 https://doi.org/10.56279/orseaj.v15i2.9

Hybrid Metaheuristic Algorithm for the Linear Ordering Problem: The Case of Tanzanian Input-Output Tables

Allen R. Mushi[1]. and Sumaya M. Kagoya[2]

Abstract

Linear Ordering is the problem of finding an acyclic tournament in a complete directed weighted graph that maximizes the sum of the arc weights. This is equivalent to finding an order of elements of a matrix associated with a complete weighted digraph in such a way that the sum of the entries in the upper triangle is as large as possible. The problem has interesting applications, including the triangulation of input-output tables in economics. Linear Ordering has been proven to be NP-Hard and therefore, no polynomial time algorithm is known for its solution. Metaheuristic techniques have been applied with reported success in many instances, including instances of real-world problems. This paper presents a hybrid metaheuristic algorithm that combines Flex Deluge and Simulated Annealing in an attempt to find a near-optimal solution to the linear ordering problem. Tanzanian input-output tables are used as a case study. The results are compared with the performances of Simulated Annealing, Great Deluge, and Flex Deluge algorithms. It is concluded that the new procedure is good for the linear ordering problem with comparable performance in terms of solution quality and better performance in terms of time by 26%.

Keywords: Combinatorial Optimization, Input-output tables, Triangulation, Great Deluge Algorithm, Global heuristics

Introduction

Linear Ordering Problem (LOP) is the problem of finding an acyclic tournament in a complete weighted directed graph in such a way that the sum of the weights is as large as possible (Garmendia et al., 2024). Equivalently, it is the problem of finding an order of the elements of $n \times n$ matrix in such a way that the upper triangle entries are as large as possible (Pérez Lugo et al., 2025). LOP has interesting applications, including the aggregation of individual preferences in sports, archaeological seriation in archaeology, and the triangulation of input-output tables in economics (Benito-Marimon et al., 2025). In archaeology, the studies of ancestry relationships include studies of pottery in graveyards. The items are weighted according to their ground depth and location in graveyards. Determining the age of each pottery is a challenge that requires ordering the items according to the weights that are structured to reflect the relative strengths of the ordering; this turns into solving the LOP (Glover et al., 2020). Generally, the LOP is used to model problems that require a rank ordering of objects using aggregation of individual preferences, so that the ranking matches the individual preferences as closely as possible (Charon & Hudry, 2007). Aggregation of individual preferences has many applications, including the voting theory, ranking of players/teams in sports tournaments (Iranmanesh & Krishnamurti, 2016), and word reordering for machine translation (Tromble & Eisner, 2009). Rankability of data is another

[1] Mzumbe University, Tanzania
Email: allenmushi66@gmail.com
[2] Makerere University Business School, Uganda

application of the LOP in sports, as presented by Cameron (Cameron et al., 2021). In economics, the sectors of the economy are organized in a matrix form called input-output tables, which represent the interdependencies between the sectors of the economy. The ordering of the sectors of the economy in a way that maximizes the sum of weights in the upper triangle is called triangulation and provides very useful interpretations of the economy (Raa, 2009); this is a direct application of the LOP.

Exact Methods

The problem has been proven to be NP-hard (Hartmanis, 1982) and therefore no optimal algorithm is known for the solution of the LOP within a reasonable time. However, exact methods have been attempted mostly through integer programming formulations and solved with some success by various techniques. The most popular approach in the exact methods is the branch and cut. In this method, an integer programming formulation of the LOP polytope is relaxed into a continuous linear programming polyhedron and generates cutting planes to prune infeasibilities from the relaxed polyhedron. The deepest cutting planes are called facets, which are developed through the theory of polyhedral Combinatorics. The generated cutting planes are used to reduce infeasibilities in the polyhedron, which is then solved by the branch and bound method. Many papers are available on this approach, and the most common include the work by Groetschel (Groetschel et al., 1984) and proceedings by Marti and Reinelt (Martí & Reinelt, 2011). Mushi (Mushi, 2010) used the branch and cut method for the LOP and managed to find an optimal solution to the Irish input-output tables, which had 41 sectors of the economy. The branch and bound method is normally faced with the challenge in the selection of the branching node, and consequently, several authors have tried to develop heuristics to improve branching selection for the branch and cut algorithm; these include the primal heuristic procedure (Agrawal et al., 2019). Many techniques in the branch and cut approaches apply the simplex method to solve the relaxed linear programming problem due to its simplicity; however, some authors have attempted to use other methods, such as the work of previous researchers (Mitchell & Borchers, 1996), who applied interior-point and cutting plane methods to the relaxed problem and presented competitive results. Since the problem is NP-hard, exact methods are limited to small problem sizes.

