common reasons for software-project failure and the role of effective management in reducing project risks are discussed.

Software Metrics and Measurement

This subsection introduces software measurement, measures, metrics and indicators. Process metrics, project metrics and product metrics are distinguished. Size-oriented metrics such as Lines of Code and productivity measures are discussed. Function-oriented metrics and function-point analysis are introduced through inputs, outputs, inquiries, internal files and external interfaces. Extended function-point concepts and the use of metrics in estimation, quality assessment and process improvement are also covered. (6 Hrs)

Unit 2: Software-Project Planning and Requirements Engineering

Software-Project Planning

This subsection introduces the concept, purpose, scope and objectives of software-project planning. Project scope, assumptions, constraints, deliverables, milestones and resource requirements are discussed. Task decomposition, Work Breakdown Structures and project scheduling are introduced. Effort, cost, staffing and development-time estimation are examined. The relationship among software size, effort, cost and schedule is discussed. Project-decomposition techniques, including problem-based and process-based decomposition, are covered.

Software Estimation Models

This subsection introduces empirical estimation models and their role in project planning. Expert judgement, analogy-based estimation, top-down estimation and bottom-up estimation are discussed. The basic principles of the Constructive Cost Model and other empirical estimation approaches are

introduced. Estimation uncertainty, productivity variation, reuse, personnel capability, technological complexity and risk factors are considered. The limitations of estimation models and the need for continuous refinement of project estimates are discussed.

Requirements Engineering

This subsection introduces requirements engineering and its major activities, including inception, elicitation, elaboration, negotiation, specification, validation and management. Functional requirements, non-functional requirements, domain requirements, user requirements and system requirements are distinguished. Communication techniques such as interviews, questionnaires, observation, workshops, brainstorming, prototyping and document analysis are discussed. Challenges involving ambiguous, incomplete, conflicting, volatile and infeasible requirements are examined.

Requirements Analysis and Specification

This subsection explains the principles of requirements analysis and the characteristics of a good Software Requirements Specification. Requirements should be correct, complete, consistent, verifiable, traceable, modifiable and unambiguous. Software prototyping and its role in clarifying requirements are discussed. Throwaway and evolutionary prototypes are distinguished. Requirements prioritization, requirements validation and requirements traceability are introduced.

Analysis Modelling

This subsection introduces the elements of analysis modelling. Data modelling, functional modelling, information-flow modelling and behavioural modelling are discussed. Entity-relationship concepts, Data Flow Diagrams, process specifications, state-transition representations and control specifications are introduced. The purpose, contents and structure of a data dictionary are explained. The use of analysis models for communicating requirements among customers, analysts, designers and developers is emphasized. (6 Hrs)

Unit 3: Software Design

Software-Design Process and Principles

This subsection introduces software design as the process of transforming requirements and analysis models into a technical representation for construction. The major design activities, including data design, architectural design, interface design and component-level design, are discussed. Design concepts such as abstraction, refinement, modularity, information hiding, separation of concerns, functional independence, cohesion, coupling, patterns and refactoring are covered. The characteristics of a good software design, including understandability, maintainability, reusability, testability and extensibility, are examined.

Modular and Architectural Design

This subsection explains effective modular design. Different levels of cohesion and coupling are discussed along with their effects on maintainability and software quality. Software architecture and architectural styles are introduced. Layered architecture, client-server architecture, repository architecture, pipe-and-filter architecture and Model-View-Controller architecture are discussed. The architectural-design process, architectural decisions and evaluation of architectural alternatives are covered. Transform mapping and transaction mapping are introduced as techniques for deriving architectural structures from information-flow models.

Internal, External and Interface Design

This subsection distinguishes internal design from external design. Internal design addresses software structure, modules, algorithms and data organization, whereas external design addresses user-visible behaviour and interactions with external systems. Human-Computer Interface design is introduced through user characteristics, task analysis, usability and accessibility. Interface-design principles such as consistency, simplicity, feedback, error prevention, recoverability and user control are discussed. Interface guidelines for forms, navigation, menus, messages and data presentation are covered.

Procedural and Object-Oriented Design

This subsection introduces procedural design using algorithms, flowcharts, decision tables, decision trees and structured English. Component-level design and stepwise refinement are discussed. Object-oriented design is introduced through objects, classes, attributes, operations, relationships, inheritance, aggregation, composition and polymorphism. The transformation of analysis classes into design classes and the allocation of responsibilities among software objects are covered. (4 Hrs)

Unit 4: Software Quality Assurance and Reliability

Software Quality Concepts

This subsection introduces software quality and the different views of quality. Quality factors such as correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability and interoperability are discussed. Software-quality metrics and their role in assessing

software products, processes and projects are covered. Defect density, complexity, maintainability, test coverage, customer-reported defects and process-compliance measures are introduced.

Software Quality Assurance

This subsection introduces Software Quality Assurance as a planned and systematic set of activities for ensuring that software processes and products conform to requirements, standards and procedures. Quality planning, quality control, audits, process monitoring, defect prevention and continuous improvement are discussed. The quality movement and the role of organizational quality culture are examined. Formal technical reviews, walkthroughs, inspections and peer reviews are covered. Review planning, review roles, entry criteria, exit criteria and defect reporting are discussed.

Formal Quality Assurance and Reliability

This subsection introduces formal approaches to quality assurance, including mathematical specifications, formal verification and correctness arguments. The benefits and limitations of formal methods are discussed. Software reliability, failure, fault, error, availability and maintainability are introduced. Reliability measures such as Mean Time to Failure, Mean Time to Repair and Mean Time Between Failures are discussed.

Software-Quality Standards

This subsection introduces applicable international standards and quality models for software development and evaluation. The role of ISO 9001, ISO/IEC 25010 and related software life-cycle and quality standards is discussed at an introductory level. Process assessment and maturity concepts may also be introduced. (3 Hrs)

Unit 5: Software Testing and Debugging

Fundamentals of Software Testing

This subsection introduces the purpose, objectives and principles of software testing. Verification and validation are distinguished. Testing levels, testing strategies, test planning, test cases, test data, test suites, test environments and expected results are discussed. The relationship among errors, faults, defects and failures is explained. The limitations of testing and the principles of exhaustive testing, defect clustering, pesticide paradox and early testing are introduced.

White-Box Testing

This subsection covers white-box testing based on the internal structure and logic of a program. Control-flow testing, statement coverage, branch coverage, condition coverage and path coverage are discussed. Basis-path testing is introduced through flow graphs, regions, independent paths and cyclomatic complexity. Testing of control structures, including conditions, loops and nested loops, is covered. Data-flow testing may also be introduced.

Black-Box Testing

This subsection introduces black-box testing based on software requirements and externally observable behaviour. Equivalence-class partitioning, boundary-value analysis, decision-table testing, cause-effect graphing, state-transition testing and error guessing are discussed. The selection of effective test inputs and the reduction of redundant test cases are emphasized.

Testing Strategies

This subsection discusses unit testing, integration testing, validation testing and system testing. Top-down, bottom-up, sandwich and big-bang integration strategies are compared. The use of drivers and stubs is explained. System-testing categories such as recovery testing, security testing, stress testing, performance testing, usability testing and compatibility testing are introduced. Acceptance testing, alpha testing and beta testing are discussed. Regression testing and automated testing are introduced. The role of test-management and test-automation tools in contemporary software development is explained.

Debugging

This subsection introduces debugging as the process of locating, analysing and correcting the causes of observed failures. Brute-force debugging, backtracking, cause-elimination and program-slicing techniques are discussed. The relationship between testing and debugging is explained. Logging, assertions, breakpoints, stack traces, defect tracking and root-cause analysis are introduced. (5 Hrs)

Unit 6: Software Reusability

Fundamentals of Software Reuse

This subsection introduces software reusability and the different levels of reuse, including code reuse, component reuse, framework reuse, service reuse, design-pattern reuse and application-system reuse. Opportunistic reuse and systematic reuse are distinguished. The software-reuse process, including identification, evaluation, adaptation, integration and maintenance of reusable assets, is discussed.

Development and Management of Reusable Components

This subsection explains the characteristics of reusable software components, including generality,

modularity, low coupling, high cohesion, clear interfaces, configurability, reliability and documentation. Classification and retrieval of reusable components are discussed. Component catalogues, repositories, metadata, indexing, keyword classification and faceted classification are introduced.

Benefits, Limitations and Economics of Reuse

This subsection examines the benefits of software reuse, including reduced development time, lower cost, improved quality and greater consistency. Limitations related to component mismatch, adaptation effort, licensing, dependency management, security and maintenance are discussed. Economic considerations associated with reusable software components are introduced. The cost of developing for reuse is compared with the cost of developing with reuse. Break-even analysis and return on investment are discussed conceptually. (3 Hrs)

Unit 7: Computer-Aided Software Engineering

Introduction to CASE

This subsection introduces Computer-Aided Software Engineering and its role in automating and supporting software-development activities. The objectives, capabilities, advantages and limitations of CASE tools are discussed. The building blocks of CASE environments, including tools, workbenches, repositories, methods and integration mechanisms, are introduced.

Taxonomy of CASE Tools

This subsection discusses the classification of CASE tools according to supported life-cycle activities and functionality. Upper-CASE tools supporting planning, requirements analysis and design are distinguished from lower-CASE tools supporting coding, testing, maintenance and reverse engineering. Tools for project management, modelling, prototyping, interface design, code generation, testing, configuration management, documentation and maintenance are introduced.

Integrated CASE Environments

This subsection explains integrated CASE environments and the integration of tools across the software-development life cycle. Data integration, control integration, presentation integration, platform integration and process integration are discussed. CASE repositories and their role in storing models, requirements, design specifications, code information, test data and project metadata are covered. Version control, traceability and consistency checking are introduced.

Contemporary Software-Engineering Tools

This subsection discusses the role of modern development environments, Unified Modeling Language tools, requirements-management tools, project-tracking systems, version-control platforms, continuous-integration tools, automated-testing frameworks and code-quality-analysis tools. The use of CASE tools in planning, analysis, design, development, testing, documentation, deployment and maintenance is examined. (3 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Software Processes, Metrics and Project Management

This tutorial unit includes comparative analysis of waterfall, prototyping, RAD, incremental, evolutionary and spiral process models. Students will select appropriate process models for given project situations and justify their decisions. Exercises on software size, Lines of Code, productivity, defect density and function-point estimation will be undertaken. Students will identify the people, product, process and project dimensions of selected software-development cases and analyse common causes of project failure. (3 Hrs)

Tutorial Unit 2: Project Planning, Estimation and Requirements Analysis

This tutorial unit includes preparation of project scope statements, Work Breakdown Structures, task dependencies, milestone plans and preliminary project schedules. Students will perform effort, cost and duration estimation using decomposition, expert judgement and suitable empirical estimation models. Requirements-engineering exercises will include identifying functional and non-functional requirements, detecting ambiguity and inconsistency, prioritizing requirements and preparing structured Software Requirements Specification sections. Students will develop context diagrams, Data Flow Diagrams, entity-relationship representations, behavioural models and data dictionaries for selected systems. (3 Hrs)

Tutorial Unit 3: Software Design

This tutorial unit includes exercises on abstraction, refinement, information hiding, modularity, cohesion and coupling. Students will analyse poorly structured designs and recommend improvements.

