SURVEY TENTANG OPTIMASI KOLONI SEMUT (ANT COLONY OPTIMIZATION) PADA PENGUJIAN PERANGKAT LUNAK

Salma Luna Eka Puteri, M. Rafi Raihan Rizal dan Fenni Agustina
Universitas Gunadarma
Jl. Margonda Raya No. 100, Depok, Jawa Barat 16424
salmaluna48@gmail.com, rafi.raihanrizal@yahoo.co.id, fenni@staff.gunadarma.ac.id

ABSTRAK

Ant Colony Optimization (ACO) adalah teknik meta-heuristik yang terkenal dan berkembang pesat. Sejumlah besar masalah optimasi telah mengambil manfaat dari teknik ACO begitu pula lainnya yang sedang menggunakannya saat ini tak terhitung jumlahnya. Sejumlah upaya juga telah dilakukan oleh para peneliti untuk menerapkan ACO dalam menyelesaikan berbagai masalah pengujian perangkat lunak. Artikel ini menyajikan survei dari sembilan studi tersebut, kami mengdentifikasi hal yang berkaitan dengan penggunaan ACO dalam konsep pengujian perangkat lunak yang beragam. Pada akhirnya, sembilan studi telah dianalisis dengan teliti untuk menemukan beberapa parameter umum yang dapat dikelompokkan bersama atau dibandingkan untuk memberikan wawasan yang berguna.

Kata Kunci : *ACO, pengujian, perangkat lunak, optimasi*

PENDAHULUAN

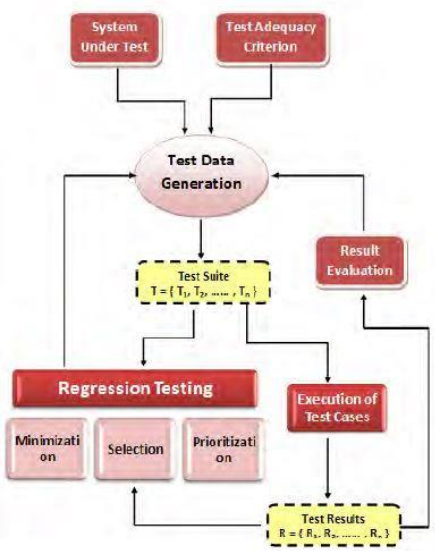
Pengujian perangkat lunak adalah tahap investigasi yang dilakukan untuk memberikan para pemangku kepentingan informasi tentang kualitas produk atau layanan yang diuji [1]. Pengujian perangkat lunak memberikan pandangan yang objektif dan independen terhadap perangkat lunak, sehingga bisnis dapat menghargai dan memahami risiko yang terlibat dalam implementasi perangkat lunak. Teknik pengujian menggabungkan (tetapi tidak terbatas pada) proses eksekusi program dengan tujuan untuk menemukan *bug* perangkat lunak, sehingga *bug* tersebut dapat diperbaiki. Pengujian tidak dapat memastikan bahwa perangkat lunak akan berfungsi dengan baik dalam semua kondisi; melainkan hanya dapat menentukan bahwa produk akan beroperasi dengan baik dalam situasi yang spesifik [2]. Beberapa survei dapat ditemukan di bidang pengujian perangkat lunak. Tinjauan sistematis pertama dalam pengujian perangkat lunak dipublikasikan di 2004 tentang *The Testing Technique Experiments* [3]. Baru-baru ini, tinjauan literatur sistematis secara khusus tentang proses Seleksi Uji Regresi dibuat oleh Ding, Kau, Li dan Yang [4]. Yoo dan Harman [5] telah mempresentasikan survei yang sangat ketat tentang Minimisasi, Seleksi, dan Prioritas Uji Regresi. Survei berbasis kuesioner umum pada Praktik Pengujian Regresi juga diterbitkan pada 2010 oleh Engstrőm dan Rumerson [6].

Ada berbagai langkah yang termasuk dalam proses pengujian perangkat lunak dan dapat diwakili dengan cara yang berbeda. Terinspirasi dari [1], langkah-langkah utama yang terlibat dalam proses pengujian perangkat lunak telah digambarkan pada Gambar 1. Pada Gambar 1, terdapat bagan yang menunjukkan sistem yang sedang diuji dan beberapa kriteria kecukupan uji yang harus dipenuhi setelah pengujian. Kriteria kecukupan pengujian umumnya didasarkan pada jenis input yang diberikan untuk proses pengujian perangkat lunak. Kriteria ini termasuk cakupan struktural, cakupan kesalahan atau beberapa kriteria gabungan. Kasus uji kemudian dihasilkan berdasarkan kriteria tersebut. Sub-proses dari *test data generation* menghasilkan sejumlah kasus uji yang membentuk *test suite. Test suite* ini kemudian dapat dieksekusi untuk menghitung hasil tes atau dapat dimasukkan sebagai input untuk teknik pengujian regresi.

