Jurnal Rekayasa Sistem Industri
Volume 13 No 2 - Oktober 2024
http://journal.unpar.ac.id/index.php/jrsi/index
ISSN 2339-1499 (*online*) – ISSN 0216-1036 (*print*)

Disampaikan : 24 Februari 2024
Direview : 2 Juni 2024
Diterima : **18 Juli 2024**

Penerapan *Lightning Search Algorithm* dengan 2-Opt *Local Search* untuk Penyelesaian *Asymmetric Traveling Salesman Problem*

Yulius Chandra Gunawan[1], Ignatius A. Sandy[2]
[1,2)] Fakultas Teknologi Industri, Jurusan Teknik Industri, Universitas Katolik Parahyangan
Jl. Ciumbuleuit 94, Bandung 40141
Email: 8132301003@student.unpar.ac.id, sandy@unpar.ac.id

Abstract

The Asymmetric Traveling Salesman Problem (ATSP) is an optimization problem where a "salesman" must visit several cities in a single trip. In the ATSP, the distance traveled from city i to j differs from the distance from city j to i. The goal of the ATSP is to minimize the total distance traveled by the "salesman." In this study, the Lightning Search Algorithm (LSA) and the 2-Opt local search algorithm are used to find solutions for the ATSP. LSA is a metaheuristic algorithm inspired by the process of lightning propagation to the earth's surface. 2-Opt is a local search algorithm that can manipulate routes to produce better solutions. This research aims to design LSA with 2-Opt for solving the ATSP and to identify the parameters that influence the ATSP solution. Three parameters are used in this study: maximum channel time (max_ctime) and forking probability (fork_prob), which are responsible for the forking phenomenon, and Boundaries (Bound), which define the solution space. ANOVA testing was conducted on 8 parameter combinations implemented on 5 ATSP cases from TSPLIB: BR17, FTV33, FTV44, FTV55, and FTV70 to determine the best parameter values. The ANOVA results show that the Bound parameter affects the solution in the FTV33, FTV44, and FTV70 cases, while the max_ctime parameter affects the solution in the FTV55 case. Based on the determined parameter values, the LSA with 2-Opt was re-implemented on the five ATSP cases. The results show that the LSA with 2-Opt was able to find the best-known solution for the BR17 case but was unable to find the best-known solution for the other cases.

***Keywords**: Asymmetric Traveling Salesman Problem, Lightning Search Algorithm, Metaheuristics, Local Search, Optimization Problem*

Abstrak

Asymmetric Traveling Salesman Problem (ATSP) adalah suatu permasalahan optimasi dimana terdapat seorang "*salesman*" yang harus mengunjungi beberapa kota dalam satu kali perjalanan. Pada ATSP, jarak yang ditempuh dari kota i ke j berbeda dengan jarak dari kota j ke i. Tujuan dari ATSP adalah meminimasi total jarak yang ditempuh oleh "*salesman*". Pada penelitian ini *Lightning Search Algorithm* (LSA) dan 2-Opt *local search algorithm* digunakan untuk menemukan solusi dari ATSP. LSA adalah sebuah algoritma metaheuristik yang terinspirasi dari proses perambatan lidah petir (*step leader*) ke permukaan bumi. 2-Opt merupakan algoritma *local search* yang dapat memanipulasi rute agar menghasilkan solusi yang lebih baik. Penelitian ini bertujuan untuk merancang LSA dengan 2-Opt dalam menyelesaikan ATSP dan menemukan parameter yang berpengaruh terhadap solusi ATSP. Tiga parameter digunakan dalam penelitian ini, yaitu *maximum channel time* (max_ctime) dan *forking probability* (fork_prob) yang bertanggungjawab atas terjadinya fenomena *forking*, dan *Boundaries* (Bound) yang mendefinisikan ruang solusi. Pengujian ANOVA dilakukan terhadap 8 kombinasi parameter yang diimplementasi ke 5 kasus ATSP dari TSPLIB, yaitu kasus BR17, FTV33, FTV44, FTV55, dan FTV70 untuk menentukan nilai parameter terbaik. Hasil ANOVA menunjukkan parameter Bound berpengaruh terhadap solusi pada kasus FTV33, FTV44, dan FTV70. Sedangkan parameter max_ctime berpengaruh terhadap solusi pada kasus FTV55. Berdasarkan nilai parameter yang ditentukan, LSA dengan 2-Opt diimplementasikan kembali ke 5 (lima) kasus ATSP. Hasil yang didapat adalah LSA dengan 2-Opt mampu menemukan *best known solution* untuk kasus BR17, tetapi tidak mampu menemukan *best known solution* untuk kasus lain-nya.

Kata kunci: *Asymmetric Traveling Salesman Problem*, *Lightning Search Algorithm*, Metaheuristik, *Local Search*, *Optimization Problem*

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Pendahuluan

Asymmetric Traveling Salesman Problem (ATSP) merupakan sebuah masalah pemilihan rute dari seorang "*salesman*" yang harus mengunjungi sekumpulan kota dan kembali ke lokasi awal dalam satu kali perjalanan. Jarak dari kota A ke kota B, dan dari kota B ke okta A bisa berbeda. Tujuan dari ATSP ini adalah meminimasi total jarak yang ditempuh oleh sang "*salesman*" (Gutin & Punnen, 2002). ATSP termasuk ke dalam kategori NP-*hard problem* (*Non-deterministic Polynomial-time Hard Problem*) yang artinya semakin kompleks suatu permasalahan, waktu yang dibutuhkan untuk mendapatkan hasil optimal dari masalah tersebut akan semakin lama (Brest & Zerovnik, 2003). ATSP dapat digunakan di dunia nyata untuk memodelkan permasalahan distribusi, penjadwalan, dan menentukan rute kendaraan (Nagata & Soler, 2012).

Penyelesaian ATSP dapat dilakukan dengan menggunakan metode analitik dan metode heuristik. Contoh metode analitik yang dapat digunakan untuk menyelesaikan ATSP adalah *branch and bound*, dan *dynamic programming*. Namun, jika jumlah kota sangat banyak, pencarian solusi menggunakan metode analitik akan memakan waktu yang sangat lama sekali. Sehingga pencarian solusi dapat dilakukan dengan menggunakan menggunakan metode heuristik.