Architectural styles will be selected for given application scenarios. Transform mapping and transaction mapping exercises will be undertaken. Students will prepare interface layouts, procedural designs and object-oriented design models for representative software problems. (2 Hrs)

Tutorial Unit 4: Software Quality and Reliability
This tutorial unit includes calculation and interpretation of basic software-quality and reliability measures. Students will analyse defect data, reliability data and review findings. Review checklists will be prepared for requirements, design and code documents. Case-based exercises on Software Quality Assurance planning, formal reviews and the selection of suitable software-quality standards will also be undertaken. (1 Hrs)
Tutorial Unit 5: Software Testing and Debugging
This tutorial unit includes the preparation of test cases using equivalence-class partitioning, boundary-value analysis, decision tables and state-transition testing. Students will draw control-flow graphs, calculate cyclomatic complexity and identify independent paths for basis-path testing. Exercises on unit, integration, validation and system-testing strategies will be undertaken. Students will analyse failure reports, distinguish symptoms from causes and apply suitable debugging and root-cause-analysis techniques. (3 Hrs)
Tutorial Unit 6: Software Reuse
This tutorial unit includes analysis of reusable-component characteristics, classification of components and selection of reusable assets from hypothetical repositories. Students will compare development with reuse and development for reuse. Simple economic exercises will be undertaken to determine the conditions under which the additional cost of producing reusable components becomes beneficial. (1 Hrs)
Tutorial Unit 7: CASE Tools and Integrated Environments
This tutorial unit includes comparison of upper-CASE, lower-CASE and integrated CASE tools. Students will identify appropriate tools for project planning, requirements management, modelling, coding, testing, configuration management and documentation. A case study will be analysed to design an integrated tool environment supporting traceability from requirements to design, implementation and testing. (2 Hrs)

Practical
Practical Credit: 1
Hours per Practical Credit: 30 Hours
Total Practical Hours: 30 Hours
The practical component may be conducted using suitable project-management, modelling, development, testing and CASE tools. Appropriate tools may include StarUML, Visual Paradigm, draw.io, Microsoft Project, ProjectLibre, Git, GitHub, GitLab, Jira, Selenium, JUnit, NUnit, PyTest, SonarQube or equivalent software-engineering tools.
Practical 1: Process-Model Selection and Project Proposal
Students will select a suitable software-development problem and prepare a concise project proposal. The proposal will include the problem statement, project objectives, stakeholders, scope, assumptions, constraints, deliverables and selected software-process model. The choice of process model will be justified based on requirement stability, project size, risk and customer involvement. (3 Hrs)
Practical 2: Project Planning and Estimation
Students will develop a Work Breakdown Structure for the selected project. Project activities, dependencies, milestones, effort, resources and duration will be identified. A project schedule or Gantt chart will be prepared using a suitable project-management tool. Software size and effort will be estimated using Lines of Code, function points, decomposition or an appropriate empirical estimation technique. The assumptions and limitations of the estimates will be documented. (3 Hrs)
Practical 3: Requirements Elicitation and Software Requirements Specification
Students will conduct a simulated requirements-elicitation activity using interviews, questionnaires, observation or brainstorming. Functional requirements, non-functional requirements, constraints and business rules will be identified. A structured Software Requirements Specification will be prepared. The document will include the system scope, user classes, functional requirements, quality requirements, external-interface requirements, assumptions and acceptance criteria. (3 Hrs)
Practical 4: Analysis Modelling
Students will develop analysis models for the selected system using an appropriate CASE or modelling tool. The practical will include a context diagram, Data Flow Diagrams, entity-relationship model, data dictionary and behavioural representation. The consistency among requirements, processes, data entities and system behaviour will be examined. (3 Hrs)
Practical 5: Object-Oriented Modelling Using UML

Students will develop Unified Modeling Language models for the selected software system. Use-case diagrams, class diagrams, sequence diagrams, activity diagrams and state-machine diagrams will be prepared as applicable. The relationships among actors, use cases, classes, objects and interactions will be documented. Traceability between requirements and design elements will be maintained. (3 Hrs)

Practical 6: Architectural, Component and Interface Design

Students will select and document an appropriate architectural style for the selected application. Major subsystems, modules, components, interfaces and data flows will be identified. A component-level design and sample user-interface prototype will be prepared. Design quality will be assessed in terms of modularity, cohesion, coupling, maintainability, usability and extensibility. (3 Hrs)

Practical 7: Version Control and Collaborative Development

Students will create and manage a software repository using a distributed version-control system. The practical will include repository initialization, commits, branches, merging, conflict resolution, tagging and maintenance of a suitable ignore file. A collaborative workflow involving feature branches, code reviews or merge requests will be demonstrated. Commit history and project documentation will be maintained systematically. (3 Hrs)

Practical 8: White-Box and Black-Box Test Design

Students will select a program module and prepare white-box and black-box test cases. A control-flow graph will be constructed, cyclomatic complexity will be calculated and independent paths will be identified. Equivalence-class partitioning, boundary-value analysis and decision-table testing will be applied. The practical will document test inputs, expected results, actual results and pass or fail status. (3 Hrs)

Practical 9: Automated Testing, Debugging and Quality Analysis

Students will implement automated unit or functional tests using an appropriate testing framework. A deliberate set of defects may be introduced to practise failure reproduction, logging, breakpoint analysis, stack-trace interpretation and root-cause identification. Static analysis or code-quality tools will be used to identify complexity, duplication, coding-standard violations, vulnerabilities or maintainability issues. Corrective actions and regression-test results will be documented. (3 Hrs)

Practical 10: Integrated CASE-Based Mini Project

Students will integrate project planning, requirements specification, analysis modelling, software design, implementation, testing, configuration management and documentation into a small software-engineering project. The project will use appropriate CASE and development tools and will demonstrate traceability among requirements, design elements, source-code modules and test cases. The final submission will include project documents, models, source code, test reports, repository history and a brief project presentation. (3 Hrs)

Reference Books

Primary References

1. Pressman, R. S. and Maxim, B. R., *Software Engineering: A Practitioner's Approach*, McGraw-Hill.
2. Jalote, P., *An Integrated Approach to Software Engineering*, Springer.
3. Mall, R., *Fundamentals of Software Engineering*, PHI Learning.

Additional References

1. Sommerville, I., *Software Engineering*, Pearson.
2. Pfleeger, S. L. and Atlee, J. M., *Software Engineering: Theory and Practice*, Pearson.
3. Schach, S. R., *Object-Oriented and Classical Software Engineering*, McGraw-Hill.
4. Humphrey, W. S., *Managing the Software Process*, Addison-Wesley.

PG2-COMP-3012: Machine Learning

Total Credits: 4

Total Contact Hours: 75 Hours

Course Objectives

The course aims to provide students with a comprehensive understanding of the fundamental concepts, mathematical foundations, algorithms and practical applications of machine learning. It introduces

supervised and unsupervised learning, regression, probabilistic classification, nearest-neighbour methods, neural networks, decision trees, ensemble learning, clustering, dimensionality reduction, Support Vector Machines and time-series modelling. The lecture component develops the theoretical and mathematical foundations of machine-learning methods, the tutorial component strengthens analytical and problem-solving skills through numerical and algorithmic exercises, and the practical component provides hands-on experience in data preparation, model implementation, performance evaluation and the development of machine-learning solutions using suitable programming and analytical tools.

Course Learning Outcomes

On successful completion of the course, the students will be able to:

- Explain the fundamental concepts, categories, problems, tools and applications of machine learning;
- Prepare, visualize and partition datasets into training, validation and testing sets;
- Apply linear-regression techniques using gradient descent, normal equations and suitable feature representations;
- Identify overfitting and control model complexity through appropriate validation and evaluation methods;
- Apply probabilistic classification techniques, including Bayes optimal classification, Naive Bayes and Gaussian class-conditional models;
- Develop classification models using logistic regression, linear classifiers, nearest-neighbour methods and neural networks;
- Construct decision trees and apply ensemble-learning methods such as bagging, random forests and boosting;
- Apply clustering techniques, including K-Means, Hierarchical Agglomerative Clustering and Expectation-Maximization;
- Use Principal Component Analysis and latent-space methods for dimensionality reduction and data representation;
- Explain VC dimension, structural risk minimization, margin-based learning and Support Vector Machines;
- Analyse sequential and time-dependent data using Markov and autoregressive models; and
- Implement suitable machine-learning algorithms and develop solutions for real-world applications using programming and analytical tools.

Lecture

Lecture Credit: 2

Total Lecture Hours: 30 Hours

Unit 1: Introduction, Data Preparation, Regression and Basic Classification

This unit introduces the definition, scope, objectives and applications of machine learning, along with the distinction among supervised, unsupervised and semi-supervised learning. It discusses machine-learning problems, datasets, attributes, labels, data types, analytical tools and programming environments such as Python, R, MATLAB or equivalent platforms. Data visualization, exploratory data analysis, handling of missing values, categorical-variable encoding, normalization, standardization and partitioning of datasets into training, validation and testing sets are covered. The unit introduces simple and multiple linear regression, hypothesis functions, the Sum of Squared Errors, Mean Squared Error, gradient-descent optimization, learning rate, convergence, closed-form solutions and normal equations. Feature representation, polynomial features, interaction features and basic feature engineering are discussed. Overfitting, underfitting, bias, variance, model complexity, regularization and validation are introduced. Classification problems, class labels, decision boundaries, distance measures and nearest-neighbour methods, including the K-Nearest Neighbours algorithm, are also covered. (8 Hrs)

Unit 2: Probability, Probabilistic Classification and Neural Networks

This unit introduces the probability concepts required for machine learning, including random variables, joint probability, conditional probability, marginal probability, prior probability, posterior probability, likelihood and Bayes' rule. Bayes optimal decisions, minimum-error classification and probabilistic decision boundaries are discussed. The Naive Bayes model, conditional-independence assumption, Bernoulli Naive Bayes, Gaussian Naive Bayes and Gaussian class-conditional distributions are covered. Linear classifiers, linear decision functions and discriminant functions are introduced. Logistic regression, the sigmoid function, log-loss, maximum-likelihood estimation, gradient descent and online gradient descent are discussed. The unit also introduces artificial neural networks through biological inspiration, artificial neurons, weights, biases, activation functions, input layers, hidden layers and output

layers. Feed-forward computation, loss functions, gradient-based training, backpropagation, learning rate, epochs, batch learning, stochastic learning, regularization and common neural-network applications are covered. (7 Hrs)

Unit 3: Decision Trees, Ensemble Learning and Clustering

This unit introduces decision-tree learning for classification and regression. It discusses tree structure, root nodes, decision nodes, branches, leaf nodes, recursive partitioning, entropy, information gain, gain ratio, Gini impurity, stopping criteria and pruning. The construction, interpretation and evaluation of decision trees are covered. Ensemble-learning concepts are introduced through model diversity, voting and averaging. Bagging, bootstrap sampling, Random Forests, random feature selection, boosting, weak learners, sequential error correction, AdaBoost and the basic principles of gradient boosting are discussed. The unit introduces unsupervised learning and clustering through similarity, dissimilarity, distance measures, cluster compactness and separation. K-Means clustering, centroid initialization, assignment, centroid updating and convergence are covered. Hierarchical Agglomerative Clustering, proximity matrices, single linkage, complete linkage, average linkage and dendrogram interpretation are discussed. Advanced clustering concepts, probabilistic clustering, mixture models and the Expectation-Maximization algorithm are introduced, along with their applications and limitations. (8 Hrs)

Unit 4: Latent-Space Models, Text Classification, Support Vector Machines and Time-Series Models

This unit introduces latent-space methods and dimensionality reduction. Principal Component Analysis is discussed through mean centring, covariance matrices, eigenvalues, eigenvectors, principal components, variance preservation and projection into lower-dimensional spaces. Text representation techniques, including tokenization, stop-word removal, stemming, lemmatization, bag-of-words representation, term frequency and TF-IDF, are introduced. Naive Bayes and multinomial probabilistic models for text classification are covered, along with clustering and latent-space representations of textual data. The unit introduces learning theory through hypothesis spaces, empirical risk, generalization, VC dimension, structural risk minimization and the relationship between model complexity and generalization. Margin-based learning, separating hyperplanes, maximum-margin classifiers, support vectors, hard-margin Support Vector Machines, soft-margin Support Vector Machines and kernel-based classification are discussed. Time-series data, temporal dependence, stationarity, trend and sequential observations are introduced. Markov models, transition probabilities, state sequences, autoregressive models, lagged variables and their applications in forecasting and sequential-data analysis are also covered. (7 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Data Preparation, Regression and Nearest-Neighbour Classification