Heuristic Methods

Given the complexity of the problem, most of the recent research work is based on heuristic algorithms. Global heuristic methods have been very successful in many Combinatorial Optimization problems. These methods work by trying to avoid falling into local optima by instituting various measures, including acceptance of bad solutions at some stages in anticipation of better solutions in the future; see (Reeves C, 1993)Click or tap here to enter text. for a thorough description of global heuristic techniques. Popular global heuristic methods include Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA). Duarte (Duarte et al., 2011) presented a TS algorithm for a special type of LOP with cumulative costs. They experimented with 218 instances and obtained good results within a reasonable time. However, they applied real data from the University of Padova with a maximum of only 15 nodes; the rest of the data were generated randomly or collected from the Linear Ordering Library (LOLIB) as described by Marti (Martí et al., 2012). The book by Martí and Reinelt (Martí & Reinelt, 2011) demonstrates the applications of TS, SA, and GA to the LOP and applied instances from the LOLIB. Population-based heuristics have also been applied to the LOP, including the work by Campos (Campos et al., 1999) on the application of scatter search and the articles by Lugo (Lugo et al., 2022) on the

application of a memetic algorithm. They both applied instances from the LOLIB and presented improvements compared to previous work on the same set of data. Other heuristics employ the structure of the Integer Programming formulation and techniques to derive heuristics, such as the articles by Iranmanesh (Iranmanesh & Krishnamurti, 2016), and (Manuel V., 2024) which applied Mixed Integer Programming (MIP) formulation to generate a set of neighborhoods; and (Belloni & Lucena, 2003) who applied the Lagrangian method to the MIP formulation to generate heuristic solutions. They both applied their own randomly generated data or data obtained from LOLIB.

More recent heuristics include the paper on the application of a genetic algorithm with local search (Cergibozan & Tasan, 2019), iterated local search (Valdez & Medina, 2012), differential evolution (Baioletti et al., 2016), and memetic algorithms by Ye (Ye et al., 2014) and (Pérez Lugo et al., 2025). Josu Ceberio et al (Ceberio et al., 2015) introduced a restricted neighborhood structure that identifies areas that cannot generate good solutions. The restricted neighbourhood was applied to memetic and iterated local search heuristics with good success. Pop and Matei (Pop & Matei, 2012) proposed genetic programming heuristic that shows promising results in solving the LOP, performing well against established methods in terms of solution quality. These are only a few articles available from the literature on the applications of heuristic techniques for the LOP. However, most of the research works in the literature applied randomly generated data from their methods or those collected from the LOLIB. It would be interesting to see more research on applications from real-world problems; consequently, this work is based on the real problem of the input-output tables of the Tanzanian economy, which has 79 sectors.

Hybrid Metaheuristic

The work employs a new hybrid heuristic that applies Flex Deluge and Simulated Annealing algorithms in two phases. Flex Deluge (FD) is an extension of the Great Deluge Algorithm (GDA), which is one of the global heuristic techniques. So far, the GDA has only been applied once to the LOP, and it is the article by (Mathias & Mushi, 2015). The next section provides a mathematical programming formulation of the LOP. Then, a section that follows describes the Flex Deluge algorithm with a non-linear decay rate and its adaptation to the LOP. The application of Simulated Annealing to the LOP has been described in (Mushi, 2020) and is therefore not discussed in this work. The representation of the LOP in the research in a way that is suitable for the new heuristic procedure is described next. This is followed by the introduction of the hybrid two-phase heuristic. A summary of results is presented next, followed by a conclusion and proposed further research directions.

Methodology

Mathematical Formulation

Given a complete digraph of n nodes $D_n = (V_n, A_n)$ with a non-negative weight function $C: A \rightarrow R_+$ it is required to find an acyclic tournament of maximum total weight.

Define a variable $x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in D_n \\ 0 & \text{Otherwise} \end{cases}$ and since only one of the edges (i,j) and (j,i) can be in the tournament, then the equation $x_{ij} + x_{ji} = 1$ (1)

applies for all $i, j \in V_n$, $i < j$. Equation (1) is called a minimal equation system. Furthermore, it is necessary to avoid cycles in the constructed complete tournament; it has been established that 3-dicyle inequalities are sufficient to remove all cycles (Groetschel et al., 1984)Click or tap here

to enter text., and these are defined as in (2) to ensure that every three nodes i, j, k of the tournament cannot form a cycle, that is;

$x_{ij} + x_{ji} \leq 2$ for all $i, j, k \in V_n$ and $i \neq j \neq k$ (2)

Where $x_{ij} \in \{0,1\}$ for all $i, j \in V_n$ (3)

The integer programming formulation of the LOP is therefore defined as follows: given the weights c_{ij} between nodes i and j in a complete digraph $D_n = (V_n, A_n)$;

Maximize $\sum_{(i,j)\in A_n} c_{ij} x_{ij}$ (4)

Subject to:

$x_{ij} + x_{ji} = 1$ for all $i, j \in V_n$, $i < j$

$x_{ij} + x_{ji} \leq 2$ for all $i, j, k \in V_n$ and $i \neq j \neq k$

$x_{ij} \in \{0,1\}$ for all $i, j \in V_n$

The acyclic tournament in this formulation can be identified as the entries in the upper triangle of the weighted matrix representation of the problem. More details on the representation of the LOP as a triangulation problem are described in (Mathias & Mushi, 2015)Click or tap here to enter text.. The exact approaches involve the relaxation of the integrality constraints (3) and developing facets to prune infeasibilities from the relaxation of (4). These include k-Fence and Mobius ladder inequalities, as presented by (Groetschel et al., 1984)Click or tap here to enter text.. However, this paper only considers the basic model as defined in (4) since it is sufficient for heuristic implementation; the Flex Deluge heuristic is discussed next.