Beberapa penelitian sebelumnya sudah mencoba menyelesaikan kasus ATSP dengan menggunakan algoritma metaheuristik, yaitu penelitian oleh Kevin (2016) yang menggunakan *Harmony Search Algorithm* (HSA), penelitian Santosa (2017) yang menggunakan *Elephant Herding Optimization* (EHO) dan penelitian Elim (2018) yang menggunakan *Lion Optimization Problem* (LOA). Ketiga penelitian tersebut menggunakan contoh kasus ATSP yang tersedia di TSPLIB (Reinelt, 1991), antara lain: kasus BR17 dengan jumlah 17 kota, FTV33 dengan jumlah 34 kota, FTV44 dengan jumlah 45 kota, FTV55 dengan jumlah 56 kota, dan FTV70 dengan jumlah 71 kota. *Best known solution* untuk setiap kasus tersebut dan solusi yang diperoleh EHO, HSA, dan LOA dapat dilihat di Tabel 1.

Dapat dilihat bahwa untuk kasus BR17, ketiga algoritma berhasil menemukan *best-known solution*, yaitu 39. Sedangkan pada kasus FTV33, FTV44, dan FTV55 hanya EHO yang mampu menemukan solusi yang sama dengan *best known solution*. Sedangkan pada kasus FTV70 ketiga algoritma tidak ada yang berhasil menemukan solusi yang sama dengan *best known solution*. Penelitian yang dilakukan oleh Santosa (2017), Kevin (2016), dan Elim (2018) tidak menggunakan tambahan *local search algorithm* untuk meng-eksploitasi lebih lanjut solusi yang telah dihasilkan oleh algoritma metaheuristik.

Tabel 1. *Best known solution* dan solusi dari EHO, HSA, dan LOA

Kasus	***Best Known Solution***	EHO	HSA	LOA
BR17	39	39	39	39
FTV33	1286	1286	1388	1534
FTV44	1613	1613	1849	2128
FTV55	1608	1608	2110	3521
FTV70	1950	2089	2813	3728

Pada penelitian kali ini digunakan *Lightning Search Algorithm* (LSA), yaitu sebuah algoritma metaheuristik yang terinspirasi dari proses perambatan lidah petir (*step leader*) ke permukaan bumi. Penelitian ini juga akan menggunakan tambahan 2-Opt *local search* untuk penyelesaian ATSP. 2-Opt dipilih sebagai tambahan untuk *local search* karena 2-Opt merupakan algoritma yang mudah digunakan, praktis dan terbukti efektif untuk mencari solusi dari TSP yang optimal atau mendekati optimal (Johnson & McGeoch, 2003) Tujuan dari penelitian ini ada tiga, pertama untuk mengetahui cara penerapan LSA dengan 2-Opt *local search* untuk penyelesaian kasus ATSP. Kedua, untuk mengetahui pengaruh parameter-parameter LSA terhadap performansi-nya. Ketiga, untuk mengetahui performansi LSA dengan 2-Opt *local search* untuk penyelesaian kasus ATSP. Penelitian ini menggunakan 5 (lima) contoh kasus ATSP dari TSPLIB (Reinelt, 1991) yang juga digunakan oleh Santosa (2017), Kevin (2016), dan Elim (2018), yaitu kasus BR17, FTV33, FTV44, FTV55, dan FTV70.

Tinjauan Pustaka

Beberapa studi telah dilakukan terkait dengan penelitian yang dilakukan, yaitu mengenai ATSP, *Lightning Search Algorithm*, *Elephant Herding Optimization*, *Harmony Search Algorithm*, 2-Opt *local search algorithm*, serta *random-key encoding*.

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Asymmetric Traveling Salesman Problem

ATSP adalah salah satu variasi dari TSP (Gutin & Punnen, 2002). Perbedaannya adalah, pada ATSP jarak yang ditempuh oleh *salesman* dari *node* i ke *node* j, dan dari *node* j ke *node* i berbeda. Tujuan utama dari permasalahan ATSP ini adalah menemukan jarak yang paling minimum untuk mendatangi setiap kota dalam satu kali perjalanan dan kembali ke kota awal. Maka, permasalahan ATSP dapat dirumuskan sebagai berikut.

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad \text{Pers. 1}$$

Dengan batasan:

$$\sum_{i=1}^{n} x_{ij} = 1;\ \forall j \quad \text{Pers. 2}$$

$$\sum_{j=1}^{n} x_{ij} = 1;\ \forall i \quad \text{Pers.3}$$

$$u_i - u_j + n \times x_{ij} \leq n-1 \quad \text{Pers.4}$$

$$x_{ij} \in \{0,1\} \quad \text{Pers.5}$$

$$i \neq j \quad \text{Pers.6}$$

Keterangan:

n = jumlah *node*

i = *node* asal

j = *node tujuan*

c_{ij} = jarak dari *node* i ke *node* j

x_{ij} = keputusan pemilihan jalur dari *node* i ke *node* j

u_i = urutan tiap kota yang dikunjungi

Persamaan 1 merupakan fungsi tujuan permasalahan ATSP yang mana jarak yang ditempuh oleh *salesman* harus diminimasi. C_{ij} merupakan jarak yang ditempuh dari *node i* ke *node j*, dan x_{ij} merupakan variabel keputusan yang menyatakan rute dari *node i* ke *j* akan diambil atau tidak. Jika rute tersebut diambil, maka nilai $x_{ij} = 1$, jika rute tersebut tidak diambil, maka nilai $x_{ij} = 0$. Arti dari Persamaan 2 dan 3 adalah dari setiap kota yang ada, sang *salesman* hanya akan mendatangi kota tersebut satu kali saja. Persamaan 4 memastikan bahwa *node i* sampai *node j* terhubung dalam 1 jalur. Persamaan 6 menunjukkan bahwa *node i* dan *node j* berbeda (Lawler, Lenstra, Kan, & Shmoys, 1985).

***Lightning Search Algorithm* (LSA)**

Lightning Search Algorithm (Shareef, Ibrahim, & Mutlag, 2015) adalah algoritma metaheuristik yang terinspirasi dari perambatan *step leader* ke permukaan bumi. Perambatan yang dimaksud adalah "perpindahan posisi" dari *step leader*. Perpindahan posisi ini dibantu oleh tiga jenis *projectile*, yaitu: *transition projectile* akan membentuk posisi awal dari setiap anggota populasi *step leader*, *space* & *lead projectile* untuk mencari posisi baru untuk *step leader*.