This tutorial unit includes exercises on identifying machine-learning problem types, classifying variables, handling missing values, encoding categorical attributes, normalizing numerical data and dividing datasets into training, validation and testing subsets. Students will solve linear-regression problems using the Sum of Squared Errors, gradient-descent iterations and normal equations. Exercises on feature representation, polynomial regression, model complexity, overfitting, underfitting and validation-error interpretation will be undertaken. Distance calculations, decision boundaries and K-Nearest Neighbours classification problems will also be solved. (4 Hrs)

Tutorial Unit 2: Probability, Bayes Classification, Logistic Regression and Neural Networks

This tutorial unit includes numerical exercises on joint, conditional and posterior probability, Bayes' rule and Bayes optimal decisions. Students will calculate class probabilities using Naive Bayes and Gaussian class-conditional models. Logistic-regression exercises will include sigmoid calculations, decision probabilities, log-loss and simple gradient updates. Artificial-neuron computations, activation functions, forward propagation, output calculation and basic backpropagation steps will also be practised. (4 Hrs)

Tutorial Unit 3: Decision Trees, Ensemble Methods and Clustering

This tutorial unit includes calculations of entropy, information gain and Gini impurity for selecting decision-tree splits. Students will construct small decision trees and analyse the effects of pruning and overfitting. Exercises on bagging, majority voting, Random Forest predictions and AdaBoost weight updating will be undertaken. Manual iterations of K-Means clustering, linkage calculations for

hierarchical clustering, dendrogram interpretation and simple Expectation-Maximization calculations will also be included. (4 Hrs)

Tutorial Unit 4: PCA, Text Models, Support Vector Machines and Time-Series Analysis

This tutorial unit includes exercises on mean centring, covariance calculation, eigenvalue interpretation, principal-component selection and dimensionality reduction. Students will construct bag-of-words and TF-IDF representations and apply multinomial Naive Bayes to small text-classification examples. Margin calculations, separating hyperplanes, support-vector identification and the effect of regularization in Support Vector Machines will be discussed. Markov-transition calculations and basic autoregressive predictions using lagged values will also be undertaken. (3 Hrs)

Practical

Practical Credit: 1

Hours per Practical Credit: 30 Hours

Total Practical Hours: 30 Hours

The practical component may be conducted using Python, R, MATLAB or another suitable programming environment. Appropriate libraries and tools may include NumPy, Pandas, Matplotlib, scikit-learn, SciPy, TensorFlow, Keras, PyTorch or equivalent machine-learning frameworks.

Practical 1: Dataset Loading, Exploration and Visualization

Students will load a suitable dataset, inspect its dimensions, attributes, data types, missing values and summary statistics, and create visual representations such as histograms, scatter plots, box plots, bar charts and correlation matrices to understand distributions, relationships and unusual observations. (3 Hrs)

Practical 2: Data Preprocessing and Dataset Partitioning

Students will perform missing-value treatment, duplicate removal, categorical-variable encoding, feature scaling and outlier handling. The processed dataset will be divided into training, validation and testing subsets, and the differences between random, stratified and time-based splitting approaches will be examined. (3 Hrs)

Practical 3: Linear Regression Using Gradient Descent and Normal Equations

Students will implement simple or multiple linear regression using gradient descent and compare the result with the closed-form normal-equation solution or a suitable library implementation. Predictions, regression coefficients, residuals, Mean Squared Error and coefficient of determination will be analysed. (3 Hrs)

Practical 4: K-Nearest Neighbours and Logistic Regression

Students will implement K-Nearest Neighbours and logistic-regression classifiers on a suitable dataset. The effects of feature scaling, distance metrics, neighbourhood size and classification thresholds will be studied. The models will be evaluated using confusion matrices, accuracy, precision, recall and F1-score. (3 Hrs)

Practical 5: Naive Bayes and Gaussian Classification

Students will develop probabilistic classification models using Naive Bayes and Gaussian class-conditional distributions. Prior probabilities, likelihoods and posterior probabilities will be examined. The models will be applied to categorical, numerical or text-based datasets and their classification performance will be interpreted. (3 Hrs)

Practical 6: Artificial Neural Network

Students will construct and train a basic feed-forward artificial neural network for classification or regression. The practical will include selection of network architecture, activation functions, loss function, optimizer, batch size and number of epochs. Training and validation performance will be compared to identify underfitting or overfitting. (3 Hrs)

Practical 7: Decision Trees and Ensemble-Learning Models

Students will implement a Decision Tree, Random Forest and boosting-based classifier on a suitable dataset. Tree depth, splitting criteria, pruning, number of estimators and learning rate will be examined. Feature importance and the comparative performance of individual and ensemble models will be analysed. (3 Hrs)

Practical 8: Clustering Using K-Means, Hierarchical Clustering and Expectation-Maximization

Students will apply K-Means and Hierarchical Agglomerative Clustering to an unlabeled dataset and visualize the resulting clusters. The number of clusters will be examined using suitable criteria such as the elbow method or silhouette score. A Gaussian mixture or Expectation-Maximization-based clustering model will also be implemented and compared with distance-based clustering. (3 Hrs)

Practical 9: Principal Component Analysis and Support Vector Machines
Students will apply Principal Component Analysis to a multidimensional dataset and analyse the explained-variance ratio and reduced feature representation. A Support Vector Machine classifier will be trained on the original and reduced datasets. Linear and kernel-based Support Vector Machines will be compared in terms of accuracy, training time and decision boundaries. (3 Hrs)
Practical 10: Text Classification or Time-Series Modelling Case Study
Students will complete a machine-learning case study involving either text classification or time-series prediction. The text-classification option will include text preprocessing, bag-of-words or TF-IDF representation and classification using multinomial Naive Bayes or another suitable model. The time-series option will include temporal ordering, lag-feature generation, training of an autoregressive model and evaluation of forecasting performance. (3 Hrs)
Reference Books
Primary References

1. Mitchell, T. M., *Machine Learning*, McGraw-Hill;
2. Alpaydin, E., *Introduction to Machine Learning*, 2nd Edition, MIT Press;
3. Marsland, S., *Machine Learning: An Algorithmic Perspective*; and
4. Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer.

Additional References

1. Hastie, T., Tibshirani, R. and Friedman, J., *The Elements of Statistical Learning*, Springer;
2. Géron, A., *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*, O'Reilly Media;
3. Murphy, K. P., *Machine Learning: A Probabilistic Perspective*, MIT Press; and Goodfellow, I., Bengio, Y. and Courville, A., *Deep Learning*, MIT Press.

PG2-COMP-3013: Elective -I

PG2-COMP-3014: Technical Seminar

Course Objectives

The objectives of this course are to enable students to:

- Develop the ability to study, analyze, and present advanced technical topics in Computer Science.
- Enhance technical communication, presentation, and report writing skills.
- Develop the ability to critically review recent research and emerging technologies.
- Improve professional communication and technical discussion skills.
- Foster independent learning and lifelong learning through technical literature.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Study and critically analyze advanced topics in Computer Science.
2. Prepare professional technical reports and seminar documents.

3. Deliver effective technical presentations using appropriate presentation tools.
4. Participate in technical discussions, defend ideas, and answer technical questions.
5. Develop independent learning skills through literature survey and self-study.
6. Demonstrate professional ethics and academic integrity in technical communication.

Practical / Seminar Syllabus (120 Hours)

The seminar shall be based on **advanced and emerging topics in Computer Science**. Seminar topics shall be announced by the Department from time to time in accordance with current research trends and technological developments.

The seminar shall include the following components:

1. Selection and approval of seminar topic.
2. Literature survey using journals, conference proceedings, technical reports, patents, standards, and other scholarly resources.
3. Critical review and analysis of the selected topic.
4. Preparation of seminar synopsis and technical report.
5. Scientific writing, citation, referencing, and plagiarism awareness.
6. Preparation of professional presentation slides and technical illustrations.
7. Oral seminar presentation using multimedia tools.
8. Demonstration, simulation, prototype, or case study wherever applicable.
9. Technical discussion, question-answer session, and defense of the presented work.
10. Participation in seminars presented by fellow students and peer evaluation.
11. Preparation of the final seminar report incorporating suggestions received during evaluation.

Guidelines for Evaluation

The evaluation may be based on:

- Topic selection and relevance
- Depth of literature survey
- Technical understanding and analysis
- Quality of seminar report
- Quality of presentation slides
- Presentation skills and communication
- Ability to answer questions and defend the work
- Participation in technical discussions
- Originality, academic integrity, and overall performance

Suggested References

1. Robert A. Day and Barbara Gastel, *How to Write and Publish a Scientific Paper*, Cambridge University Press.

2. Wayne C. Booth, Gregory G. Colomb, Joseph M. Williams, *The Craft of Research*, University of Chicago Press.
3. Kate L. Turabian, *A Manual for Writers of Research Papers, Theses, and Dissertations*, University of Chicago Press.
4. M. Ashraf Rizvi, *Effective Technical Communication*, McGraw-Hill Education.
5. Meenakshi Raman and Sangeeta Sharma, *Technical Communication: Principles and Practice*, Oxford University Press.

PG2-COMP-3015: Minor Project

Course Objectives

The objectives of this course are to enable students to:

- Apply the knowledge acquired during the programme to solve a real-world computing problem.
- Develop the ability to design, implement, test, and document a software or hardware solution.
- Enhance skills in literature survey, project planning, technical writing, and professional presentation.
- Prepare students for undertaking an independent Major Project in the subsequent semester.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Identify and formulate a computing problem suitable for independent study.
2. Design and develop a prototype or proof-of-concept solution using appropriate tools and technologies.
3. Conduct literature survey and analyze existing approaches related to the selected problem.
4. Prepare professional project documentation and present technical work effectively.
5. Demonstrate teamwork, project management, and ethical professional practices.
6. Plan and justify future extensions of the work toward a Major Project.

Project Guidelines (120 Hours)

The Minor Project shall be carried out individually or in a small group as approved by the Department. The project should involve the design and implementation of a software system, hardware prototype, computational model, algorithm, analytical study, or research-oriented investigation in an area of Computer Science.

The project shall include the following components:

1. Identification of the project problem and preparation of the project proposal.
2. Literature survey and study of existing techniques and related work.
3. Requirement analysis and project planning.
4. System design, algorithm development, or architectural design.
5. Prototype development and implementation.
6. Testing, validation, and performance evaluation.
7. Preparation of project documentation.
8. Demonstration of the developed system.
9. Seminar presentation and viva voce.
10. Preparation of future work for extension into the Major Project.

Guidelines and Regulations

1. **The project topic shall be approved by the Department before commencement of the work.**
2. **The topic selected for the Minor Project shall not be the same as the topic presented in the Technical Seminar course.** However, both may belong to the same broad research area with clearly different objectives and scope.
3. The Minor Project should preferably address a practical problem, prototype development, algorithm implementation, experimental study, or proof-of-concept relevant to Computer Science.
4. The scope of the Minor Project shall be limited so that it can be completed within the semester while leaving sufficient scope for further enhancement in the Major Project.
5. Students are encouraged to select project topics that can be **extended, improved, or experimentally evaluated in the subsequent Major Project**.
6. A comprehensive literature survey shall be carried out before implementation, and relevant research papers shall be cited appropriately.
7. Students shall maintain a project diary or progress log documenting weekly activities, design decisions, implementation progress, and testing.
8. The project shall follow good software engineering practices, including modular design, coding standards, version control, testing, and documentation.
9. The use of open-source libraries, frameworks, datasets, and AI-assisted development tools is permitted, provided their use is appropriately acknowledged and complies with applicable licenses and the policy of the University.
10. Plagiarism in code, documentation, datasets, or reports shall not be permitted. All submissions shall comply with the academic integrity policy of the University.
11. The final submission shall include:
    - Project report
    - Source code and supporting files
    - User manual (where applicable)
    - Installation and execution instructions
    - Presentation slides
    - Demonstration of the working system
12. The project shall be evaluated through periodic progress reviews, final report evaluation, software demonstration, presentation, and viva voce.