Flex Deluge Algorithm

Flex Deluge algorithm is an extension of the Great Deluge algorithm that was introduced by Burke and Yuri (Burke & Yuri, 2006)Click or tap here to enter text.. The Great Deluge algorithm's strength lies in the acceptance of bad moves within the control of a level parameter L that is determined by a decay rate function that determines a decay rate value, ΔL. One of the main challenges associated with many global heuristic techniques is the sheer number of parameters that have to be selected and their sensitive effect on the best solution. A Great Deluge algorithm was designed to address such a problem by minimizing the number of parameter selections while maintaining a good solution. The only input parameter in this algorithm is the decay rate value ΔL. The general idea is a simulation of an object in a mountainous space that is under pouring rain. The object wanders randomly in the space, but there is a water level below which it cannot go because of the water. If this water level is L, then the object accepts any area that has a value greater than L. As time goes on, L rises slowly and is finally forced up onto a peak (and the rain stops). However, it has been observed by (Burke & Yuri, 2006) controlling the mechanism of accepting bad moves and slowing down downhill movements may improve the performance of the algorithm. Consequently, they introduced a flexibility mechanism through a coefficient $k_f (0 \leq 0 \leq 1)$ that controls the uphill and downhill movements. For the minimization problem, (Burke & Yuri, 2006) proposed that, at every iteration, if the current solution is X_{curr}, the algorithm should accept a new candidate solution X_{new} if it satisfies the conditions;

$X_{new} \leq X_{curr}$ when $X_{curr} \geq L$

$X_{new} \leq X_{curr} + k_f(L - X_{curr})$ when $X_{curr} < L$

This criterion implies that acceptance of bad moves should not exceed the value of $k_f(L - X_{curr})$ beyond the current move. Note that when $k_f = 0$ the algorithm turns into a greedy one, where only good moves are accepted; when $k_f = 1$ the algorithm turns into the original Great Deluge. Thus k_f is called a 'flexible bumper' and it represents the degree of flexibility. Varying the value of k_f between 0 and 1 allows the algorithm to accept moves flexibly within the range specified and has improved performance over the original Great Deluge algorithm, as reported by (Burke & Yuri, 2006) when applied to the examination timetabling problem. These conditions were stated for a minimization problem, while LOP is a maximization problem, and therefore, the acceptance conditions used change to the following;

$X_{new} \geq X_{curr}$ when $X_{curr} \leq L$

$X_{new} \geq X_{curr} - k_f(L - X_{curr})$ when $X_{curr} > L$

This means if the current solution (X_{curr}) is below the level (L), the new move (X_{new}) is accepted if it is better than the current solution (X_{curr}). In this case, the algorithm accepts only better solutions (greedy). However, when the current solution is better than the water level (L), then the algorithm may accept bad moves that are below the current solution by a value $k_f(L\text{-}X_{curr})$ in anticipation of better moves in the future by avoiding falling into local solutions. The general algorithm is shown in Figure 1.

Flex Deluge Algorithm
Set the Initial Solution X_o;
Calculate Initial cost $f(X_o)$;
Initial Level L = 0;
Specify input parameter ΔL; //decay rate
Specify flexibility coefficient $0 \leq k_f \leq 1$;
While stopping criteria is not satisfied {
Define neighbourhood $N(X_o)$
Randomly select a candidate solution $X \in N(X_o)$;
    If $f(X) \geq f(X_o)$ AND $f(X) \leq L$
    Accept new solution ($X_o = X$);
    Else if $f(X) \geq (f(X_o) - k_f(L - f(X_o)))$ AND $f(X) > L$
    Accept new solution ($X_o = X$);
Else
    Reject new solution;
Update Level by decay rate
Function L = f(L, ΔL);}
X_o is the best solution;
End Flex Deluge

Figure 1: Flex Deluge Algorithm

Given this general heuristic procedure, it is necessary to specify how the algorithm is to be implemented for the LOP. These are: the representation of the problem, the structure of the objective function, how to generate an initial solution, the structure of the neighbourhood, the choice of the decay rate, and stopping criteria; these are discussed next.

Problem Representation

The solution is a tournament $T_n \subseteq D_n$ and is represented by a matrix $X_{n\times n}$ where $x_{ij} = \begin{cases} 1 & if\ (i,j) \in T_n \\ 0 & otherwise \end{cases}$ for all $x_{ij} \in X$.

The weights of the digraph are represented by a matrix $c_{n\times n}$ where c_{ij} is the weight of an edge between nodes i and j, and the objective function is defined as: $f(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$.

Generating an initial solution

It is noted that a quick initial feasible solution can easily be obtained by picking all entries of the upper triangle in the solution matrix, i.e., define X_0 such that:

$x_{ij} = \begin{cases} 1 & if\ i < j \\ 0 & otherwise \end{cases}$ for all $x_{ij} \in X_0$. These entries ensure that only one of the edges (i,j) and $(j,i)$is selected, and therefore the basic inequalities, i.e., constraints (1) are satisfied. Furthermore, since all upper triangle entries are selected, the chosen initial solution forms a tournament that also satisfies the 3-dicycle inequalities, i.e., constraints (2) and therefore acyclic. Table 1 shows an example of a weighted matrix and Table 2 depicts the initial solution (X_0) as described above. The solution value, which is the sum of weights of the upper triangle, is $f(X_0) = (7 + 2 + 3 + 9 + 8 + 1) = 30$. The associated tournament is shown in Figure 2, which is an acyclic digraph.