Sebelumnya telah disebutkan bahwa posisi awal dari setiap anggota populasi *step leader* dibentuk menggunakan *transition projectile*, sehingga jika terdapat sebuah populasi yang terdiri dari N buah *step leader* $SL = [sl_1,\ sl_2, sl_3\ , \dots, sl_N]$, dibutuhkan pula N buah *transition projectile* $P^T = [p_1^T, p_2^T, p_3^T, \dots p_N^T]$. *Transition projectile* menggunakan bilangan acak berdistribusi *uniform*(LB,UB) sebagai posisi awal dari setiap *step leader*, dengan LB adalah batas bawah, dan UB adalah batas atas. Setiap *step leader* akan memiliki energi (E_{sl}) berdasarkan posisinya.

Posisi baru oleh *lead projectile* dimodelkan sebagai bilangan acak yang berdistribusi normal. Oleh sebab itu, pencarian posisi baru oleh *lead projectile* dilakukan dengan Persamaan 7.

$$p_{new}^{L} = p^{L} + normrand(\mu_L, \sigma_L) \quad \text{Pers. 7}$$

Dengan p_{new}^{L} adalah posisi baru, p^L posisi awal atau posisi *step leader*, dan $normrand(\mu_L, \sigma_L)$ adalah bilangan acak berdistribusi normal dengan *shape parameter* μ_L dan *scale parameter* σ_L. *Step leader* hanya akan berpindah posisi apabila $E_{sl} < E_{p_{new}^{L}}$.

Posisi baru oleh *space projectile* dimodelkan sebagai bilangan acak yang berdistribusi eksponensial. Oleh sebab itu, pencarian posisi baru oleh *lead projectile* dilakukan dengan Persamaan 8.

$$p_{i\_new}^{S} = p_i^{S} \pm \exp rand(\mu_i) \quad \text{Pers. 8}$$

dengan $p_{i\_new}^{S}$ adalah posisi baru, p_i^S adalah posisi awal atau posisi *step leader*, dan $exprand(\mu_i)$ adalah bilangan acak berdistribusi eksponensial dengan parameter μ_i. μ_i adalah jarak antara p_i^S dengan *step leader* terbaik. *Step leader* hanya akan berpindah posisi apabila $E_{sl} < E_{p_{i\_new}^{S}}$.

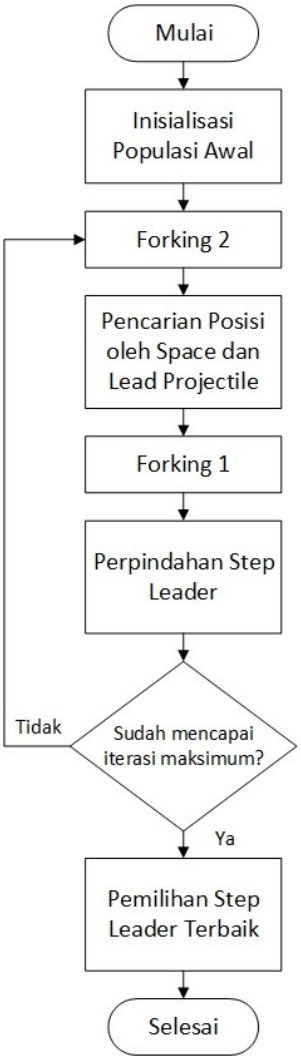
Dalam melakukan pencarian solusi, terdapat kemungkinan terjadinya fenomena percabangan atau *forking*. Fenomena *forking* ini direalisasikan dengan dua cara: pertama, cabang yang simetris akan terbentuk dengan posisi yang berlawanan dengan posisi dari *projectile*.

$$\bar{p_i} = LB + UB - p_i \quad \text{Pers. 9}$$

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Dimana $\bar{p}_i$ dan p_i beruturt-turut adalah *opposite projectile* dan *original projectile*. Sedangkan LB dan UB adalah batas-batas. Jika *forking* tidak memperbaiki jalur perambatan, maka titik *forking* akan dieliminasi untuk mempertahankan ukuran populasi. Cara kedua adalah mengganti posisi *step leader* yang gagal berpindah posisi, dengan posisi *step leader* terbaik. Pergantian posisi ini didasarkan pada asumsi munculnya jalur percabangan baru pada *step leader* terbaik, menggantikan *step leader* terburuk.

Berdasarkan literatur mengenai LSA, didapatkan langkah-langkah besar dalam melakukan pencarian solusi menggunakan LSA seperti pada Gambar 1. Istilah *Forking* 2 pada Gambar 1 merupakan *forking* dengan cara yang kedua, sedangkan *Forking* 1 adalah *forking* dengan cara yang pertama. Sedangkan iterasi maksimum adalah jumlah maksimum pengulangan/siklus pencarian solusi yang harus dilakukan.



Gambar 1. Langkah besar pencarian solusi LSA

***Elephant Herding Optimization* (EHO)**

Elephant Herding Optimization (Wang, Deb, & Coelho, 2015) terinspirasi dari perilaku penggembalaan gajah dalam mencari sumber makanan dan air di alam liar. Para gajah dalam berkelompok menggunakan pencarian kolaboratif dan strategi adaptif dalam mencari sumber daya. Selama proses pencarian, gajah-gajah berkomunikasi dan berkolaborasi satu sama lain untuk mencari solusi terbaik. Mereka saling berbagi informasi dan pengetahuan tentang lingkungan sekitar. Kolaborasi ini dapat terjadi melalui mekanisme seperti pertukaran informasi, penggabungan solusi, atau adopsi strategi terbaik dari gajah-gajah lain dalam kelompok.

EHO juga melakukan keseimbangan antara eksplorasi dan eksploitasi untuk meningkatkan kualitas solusi. Dengan melakukan keseimbangan ini EHO dapat mencapai konvergensi menuju solusi optimal atau sub-optimal dalam ruang pencarian.