Suggested Evaluation Components **40 + 10 (I.A.)**

Component	Marks (40)
Project Proposal and Literature Survey	5
Design and Methodology	6
Implementation and Technical Quality	10
Testing, Evaluation and Result	4
Documentation and Report	5
Presentation and Demonstration	5
Viva Voce	5

Suggested References

The references shall depend on the selected project topic and may include:

1. IEEE Xplore Digital Library.
2. ACM Digital Library.
3. SpringerLink.
4. ScienceDirect.
5. arXiv.
6. Relevant textbooks, standards, technical reports, and software documentation related to the project domain.

PG2-COMP-4011: Deep Learning

Course Objectives

The objectives of this course are to enable students to:

- Understand the theoretical foundations of deep learning and artificial neural networks.
- Design, train, and evaluate deep neural network models for solving real-world problems.
- Apply convolutional, recurrent, and transformer-based architectures to image, text, and

sequential data.
- Utilize modern deep learning frameworks for model development and deployment.
- Analyze recent advances in deep learning and their applications across various domains.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Explain the principles and architectures of deep neural networks.
2. Design and train deep learning models for computer vision and natural language processing tasks.
3. Apply optimization, regularization, and transfer learning techniques to improve model performance.
4. Develop deep learning applications using TensorFlow or PyTorch.
5. Analyze modern deep learning architectures, including CNNs, RNNs, LSTMs, Transformers, and Generative AI models.
6. Evaluate deep learning models using appropriate performance metrics and ethical considerations.

Unit I: Foundations of Deep Learning (5 Hours)

Introduction to Artificial Intelligence, Machine Learning and Deep Learning. Biological neuron and artificial neuron. Perceptron and Multilayer Perceptron (MLP). Activation functions. Loss functions. Forward and backward propagation. Gradient Descent and Backpropagation. Deep Feedforward Neural Networks. Weight initialization. Model evaluation metrics.

Unit II: Optimization and Convolutional Neural Networks (5 Hours)

Optimization techniques including Stochastic Gradient Descent, Momentum, RMSProp and Adam optimizer. Batch normalization, dropout, regularization, hyperparameter tuning and early stopping. Convolution operation, pooling, padding, stride, feature extraction. CNN architectures including LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, DenseNet and EfficientNet. Transfer learning and fine tuning.

Unit III: Sequence Models and Transformer Architectures (5 Hours)

Sequential data processing. Recurrent Neural Networks (RNN). Backpropagation Through Time (BPTT). Long Short-Term Memory (LSTM). Gated Recurrent Units (GRU). Encoder-Decoder architecture. Attention mechanism. Transformer architecture. Self-attention. Multi-head attention. Positional encoding. Encoder-only and decoder-only Transformers. BERT, GPT, T5 and Vision Transformers (ViT).

Unit IV: Large Language Models and Generative AI (6 Hours)

Introduction to Foundation Models and Large Language Models (LLMs). Tokenization and embeddings. Pre-training objectives. Scaling laws. Fine-tuning techniques including parameter-efficient fine tuning (PEFT), LoRA and adapters. Instruction tuning. Reinforcement Learning from Human Feedback (RLHF). Prompt engineering. In-context learning. Retrieval-Augmented Generation (RAG). Hallucination and grounding. Autoencoders, Variational Autoencoders (VAE), Generative Adversarial Networks (GANs), Diffusion Models and multimodal foundation models.

Unit V: Applications and Emerging Trends (5 Hours)

Deep learning applications in computer vision, natural language processing, speech recognition, recommendation systems, healthcare, bioinformatics, cybersecurity, finance and autonomous systems. Explainable AI (XAI). Model interpretability. AI ethics, fairness, bias and privacy. Federated learning. Model compression, pruning and quantization. Edge AI. Current research trends in deep learning and Generative AI.

Unit VI: Deep Learning Frameworks and Model Deployment (4 Hours)

TensorFlow, Keras and PyTorch frameworks. Hugging Face Transformers ecosystem. Model training pipelines. Dataset preparation and augmentation. Model deployment using ONNX, TensorFlow Lite and TorchScript. GPU computing basics. Introduction to MLOps and deployment of deep learning models.

Tutorial Syllabus (15 Hours)

1. Discussion on neural network architectures, activation functions, loss functions and backpropagation with illustrative examples.
2. Guided problem-solving on optimization algorithms, regularization methods and hyperparameter tuning.
3. Discussion on CNN architectures, transfer learning and computer vision applications.
4. Discussion on RNNs, LSTMs, attention mechanisms and Transformer architectures.
5. Guided discussion on Large Language Models including BERT, GPT, LLaMA, Gemma, Mistral and DeepSeek; prompt engineering, fine-tuning, RLHF and Retrieval-Augmented Generation (RAG).
6. Discussion on Generative AI models including GANs, VAEs, Diffusion Models and multimodal AI systems.
7. Guided analysis of recent research papers, ethical issues, Explainable AI, trustworthy AI and emerging trends in deep learning.

Practical Syllabus (30 Hours)

1. Introduction to TensorFlow, Keras and PyTorch.
2. Implementation of Perceptron and Multilayer Perceptron.
3. Image classification using Feedforward Neural Networks.
4. Optimization using SGD, RMSProp and Adam.
5. Handwritten digit recognition using the MNIST dataset.
6. Image classification using Convolutional Neural Networks.
7. Transfer learning using ResNet, VGG16 or EfficientNet.
8. Sequence prediction using Recurrent Neural Networks.
9. Time-series prediction using LSTM/GRU.
10. Text classification using a pre-trained BERT model.
11. Prompt engineering using a pre-trained Large Language Model.
12. Fine-tuning Transformer models using the Hugging Face Transformers library.
13. Retrieval-Augmented Generation (RAG) using LangChain/LlamaIndex and vector databases.
14. Image generation using Autoencoders, GANs or Diffusion Models.
15. Mini project on Deep Learning, Generative AI or Large Language Models.

Text / Reference Books

1. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press.
2. François Chollet, *Deep Learning with Python*, 2nd Edition, Manning Publications.
3. Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd Edition, O'Reilly Media.
4. Charu C. Aggarwal, *Neural Networks and Deep Learning*, Springer.
5. Simon J. D. Prince, *Understanding Deep Learning*, MIT Press.
6. Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, *Dive into Deep Learning*, Cambridge University Press.
7. Jay Alammar and Maarten Grootendorst, *Hands-On Large Language Models*, O'Reilly Media.
8. Lewis Tunstall, Leandro von Werra, and Thomas Wolf, *Natural Language Processing with Transformers*, O'Reilly Media.
9. Denis Rothman, *Transformers for Natural Language Processing*, Packt Publishing.
10. Sebastian Raschka, *Build a Large Language Model (From Scratch)*, Manning Publications.
11. Christopher M. Bishop and Hugh Bishop, *Deep Learning: Foundations and Concepts*, Springer.
12. Kevin P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press.

PG2-COMP-4012: Distributed Systems

Total Credits: 4
Total Contact Hours: 75 Hours

Course Objectives

The course aims to provide students with a clear and beginner-friendly understanding of distributed systems and the principles used to design, implement and manage them. It introduces distributed-system characteristics, architectural models, communication mechanisms, clock synchronization, event ordering, coordination algorithms, distributed transactions, concurrency control, deadlock handling, distributed shared memory, naming services and security. The lecture component develops the conceptual foundation of distributed computing, the tutorial component strengthens analytical understanding through algorithms and problem-solving exercises, and the practical component provides hands-on experience in developing concurrent and distributed applications using the Go programming language.

Course Learning Outcomes

On successful completion of the course, students will be able to achieve the following outcomes:

- Explain the basic concepts, characteristics, objectives, benefits and challenges of distributed systems;
- Distinguish among centralized, client-server, multi-tier, peer-to-peer, layered, event-based and service-oriented architectures;
- Describe communication mechanisms such as message passing, sockets, Remote Procedure Calls and REST-based communication;
- Apply physical-clock and logical-clock synchronization techniques;
- Analyse distributed mutual-exclusion and election algorithms;
- Explain distributed transactions and apply suitable concurrency-control techniques;
- CO7 identify distributed deadlocks and describe prevention, detection and recovery methods;

- Explain distributed shared-memory architectures and consistency models;
- CO9 describe naming systems, name resolution, caching and the Domain Name System;
- Identify common security threats and basic protection mechanisms in distributed systems;
- Develop concurrent client-server and distributed applications using Go; and
- Select appropriate distributed-system techniques for solving practical computing problems.

Lecture

Lecture Credit: 2
Total Lecture Hours: 30 Hours

Unit 1: Fundamentals of Distributed Systems

This unit introduces distributed systems through definitions, characteristics and real-life examples. Centralized and distributed systems are compared, and the main goals of distributed computing, including resource sharing, transparency, openness, scalability, reliability, availability, performance and fault tolerance, are discussed. Different forms of transparency, such as access, location, migration, replication, concurrency and failure transparency, are introduced. The unit also explains heterogeneity, security, communication delays, partial failures and the absence of a global clock as major challenges in distributed computing. Hardware and software concepts, distributed operating systems, network operating systems, middleware and commonly provided distributed services are covered, together with applications such as cloud computing, online banking, distributed databases, content-delivery networks and large-scale web platforms. (4 Hrs)

Unit 2: Distributed-System Architectures and Communication

This unit introduces the major architectural styles used in distributed systems, including client-server, multi-tier, peer-to-peer, hybrid, layered, object-based, data-centred, event-based and service-oriented architectures. A basic overview of microservices and their relationship with distributed-system design is included. The unit explains processes, threads and inter-process communication, followed by synchronous and asynchronous message passing, blocking and non-blocking communication, buffering and message delivery. Socket-based communication, Remote Procedure Calls, Remote Method Invocation, message-oriented middleware and basic REST-based communication are discussed. The unit also covers request-response interaction, serialization, JSON-based messaging, communication failures, retries, duplicate requests, timeout handling and the need for idempotent operations in distributed applications. (5 Hrs)

Unit 3: Time, Coordination and Distributed Algorithms

This unit explains why time and event ordering are difficult to manage in distributed systems. Physical clocks, clock drift, clock skew and clock-synchronization requirements are introduced, followed by Cristian's algorithm, the Berkeley algorithm and an overview of the Network Time Protocol. Logical time is discussed through Lamport logical clocks, vector clocks, the happened-before relationship, concurrent events and causal ordering. Distributed mutual exclusion is introduced through safety, liveness, fairness and fault-tolerance requirements. The centralized mutual-exclusion algorithm, Lamport's algorithm, the Ricart–Agrawala algorithm and the token-ring algorithm are studied and compared in terms of message complexity, synchronization delay and failure behaviour. The unit also introduces leader-election problems and covers the bully algorithm and ring election algorithm. (6 Hrs)

Unit 4: Distributed Transactions, Concurrency Control and Deadlocks

This unit introduces transactions, transaction states and the ACID properties of atomicity, consistency, isolation and durability. Local and distributed transactions are distinguished, and transaction coordinators, participants and failure conditions are discussed. The two-phase commit protocol is studied through its prepare and commit phases, along with coordinator failure, participant failure and blocking problems. A conceptual overview of the three-phase commit protocol is also included. The unit introduces serial and concurrent execution, schedules, conflict serializability and recoverability. Lock-based concurrency control, shared and exclusive locks, two-phase locking, timestamp ordering and optimistic concurrency control are explained. Distributed deadlocks are discussed using wait-for graphs and resource-allocation graphs, followed by deadlock prevention, avoidance, centralized and distributed detection and recovery through transaction termination or resource pre-emption. (6 Hrs)