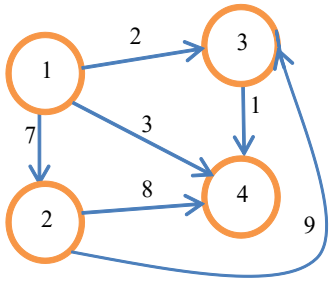
Table 1 Example of LOP Weight matrix

	1	**2**	**3**	**4**
1	0	**7**	**2**	**3**
2	9	0	**9**	**8**
3	4	8	0	**1**
4	5	6	7	0

Table 2: Example of initial solution

	1	**2**	**3**	**4**
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

Figure 2: Example of a solution tournament



Neighbourhood Solution

Several neighbourhood structures have been proposed in the literature, and the most common is the ***Insert*** and ***Interchange*** moves, as described by (Sakuraba & Yagiura, 2010). ***Insert*** move involves taking a node in position i and insert it after another node in position j $(i < j)$ or insert it before another node in position j $(i > j)$. The ***Interchange*** move, on the other hand, involves the exchange of nodes in position i and position j. Although (Schiavinotto & Stützle, 2003) reported that the interchange move gives worse results than the insert move in their research, this work prefers the interchange move due to the nature of the solution structure and ease of generating feasible solutions. However, the results were compared with insert moves. With the interchange move, hereby called the Swap move, every move involves the removal of one edge and the addition of another in the solution and therefore always ensures that the basic inequalities are satisfied. Example in Table 3 displays the solution X_1 after a swap of node 1 and node 2 positions, as highlighted. Table 4 displays the resulting solution value (32) after the swap.

Table 3: X_1; Swap of $(1, 2)$

	1	2	3	4
1	0	**0**	1	13
2	**1**	0	1	1
3	0	0	0	1
4	0	0	0	0

Table 4: C; $f(X_1) = 32$

	1	2	3	4
1	0	**7**	2	3
2	**9**	0	9	3
3	4	8	0	1
4	5	6	7	0

The positions of nodes i and j for swap moves are randomly selected and therefore create a set of neighbourhood solutions as candidates to the algorithm. After three more swap moves (3,1), (4,1), and (4,3) in the example, as shown in Note that the solution is always obtained by the product CX where the changing values are in X while the values of C remain constant.

Table 5: X_4; Swaps (3,1), (4,1), (4,3)

	1	2	3	4
1	**0**	**0**	**0**	**0**
2	**1**	**0**	**1**	**1**
3	**1**	**0**	**0**	**0**
4	**1**	**0**	**1**	**0**

Table 6: C; $f(X_4) = 42$

	1	2	3	4
1	0	**7**	**2**	**3**
2	**9**	0	9	8
3	**4**	8	0	**1**
4	**5**	6	**7**	0

, the solution is presented in Table 6 with a value of 42. Note that the solution is always obtained by the product CX where the changing values are in X while the values of C remain constant.

Table 5: X_4; Swaps (3,1), (4,1), (4,3)

	1	2	3	4
1	**0**	**0**	**0**	**0**
2	**1**	**0**	**1**	**1**
3	**1**	**0**	**0**	**0**
4	**1**	**0**	**1**	**0**

Table 6: C; $f(X_4) = 42$

	1	2	3	4
1	0	**7**	**2**	**3**
2	**9**	0	9	8
3	**4**	8	0	**1**
4	**5**	6	**7**	0

Extracting Linear Order from Solution

Once the solution has been generated, as in Note that the solution is always obtained by the product CX where the changing values are in X while the values of C remain constant.

Table 5: X_4; Swaps (3,1), (4,1), (4,3)

	1	2	3	4
1	**0**	**0**	**0**	**0**
2	**1**	**0**	**1**	**1**
3	**1**	**0**	**0**	**0**
4	**1**	**0**	**1**	**0**

Table 6: C; $f(X_4) = 42$

	1	2	3	4
1	0	**7**	**2**	**3**
2	**9**	0	9	8
3	**4**	8	0	**1**
4	**5**	6	**7**	0

the linear order is simply extracted by summing the columns of the solution matrix. The least total is the first node in the order, followed by the next until the last total has been used, as demonstrated in Table 7. In case of ties in the column sums, pick the first encountered column from left to right. The order in the above example is 2, 4, 3, 1, with a value of 42 as shown in Table 8. The input parameters in the algorithm are the decay rate function and the flexibility coefficient k_f.

Table 7 Extraction of Linear Order

	1	2	3	4
1	0	**0**	**0**	**0**
2	**1**	0	1	1
3	**1**	0	0	**0**
4	**1**	0	**1**	0
Sum	**3**	**0**	**2**	**1**
Position	**4**	**1**	**3**	**2**

Table 8: Weight Matrix for the Linear order (2,4,3,1).