***Harmony Search Algorithm* (HSA)**

Harmony Search Algorithm (Geem, Kim, & Loganathan, 2001) terinspirasi dari proses penciptaan musik dimana harmoni tercipta melalui penyesuaian bertahap terhadap elemen-elemen yang ada. Pada setiap iterasi, HSA menciptakan sebuah harmoni baru dengan mengkominasikan elemen-elemen dari solusi yang ada. Harmoni baru ini merepresentasikan sebuah kemungkinan solusi baru yang dapat dievaluasi kualitasnya. Penyesuaian ini dilakukan dengan berbagai cara, seperti pemilihan elemen secara acak dari solusi yang ada, penggabungan elemen dari beberapa solusi, atau penggunaan aturan heuristik untuk menghasilkan solusi baru.

***Lion Optimization Algorithm* (LOA)**

Lion Optimization Algorithm (Yazdani & Jolai, 2016) terinspirasi dari perilaku berburu singa dalam mencari mangsa di padang rumput Afrika. Setiap singa merepresentasikan sebuah solusi. Proses pencarian solusi LOA diawali dengan berburu, dimana pada setiap iterasi singa-singa akan mencari mangsa (solusi optimal) dalam lingkungan mereka. Singa jantan dan betina mungkin bekerja sama dalam kelompok untuk meningkatkan peluang keberhasilan perburuan. Setelah mencoba menemukan mangsa, kemampuan berburu dari setiap singa dievaluasi berdasarkan kinerja

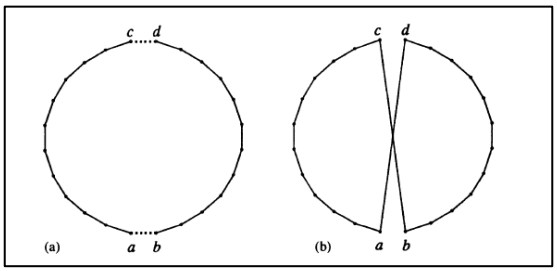
DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

mereka dalam menemukan solusi yang baik. Evaluasi ini dapat dilakukan dengan menggunakan fungsi objektif dari permasalahan yang relevan.

2-Opt *Local Search Algorithm*

Menurut Johnson dan McGeoch (1997), 2-Opt merupakan algoritma *local improvement* untuk permasalahan TSP yang pertama kali digagas oleh Croes (1958). Algoritma ini akan memanipulasi solusi TSP dengan melakukan pemotongan pada sebuah tur sehingga didapatkan dua buah sub-tur. Kemudian kedua sub-tur ini digabungkan kembali dengan cara lain yang memungkinkan. Proses ini akan terus diulang sampai tidak ditemukan lagi tur yang lebih baik.

Sebagai contoh, terdapat sebuah tur yang terdiri dari 4 kota, yaitu B - D - C - A dengan total jarak sebesar 100 satuan. Tur ini dipotong menjadi 2 bagian, yaitu sub-tur B - D dan sub-tur C - A. Urutan Sub-tur C - A diputar sehingga menghasilkan A - C. Kemudian kedua sub-tur ini digabungkan kembali agar membentuk tur B - D - A - C. Jika total jarak tur baru < total jarak tur lama, maka tur baru yang dipilih. Ilustrasi algoritma 2-Opt dapat dilihat pada Gambar 2.



Gambar 2. 2-Opt *algorithm*: (a) Tur awal (b) Tur setelah 2-Opt.
(Sumber: Johnson & McGeoch, 1997)

Algoritma 2-Opt merupakan algoritma mudah digunakan, praktis dan terbukti efektif untuk mencari solusi dari TSP yang optimal atau mendekati optimal (Johnson & McGeoch, 2003).

Beberapa penelitian sebelumnya sudah mencoba menyelesaikan ATSP dengan algoritma metaheuristik. Santosa (2017) menggunakan EHO, Kevin (2016) menggunakan HSA, dan Elim (2018) menggunakan LOA. Ketiga penelitian tersebut hanya memanfaatkan metode eksplorasi dan eksploitasi yang ada dalam algoritma metaheuristiknya. Jika dilihat pada Tabel 1, hanya EHO yang mampu menemukan *best-known solution* untuk kasus FTV33, FTV44, dan FTV55. Pada kasus FTV70 ketiga algoritma tidak dapat menemukan *best-known solution*. Penelitian ini menggunakan LSA untuk penyelesaian ATSP. Selain itu penelitian ini juga akan menggunakan 2-Opt *local search* untuk membantu dalam eksploitasi solusi yang diperoleh dari LSA.

Metode Penelitian

Langkah-langkah yang dilakukan pada penelitian ini adalah sebagai berikut.

Perancangan Algoritma LSA dengan 2-Opt *Local Search*.

Perancangan diawali dengan melakukan identifikasi elemen-elemen dari ATSP dan LSA, untuk mengetahui elemen ATSP yang harus ditranslasi ke elemen LSA agar LSA dapat melakukan pencarian solusi ATSP. Elemen-elemen ATSP dapat dilihat pada Tabel 2.

Tabel 2. Elemen-elemen ATSP

No.	Elemen
1	Jumlah kota
2	Jarak antar kota
3	Urutan kota
4	Total jarak ditempuh

Elemen-elemen dari LSA yang telah diidentifikasi dapat dilihat pada Tabel 3. Dari hasil identifikasi elemen, elemen ATSP yang ditranslasi ke elemen LSA dapat dilihat di Tabel 4.

Selanjutnya dirancang LSA dengan 2-Opt untuk memproses pencarian solusi. Pada LSA langkah pertama proses pencarian solusi adalah inisialisasi *step leader*. Untuk kasus ATSP, setiap *step leader* menyimpan 1 (satu) urutan kota. Sehingga jika terdapat P buah *step leader*, maka ada sebanyak P buah urutan kota.

LSA menggunakan nilai kontinyu untuk melakukan pencarian solusi. Sedangkan ATSP merupakan permasalahan diskrit, dimana solusi-nya berbentuk permutasi urutan kota. Beberapa penelitian sebelumnya mengenai penyelesaian ATSP dengan metaheuristik (Santosa, 2017; Kevin, 2016; Elim, 2018) menggunakan *random key encoding* untuk mentranslasi urutan kota menjadi nilai vektor kontinyu. Talbi (2009) menyebutkan bahwa *random key* cocok untuk permasalahan permutasi salah satunya adalah TSP. Sehingga pada penelitian ini untuk mentranslasi urutan

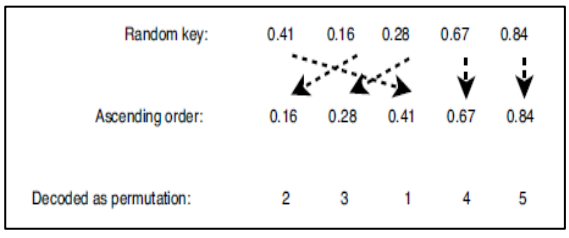
kota menjadi nilai kontinyu, digunakan *random-key encoding*. Contoh *random key encoding* dijelaskan oleh Gambar 3.