ACO adalah pendekatan meta-heuristik yang terinspirasi dari perilaku koloni semut. Pendekatan ini memanfaatkan kemampuan semut untuk menemukan jalur terpendek dari sarangnya ke sumber makanan dengan bantuan fenomena yang disebut *stigmergy*. Berasal dari perilaku semut yang nyata ini, Dorigo mengusulkan *Ant Colony Optimization* (ACO) pada 1990-an [7].

Teknik ini telah diterapkan untuk menyelesaikan berbagai masalah optimisasi kombinatorial termasuk masalah *knapsack*, *travelling salesman*, jaringan terdistribusi, *data mining*, jaringan telekomunikasi, perutean kendaraan, pembuatan data uji [7, 8, 9, 10] dan lain-lain. Tahun 2003 menandai dimulainya penerapan ACO untuk menyelesaikan berbagai konsep pengujian perangkat lunak [11, 12] dan telah berkembang sejak saat itu. Pada akhirnya, makalah ini mencoba untuk mengelompokkan berbagai aplikasi dan efek ACO ketika diterapkan pada berbagai bidang dalam pengujian perangkat lunak.

Dalam jurnal survei ini, penulis mencoba menganalisis beberapa penelitian yang telah menggunakan teknik ACO dalam setiap langkah proses pengujian perangkat lunak. Bagian selanjutnya memberikan rincian yang telah diakumulasikan dari penelitian ini, seperti pada Gambar 1.



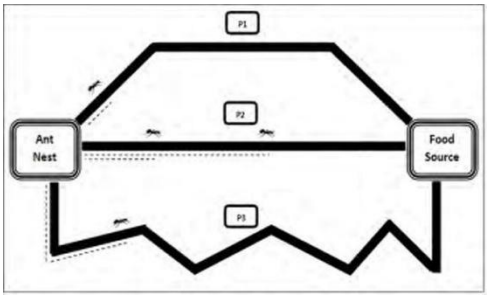
Gambar 1. *Proses Pengujian Perangkat Lunak*

TINJAUAN PUSTAKA

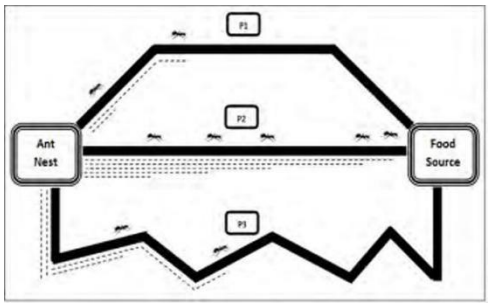
Ahli entomologi telah menemukan bahwa kekuatan semut berada di otak koloni mereka [7]. ACO adalah metode pencarian berbasis populasi untuk menyelesaikan berbagai masalah optimisasi kombinatorial. ACO didasarkan pada konsep *stigmergy* - komunikasi tidak langsung antara anggota populasi melalui interaksi dengan lingkungan. Contoh *stigmergy* adalah komunikasi semut selama proses pencarian makan: semut secara tidak langsung berkomunikasi satu sama lain dengan mendepositkan jejak feromon di tanah dan dengan demikian memengaruhi proses keputusan semut lain. Konsep dasar koloni semut:

1) Semut berjalan kesana kemari dari sarang ke sumber makanan sambil menyebarkan zat yang disebut "feromon" di jalurnya.
2) Semut lain dapat mencium bau feromon ini dan mempengaruhi pilihan jalurnya, karena semut cenderung mengikuti konsentrasi feromon yang lebih kuat.
3) Feromon yang disimpan di tanah membentuk jejak feromon. Hal ini memungkinkan semut lain menemukan sumber makanan yang sebelumnya telah diidentifikasi oleh semut koloni mereka.
4) Feromon terus menguap dengan sejalannya waktu pada tingkat penguapan tertentu.
5) Semut yang memilih jalur yang lebih pendek akan menjadi yang pertama untuk kembali ke sarang. Hal ini disebabkan oleh probabilitas yang sangat tinggi dari semut tersebut untuk memilih jalur yang lebih pendek yang sama pada perjalanan pulangnya (karena memiliki jejak feromon yang lebih kuat).
6) Oleh karena itu, setelah beberapa waktu, semua semut koloni berkumpul untuk mengikuti jalur terpendek.

