CO3091 Computational Intelligence and Software Engineering

Credits: *20* **Convenor:** *Dr. L. Minku* **Semester:** *1st*

Prerequisites:	*Essential: CO1003,CO1005*		*Desirable: CO2006, CO2015*		
Assessment:	*Coursework: 40%*		*Three hour exam in January: 60%*		
Lectures:	*29*	*hours*	**Problem Classes:**	*5*	*hours*
Surgeries:	*6*	*hours*	**Class Tests:**	*0*	*hours*
Laboratories:	*5*	*hours*	**Private Study:**	*105*	*hours*

Subject Knowledge

Aims Computational intelligence is a field of artificial intelligence concerned with so-called heuristic algorithms, which aim to produce good solutions to problems in a reasonable amount of time. These algorithms are widely used for several real world applications, e.g., routing problems; assignment and scheduling problems; medical, biomedical and bioinformatics problems; forecasting problems; etc. More recently, they have also started to be used to help solving software engineering problems. In particular, due to the increased size and complexity of software systems, software engineering tasks such as software project planning, software testing and maintenance have become increasingly time consuming and error prone. Computational intelligence techniques can be used as decision support tools in order to produce higher quality software faster, helping to overcome the challenges posed by large and complex software systems. This module explains computational intelligence approaches that can be used for solving problems from several different domains. It also explores the synergies between computational intelligence and software engineering, explaining how computational intelligence approaches can be used to help solving software engineering problems.

Learning Outcomes By completion of this module, students should be able to: recognise which software engineering problems can be formulated as computational intelligence optimisation or machine learning problems; formulate such software engineering problems as optimisation or machine learning problems; demonstrate an understanding of the core techniques used in the computational intelligence approaches to solve such problems; communicate such core techniques to software engineers; build models able to support software engineering learning tasks; use optimisation algorithms to support software engineering optimisation problems; evaluate, analyse and critique computational intelligence approaches for software engineering. It is also expected that students will be able to use their knowledge to non-software engineering problems.

Methods Class sessions together with lecture slides; recommended book chapters, articles and research papers; web resources; worksheets.

Assessment Marked coursework and written examination.

Skills

Aims Students will learn how to: research current issues in computational intelligence for software engineering; write reports; solve problems; and use computational intelligence toolboxes.

Learning Outcomes Upon completion of this module, students should be able to: research a given topic using a variety of sources including books, current articles and research papers; write reports; and use computational intelligence toolboxes.

Methods Class sessions; lab sessions; articles and research papers; and web resources.

Assessment Marked coursework and written examination.

Explanation of Prerequisites Coursework and lab sessions involving programming will require CO1003 and CO1005, besides experience of programming in Java. The module will give a computational intelligence perspective to some of the problems introduced in CO2006 and CO2015. Having taken these modules is not required for CO3091, but would provide useful background knowledge.

Course Description Computational intelligence is a field of artificial intelligence concerned with so-called heuristic algorithms, which aim to produce good solutions to problems in a reasonable amount of time. These algorithms are widely used for several real world applications, e.g., routing problems; assignment and scheduling problems; medical, biomedical and bioinformatics problems; forecasting problems; etc. More recently, they have also started to be used to help solving software engineering problems. In particular, software systems have become ever larger and more complex, making software engineering tasks such as software project planning, software testing and maintenance increasingly costly and error prone. Many of these tasks can be formulated as problems that can be solved with the help of computational intelligence. If properly designed, computational intelligence approaches can be used as decision support tools to increase software development cost-effectiveness and improve software quality.

This module teaches computational intelligence approaches that can be applied to problems from several different domains. It also explores the synergies between computational intelligence and software engineering. It explains (1) software engineering problems that can be solved with the help of computational intelligence, (2) when and why computational intelligence is important for solving these problems, (3) how to formulate these problems in order to use computational intelligence approaches to solve them, (4) what computational intelligence approaches can be used to solve these problems, (5) how these approaches work and (6) how to use these approaches.

In particular, this module explains both optimisation and machine learning approaches. Optimisation problems are concerned with finding a solution that achieves one or more pre-defined goals. For example, software test case generation can be seen as the problem of finding a set of test cases that maximise code coverage and minimise testing effort. Computational intelligence algorithms such as evolutionary algorithms inspired by Darwin's evolutionary theory can be used to solve this problem, saving the time to create a good set of test cases manually and helping to improve software quality.

Machine learning problems are concerned with creating models able to infer knowledge from data. For example, models can be created to learn the relationship between software code metrics and information on whether the corresponding piece of code contains bugs or not. These models can then be used to predict whether new pieces of code are likely to contain bugs, providing an insight into what pieces of code require more or less testing effort. Computational intelligence algorithms such as naive bayes can be used to solve this problem.