Tabel 3. Elemen-elemen LSA

No.	Elemen
1	***Population size* (P)** Merupakan jumlah *Step Leader* yang digunakan untuk pencarian solusi.
2	***Boundaries* (Bound)** Merupakan batasan yang mendefinisikan ruang solusi. Nilai *lower bound* adalah (-Bound) sedangkan nilai *upper bound* adalah (Bound).
3	***Max. channel time* (max_ctime)** Merupakan jumlah iterasi/siklus yang dibutuhkan untuk terjadinya *Forking* 2 (*forking* dengan cara kedua). Nilai max_ctime merupakan persentase dari *population size*.
4	***Forking probability* (fork_prob)** Merupakan batas probabilitas terjadinya *Forking* 1 (*forking* dengan cara pertama).
5	***Maximum iteration* (T)** Merupakan jumlah maksimum pengulangan/siklus pencarian solusi oleh algoritma.
6	***Step leader*** Merupakan elemen yang menyimpan solusi dari permasalahan.
7	***Step leader energy*** Merupakan *fitness* dari solusi yang disimpan oleh *step leader*.
8	***Lightning strike*** Merupakan *step leader* terbaik saat proses pencarian solusi sudah mencapai iterasi maksimum.
9	***Lightning strike energy*** Merupakan *fitness* dari *step leader* terbaik saat pencarian solusi sudah mencapai iterasi maksimum.

Tabel 4. Elemen ATSP yang ditranslasi ke elemen LSA

Elemen ATSP	Elemen LSA
Urutan kota	*Step leader*
Total jarak ditempuh	*Step leader energy*



Gambar 3. *Random key representation*
(Sumber: Talbi, 2009)

Dalam penelitian ini, sebuah *step leader* akan menyimpan 1 urutan kota. Dengan *random key encoding*, setiap kota akan direpresentasikan dengan 1 nilai. Nilai ini berdistribusi *uniform*(UB,LB) yang di-*generate* oleh *transition projectile*. Nilai-nilai tersebut kemudian diurutkan dari terkecil ke terbesar untuk mendapatkan urutan awal. Sebagai contoh, misal nilai (UB,LB) adalah (10,-10). Proses *encoding* dapat dilihat pada Tabel 5.

Tabel 5. Proses *encoding*

Posisi transition projectile	2	-1	3	-5	8
Pengurutan	-5	-1	2	3	8
Urutan kota awal	**4**	**2**	**1**	**3**	**5**

Dari tahap *encoding* di atas, maka *step leader* akan menyimpan urutan kota 4 - 2 - 1 - 3 - 5 dengan posisi *step leader* (-5, -1, 2, 3, 8). Posisi *step leader* ini yang kemudian akan diproses eksplorasi dan eksploitasi oleh LSA.

Pada LSA, tahap eksplorasi dilakukan saat proses pencarian posisi baru oleh *space projectile*. Sedangkan eksploitasi dilakukan saat pencarian posisi baru oleh *lead projectile*. Pada penelitian ini, 2-Opt ditambahkan setelah proses pencarian posisi baru oleh *space* dan *lead projectile*. Jika 2-Opt berhasil menemukan posisi yang lebih baik dari posisi baru tersebut, maka posisi baru akan diganti dengan posisi yang diperoleh 2-Opt.

Validasi Algoritma

Proses validasi algoritma dengan cara menyelesaikan kasus ATSP sederhana. Jika algoritma tidak dapat menyelesaikan kasus ATSP, maka algoritma yang dirancang dimodifikasi sampai algoritma berhasil menyelesaikan ATSP.

Perancangan Program Komputer

Setelah algoritma divalidasi, dirancang program komputer sesuai dengan algoritma yang telah dibuat untuk memudahkan proses implementasi. Perancangan program menggunakan *software* MATLAB R2018a.

Verifikasi dan Validasi Program Komputer

Program komputer diverifikasi untuk memastikan program yang dibuat sudah sesuai dengan langkah-langkah dalam algoritma. Setelah verifikasi dilanjutkan proses validasi program dengan menyelesaikan kasus ATSP sederhana.

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Implementasi Algoritma

Setelah program valid, maka program diimplementasi ke lima kasus ATSP dari TSPLIB (Reinelt, 1991), yaitu kasus BR17 (17 kota), FTV33 (34 kota), FTV44 (45 kota), FTV55 (56 kota), FTV70 (71 kota). Eksperimen dilakukan untuk menentukan nilai parameter terbaik dari setiap kasus ATSP. Eksperimen dilakukan dengan menyelesaikan kasus ATSP tersebut dengan beberapa kombinasi nilai parameter. Hasil eksperimen dilakukan uji ANOVA untuk mengetahui parameter dan/atau interaksi parameter yang memiliki pengaruh terhadap solusi yang ditemukan.

Setelah menemukan nilai parameter terbaik, algoritma diimplementasi untuk mencari solusi terbaik. Implementasi dilakukan sebanyak 10 replikasi terhadap setiap kasus. Hasil terbaik adalah solusi algoritma untuk kasus tersebut.

Hasil dan Diskusi

Penyelesaian ATSP menggunakan LSA dengan 2-Opt diawali dengan mentranslasi permutasi urutan kota menjadi nilai vektor kontinyu menggunakan *random key encoding* untuk kemudian diproses oleh LSA. Selain itu, 2-Opt ditambahkan setelah proses pencarian posisi baru oleh *space* dan *lead projectile* untuk membantu proses eksploitasi. Hasil perancangan LSA dengan 2-Opt dapat dilihat pada Tabel 6.

LSA memiliki 5 parameter, namun hanya 3 parameter yang akan dilakukan eksperimen dengan 2^k *factorial design* untuk menentukan nilai terbaiknya, yaitu *maximum channel time* (max_ctime), *Boundaries* (Bound), dan *forking probabilty* (fork_prob). *Population size* (N) dan *maximum iteration* (T) tidak dilakukan eksperimen karena semakin besar nilainya maka semakin besar kemungkinan untuk menemukan solusi terbaik.