Unit 5: Distributed Shared Memory, Naming and Security

This unit introduces distributed shared memory as a mechanism for providing a shared-memory abstraction across networked computers. Its motivation, architecture, data location, data access and page-based and object-based implementations are discussed. Consistency models, including strict, sequential, causal and eventual consistency, are explained, together with a basic overview of replication and consistency management. The naming section introduces names, identifiers, addresses, name spaces, hierarchical naming, name resolution, name servers and name caching. The Domain Name System is studied through domain hierarchy, DNS records, recursive and iterative resolution and caching. The security section introduces authentication, authorization, access control, confidentiality, integrity and availability. Secure communication, encryption, digital signatures, identity verification and common threats such as eavesdropping, spoofing, replay attacks, denial-of-service attacks and unauthorized access are discussed along with basic protection mechanisms. (5 Hrs)

Unit 6: Distributed Computation Using Go

This unit introduces the Go programming language as a practical platform for developing concurrent and distributed applications. It covers program structure, packages, variables, data types, control statements, functions, arrays, slices, maps, structures, interfaces and error handling. Go concurrency is introduced through goroutines, channels, buffered and unbuffered communication, channel direction, select statements, wait groups and mutexes. Concurrent-task execution, race conditions and synchronization are discussed. The unit introduces TCP client-server programming, basic HTTP client and server development, JSON encoding and decoding and handling multiple requests concurrently. Timeout management, context-based cancellation and error handling are covered. Remote Procedure Call support in Go is introduced, and the design of a small distributed service combining communication, concurrency and failure handling is discussed. (4 Hrs)

Tutorial

Tutorial Credit: 1
Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Distributed-System Concepts and Architectural Selection

This tutorial unit includes comparative exercises on centralized and distributed systems and the identification of transparency, scalability, reliability, availability and fault-tolerance requirements in practical applications. Students will analyse application scenarios and select appropriate client-server, multi-tier, peer-to-peer, layered, event-based, service-oriented or microservice-based architectures, providing clear reasons for their selections. (2 Hrs)

Tutorial Unit 2: Communication and Failure Handling

This tutorial unit includes exercises on synchronous and asynchronous communication, blocking and non-blocking message passing, sockets, Remote Procedure Calls, Remote Method Invocation and REST-based communication. Students will trace request-response sequences, identify possible communication failures and prepare simple strategies for timeouts, retries, duplicate-message handling and idempotent request processing. (2 Hrs)

Tutorial Unit 3: Clock Synchronization, Logical Time and Coordination

This tutorial unit includes numerical exercises on clock drift, clock skew and synchronization using Cristian's and Berkeley algorithms. Students will assign Lamport timestamps and vector timestamps to distributed events, determine happened-before relationships and identify concurrent events. Exercises on centralized, Lamport, Ricart–Agrawala and token-ring mutual-exclusion algorithms will be undertaken, together with message-complexity comparisons and step-by-step execution of bully and ring election algorithms. (3 Hrs)

Tutorial Unit 4: Distributed Transactions, Concurrency and Deadlocks

This tutorial unit includes transaction-state analysis, preparation of two-phase commit message sequences and examination of coordinator and participant failures. Students will analyse schedules for serializability and apply locking, two-phase locking, timestamp ordering and optimistic concurrency-control techniques. Exercises on wait-for graphs, deadlock identification, prevention, detection and recovery will also be included. (3 Hrs)

Tutorial Unit 5: Shared Memory, Consistency, Naming and Security

This tutorial unit includes comparative exercises on strict, sequential, causal and eventual consistency. Students will select suitable consistency models for distributed applications such as collaborative editing, banking, social networking and replicated content services. Naming exercises will cover hierarchical name spaces, DNS records, recursive resolution, iterative resolution and caching. Security case studies will include authentication, authorization, replay attacks, spoofing, data tampering and denial-of-service threats. (2 Hrs)

Tutorial Unit 6: Go Concurrency and Distributed-Application Design

This tutorial unit includes dry-run exercises on goroutines, channels, buffered channels, select statements, wait groups and mutexes. Students will analyse concurrent program traces, identify race conditions and suggest suitable synchronization methods. They will also prepare communication flows and basic designs for TCP, HTTP, JSON-based and RPC-based distributed applications developed using Go. (3 Hrs)

Practical

Practical Credit: 1
Hours per Practical Credit: 30 Hours
Total Practical Hours: 30 Hours

The practical component will primarily use the Go programming language. Suitable supporting tools may include the Go compiler and toolchain, Visual Studio Code or another development environment, Git, command-line networking tools and local or virtual network environments.

Practical 1: Go Programming Fundamentals

Students will install and configure the Go development environment and develop programs using variables, data types, control statements, functions, arrays, slices, maps, structures, interfaces, packages and error handling. The practical will establish the programming foundation required for later concurrent and distributed applications. (3 Hrs)

Practical 2: Goroutines and Concurrent Task Execution

Students will develop programs that execute multiple functions concurrently using goroutines. The behaviour of sequential and concurrent execution will be compared, and suitable timing or logging mechanisms will be used to observe the execution order of independent tasks. (3 Hrs)

Practical 3: Channels, Synchronization and Race-Condition Handling

Students will implement communication among goroutines using buffered and unbuffered channels. Wait groups and mutexes will be used for synchronization and shared-data protection. Race conditions will be demonstrated, detected using suitable Go tools and corrected through safe synchronization techniques. (3 Hrs)

Practical 4: TCP Client-Server Application

Students will develop a basic TCP server and client using Go. The client will send a request to the server, and the server will process the request and return a response. Connection establishment, reading, writing, connection closure and basic network-error handling will be implemented. (3 Hrs)

Practical 5: Concurrent Multi-Client Server

Students will extend the TCP server to handle multiple clients concurrently using goroutines. Client connections will be processed independently, and the application will include safe shared-state handling, timeout management and graceful connection termination. (3 Hrs)

Practical 6: HTTP and JSON-Based Distributed Service

Students will develop an HTTP server exposing basic service endpoints and a client that communicates with the server. Request parameters, response status codes, JSON encoding and decoding and concurrent-request handling will be implemented. Basic validation, timeout handling and error responses will also be included. (3 Hrs)

Practical 7: Remote Procedure Call Using Go

Students will develop a simple RPC server and client using Go's RPC facilities. A remote service will expose one or more methods, and the client will invoke those methods as though they were locally available. Students will examine request serialization, method registration, remote invocation and error handling. (3 Hrs)

Practical 8: Logical Clock and Event-Ordering Simulation

Students will implement a simulation of Lamport logical clocks for multiple communicating processes. Events and message exchanges will update local timestamps, and the final event order will be displayed. The practical may be extended to vector clocks to identify causal and concurrent events. (3 Hrs)

Practical 9: Coordination, Election or Transaction Simulation

Students will implement a simplified distributed coordination algorithm such as the bully election algorithm, ring election algorithm or token-based mutual exclusion. Alternatively, a two-phase commit

simulation may be developed with one coordinator and multiple participants. The program will display state transitions, exchanged messages and the result under normal and selected failure conditions. (3 Hrs)

Practical 10: Go-Based Distributed Application

Students will develop a small distributed application such as a concurrent key-value service, distributed task processor, file-metadata service, chat system or replicated information service. The application will combine goroutines, channels, network communication, JSON or RPC messaging, timeout handling, synchronization and basic fault handling. The final submission will include source code, system architecture, communication flow, testing results and a short demonstration. (3 Hrs)

Reference Books

Primary References

1. Tanenbaum, A. S. and Van Steen, M., *Distributed Systems: Principles and Paradigms*, Pearson;
2. Coulouris, G., Dollimore, J., Kindberg, T. and Blair, G., *Distributed Systems: Concepts and Design*, Pearson; and
3. Donovan, A. A. A. and Kernighan, B. W., *The Go Programming Language*, Addison-Wesley.

Additional References

1. Tanenbaum, A. S. and Bos, H., *Modern Operating Systems*, Pearson;
2. Kshemkalyani, A. D. and Singhal, M., *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press;
3. Ghosh, S., *Distributed Systems: An Algorithmic Approach*, CRC Press; and Burns, B., *Designing Distributed Systems*, O'Reilly Media.

PG2-COMP-4013: Elective-II

PG2-COMP-4014: Seminar & Grand Viva

Course Objectives

The objectives of this course are to enable students to:

- Demonstrate comprehensive understanding of Computer Science concepts acquired during the programme.
- Present and defend the outcomes of the Major Project effectively before an expert panel.
- Develop professional communication, technical presentation, and scientific discussion skills.
- Integrate knowledge from different areas of Computer Science to analyze and solve complex problems.
- Prepare students for higher studies, research, and professional careers.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Present and defend technical and research work in a professional manner.
2. Demonstrate comprehensive knowledge of core and advanced Computer Science subjects.
3. Critically analyze and justify design decisions, methodologies, and experimental results.
4. Communicate technical ideas effectively through oral presentations and discussions.
5. Demonstrate professional ethics, independent learning, and problem-solving abilities.
6. Respond confidently to technical questions through a comprehensive viva voce examination.

Syllabus (120 Hours)

The Seminar & Grand Viva shall be conducted during the final semester of the programme and shall consist of comprehensive technical presentations, project defense, and oral examination covering both the Major Project and the overall curriculum.

The course shall include the following components:

1. Preparation of seminar based on the Major Project and related research work.
2. Preparation of professional presentation slides, technical illustrations, and demonstration materials.
3. Presentation of the Major Project before the Departmental Evaluation Committee.
4. Demonstration and defense of the developed software, hardware, prototype, or research outcome.
5. Discussion of project objectives, methodology, implementation, experimental evaluation, limitations, and future scope.
6. Critical review of related literature and comparison with existing approaches.
7. Technical discussion on recent developments and emerging trends related to the project area.
8. Grand Viva Voce covering the core and elective subjects studied during the M.Sc. Computer Science programme.
9. Assessment of professional communication, technical competence, analytical ability, and problem-solving skills.

Guidelines

1. Every student shall deliver a seminar based on the Major Project carried out during the fourth semester.
2. The seminar presentation shall include:
    - Introduction and motivation
    - Problem statement
    - Literature review
    - Objectives
    - Methodology
    - System design or algorithm
    - Experimental results and performance evaluation
    - Conclusions
    - Future scope
3. The student shall demonstrate the developed system, software, hardware prototype, or research outcome wherever applicable.
4. The seminar report shall be submitted in the Department-prescribed format before the presentation.
5. The Grand Viva Voce shall assess the student's understanding of:
    - Major Project
    - Core courses of the M.Sc. programme
    - Elective courses
    - Current trends in Computer Science

    - Professional and research ethics
6. Students shall be expected to justify design decisions, implementation choices, algorithms, and experimental results during the viva examination.
7. Academic integrity shall be maintained in all reports, presentations, demonstrations, and discussions.
8. The Seminar & Grand Viva shall ordinarily be conducted by an Examination Committee consisting of Internal and External Examiners appointed by the University.

Suggested Evaluation Scheme

Component	**Weight (40) + 10 I.A.**
Seminar Report	5
Quality of Presentation	10
Technical Content and Analysis	10
Grand Viva Voce	15
Total	**40**

Suggested References

The references shall depend on the student's Major Project and may include:

1. IEEE Xplore Digital Library.
2. ACM Digital Library.
3. SpringerLink.
4. ScienceDirect.
5. arXiv.
6. Standard textbooks, technical manuals, software documentation, and research publications relevant to the project area.

Relationship with Other Courses

- **Technical Seminar (Semester III):** Focuses on the study and presentation of an emerging topic in Computer Science, independent of the project work.
- **Minor Project (Semester III):** Involves the development of a prototype or proof-of-concept and preparation for the Major Project.
- **Major Project (Semester IV):** Involves complete design, implementation, experimentation, and documentation.
- **Seminar & Grand Viva (Semester IV):** Serves as the **culminating academic assessment**, where students present and defend their Major Project and demonstrate comprehensive knowledge gained throughout the M.Sc. programme.

This progression ensures that the four courses complement one another without overlap and provide a

coherent pathway from topic exploration to project execution and final comprehensive evaluation.