	2	4	3	1
2	0	**8**	**9**	9
4	6	0	7	5
3	8	**1**	0	4
1	**7**	**3**	**2**	0

Decay rate function

The standard decay rate function for the Great Deluge is a linear function $L = L + \Delta L$. Landa-Silva (Landa-Silva & Obit, 2008) proposed the use of a non-linear decay rate function to improve the search for a better-quality solution, defined as $L = L \times e^{-\delta(random[(Min,Max)])} + \beta$. Parameters δ, Min and Max are used to control the speed of decay, while β controls the shape of the decay function. The higher the values of Min and Max, the faster the water level goes up, and consequently, the search quickly achieves improvement, but at the expense of getting stuck early in the local optima. Parameter β represents the minimum expected penalty corresponding to the best solution. Apart from β the best values of other parameters found in this application were obtained through experimentation. The non-linear decay function was designed to improve the water level changes to efficiently guide the search process for a better solution, and it worked very well for applications in academic timetabling problems (Landa-Silva & Obit, 2008). It is worth applying the same to the LOP.

Stopping criteria ($MaxIter$)

The algorithm must be run for a sufficient number of iterations to ensure a complete search of the solution space, creating ample chances of avoiding local optima. This value is the maximum number of iterations ($MaxIter = 1{,}000{,}000$) and is obtained through experimentation. The next section presents the proposed hybrid heuristic that combines Flex Deluge and Simulated Annealing.

A hybrid metaheuristic for the LOP

Simulated Annealing is a very popular heuristic that has been successful in many Combinatorial Optimization problems, including academic timetabling (Elmohamed et al., 1998), traveling salesman (Elmohamed et al., 1998), mobile wireless systems (Kwak & Shroff, 2019), (Zomaya et al., 2003), therapy treatment(Palmqvist et al., 2015), (Li et al., 2014) to name but a few. A discussion on the applications of Simulated Annealing is also provided by prior researchers (Welsh, 1989). Simulated Annealing is a simulation of the physical cooling process, where a solution is accepted if it has improved, but a bad solution can also be accepted by a probability function $f(T, \sigma) = e^{\frac{-\sigma}{T}}$. More discussion on the application of this algorithm to the linear ordering problem, with application to Tanzanian input-output tables, is found in the work of (Mushi, 2020) Click or tap here to enter text.. However, the performance of the Simulated Annealing greatly depends on the selection of the initial solution. In this case, Flex Deluge is used to provide a good, feasible, and quick initial solution that is fed into Simulated Annealing in the second phase. The algorithm is presented in Figure 3 below;

Hybrid Metaheuristic Algorithm

Phase I: Flex Deluge
Set the Initial Solution X_o;
Calculate Initial cost $f(X_o)$;
Initial Level L = 0;
$X_{new}=X_o$;
//flexibility
Specify flexibility coefficient $0\leq k_f\leq 1$;
While stopping criteria is not satisfied {
Define neighbourhood $N(X_{new})$
Randomly select a candidate solution $X\in N(X_{new})$;
If $f(X)\geq f(X_{new})$ AND $f(X)\leq L$
Accept new solution
($X_{new}=X$);
Else if $f(X)\geq(f(X_{new})-k_f(L-f(X_{new})))$ AND $f(X)>L$
Accept new solution
($X_{new}=X$);
Else
Reject new solution;

Update the level $L = L \times (e^{-\delta(random([Min,Max])}) + \beta$;
}
X_{new} is the best solution so far;

Phase II: Simulated Annealing
σ =objective(X_{new})-objective(X);
Initialize temperature T;
Initialize freezing point $Freeze$
While $T > Freeze$ {
If σ>0 accept solution
X_{new} = X;
Else
Generate a random
Number r between 0 and 1
If $r > e^{\frac{-\sigma}{T}}$ accept solution
X_{new} = X;
Else reject solution
Update temperature $T = \alpha T$;;
Where $0.8 \leq \alpha \leq 1$
Find X in the neighborhood of X_{new}, X=neighbour (X_{new});
σ =objective(X_{new})-objective(X);
}
X_{new} is the best solution found

End hybrid algorithm

Figure 3: hybrid metaheuristic algorithm

The best values of α are $\alpha = 0.99$ and $Freeze = 0.001$ which were found through experimentation. The next section gives a summary of the results.

Summary of Research Results and Discussion

The algorithm was implemented in a C++ compiler (fixed seed, CPU time measurement) and run on a Windows 11, Intel® Core i5, 1.6 GHz processor, and applied to the Tanzanian input-output tables, which have 79 sectors of the economy. The same data were used in previous work with a Great Deluge (Mathias & Mushi, 2015) and Simulated Annealing (Mushi, 2020) Click or tap here to enter text..

Parameter Tuning

The performance of metaheuristics is highly dependent on parameter settings. A preliminary hyperparameter search using the Tanzanian Input Output Tables instances was conducted to establish effective general parameters, and the results are presented in
Table 9.