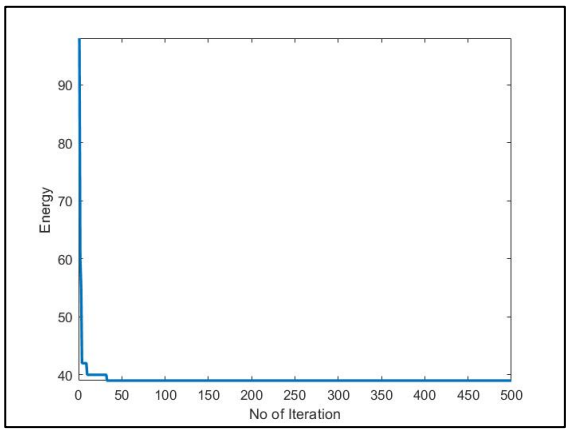
Population size (N) yang digunakan dalam penelitian ini adalah sebanyak 200 *step leader* agar proses pencarian solusi maksimal dengan jumlah yang banyak. Sedangkan untuk nilai iterasi maksimum disesuaikan dengan masing-masing kasus. Penyesuaian iterasi dilakukan dengan melakukan *trial & error* pada setiap kasus ATSP dan dilihat pada iterasi berapa solusi mengalami stagnansi. Pada percobaan ini nilai parameter akan tetap, yaitu N = 200, Bound = 100, max_ctime = 5%, dan fork_prob = 0,5.

Tabel 6. Rancangan LSA dengan 2-Opt

	LSA dengan 2-Opt
1	Input parameter LSA: P, Bound, max_ctime, T, fork_prob
2	Input matrix jarak dan jumlah kota
3	Inisialisasi posisi awal populasi *step leader* dengan batas (-B,B)
4	Set ctime = 0
5	While (t < T) :
6	Tingkatkan ctime sebesar 1
7	Jika ctime = max_ctime, maka :
8	Forking 2 : ubah posisi *leader* terburuk menjadi posisi step leader terbaik
9	*Reset* ctime menjadi 0
10	Jika lainnya, maka lanjut ke langkah 11
11	Urutkan *step leader* berdasarkan energi terkecil ke energi terbesar
12	Untuk setiap *step leader* :
13	Pencarian posisi baru oleh *Space* & *Lead Projectile*
14	Jika probabilitas forking 1 < fork_prob
15	Forking 1 : cek energi di posisi forking
16	Jika energi forking < energi posisi baru, maka posisi baru = posisi forking
17	Jika lainnya, maka lanjut ke langkah 19
18	Jika lainnya, maka lanjut ke langkah 19
19	2-Opt Algorithm
20	Jika energi posisi baru < energi posisi lama, maka posisi lama = posisi baru
21	Pilih *step leader* terbaik sebagai *lightning strike*

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Percobaan pada kasus BR17 dengan T = 500 iterasi didapatkan hasil seperti pada Gambar 4 yang menunjukkan solusi mengalami stagnansi dari iterasi 33 sampai iterasi ke 500. Sehingga untuk kasus BR17 cukup menggunakan 100 iterasi.



Gambar 4. Penentuan iterasi untuk kasus BR17

Dengan menggunakan metode penentuan iterasi yang sama, maka jumlah iterasi yang digunakan pada masing-masing kasus dapat dilihat pada Tabel 7.

Tabel 7. Jumlah iterasi pada setiap kasus ATSP

Kasus	Jumlah Iterasi
BR17	100
FTV33	2000
FTV44	2000
FTV55	3000
FTV70	3000

Untuk nilai parameter max_ctime, Bound, dan fork_prob dilakukan eksperimen dengan metode 2^k *factorial design*. Sehingga kombinasi nilai faktor dieksperimen dapat dilihat di Tabel 8.

Tabel 8. Kombinasi nilai parameter

Kombinasi	max_ctime	fork_prob	Bound
1	1%	0,1	100
2	5%	0,1	100
3	1%	0,9	100
4	5%	0,9	100
5	1%	0,1	1000
6	5%	0,1	1000
7	1%	0,9	1000
8	5%	0,9	1000

Nilai max_ctime yang dieksperimen hanya 1% dan 5% dari iterasi maksimum karena menurut Shareef, Ibrahim, & Mutlag (2015) nilai 1% - 5% dari iterasi maksimum dapat menghasilkan solusi yang baik, dan dalam eksperimen ini diambil 1% dan 5% untuk merepresentasikan nilai terendah dan nilai tertinggi. Nilai parameter fork_prob yang diuji adalah 0,1 dan 0,9 dan nilai parameter Bound yang diuji adalah 100 dan 1000 karena merepresentasikan nilai terendah dan nilai tertinggi.

Kombinasi nilai parameter tersebut diterapkan pada 5 kasus ATSP. Penerapan dilakukan sebanyak 5 replikasi. Solusi dari setiap kasus kemudian diuji ANOVA untuk mencari tahu pengaruh setiap parameter terhadap solusi pada masing-masing kasus. Hasil uji ANOVA dapat dilihat pada Tabel 9.

Tabel 9. Hasil uji ANOVA

Source of Variation	Kasus				
	BR 17	FTV 33	FTV 44	FTV 55	FTV 70
max_ctime	Tidak	Tidak	Tidak	Ya	Tidak
fork_prob	Tidak	Tidak	Tidak	Tidak	Tidak
Bound	Tidak	Ya	Ya	Tidak	Ya
max_ctime* fork_prob	Tidak	Tidak	Tidak	Tidak	Tidak
max_ctime* Bound	Tidak	Tidak	Tidak	Tidak	Tidak
fork_prob* Bound	Tidak	Tidak	Tidak	Tidak	Tidak
max_ctime* fork_prob* Bound	Tidak	Tidak	Tidak	Tidak	Tidak

Dapat dilihat pada Tabel 9. bahwa parameter max_ctime berpengaruh pada kasus FTV55. Dan parameter Bound berpengaruh pada kasus FTV33, FTV44, dan FTV70. Sedangkan parameter fork_prob dan interaksi antar parameter tidak berpengaruh pada semua kasus. Untuk parameter yang berpengaruh, dilakukan *trial and error* dengan 5 replikasi untuk mendapatkan nilai parameter terbaik. Hasil dari *trial and error* parameter Bound kasus FTV33, FTV44, dan FTV70 berturut-turut dapat dilihat pada Gambar 5, Gambar 6, dan Gambar 7. Rata-rata solusi yang dihasilkan algoritma minimum pada nilai Bound = 125 untuk kasus FTV33 dan FTV44. Sedangkan untuk kasus FTV70 rata-rata solusi yang dihasilkan minimum pada nilai Bound = 100. Nilai Bound hasil *trial and error* ini yang akan digunakan pada implementasi.