PG2-COMP-4015: Major Project

Course Objectives

The objectives of this course are to enable students to:

- Undertake an independent project to solve a significant problem in Computer Science.
- Apply theoretical knowledge and practical skills for the design, implementation, testing, and evaluation of a complete computing solution.
- Develop research aptitude through literature survey, experimentation, technical analysis, and scientific documentation.
- Demonstrate professional competence through project presentation, software demonstration, and technical defense.
- Prepare students for research, higher studies, and professional careers.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Identify and formulate a significant problem in Computer Science and propose an appropriate solution.
2. Design, develop, implement, and evaluate a software, hardware, or research-based system.
3. Conduct a comprehensive literature survey and apply suitable methodologies to solve the selected problem.
4. Prepare a professional project dissertation following standard scientific writing practices.
5. Present and defend the project before an expert evaluation committee.
6. Demonstrate independent learning, project management, ethical practices, and technical competence.

Syllabus (120 Hours)

The Major Project shall be carried out individually or in a group, as approved by the Department, under the supervision of a faculty member. The project shall involve the complete lifecycle of a software, hardware, computational, or research-oriented problem from conception to implementation and evaluation.

The project may be an extension of the Minor Project undertaken in the previous semester or a new project approved by the Department with adequate technical depth and scope.

Break-up of 120 Hours

Activity	Hours
• Problem identification, project planning, and requirement analysis	**10**

- Literature survey and review of related work **15**
- System design, algorithm design, or architecture development **15**
- Software development / implementation / prototype development **40**
- Testing, debugging, validation, performance evaluation, and result analysis **15**
- Dissertation writing, technical documentation, and user manual preparation **10**
- Seminar presentation, software demonstration, project defense, and final revisions **15**

Project Guidelines

1. The project topic shall be approved by the Department before the commencement of the work.
2. The Major Project may be an extension of the Minor Project carried out in the previous semester with substantial enhancement in objectives, methodology, implementation, experimentation, and evaluation, or it may be a new project approved by the Department.
3. The project shall address a significant problem in Computer Science and demonstrate originality in design, implementation, experimentation, or application.
4. Every project shall be carried out under the supervision of a faculty member assigned by the Department.
5. Students shall submit a project proposal containing the objectives, methodology, work plan, and expected outcomes before commencing the implementation.
6. Students shall maintain a **Project Diary/Progress Log Book** recording weekly progress, meetings with the supervisor, implementation details, testing, and future plans.
7. A comprehensive literature survey shall be conducted using books, journals, conference proceedings, patents, standards, and other scholarly resources.
8. The project shall follow standard software engineering or research methodologies, including requirement analysis, system design, implementation, testing, validation, and documentation.
9. Students are encouraged to use appropriate project management practices, version control systems (such as Git), issue tracking, coding standards, and software testing methodologies.
10. The use of open-source software, libraries, frameworks, datasets, and AI-assisted development tools is permitted, provided their usage is properly acknowledged and complies with applicable licenses and academic integrity policies.
11. All project reports shall follow the dissertation format prescribed by the Department. Proper citation and referencing shall be maintained, and plagiarism shall be within the permissible limits prescribed by the University.
12. The final submission shall include:
    - Project dissertation
    - Source code
    - Executable software or prototype (where applicable)
    - User Manual
    - Installation Guide
    - Test results and performance analysis
    - Presentation slides
13. Every student shall deliver a **seminar presentation** on the Major Project before the Department and demonstrate the developed system (where applicable). The presentation shall be followed by a **technical defense (Project Viva Voce)** before the Examination Committee.

14. The project shall demonstrate the student's ability to independently analyze, design, implement, evaluate, and document a complete computing solution.

Evaluation Scheme

Component	Marks
Internal Assessment	**10**
End Semester Examination	**40**
Dissertation Evaluation	10
Project Seminar Presentation	10
Software Demonstration	10
Project Viva Voce	10
Total	**50**

Suggested References

The references shall depend upon the selected project topic and may include:

1. IEEE Xplore Digital Library.
2. ACM Digital Library.
3. SpringerLink.
4. ScienceDirect.
5. arXiv.
6. Relevant books, standards, software documentation, technical manuals, and research publications related to the selected project area.

PG2-COMP-3013: Elective -I-A: Advanced Algorithms

Course Objectives

The objectives of this course are to enable students to:

- Study advanced data structures and sophisticated algorithmic techniques for solving complex computational problems.
- Analyze and design efficient algorithms using advanced paradigms such as branch-and-bound, approximation, and cache-aware computation.
- Apply advanced graph, string, computational geometry, and optimization algorithms to practical applications.
- Develop the ability to analyze algorithmic trade-offs in terms of time, space, scalability, and computational complexity.
- Introduce students to emerging trends in algorithm design and engineering.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Design efficient algorithms using advanced algorithmic paradigms.
2. Select suitable advanced data structures for large-scale computational problems.
3. Apply computational geometry and advanced graph algorithms to real-world applications.
4. Analyze optimization problems using approximation, greedy, dynamic programming, and branch-and-bound techniques.
5. Implement advanced string algorithms for bioinformatics and information retrieval.
6. Evaluate modern algorithmic techniques for high-performance computing and large-scale data processing.

Theory Syllabus (30 Hours)

Unit I: Advanced Data Structures (6 Hours)

Threaded Binary Trees. Self-balancing trees. AVL Trees. Heap data structures. Binary Heap, Binomial Heap, Fibonacci Heap (overview), Pairing Heap (overview). Double-ended Heap. Priority Queues and their implementations. Heap Sort. Applications of advanced tree and heap data structures. Augmenting Data structure.

Unit II: Advanced Graph Algorithms and Optimization (5 Hours)

Single-source shortest path algorithms. All-pairs shortest path algorithms. Johnson's Algorithm. Matching, Maximum Matching in Bipartite Graphs. Hungarian Algorithm (overview). Network optimization. Branch-and-Bound technique. Advanced Greedy algorithms. Advanced Dynamic Programming techniques.

Unit III: Computational Geometry and Approximation Algorithms (5 Hours)

Introduction to Computational Geometry. Convex Hull algorithms (Graham Scan and Jarvis March). Line Segment Intersection. Closest Pair of Points. Sweep Line technique. Delaunay Triangulation and its dual. Approximation algorithms. Approximation ratio. Greedy approximation algorithms. Vertex cover. Ub set sum and its applications. Set Cover approximation. Traveling Salesman approximation. Introduction to PTAS and FPTAS.

Unit IV: Advanced String Algorithms and Bioinformatics (7 Hours)

Advanced string processing. Trie and Suffix Trie. Suffix Array and Suffix Tree (overview). Z-Algorithm. Aho-Corasick multiple pattern matching. Sequence Alignment. Edit Distance. Levenshtein Distance. Needleman-Wunsch Algorithm. Smith-Waterman Algorithm. Gap models and Gap Penalty.

Unit V: Modern Algorithmic Techniques (7 Hours)

Cache-aware and Cache-Oblivious Algorithms. Randomized Algorithms. Amortized Analysis (advanced applications). Online Algorithms. Introduction to Parallel algorithms. Polynomial operations, DFT, FFT.

Tutorial Syllabus (15 Hours)

1. Discussion on threaded trees, AVL trees, heap structures, and priority queue implementations.
2. Guided analysis of shortest path, all-pairs shortest path, matching, and network optimization algorithms.
3. Discussion on advanced dynamic programming, branch-and-bound, and approximation algorithms.
4. Problem-solving using computational geometry algorithms including convex hull and line segment intersection.
5. Discussion on sequence alignment, edit distance, Needleman-Wunsch algorithm, gap penalty, and bioinformatics applications.
6. Discussion on cache-oblivious algorithms, randomized algorithms,

---

Practical Syllabus (30 Hours)

1. Implementation of Threaded Binary Tree.

2. Implementation of AVL Tree.
3. Implementation of Binary Heap and Priority Queue.
4. Heap Sort implementation and performance analysis.
5. Implementation of Double-ended Heap.
6. Single-source shortest path using Dijkstra and Bellman-Ford algorithms.
7. All-pairs shortest path using Floyd-Warshall algorithm.
8. Bipartite graph matching implementation.
9. Branch-and-Bound solution for the Traveling Salesman Problem or 0/1 Knapsack.
10. Convex Hull using Graham Scan.
11. Line Segment Intersection algorithm.
12. Sequence Alignment using Needleman-Wunsch Algorithm.
13. Edit Distance and Levenshtein Distance implementation.
14. Trie and Aho-Corasick pattern matching.
15. Comparative performance analysis of advanced algorithms on benchmark datasets.
16. Polynomial DFT, FFT

Total Practical Hours: 30

Text / Reference Books

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 4th Edition, MIT Press.
2. Steven S. Skiena, *The Algorithm Design Manual*, 3rd Edition, Springer.
3. Jon Kleinberg and Éva Tardos, *Algorithm Design*, Pearson.
4. Mark Allen Weiss, *Data Structures and Algorithm Analysis in C++*, Pearson.
5. Donald E. Knuth, *The Art of Computer Programming*, Volumes 1–3, Addison-Wesley.
6. Dan Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press.
7. Joseph O'Rourke, *Computational Geometry in C*, 2nd Edition, Cambridge University Press.
8. Vijay V. Vazirani, *Approximation Algorithms*, Springer.
9. David P. Williamson and David B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press.
10. D. Eppstein, *Computational Geometry*, Lecture Notes (for supplementary reading).
11. Phillip Rogaway and Thomas Shrimpton (selected papers on algorithm engineering) and recent survey articles on cache-oblivious and streaming algorithms (supplementary reading).

PG2-COMP-3013: Elective -I-B: Quantum Computing

Objectives

v Introduce the basic principles of quantum computing.
v Develop understanding of qubits, quantum gates, and quantum circuits.
v Familiarize students with quantum algorithms and applications.
v Provide hands-on exposure to quantum programming tools.
v Encourage creative and research-oriented thinking.

Course Outcomes (CO)

Upon successful completion of this course, students will be able to:

. Explain the fundamental principles of quantum computing, including qubits, superposition, entanglement, quantum measurement, and the Bloch sphere, and distinguish quantum computing from classical computing.

. Design and analyze quantum circuits using quantum gates such as Pauli, Hadamard, phase, and controlled gates, and demonstrate the preparation of Bell states and basic quantum communication protocols.

. Analyze and implement fundamental quantum algorithms, including Deutsch, Deutsch–Jozsa, Grover's Search, Quantum Fourier Transform, and explain the significance of Shor's algorithm and quantum computational complexity.

. Develop and execute quantum programs using Python and the Qiskit framework, simulate quantum circuits, and evaluate the effects of noise by comparing simulation results with executions on IBM Quantum hardware.

. Apply quantum computing concepts to emerging application domains such as quantum cryptography, quantum key distribution, quantum machine learning, artificial intelligence, finance, and quantum communication systems, while evaluating future challenges and ethical considerations.

Syllabus (Theory)

Credit 3 **45 hours**

UNIT I: Introduction to Quantum Computing

Recommended Teaching Hours: 05 Hours

- l Limitations of classical computing
- l Introduction to quantum computing
- l Basic principles of quantum mechanics
- l Qubits
- l Superposition and measurement
- l Entanglement and quantum states
- l Bloch sphere

UNIT II: Quantum Gates and Quantum Circuits

Recommended Teaching Hours: 10 Hours

- l Quantum gates
- l Pauli gates
- l Phase shift gates
- l Hadamard gate
- l Controlled gates (CNOT)
- l Quantum circuits and circuit representation
- l Bell states and quantum teleportation

UNIT III: Quantum Algorithms

Recommended Teaching Hours: 10 Hours

- l Grover's search algorithm
- l Deutsch algorithm
- l Deutsch–Jozsa algorithm
- l Quantum Fourier Transform
- l Introduction to Shor's algorithm
- l Classical vs quantum complexity

UNIT IV: Quantum Programming and Simulators

Recommended Teaching Hours: 10 Hours

- l Introduction to Python for quantum computing
- l Qiskit framework
- l Quantum simulators
- l Running circuits on IBM Quantum platform
- l Quantum circuit implementation
- l Noise in quantum systems

UNIT V: Modern Applications and Emerging Trends

Recommended Teaching Hours: 10 Hours

- l Quantum cryptography
- l Quantum key distribution
- l Quantum machine learning
- l Quantum communication systems
- l Quantum computing in AI and finance
- l Ethical and future challenges of quantum computing

Quantum Computing (Tutorial)

Credit: 1

Teaching

Hours: 15 Hours

UNIT I Tutorials: Introduction to Quantum Computing

.