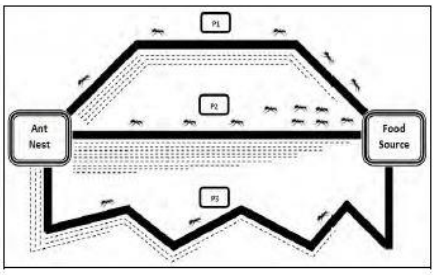
Hal yang sama digambarkan pada Gambar 1 - 4. Contoh ini menunjukkan tiga jalur yang mungkin dilewati dari sarang semut ke sumber makanan (P1, P2 dan P3). Dari tiga jalur, P2 adalah yang terpendek (Gambar 1 - 4). Awalnya (Gambar 1) semut memilih jalurnya secara acak dan menyimpan feromon (ditunjukkan di gambar di bawah jalur yang diikuti oleh semut). Kemudian, banyak semut mulai berangkat dari sarang. Sementara itu proses konvergensi ke jalur terpendek telah dimulai (Gambar. 2 - 3). Akhirnya, semua semut bertemu di jalur terpendek yang juga memiliki jumlah maksimum feromon yang tersimpan di dalamnya (Gambar 4).



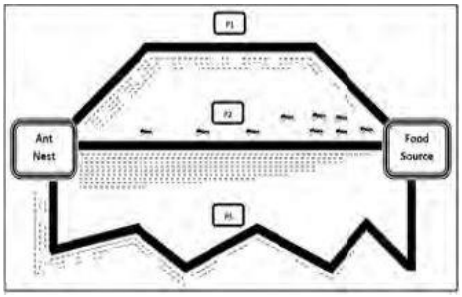
Gambar 2. *Cara kerja dari ACO, awalnya 4 semut mulai dari sarang.*



Gambar 3. *Cara kerja dari ACO, semakin banyak semut menyimpan banyak feromon di berbagai jalur.*



Gambar 4. *Cara kerja dari ACO, beberapa semut yang memulai perjalanan dari sarangnya, sekarang menyatu menuju jalur yang lebih pendek.*



Gambar 5. *Cara kerja dari ACO, akhirnya semua semut bersatu untuk mengikuti jalur terpendek.*

HASIL DAN PEMBAHASAN

Langkah-langkah utama dari proses pengujian perangkat lunak meliputi pembuatan urutan pengujian atau jalur pengujian, yang pada gilirannya membantu dalam menghasilkan data pengujian. Data uji yang dihasilkan ini kemudian dioptimalkan (seperti yang dijelaskan pada Gambar 1). Gambar 6 merupakan distribusi ACO pada berbagai langkah proses pengujian perangkat lunak, seperti yang digunakan oleh berbagai peneliti di lapangan. Lebih dari setengah penelitian melaporkan Pembuatan Data Uji menggunakan ACO. Sejumlah studi juga mewakili penggunaan ACO dalam jalur atau urutan pembuatan atau optimasi data uji. Hanya satu studi yang terlihat sebagai survei tentang teknik evolusi (termasuk ACO) dalam Rekayasa Perangkat Lunak (termasuk Pengujian Perangkat Lunak).

Gambar 6. *Distribusi aplikasi ACO dalam berbagai langkah pengujian perangkat lunak, TDG - Generasi Data Uji, PG / SG - Generasi Path / Sequence, OPT - Optimasi*

Data Uji, Survei - Survei Pengujian Evolusi.