Hasil *trial and error* parameter max_ctime pada kasus FTV55 dapat dilihat pada

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

Gambar 8. Pada kasus FTV55 rata-rata solusi cenderung naik, namun minimum saat max_ctime berada pada nilai 2%. Sehingga untuk kasus FTV55 akan digunakan nilai parameter max_ctime sebesar 2% dari nilai populasi.

Untuk parameter yang tidak berpengaruh, yaitu max_ctime ditetapkan 1% dari nilai populasi karena max_ctime akan menentukan seberapa sering Forking 2 akan terjadi. Semakin banyak Forking 2, maka semakin besar potensi ditemukan solusi yang lebih baik. Untuk parameter fork_prob digunakan nilai sebesar 0,5 karena nilai tersebut netral untuk probabilitas, dalam artian kemungkinan terjadi atau tidaknya Forking 1 akan sama. Sedangkan parameter Bound ditetapkan sebesar 100 agar solution *space* tidak terlalu besar sehingga membuat pencarian solusi menjadi lama. Rekapitulasi nilai parameter yang digunakan setiap kasus dapat dilihat pada Tabel 10.

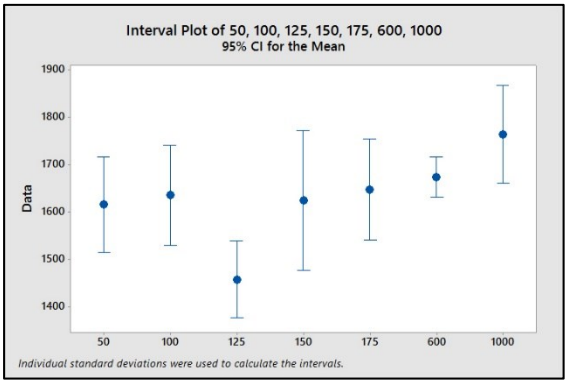
Tabel 10. Nilai parameter pada tiap kasus

Kasus	Parameter		
	max_ctime	fork_prob	Bound
BR17	1%	0,5	100
FTV33	1%	0,5	125
FTV44	1%	0,5	125
FTV55	2%	0,5	100
FTV70	1%	0,5	100

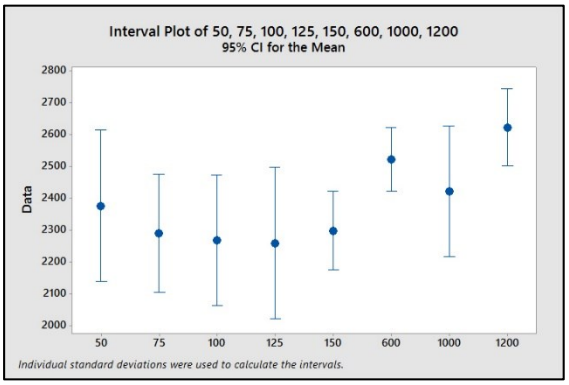
Setelah didapatkan nilai parameter terbaik dilakukan implementasi. Setiap kasus dilakukan replikasi sebanyak 10 kali. Hasil implementasi dapat dilihat pada Tabel 11.

Tabel 11. Hasil implementasi

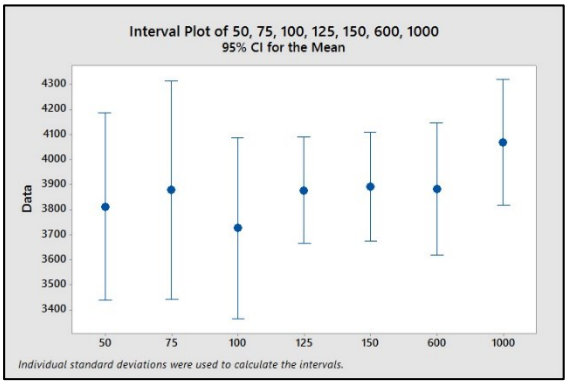
Replikasi	Kasus				
	BR 17	FTV 33	FTV 44	FTV 55	FTV 70
1	39	1556	1976	2792	3772
2	39	1483	2263	2740	3914
3	39	1491	2265	2673	3617
4	39	1606	2034	2909	3774
5	39	1578	2202	3028	3587
6	39	1431	2161	2414	3804
7	39	1588	2224	2439	3322
8	39	1375	2274	2773	3579
9	39	1471	2103	2793	3275
10	39	1437	2251	2496	3671
Min	39	1375	1976	2414	3275
Best Known Solution	39	1286	1613	1608	1950



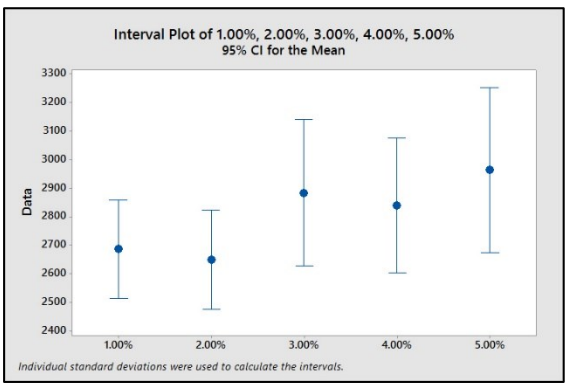
Gambar 5. *Trial* & *error* Bound pada kasus FTV33



Gambar 6. *Trial* & *error* Bound pada kasus FTV44



Gambar 7. *Trial* & *error* Bound pada kasus FTV70



Gambar 8. *Trial* & *error* max_ctime pada kasus FTV55

Hasil implementasi dibandingkan dengan hasil dari EHO, HSA, dan LOA. Perbandingan tersebut ditunjukkan oleh Tabel 12.