Detailed Syllabus

Examples of computational intelligence approaches covered by this module are:

- Optimisation approaches: evolutionary algorithms and multi-objective evolutionary algorithms.
- Machine learning approaches: k-nearest neighbour, decision trees and naive-bayes.

Examples of software engineering problems covered by this module are:

- Requirements engineering: requirements selection.
- Project management: software effort estimation and software project scheduling.
- Development and maintenance: code optimisation for non-functional requirements.
- Testing and debugging: software defect prediction and software test suite generation.

This module will also explain procedures that can be used to evaluate the suitability of computational intelligence approaches to a given problem.

Reading List

[B] Russel, S.; Norvig, P.; John F. Canny, *Artificial Intelligence – A Modern Approach, Section 4.1: Local Search Algorithms and Optimization Problems, 3rd Edition (New International Edition),* Pearson Education, 2014.

[B] Michel, B.; Mancoridis, S., *Using Heuristic Search Techniques To Extract Design Abstractions From Source Code,* Proceedings of the Genetic and Evolutionary Computation Conference, Pages 1375-1382, 2002. Read section 3 until the end of section 3.1.1.

[B] Menzies, T.; Kocaguneli, E.; Minku, L.; Peters, F. and Turhan, B., *Sharing Data and Models in Software Engineering, Chapters 7 (Data Mining and Software Engineering), 8 (Defect Prediction), 9 (Effort Estimation), 10.10 (Extensions [of Decision Trees] to Continuous Classes),* Morgan Kaufmann, 2014.

[B] Eiben, A.; Smith, J., *Introduction to Evolutionary Computing, Chapters 1–8,* Springer, 2003.

[B] Turhan, B.; Menzies, T.; Bener, A.; Di Stefano, J., *On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction, Section 3.2 (Naive Bayes Classifier),* Empirical Software Engineeing 14:540578, 2009.

[B] Fraser, G.; Arcuri, A., *Whole Test Suite Generation,* IEEE Transactions on Software Engineering 39(2):276-291, 2013.

[B] Minku, L.; Sudholt, D.; Yao, X., *Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis,* IEEE Transactions on Software Engineering 40(1):83-102, 2014.

[B] Linares-Vasquez, M.; Bavota, G.; Cardenas, C.; Oliveto, R.; Di Penta, M.; Poshyvanyk, D., *Optimizing Energy Consumption of GUIs in Android Apps: A Multi-objective Approach,* ACM SIGSOFT Symposium on the Foundations of Software Engineering, p. 143-153, 2015.

[B] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T., *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,* IEEE Transactions on Evolutionary Computation, 6(2):182-197, 2002.

[B] Chen, W.; Zhang, J., *Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler,* IEEE Transactions on Software Engineering 39(1):1-17, 2013.

[B] Rosen, C.; Grawi, B.; Shihab, E., *Commit Guru: Analytics and Risk Prediction of Software Commits,* Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, p. 966-969, 2015.

[C] Le Goues, C.; Nguyen, T.; Forrest, S.; Weimer, W., *GenProg: A Generic Method for Automatic Software Repair,* IEEE Transactions on Software Engineering 38(1):54-72, 2012.

[C] Shin Yoo, *Evolving Human Competitive Spectra-Based Fault Localisation Techniques,* Symposium on Search Based Software Engineering, Lecture Notes in Computer Science 7515:244-258, 2012.

[C] Harman, M.; Jones, B., *Search-based software engineering,* Information and Software Technology 43(14):833-839, 2001.

[C] Fraser, G.; Arcuri, A., *A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite,* ACM Transactions on Software Engineering and Methodology 24(2):8, 2014.

[C] Le Goues, C.; Dewey-Vogt, M.; Forrest, S.; Weimer, W., *A Systematic Study of Automated Program Repair: Fixing 55 out of 105 bugs for $8 Each,* International Conference on Software Engineering, p. 3-13, 2012.

[C] Kamei, Y.; Shihab, E.; Adams, B.; Hassan, A.; Mockus, A.; Sinha, A.; Ubayashi, N., *A Large-Scale Empirical Study of Just-In-Time Quality Assurance,* IEEE Transactions on Software Engineering 39(6):757-773, 2013.

[C] Eiben, A.; Smith, J., *Introduction to Evolutionary Computing,* Springer, 2003.

Resources Lecture slides; books, articles and research papers; web page; study guide; worksheets; lecture rooms with projectors; labs with projectors; open source computational intelligence toolboxes.

Module Evaluation Course questionnaires, course review.