Shunkun Yang *et al.* [13] menemukan kekurangan dari ACO, yaitu feromon pencarian awal relatif langka, efisiensi pencariannya rendah, model pencarian terlalu sederhana, dan mekanisme umpan balik positif mudah untuk mencegah fenomena stagnasi dan prekursor. Berdasarkan hal tersebut mereka telah meningkatkan ACO, membangun jalur pencarian koloni semut, dan mengedepankan pendekatan yang ditingkatkan: *ILPACO, IPVACO (Improved Pheromone Volatilization coefficient for Ant Colony Optimization), IGPACO (Improved the Global Path pheromone update strategy for Ant Colony Optimization),* dan *ACIACO (A Comprehensive Improved Ant Colony Optimization).* Teknik yang diusulkan dibandingkan dengan RND (*Random Algorithm*) dan GA (*Genetic Algorithm*) dalam hal efisiensi dan cakupan. Melalui perbandingan dan analisis CTCP dan TCAS, metode yang ditingkatkan dapat secara efektif meningkatkan efisiensi pencarian, menahan prekursor, mempromosikan cakupan kasus uji, dan mengurangi jumlah iterasi. Kekurangan dalam optimasi koloni semut yang dapat diatasi dengan menggunakan algoritma genetika, optimasi kerumunan partikel, koloni lebah buatan, atau algoritma heuristik lain untuk bergabung dengan optimisasi koloni semut secara efektif, saling melengkapi satu sama lain. Algoritma yang komprehensif akan meningkatkan kualitas kasus uji perangkat lunak yang dihasilkan secara efektif. Hasil menunjukkan bahwa metode yang ditingkatkan dapat secara efektif meningkatkan efisiensi pencarian, menahan prekursor, mempromosikan cakupan kasus, dan mengurangi jumlah iterasi.

Noguchi *et al.* [14] mengusulkan kerangka kerja berbasis riwayat pelaksanaan pengujian yang dikumpulkan dari produk sebelumnya yang serupa, untuk memprioritaskan kasus pengujian untuk pengujian kotak hitam (*black box*) pada produk baru menggunakan ACO. Simulasi menggunakan dua produk nyata menunjukkan bahwa kerangka kerja mereka dapat meningkatkan efektivitas pengujian kotak hitam dalam waktu yang praktis.

Ghiduk [15] menggunakan algoritma ACO dalam masalah pengujian aliran data perangkat lunak untuk menghasilkan serangkaian jalur optimal untuk mencakup semua asosiasi penggunaan definisi (*du-pair*) dalam program yang sedang diuji. Kemudian, pendekatan ini menggunakan optimisasi koloni semut untuk menghasilkan paket data uji untuk memuaskan set path yang dihasilkan.

Biswas *et al.* [16] menerapkan ACO untuk menghasilkan jalur optimal dalam grafik aliran kontrol (CFG). Jalur optimal akan mencakup seluruh perangkat lunak dengan redundansi minimum. Dalam algoritma yang diusulkan, setelah jalur diprioritaskan dengan cara baru, sehingga kita dapat memutuskan jalur mana yang akan diuji terlebih dahulu. Probabilitas menemukan kesalahan pada tahap sebelumnya mengurangi banyak biaya pengembangan perangkat lunak. Kemudian ACO telah digunakan untuk menghasilkan urutan data uji yang akan digunakan sebagai input untuk mengeksekusi setiap jalur yang dihasilkan sebelumnya. Algoritma pembuatan data uji telah dikembangkan dengan tujuan mencakup seluruh domain dan tidak termasuk dalam pencarian parsial. Akhirnya algoritma yang diusulkan telah didemonstrasikan dan diterapkan ke dalam program pencarian biner untuk menghasilkan jalur, serta input. Manfaat dari penelitian yang ia lakukan adalah: 1) Cakupan jalur penuh melalui CFG (simpul, cabang, *loop*, kondisi); 2) Menggunakan ACO untuk menghasilkan jalur serta menguji data; 3) Prioritaskan jalur untuk memastikan pengujian yang efektif; 4) Hapus redundansi. Ia menyimpulkan dari analisis pendekatan yang diusulkan bahwa algoritma ACO dapat digunakan di bidang pengujian perangkat lunak untuk menghasilkan hasil yang optimal. Teknik meta-heuristik yang berbeda juga dapat digabungkan dengan pendekatan yang diusulkan untuk membangun aplikasi pengujian perangkat lunak.

Li *et al.* [17] menyajikan metode berdasarkan kriteria cakupan jalur untuk menghasilkan data uji, dan algoritma koloni

semut diadopsi untuk mencari model yang dibangun pada domain input program untuk mendapatkan data yang memenuhi jalur yang dipilih. Model yang diusulkan berbeda dari yang tradisional. Dari hasil percobaan kami dapat menemukan bahwa strategi "*choose the poorest*" dapat secara efektif mencegah situasi optimisasi lokal terjadi. Dalam tulisan ini, mereka menggabungkan pendekatan algoritma koloni semut (ACO) dengan teknik fungsi cabang untuk secara otomatis menghasilkan data uji berdasarkan kriteria cakupan jalur diusulkan. Pendekatan ini dapat mencari dalam model yang dibangun oleh domain input program, dan memperoleh data uji yang dapat memenuhi jalur yang dipilih. Akhirnya, efisiensi pendekatan divalidasi menggunakan program segitiga. Namun, metode ini juga memiliki keterbatasan. Dalam tulisan ini, kita hanya membahas bagaimana membangun model berdasarkan pada domain input tipe numerik; alih-alih memikirkan bagaimana menyelesaikan masalah tipe karakter.