Table 9: Parameter tuning

Parameter	Phase I (FD)	Phase II (SA)	Value
Decay Rate	$L = L \times e^{-\delta(random[(Min,Max)])} + \beta.$	N/A	N/A
Flexibility Coefficient k_f	0.5	N/A	0.5
β	1	N/A	1
δ	0.00079	N/A	0.00079
Type of Moves	Swap, Insert	Swap	Swap (best)
Min	10,000	N/A	10,000
Max	15,000	N/A	15,000
Stopping Criteria	MaxIter	Freeze	1,000,000
		0.001	
Cooling function	N/A	Geometric $T = \alpha T$	N/A
Initial Temperature	Hot enough to allow exploration of the solution space	N/A	1,000
α	N/A	$0.8 \leq \alpha \leq 1$	0.99

These parameters were obtained through step-by-step procedures involving 30 runs of different random number seeds and a confidence test. The first step was to find the good parameters for the non-linear function in the Flex Deluge algorithm. The value of $k_f = 0.5$ was found to perform better as the flexibility coefficient. This was done by fixing all other parameters and performing 30 seed runs, varying k_f by 0.1 intervals from 0 to 1, where $k_f = 0.5$ had the highest mean (837,838) and median (838,800). Furthermore, through similar testing, the value of $\beta = 1.0$ was found to perform better. For values of $\beta < 1$ the algorithm did not converge and had to be halted,

while the values of $\beta > 1$ resulted in premature convergence into low-quality solutions. Similarly, experimentation was done by fixing the minimum and maximum values to 10,000 and 15,000, respectively, and finding the corresponding best δ value. Table 10 shows the results of varying the values of δ on the solution (Sum of the upper triangle weights) and execution time. The best value of $\delta = 0.00079$, resulting in a better solution at a shorter period of time (48.8 seconds), is highlighted.

Table 10: Finding the best delta (δ) value

Fixed $\beta = 1.0$; $min = 10,000$; $max = 15,000$		
δ	**Solution**	**Time (Sec)**
0.0007	838,094	17.836
0.00071	839,515	19.897
0.00072	833,764	22.206
0.00073	839,516	24.911
0.00074	839,617	28.062
0.00075	839,258	31.212
0.00076	839,401	35.099
0.00077	839,828	39.332
0.00078	839,835	43.657
0.00079	839,842	48.798
0.0008	839,842	55.421
0.0085	839,842	97.102
0.009	839,842	168.062

Time increases sharply as δ increases, especially beyond δ = 0.0008, indicating a nonlinear growth in computational effort as shown in the Error Bar (Figure 4). The solution values are tightly clustered around 839,842, with a few lower outliers (e.g., 833,764). Time values show a widespread, with a median around 39 seconds and a long tail reaching up to 168 seconds. This contrast highlights that while the solution quality stabilizes, the time cost escalates significantly with δ (see Boxplot in Figure 5).

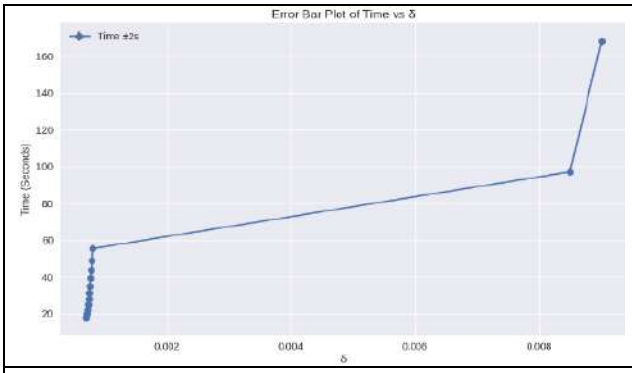


Figure 4: Error Bar Plot: Time vs δ

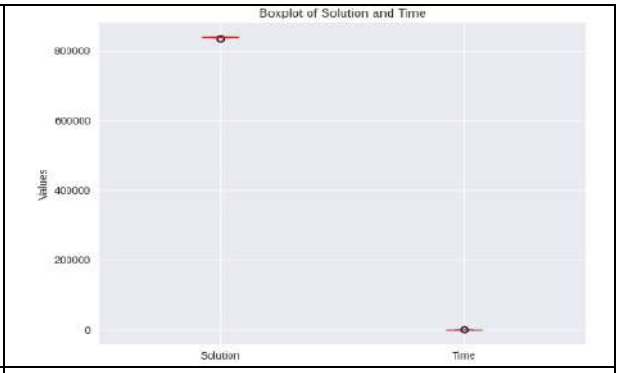


Figure 5: Boxplot: Solution vs Time

The results of the Swap and Insert moves are compared as shown in

Table 11. The mean difference between Swap and Insert solutions is 22,035.81, with a standard deviation of 30,140.30. The 95% confidence interval for the mean difference is (5,975.18, 38,096.44). This indicates that, on average, the Swap Solution values are about 22,036 units higher than the Insert Solution. The Swap algorithm has a mean execution time of 0.781 sec with a 95% confidence interval of (0.697, 0.865). The Insert algorithm has a mean execution time of 1.053 sec with a 95% confidence interval of (0.895, 1.211). Insert is slower on average and shows greater variability. This is expected because of the structure of the solution (0-1 matrix), where more time is required to move values in the permutation matrix in order to allow space for the insertion of an item.