Tabel 12. Perbandingan hasil LSA dengan hasil dari EHO, HSA, dan LOA

Kasus	LSA	EHO	HSA	LOA	*Best Known Solution*
BR17	39	39	39	39	39
FTV33	1375	1286	1388	1534	1286
FTV44	2034	1613	1849	2128	1613
FTV55	2414	1608	2110	3521	1608
FTV70	3275	2089	2813	3728	1950

LSA dengan 2-Opt dapat menemukan *best-known solution* pada kasus BR17 (17 kota). Namun, untuk kasus dengan jumlah kota yang lebih banyak seperti FTV33 (34 kota) atau FTV70 (71 kota), LSA dengan 2-Opt tidak dapat menemukan *best-known solution*.

Setelah dilakukan analisa terhadap hasil implementasi, didapatkan bahwa penggunaan *random key encoding* untuk LSA rentan menyebabkan solusi terjebak di *local optima* saat proses pencarian posisi baru oleh *space* & *lead projectile*. Sebagai contoh, misal ada urutan kota 1 - 2 - 3 yang direpresentasikan dengan bilangan acak 0,1 - 0,2 - 0,3. Pada saat proses pencarian posisi baru, *space* & *lead projectile* menggunakan bilangan acak. Sehingga ada kemungkinan hasil dari pencarian posisi baru adalah 0,3 - 0,5 - 0,9 yang jika diterjemahkan menjadi urutan kota akan menghasilkan urutan yang tetap sama, yaitu 1 - 2 - 3. Jika sudah terjebak di *local optima*, penggunaan 2-Opt juga tidak membantu karena 2-Opt hanya mencari kemungkinan solusi yang lebih baik di sekitar solusi yang ditemukan oleh LSA.

Kesimpulan

Berdasarkan penelitian yang telah dilakukan, LSA dengan 2-Opt dapat menyelesaikan ATSP dengan cara mentranslasikan ATSP ke dalam LSA menggunakan *random key encoding*. 2-Opt ditambahkan setelah tahap pencarian posisi baru oleh *projectile* untuk membantu eksploitasi solusi. Rancangan algoritma dapat dilihat pada Tabel 6. Hasil implementasi menunjukkan bahwa LSA dengan 2-Opt yang dirancang dapat menemukan solusi dari kasus ATSP.

Ada 3 (tiga) parameter dalam penelitian ini, max_ctime, Bound, dan fork_prob. Parameter Bound hanya berpengaruh pada kasus FTV33, FTV44, dan FTV70. Pada kasus FTV33 dan FTV44 rata-rata solusi akan minimum pada Bound = 125 dan nilai solusi akan kembali naik saat nilai Bound bertambah. Sedangkan untuk kasus FTV70 rata-rata solusi minimum di nilai Bound = 100 dan akan kembali meningkat saat nilai Bound meningkat. Parameter max_ctime hanya berpengaruh pada kasus FTV55. Parameter fork_prob dan interaksi antar parameter tidak memiliki pengaruh pada setiap kasus.

LSA dengan 2-Opt dapat menemukan *best-known solution* untuk kasus BR17, tetapi tidak dapat menemukan *best-known solution* untuk kasus-kasus lainnya.

Pada penelitian penerapan LSA dengan *local search* untuk menyelesaikan masalah ATSP berikutnya dapat menggunakan metode *encoding* dan *decoding* yang lain, atau menggunakan metode heuristik lainnya untuk melakukan *local search*.

Daftar Pustaka

Brest, J., & Zerovnik, J. (2003). An Approximation Algorithm for the Asymmetric Traveling Salesman Problem. *Elektrotehniski Vestnik/Electrotechnical Review, 70*(1).

Croes, G. (1958). A Method for Solving Traveling-Salesman Problems. *Operations Research, 6*(6), 791-812. doi:https://doi.org/10.1287/opre.6.6.791

Elim, Y. (2018). *Penerapan lion optimization algorithm untuk menyelesaikan kasus asymmetric traveling salesman problem.* Skripsi, Universitas Katolik Parahyangan, Industrial Engineering, Bandung.

Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A New Heuristic Optimization Algorithm: Harmony Search. *Simulation, 76*(2), 60-68. doi:https://doi.org/10.1177/003754970107600201

Gutin, G., & Punnen, A. (2002). *The Traveling Salesman Problem and Its Variations* (Vol. 12). Springer Science & Business Media.

Johnson, D. S., & McGeoch, L. A. (2003). The traveling salesman problem: a case study. In E. Aarts, & J. K. Lenstra, *Local Search in Combinatorial Optimization* (pp. 215-310). Princeton: Princeton University Press. doi:https://doi.org/10.1515/9780691187563-011

Kevin, P. (2016). *Penerapan harmony search algorithm untuk menyelesaikan kasus asymmetric traveling salesman problem.* Skripsi, Universitas Katolik Parahyangan, Industrial Engineering, Bandung.

Lawler, E. L., Lenstra, J. K., Kan, R. A., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Chichester: Wiley.

Reinelt, G. (1991). TSPLIB-A Traveling Salesman Problem Library. *ORSA Journal on Computing, 3*(4), 376-384.

Santosa, I. (2017). *Penyelesaian kasus asymmetric traveling salesman problem untuk meminimasi jarak tempuh menggunakan algoritma elephant herding optimization.* Skripsi, Universitas Katolik Parahyangan, Industrial Engineering, Bandung.

Shareef, H., Ibrahim, A. A., & Mutlag, A. H. (2015). Lightning search Algorithm. *Applied Soft Computing, 36*, 315-333. doi:https://doi.org/10.1016/j.asoc.2015.07.028

Talbi. (2009). *Metaheuristics: From Design to Implementation.* New Jersey: John Wiley & Sons, Inc.

Wang, G.-G., Deb, S., & Coelho, L. d. (2015). Elephant Herding Optimization. *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*, (pp. 1-5). Bali. doi:10.1109/ISCBI.2015.8.

Winston, W. L., & Goldberg, J. B. (2004). *Operations Research: Applications and Algorithms* (4 ed.). Canada: Thomson Learning-Brooks/Cole.

Yazdani, M., & Jolai, F. (2016). Lion Optimization Algorithm (LOA): A nature-inspired metaheuristic algorithm. *Journal of Computational Design and Engineering, 3*(1), 24-36. doi:https://doi.org/10.1016/j.jcde.2015.06.003

DOI: https://doi.org/10.26593/jrsi.v13i2.7760.191-202

This page is intentionally left blank