Mao *et al.* [18] mereformasi ACO untuk bekerja secara kolaboratif dengan driver uji perangkat lunak. Pekerjaan utamanya adalah mendefinisikan ulang tiga aturan: Aturan transfer lokal, aturan transfer global, dan aturan pembaruan feromon. Untuk memvalidasi solusi yang mereka usulkan untuk menghasilkan data uji, lima program benchmark terkenal digunakan untuk melakukan analisis eksperimental. Hasilnya menunjukkan bahwa algoritma mereka mengungguli algoritma Annealing dan genetik yang ada untuk metrik, seperti cakupan rata-rata (*Average Coverage*), tingkat keberhasilan (*Successful Rate*) dan rata-rata generasi konvergensi (*Average convergence Generation*).

Gao *et al.* [19] mengusulkan suatu algoritma yang memprioritaskan kasus uji berdasarkan optimasi koloni semut (ACO), mempertimbangkan tiga faktor: jumlah kesalahan terdeteksi, waktu eksekusi dan tingkat keparahan kesalahan, dan ketiga faktor ini digunakan dalam algoritma optimasi koloni semut untuk membantu mengungkap kesalahan yang lebih parah pada tahap awal proses pengujian regresi. Keefektifan algoritma diperlihatkan menggunakan metrik bernama APFD, dan hasil percobaan menunjukkan algoritma mengoptimalkan pemesanan test case secara efektif.

Srivastava [20] mengusulkan suatu algoritma yang menggunakan konsep *Ant Colony Optimization* (ACO) untuk bekerja pada CFG dan untuk menemukan jalur optimal untuk menguji perangkat lunak. Perilaku pelepasan feromon semut digunakan dalam algoritma ini untuk mengekstraksi jalur optimal. Algoritma ini menghasilkan jalur yang sama dengan kompleksitas siklomatik.

Srivastava *et al.* [21] menggunakan model penggunaan rantai Markov untuk memeriksa kelayakan urutan uji saat sedang dihasilkan. Data statistik tentang perkiraan penggunaan telah digunakan untuk membangun model stokastik dari perangkat lunak yang diuji. Teknik yang diusulkan menggunakan optimisasi koloni semut sebagai dasarnya dan juga memasukkan faktor-faktor, seperti biaya dan kekritisan berbagai pernyataan dalam model. Penggunaan optimisasi koloni semut (ACO) telah terbukti efektif dan menyediakan cara yang efisien untuk menghasilkan urutan pengujian terutama dalam model berbasis grafik. Kompleksitas algoritma yang digunakan dalam teknik ini adalah O (n^2) untuk traversal dan karenanya terbukti efisien dan komputasi yang kurang intensif.

PENUTUP

Makalah ini menyajikan survei untuk teknik optimasi ACO yang diterapkan untuk pengujian perangkat lunak. ACO telah digunakan dalam berbagai bidang pengujian perangkat lunak, seperti Pembuatan Data Uji, Optimasi Tes Suite, Pembuatan Jalur, dan lain-lain. Dari sembilan studi yang dipilih, empat studi menggunakan ACO untuk data uji (kasus uji) / urutan / pembuatan jalur, sementara lima studi mengusulkan teknik untuk uji kasus / optimasi jalur. Meskipun beberapa perbandingan disediakan dalam beberapa studi, tetapi keragaman dalam langkah-langkah proses, teknik dan contoh-contoh yang mereka terapkan, membuat perbandingan umum tidak dibuat. Di sisi lain, beberapa batasan ACO umum dilaporkan dari satu studi. Menurut

penelitian, ACO memiliki masalah konvergensi solusi optimal lokal, dan juga memiliki kecepatan konvergensi yang lambat pada awalnya ketika hampir tidak ada feromon.