Table 11: Comparison of results for Swap and Insert Moves

Seed	**Swap**		**Insert**	
	Solution	**Time (Sec)**	**Solution**	**Time (Sec)**
1	839,603	0.801	818,297	0.761
2	839,842	0.712	835,287	1.134
3	839,482	0.793	828,526	0.716
4	839,482	0.734	831,722	1.12
5	839,837	0.754	837,567	1.108
6	839,805	0.676	829,662	1.237
7	839,837	0.705	837,298	1.863
8	839,721	0.731	793,660	0.687
9	839,812	0.731	809,359	0.682
10	839,815	0.751	813,372	0.764
11	839,842	0.713	813,921	1.217
12	839,937	1.338	821,153	1.133
13	839,842	0.72	829,438	1.193
14	839,836	0.744	834,425	1.124
15	839,842	0.899	714,552	1.113
20	839,791	0.698	835,514	0.993
Mean	**839,770**	**0.78125**	**817,735**	**1.053**
Std	**128**	**0.1524219**	**29,168**	**0.288**

The rest of the results are generated from the swap moves. The next step was to fix the values of β and δ and change the ranges of min and max as shown in
Table 12. Similar experimentation with different values of min and max shows that the shortest execution time with a better solution found so far is 37.7 seconds at $min = 12{,}55$ and $max = 13{,}000$. The same solution was obtained from Simulated Annealing in only 0.365 seconds (Mushi, 2020)Click or tap here to enter text.. Simulated Annealing was, therefore, faster than the non-linear Flex Deluge algorithm in this particular case.

Table 12: Finding min and max values

$\boldsymbol{\beta = 1.0; \delta = 0.00079}$			
$\boldsymbol{min}$	$\boldsymbol{max}$	**Solution**	**Time (Sec)**

10,000	14,000	839,792	40.637
10,000	14,500	839,842	46.300
10,000	16,000	839,842	60.967
10,000	20,000	839,842	101.308
11,000	14,000	839,842	66.013
11,000	12,000	839,367	22.200
11,000	12,500	839,738	33.640
12,000	12,500	839,486	24.470
12,000	12,900	839,738	44.108
12,000	13,000	839,842	48.243
12,400	13,000	839,842	40.249
12,500	12,800	839,822	29.478
12,500	12,900	839,842	39.789
12,500	13,000	839,842	37.660
12,600	13,000	839,444	32.954
13,000	14,000	839,842	110.716
13,500	14,000	839,842	85.517
14,000	14,500	839,842	119.248
14,000	15,000	839,842	246.965

As explained earlier, however, the hybrid metaheuristic aims to use a non-linear Flex Deluge to find a fast initial solution that can be used to speed up convergence in the Simulated Annealing phase. It was therefore decided to use a lower range of min and max to give a fast solution without necessarily converging to the best one, and apply it as an initial solution to the Simulated Annealing algorithm.

Table 13 presents the results of an experiment in finding a fast initial solution by varying min and max values. It was proposed to pick a solution value of 603,384 obtained after 0.012 seconds with $min = 100$ and $max = 800$. This is drawn from the fact that the solution value obtained is better than the rest in the tested range, as highlighted.

Table 13: Finding min and max for the initial solution

$\beta = 1.0; \delta = 0.00079$				
Min	**Max**	**Solution**	**Iterations**	**Time (Sec)**
100	200	587,191	75.000	0.004
100	300	590,747	103.000	0.005
100	400	590,853	130.000	0.007
100	500	592,778	163.000	0.010
100	600	592,771	188.000	0.009
100	700	599,148	220.000	0.011
100	800	603,384	253.000	0.012
100	900	602,053	298.000	0.013
100	1,000	599,245	347.000	0.017

1,000	1,100	590,574	136.000	0.006
1,000	1,200	591,799	182.000	0.010
1,000	1,300	602,779	233.000	0.010
1,000	1,400	598,390	299.000	0.013
1,000	1,500	602,581	349.000	0.017

The initial solution was then taken into the second phase and applied in the Simulated Annealing algorithm. The results were then generated by varying the random seed value in the algorithm from 1 to 30, as shown in the Table 14. Results show solution values for the first phase of the algorithm (Flex-Delude) and time taken, and the second phase (Simulated Annealing) and the time taken.

Table 14: Results of the Algorithm by varying 30 seed values

Runs	Flex Deluge	Time (sec)	Hybrid	Time (sec)	Total Time(sec)
1	593,209	0.0110	839,842	0.2580	0.2690
2	592,049	0.0130	839,721	0.2890	0.3020
3	614,346	0.0130	839,104	0.1900	0.2030
4	588,451	0.0130	837,651	0.1460	0.1590
5	591,499	0.0100	839,463	0.2520	0.2620
6	590,519	0.0150	838,709	0.2430	0.2580
7	591,285	0.0140	839,346	0.1920	0.2060
8	598,453	0.0150	837,915	0.1730	0.1880
9	589,393	0.0140	839,812	0.2030	0.2170
10	591,338	0.0160	837,743	0.2650	0.2810
11	588,155	0.0160	839,797	0.2080	0.2240
12	588,962	0.0110	837,514	0.1890	0.2000
13	586,597	0.0100	839,550	0.2150	0.2250
14	590,255	0.0140	839,730	0.2460	0.2600
15	593,921	0.0160	839,738	0.2050	0.2210
16	592,617	0.0150	839,260	0.2250	0.2400
17	590,901	0.0190	834,587	0.1810	0.2000
18	591,917	0.0160	839,004	0.2260	0.2420
19	589,343	0.0160	839,841	0.1970	0.2130
20	597,856	0.0140	839,773	0.2830	0.2970
21	597,976	0.0140	838,759	0.2550	0.2690
22	591,332	0.0160	831,398	0.3840	0.4000
23	608,334	0.0170	838,546	0.1680	0.1850
24	590,027	0.0140	839,798	0.1740	0.1880
25	592,871	0.0140	839,408	0.2580	0.2720
26	590,410	0.0140	835,724	0.2680	0.2820
27	589,651	0.0140	836,098	0.3010	0.3150
28	593,442	0.0180	839,011	0.1810	0.1990
29	588,900	0.0110	839,829	0.1560	0.1670
30	588,993	0.0120	839,838	0.1450	0.1570