Compare classical bits

and quantum bits (qubits) through illustrative examples.

.

Represent qubit states

using Dirac notation and visualize them on the Bloch sphere.

.

Solve numerical

problems involving probability amplitudes and quantum measurement.

.

Analyze the effects of

superposition and entanglement using simple examples.

.

Discuss the

limitations of classical computing and identify problems suitable for quantum computation.

UNIT II Tutorials: Quantum Gates and Quantum Circuits

.

Construct single-qubit

quantum circuits using Pauli, Hadamard, and Phase gates.

- Perform matrix-based calculations to determine the outputs of quantum gates.
- Design simple quantum circuits using CNOT gates.
- Construct Bell states and explain quantum entanglement.
- Analyze the steps involved in quantum teleportation using circuit diagrams.

UNIT III Tutorials: Quantum Algorithms

- Trace the execution of Deutsch and Deutsch–Jozsa algorithms using circuit representations.
- Solve simple search problems using Grover's algorithm.
- Demonstrate the basic working principle of the Quantum Fourier Transform.
- Compare classical and quantum computational complexity using example problems.
- Discuss the significance of Shor's algorithm in modern cryptography.

UNIT IV Tutorials: Quantum Programming and Simulators

- Write simple Python programs involving matrices, vectors, and complex numbers.
- Construct and simulate basic quantum circuits using Qiskit.
- Compare outputs obtained from quantum simulators and IBM Quantum hardware.
- Analyze the effects of measurement and noise on quantum computations.
- Interpret circuit visualizations and quantum state outputs.

UNIT V Tutorials: Modern Applications and Emerging Trends

- Study case examples of quantum cryptography and Quantum Key Distribution (QKD).
- Compare classical cryptographic techniques with quantum cryptographic methods.
- Discuss the role of quantum computing in machine learning, artificial intelligence, finance, and optimization.
- Analyze recent research developments in quantum computing.
- Debate ethical,

security, and societal challenges arising from large-scale quantum computing.

Syllabus (Practical)

Credit 2 **60 hours**

l **Introduction to Python Programming**

- Variables, data types, operators
- Conditional statements and loops
- Functions and modules
- Lists, tuples, dictionaries
- Basic NumPy operations

l **Python for Scientific and Quantum Computing**

- Matrix and vector operations
- Complex numbers in Python
- Data visualization using Matplotlib
- Basic linear algebra operations

l **Installation and Introduction to Qiskit**

- Setting up Python and Qiskit
- Understanding Qiskit environment
- Introduction to IBM Quantum Experience

l **Creation of Qubits and Basic Quantum States**

- Single qubit representation
- Superposition states
- Multi-qubit systems

l **Implementation of Basic Quantum Gates**

- Pauli gates
- Hadamard gate
- Phase gates
- Controlled gates (CNOT)

l **Quantum Circuit Design and Simulation**

- Creating simple quantum circuits
- Circuit visualization
- Measurement operations

l **Bell State Generation**

- Entanglement creation
- Bell circuit implementation

l **Simple Quantum Random Number Generator**

- Random qubit measurement
- Quantum randomness analysis

l **Running Programs on IBM Quantum Experience**

- Executing circuits on real quantum hardware
- Comparing simulator and hardware outputs
- Understanding noise effects

PG2-COMP-4013: Elective-II-A: Advanced Cryptography and Network Security

Course Objectives

The objectives of this course are to enable students to:

- Understand advanced cryptographic constructions and security proofs.
- Design secure cryptographic protocols for modern distributed applications.
- Analyze vulnerabilities in web-based applications and networked systems.
- Implement advanced cryptographic algorithms and protocols using modern software libraries.
- Understand post-quantum cryptography and emerging cryptographic technologies.
- Perform security assessment of web applications and modern cryptographic systems.

Course Learning Outcomes

After successful completion of the course, the students will be able to:

1. Analyze advanced mathematical foundations and cryptographic algorithms used in modern security systems.
2. Design and evaluate secure public-key, symmetric-key, and elliptic curve cryptographic schemes.
3. Analyze advanced cryptographic protocols, digital signatures, and secure key establishment mechanisms.
4. Explain the principles of post-quantum cryptography and emerging cryptographic technologies.
5. Evaluate the security of web applications and cryptographic implementations against modern attacks.
6. Implement advanced cryptographic algorithms and security protocols using appropriate software tools.

Theory Syllabus (30 Hours)

Unit I: Advanced Number Theory and Algebraic Foundations (3 Hours)

Review of modular arithmetic and congruence relations. Euler's theorem, Carmichael function, Chinese Remainder Theorem and its applications in cryptography. Primitive roots, cyclic groups, finite abelian groups and group homomorphisms. Polynomial rings, finite field extensions, construction of extension fields $GF(p^n)$, irreducible and primitive polynomials, field isomorphisms, polynomial arithmetic over finite fields, efficient modular arithmetic, Montgomery multiplication, Barrett reduction, fast modular exponentiation algorithms, Tonelli-Shanks algorithm for modular square roots, Hensel lifting, advanced number theoretic algorithms, pseudo-random number generators, LFSR and RC4.

Unit II: Advanced Symmetric Cryptography (5 Hours)

Advanced principles of block cipher design including Feistel networks and substitution-permutation networks. Wide trail strategy and diffusion properties. AES, other symmetric cryptographic techniques, differential cryptanalysis, linear cryptanalysis, integral cryptanalysis, impossible differential attacks, boomerang attacks, biclique attacks, algebraic cryptanalysis, security evaluation of symmetric ciphers, side-channel attacks, timing attacks, cache attacks, power analysis, fault attacks and white-box cryptography.

Unit III: Advanced Public Key Cryptography and RSA (8 Hours)

Security foundations of RSA cryptosystems, multi-prime RSA, multi-power RSA, batch RSA and CRT-RSA. Advanced attacks on RSA, integer factorization algorithms, generation of strong primes and secure parameter selection.

ElGamal cryptosystem and signatures. Baby-Step Giant-Step algorithm, Pollard's Rho algorithm, Index Calculus, Schnorr signatures, BLS signatures, blind signatures, ring signatures, group signatures, aggregate signatures, threshold signatures, identity-based signatures and provable security.

Unit IV: Advanced Elliptic Curve Cryptography (6 Hours)

Mathematical foundations of elliptic curves over real numbers and finite fields. Group law, point addition, point doubling, scalar multiplication algorithms, projective and Jacobian coordinate systems, window methods, Montgomery ladder, Edwards curves, Curve25519, NIST and Brainpool curves, ECDH, ECDSA.

Unit V: Advanced Cryptographic Protocols (4 Hours)

Authenticated key exchange, authenticated key establishment, Sigma protocols, interactive proof systems, zero-knowledge proofs, Fiat-Shamir transformation, Shamir's Secret Sharing, verifiable secret sharing, threshold cryptography, secure multi-party computation, oblivious transfer, commitment schemes, coin flipping protocols, subset sum problem, Merkle-Hellman knapsack cryptosystem, cryptanalysis of knapsack cryptosystems, lattice attacks and applications of NP-complete problems in cryptography.

Unit VII: Advanced Network and Web Security (2 Hours)

Network security architecture and layered defense. Secure communication protocols including SSL/TLS 1.3, HTTPS, IPSec, SSH, and VPNs. Public Key Infrastructure (PKI), certificate authorities, certificate validation, certificate pinning, and Online Certificate Status Protocol (OCSP). Web security principles, same-origin policy, Content Security Policy (CSP), Cross-Origin Resource Sharing (CORS), session management, cookies, secure authentication, OAuth 2.0, OpenID Connect, JSON Web Tokens (JWT),

common web application attacks including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), clickjacking, server-side request forgery (SSRF), secure coding practices, Web Application Firewall (WAF), and an overview of the OWASP Top 10 vulnerabilities.

Unit VIII: Post-Quantum Cryptography and Emerging Topics (2 Hours)

Quantum computing and its impact on cryptography, Shor's algorithm, Grover's algorithm, homomorphic encryption, fully homomorphic encryption, lattice-based cryptography, NTRU cryptosystem, GGH cryptosystem, lattice-based digital signatures and current research directions.

Tutorial Syllabus (15 Hours)

1. Discussion on advanced number theory, finite fields, modular arithmetic, polynomial arithmetic and efficient modular computation.
2. Guided discussion on modern symmetric cryptographic algorithms, block cipher design principles and cryptanalysis techniques.
3. Problem-solving sessions on RSA variants, integer factorization algorithms, elliptic curve arithmetic and public-key cryptography.
4. Discussion on advanced digital signature schemes, discrete logarithm algorithms and cryptographic security proofs.
5. Discussion on authenticated key exchange, zero-knowledge proofs, secret sharing schemes and secure multi-party computation.
6. Guided discussion on subset sum cryptography, lattice-based attacks and post-quantum cryptographic algorithms.
7. Discussion on emerging cryptographic technologies, web security challenges and recent advances in modern cryptography.
8. Discussion on SSL/TLS, PKI, certificate management, HTTPS security, OWASP Top 10 vulnerabilities, secure authentication mechanisms, and contemporary web security challenges.

Practical Syllabus (30 Hours)

1. Implement Montgomery Multiplication.
2. Implement Barrett Reduction.
3. Implement Miller-Rabin and Baillie-PSW primality tests.
4. Implement Pollard's Rho factorization algorithm.
5. Implement Baby-Step Giant-Step algorithm.
6. Implement Pollard's Rho algorithm for discrete logarithm.
7. Implement elliptic curve point addition and point doubling.
8. Implement scalar multiplication algorithms.
9. Implement Elliptic Curve Diffie-Hellman (ECDH).

10. Implement Elliptic Curve Digital Signature Algorithm (ECDSA).
11. Implement Schnorr Digital Signature.
12. Implement Shamir Secret Sharing Scheme.
13. Construct Merkle Trees.
14. Implement SHA-3 and Sponge construction.
15. Implement a toy Learning With Errors (LWE)-based encryption scheme.
16. Demonstrate Kyber key encapsulation mechanism.
17. Perform differential cryptanalysis on a toy cipher.
18. Demonstrate timing attacks on cryptographic implementations.
19. Simulate side-channel analysis attacks.
20. Configure and analyze SSL/TLS communication using OpenSSL and inspect digital certificates.

Text / Reference Books

1. Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, 3rd Edition, CRC Press.
2. Douglas R. Stinson and Maura Paterson, *Cryptography: Theory and Practice*, CRC Press.
3. William Stallings, *Cryptography and Network Security: Principles and Practice*, Pearson Education.
4. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman, *An Introduction to Mathematical Cryptography*, 2nd Edition, Springer.
5. Christof Paar and Jan Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer.
6. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press.
7. Neal Koblitz, *A Course in Number Theory and Cryptography*, 2nd Edition, Springer.
8. Alfred Menezes, *Elliptic Curve Public Key Cryptosystems*, Springer.
9. Oded Goldreich, *Foundations of Cryptography*, Volumes I & II, Cambridge University Press.
10. Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons.
11. Nigel Smart, *Cryptography Made Simple*, Springer.
12. Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen (Eds.), *Post-Quantum Cryptography*, Springer.
13. Wenbo Mao, *Modern Cryptography: Theory and Practice*, Pearson Education.