Survei mengungkapkan bahwa ACO telah diterapkan pada berbagai langkah proses pengujian perangkat lunak. Penggunaan ACO dalam pembuatan data uji dengan beragam tipe input (diagram UML, kode sumber, dan lain-lain.) telah menghasilkan hasil yang menjanjikan. ACO sebagian besar telah diterapkan untuk menguji pembuatan data, sementara beberapa proses pengujian perangkat lunak lainnya belum memanfaatkan teknik ACO. Area penelitian di masa depan dapat mencakup mengotomatiskan semua aspek dari proses pengujian perangkat lunak menggunakan teknik ACO.

DAFTAR PUSTAKA

[1] G. M. Kapfhammer, *Software Testing*, Chapter in book, Department of Computer Science, Allegheny College, June 2003.

[2] C. Kaner, J. Falk, and H. Q. Nguyen, *Testing Computer Software*, Wiley, London, UK, 1993.

[3] N. Juristo, A. M. Moreno, S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Software Engineering Journal*, Vol. 1, No. 2 (7-44), 2004.

[4] W. Ding, J. Kou, K. Li, and Z. Yang, "An Optimization Method of Test Suite in Regression Test Model," *In the Proceedings of the 2009 WRI World Congress on Software Engineering (WCSE)*, Vol. 52, pp. 14-30, IEEE Computer Society Washington, DC, USA, 2010.

[5] S. Yoo. and M. Harman, "Regression testing minimization, selection and prioritization: a survey", *Software Testing Verification and Reliability*, 2010.

[6] E. Engström, P. Runeson, "A Qualitative Survey of Regression Testing Practices," *Lecture Notes on Computer Science (LNCS)*, pp. 3-16, Springer Verlag, 2010.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a colony of cooperating agents," *IEEE Trannsactions on Systems, Man and Cybernetics*, vol. B(26), pp. 29-41, 1996.

[8] H. Li, and C. Peng Lam, "Software Test Data Generation Using Ant Colony Optimization," *In Transactions on Engineering, Computing and Technology*, 2005.

[9] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 321–332, 2002.

[10] P.Zhao, P.Zhao, and X.Zhang, "New Ant Colony Optimization for the Knapsack Problem", *in the Proceedings of the 7th International Conference on Computer-Aided Industrial Design and Conceptual Design*, pp. 1-2, Nov 2006.

[11] K. Doerner, W.J. Gutjahr, "Extracting Test Sequences from a Markov Software Usage Model by ACO", *In the Proceedings of GECCO 2003*, Vol. 2724, pp. 2465-2476, LNCS Springer-Verlag Berlin Heidelberg, 2003.

[12] F. P. McMinn, M. Holcombe, "The State Problem for Evolutionary Testing", *In the Proceedings of GECCO 2003*, Vol. 2724, pp. 2488-2500, LNCS Springer Verlag, 2003.

[13] Shunkun Yang, et al., "Improved ant algorithms for software testing cases generation", *The Scientific World Journal*, vol. 14, Gale Academic OneFile, 2014.

[14] T. Noguchi, H. Washizaki, Y. Fukazawa, A. Sato and K. Ota, "History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization", *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1-2, Graz, 2015.

[15] Ahmed Ghiduk, "A New Software Data-Flow Testing Approach via Ant Colony Algorithms", *Universal Journal of Computer Science and Engineering Technology*, 2010.

[16] S. Biswas, M. S. Kaiser and S. A. Mamun, "Applying Ant Colony Optimization in software testing to generate prioritized optimal path and test data," *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, pp. 1-6, Dhaka, 2015.

[17] K.Li, Z. Zhang and W. Liu, "Automatic Test Data Generation Based on Ant Colony Optimization," *2009 Fifth International Conference on Natural Computation*, pp. 216-220, Tianjin, 2009.

[18] C. Mao, X. Yu, J. Chen and J. Chen, "Generating Test Data for Structural Testing Based on Ant Colony Optimization," *12th International Conference on Quality Software, Xi'an*, pp. 98-101, Shaanxi, 2012.

[19] D. Gao, X. Guo, and L. Zhao, "Test case prioritization for regression testing based on ant colony optimization," *6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 275-279, Beijing, 2015.

[20] Ranjaan Srivastava, "Structured testing using ant colony optimization," *In Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia (IITM '10),* Association for Computing Machinery, 203-207, New York, NY, USA, 2011.

[21] Dr. Praveen Srivastava, Jose Nitin, Barade Saudagar, and Ghosh Debopriyo, "Optimized Test Sequence Generation from Usage Models using Ant Colony Optimization", *International Journal of Software Engineering & Applications*, 2010.