PG2-COMP-4013: Elective-II-B: Blockchain Technology

Total Credits: 4
Total Contact Hours: 75 Hours
Course Objectives

The course is designed to help students understand how blockchain systems work, why they are considered trustworthy in decentralized environments, and where they can be applied effectively. It covers the foundations of distributed ledgers, cryptographic techniques, blockchain architecture, cryptocurrency systems, consensus mechanisms, permissioned and permissionless networks, Bitcoin, Ethereum, Hyperledger Fabric, smart contracts and blockchain security. The course also encourages students to examine practical blockchain applications in finance, supply chains, healthcare, identity management, government and big-data environments while developing the ability to assess the benefits, limitations and risks of blockchain-based solutions.

Course Learning Outcomes

On successful completion of the course, students will be able to achieve the following outcomes:

- Explain the fundamental concepts, architecture, components and operating principles of blockchain systems;
- Describe the use of cryptographic hashing, public-key cryptography, digital signatures and Merkle trees in blockchain networks;
- Explain the architecture and transaction-processing mechanisms of Bitcoin, Ethereum and related cryptocurrency ecosystems;
- Compare major consensus mechanisms and select suitable approaches for different blockchain applications;
- Distinguish between permissioned and permissionless blockchain networks;
- CO6 analyse the architecture, components and transaction flow of Hyperledger Fabric;
- Assess blockchain applications in finance, industry, healthcare, government and other sectors;
- Identify common attacks, vulnerabilities and protection mechanisms associated with blockchain platforms, wallets and smart contracts;
- Examine current research challenges involving scalability, interoperability, privacy, governance and sustainability; and
- Explain how blockchain can be integrated with big-data platforms for secure data sharing, provenance, storage, integrity and analytics.

Lecture

Lecture Credit: 2

Total Lecture Hours: 30 Hours

Unit 1: Blockchain Foundations, Cryptography and Cryptocurrency Systems

This unit introduces the evolution of distributed ledgers and the need for blockchain technology in environments where participants may not fully trust one another. It explains blockchain architecture through blocks, transactions, chains, nodes, peer-to-peer communication, distributed ledgers, timestamps, nonces and block validation. The cryptographic foundations of blockchain are covered through hash functions, collision resistance, public-key cryptography, private and public keys, digital signatures, address generation and Merkle trees. The unit introduces Bitcoin architecture, transaction structure, the Unspent Transaction Output model, wallets, mining, block creation, transaction verification, incentives and the wider cryptocurrency ecosystem. It also provides an introductory view of Ethereum, accounts, gas, smart contracts and decentralized applications. The purpose of consensus in distributed networks is discussed through Proof of Work, Proof of Stake and other commonly used consensus approaches, together with basic ideas of forks, confirmation and finality. (8 Hrs)

Unit 2: Permissioned Blockchains and Hyperledger Fabric

This unit introduces permissioned blockchain systems and explains why organisations may prefer controlled participation, identifiable members, restricted access and governed transaction validation. Permissioned and permissionless blockchains are compared in terms of access, identity, consensus, privacy, performance, governance and application areas. The unit introduces the Hyperledger ecosystem and studies Hyperledger Fabric in detail through organisations, peers, orderers, channels, certificate authorities, Membership Service Providers, policies, endorsement mechanisms and chaincode. Fabric's ledger structure, world state, transaction proposal, endorsement, ordering, validation, commitment and event-notification stages are explained. The unit also discusses private data, channel-based confidentiality, chaincode deployment, policy configuration and the practical role of Hyperledger Fabric in enterprise and consortium blockchain solutions. (7 Hrs)

Unit 3: Blockchain Applications and Use Cases

This unit explores how blockchain may be used beyond cryptocurrency. Applications in banking, cross-border payments, digital currencies, insurance, trade finance, asset management and tokenization are discussed. Industrial use cases include supply-chain visibility, product traceability, logistics,

provenance verification, counterfeit prevention and automated settlement. Applications in healthcare, identity management, education, digital certificates, Internet of Things and digital-rights management are also examined. Government-oriented use cases include electronic governance, public-record management, land and property records, public distribution, voting systems, licensing and citizen-service delivery. For each area, students will examine the business problem, stakeholders, trust assumptions, data flow and role of smart contracts. The unit also discusses practical limitations such as legal acceptance, privacy, interoperability, cost, scalability, data quality, governance, integration with legacy systems and the difficulty of correcting inaccurate off-chain information. (7 Hrs)

Unit 4: Blockchain Security, Emerging Research and Big-Data Integration

This unit introduces the security goals of blockchain systems, including integrity, availability, authenticity, non-repudiation and resistance to unauthorized modification. Common threats such as majority or 51-percent attacks, double spending, Sybil attacks, selfish mining, eclipse attacks, denial-of-service attacks, transaction malleability and consensus manipulation are discussed. Private-key loss, wallet compromise, phishing, insecure key storage and exchange-related risks are examined. Smart-contract vulnerabilities such as re-entrancy, access-control failures, arithmetic errors, front running, improper validation and insecure external calls are introduced along with testing, auditing and secure-development practices. Security concerns in Bitcoin, Ethereum and permissioned blockchain systems are compared. The unit also covers current research challenges involving scalability, transaction throughput, latency, interoperability, cross-chain communication, privacy, governance, regulation, sustainability and quantum-resistant cryptography. The final part examines the relationship between blockchain and big-data technologies, including blockchain-based data sharing, provenance, integrity verification, access control, distributed storage and trusted analytics, while also discussing the limitations of storing large datasets directly on a blockchain. (8 Hrs)

Tutorial

Tutorial Credit: 1

Total Tutorial Hours: 15 Hours

Tutorial Unit 1: Blockchain Structure, Cryptography and Transactions

This tutorial unit includes exercises on block structure, hash calculation, hash-linked chains, Merkle-tree construction, public and private keys, digital signatures and transaction verification. Students will trace the life cycle of a Bitcoin transaction, interpret simple transaction records, examine the Unspent Transaction Output model and compare different wallet types. Worked problems will also cover mining difficulty, nonce selection, block confirmation and the basic operation of Proof of Work and Proof of Stake. (4 Hrs)

Tutorial Unit 2: Consensus and Permissioned Blockchain Architecture

This tutorial unit includes comparative analysis of Proof of Work, Proof of Stake, Practical Byzantine Fault Tolerance, Proof of Authority and other suitable consensus mechanisms. Students will select an appropriate consensus approach for given public, private and consortium blockchain scenarios. Exercises will also cover the identification of peers, orderers, organisations, channels, certificate authorities, endorsement policies and Membership Service Providers in a Hyperledger Fabric network, followed by step-by-step tracing of Fabric's transaction flow. (4 Hrs)

Tutorial Unit 3: Blockchain Application Design

This tutorial unit includes case-based exercises in which students identify whether blockchain is suitable for a proposed application. They will analyse stakeholders, trust boundaries, data ownership, privacy needs, transaction volume, governance rules and integration requirements for applications in banking, supply chains, healthcare, identity management, government records and the Internet of Things. Students will prepare simple blockchain solution models and distinguish between information that should be stored on-chain and information that should remain off-chain. (3 Hrs)

Tutorial Unit 4: Security, Smart Contracts and Research Challenges

This tutorial unit includes the analysis of attack scenarios involving double spending, majority attacks, Sybil attacks, selfish mining, compromised private keys and vulnerable smart contracts. Students will review small smart-contract examples, identify unsafe logic and suggest suitable controls. Exercises will also examine scalability, privacy, interoperability, sustainability and blockchain–big-data integration. Students will compare proposed solutions and discuss the technical and practical trade-offs associated with each approach. (4 Hrs)

Practical

Practical Credit: 1
Hours per Practical Credit: 30 Hours
Total Practical Hours: 30 Hours
The practical component may be conducted using Python, JavaScript, Solidity, Remix IDE, Ganache, MetaMask, Web3 libraries, Hyperledger Fabric test networks or other suitable blockchain-development platforms. The selected tools may vary according to the available laboratory environment.

Practical 1: Cryptographic Hashing and Block Linking
Students will implement cryptographic hashing for sample data and examine how minor changes in the input produce different hash values. A simple block structure containing an index, timestamp, transaction data, previous-block hash, nonce and current hash will be created. Multiple blocks will be linked to form a basic blockchain, and attempts to modify previous block data will be used to demonstrate tamper evidence. (3 Hrs)

Practical 2: Digital Signatures and Transaction Verification
Students will generate public and private key pairs, sign a sample transaction and verify its signature using a suitable cryptographic library. The practical will demonstrate transaction authenticity, integrity and non-repudiation. Students will also examine the consequences of modifying the signed message or using an incorrect verification key. (3 Hrs)

Practical 3: Merkle Tree Construction
Students will develop a program to calculate transaction hashes and construct a Merkle tree. The Merkle root will be generated for a set of transactions, and the effect of modifying one transaction will be observed. A simple Merkle proof may also be prepared to demonstrate how transaction inclusion can be verified without processing the complete block. (3 Hrs)

Practical 4: Simulation of Mining and Proof of Work
Students will implement a basic Proof-of-Work mining process by varying a nonce until the block hash satisfies a specified difficulty condition. Mining time under different difficulty levels will be measured and compared. The practical will help students understand computational effort, block validation, mining difficulty and the relationship between security and resource consumption. (3 Hrs)

Practical 5: Cryptocurrency Transaction and Wallet Analysis
Students will create or use a local test wallet, examine account addresses, balances and transaction details, and perform a simulated cryptocurrency transfer on a test or local blockchain network. They will inspect transaction hashes, sender and receiver addresses, block numbers, confirmation status and transaction fees without using real financial assets. (3 Hrs)

Practical 6: Smart-Contract Development
Students will write and deploy a simple smart contract using Solidity or an equivalent smart-contract language. The contract may manage a value, voting record, certificate, asset or simple transaction. Students will compile, deploy and interact with the contract, invoke its functions and observe changes in the contract state through a local or test blockchain environment. (3 Hrs)

Practical 7: Decentralized Application Interaction
Students will connect a simple web or command-line application to a deployed smart contract using an appropriate Web3 library. The application will read blockchain data, submit transactions and display transaction results. The practical will demonstrate the relationship among a user interface, wallet, smart contract, blockchain node and transaction confirmation process. (3 Hrs)

Practical 8: Hyperledger Fabric Network and Chaincode
Students will set up or use a preconfigured Hyperledger Fabric test network containing organisations, peers, an ordering service and a channel. They will deploy sample chaincode, invoke transactions and query the ledger. The transaction path from proposal and endorsement to ordering, validation and commitment will be observed through network logs and command outputs. (3 Hrs)

Practical 9: Blockchain Security and Smart-Contract Testing
Students will analyse a smart contract containing selected security weaknesses such as missing access control, unsafe external interaction, re-entrancy risk or inadequate input validation. Suitable fixes will be applied and tested. The practical may use static-analysis, testing or auditing tools to identify vulnerabilities and compare the contract's behaviour before and after correction. (3 Hrs)

Practical 10: Blockchain Application Case Study
Students will design a small blockchain-based solution for a real-life problem such as supply-chain traceability, academic-certificate verification, healthcare-record consent, land-record management or digital identity. The work will include problem definition, stakeholders, blockchain type, consensus approach, data model, smart-contract logic, on-chain and off-chain data selection, security considerations and a basic prototype or demonstration. (3 hrs)

Reference Books

Primary References

1. Narayanan, A., Bonneau, J., Felten, E., Miller, A. and Goldfeder, S., *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, 2016;
2. Antonopoulos, A. M., *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*; and Nakamoto, S., *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Additional References

1. Wood, G., *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, Ethereum Yellow Paper, 2014;
2. Atzei, N., Bartoletti, M. and Cimoli, T., "A Survey of Attacks on Ethereum Smart Contracts";
3. Bashir, I., *Mastering Blockchain*; and Drescher, D., *Blockchain Basics: A Non-Technical Introduction in 25 Steps*.