Mean	**592,767**	**0.0142**	**838,554**	**0.2225**	**0.2367**
Std	**5,735**	**0.0022**	**1,876**	**0.0522**	**0.0523**

The mean of the Hybrid algorithm solution is approximately 838,554, with a standard deviation of 1,876. The 95% confidence interval for the mean is (837,841.89, 839,267.31). Furthermore, the mean of the total time is 0.22 with a standard deviation of 0.052. The 95% confidence interval shows that the true mean total time lies between 0.21684 seconds and 0.25656 seconds. In the test runs of the results, the maximum value is 839,842, obtained at a seed of 1, and took 0.269 seconds. Compared with the SA, the same solution was obtained at 0.365 seconds; the hybrid metaheuristic shows an improvement of 26% in terms of time. Furthermore, the behaviour of the performance of the hybrid algorithm is compared with previous works on the same problem in terms of the quality of the solution and time in seconds. These previous works are the Simulated Annealing and the Great Deluge algorithms. These are also compared with the non-linear Flex Deluge results obtained before the hybrid implementation. Table 15 shows a sample of results obtained by iterations of the four algorithms.

Table 15: Sample iterations against solution values

Iterations	Flex Deluge	SA	GDA	Hybrid
1	585481	585481	585481	585481
2	585481	585481	585481	585481
3	585501	585481	585501	585481
4	585501	585481	585501	585481
5	585589	585481	585589	585481
6	585589	585481	585589	585481
7	585589	585488	585589	585481
8	585589	585488	585589	585481
9	585604	585517	585604	585481
10	585611	585517	585611	585481
22545	839842	839842	836924	839603
22546	839842	839842	838924	839603
22547	839842	839842	838924	839603
22548	839842	839842	838924	839842
22549	839842	839842	838924	839842
22550	839842	839842	838924	839842
22551	839842	839842	838924	839842
22552	839842	839842	838924	839842
22553	839842	839842	838924	839842
22554	839842	839842	838924	839842
22555	839842	839842	838924	839842

A clear picture of the convergence behavior of the four algorithms is shown in Figure 6.

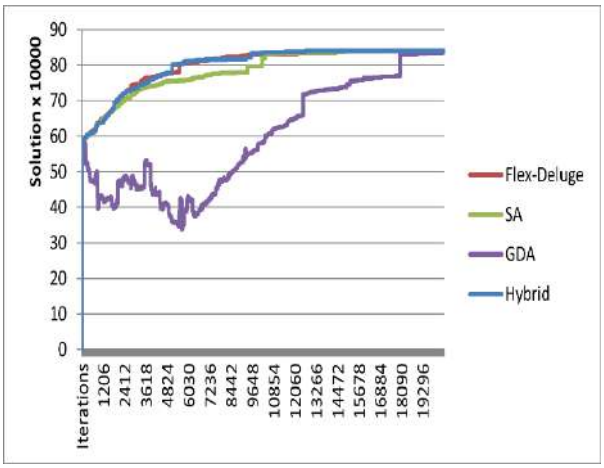


Figure 6: Solution values against iterations

The four heuristics show different paths taken before convergence, where the Great Deluge follows a significantly different path with major fluctuations at the initial stages before convergence to the best solution found. On the other hand, the Flex Deluge and hybrid algorithms follow a very closely related path in the initial stages and join the Simulated Annealing path in the final stages. This is expected since the hybrid heuristic starts with Flex Deluge and ends with Simulated Annealing algorithms. Initially, Simulated Annealing follows a slow convergence path compared to the hybrid heuristic, but later joins the same path, indicating that Flex Deluge performs better in the initial stages than Simulated Annealing. In general, the presented hybrid heuristic outperformed the compared heuristics in terms of execution time while converging to the same best solution.

Conclusion and Further Research Directions

This project aimed to develop a hybrid metaheuristic algorithm that combines Flex Deluge and Simulated Annealing algorithms in finding a solution to the LOP, which is the first attempt by this heuristic algorithm. The heuristic is applied to the Tanzanian input-output tables, which have 79 sectors of the economy. The Great Deluge and Simulated Annealing algorithms have been previously applied to the same problem and the same test data, and therefore compared the performances of the four heuristics in terms of solution quality and time. The hybrid heuristic has shown better performance in terms of time (by 26%) and compares well with the Simulated Annealing; both algorithms lead to the same good solution value. After thorough experimentation, the best choice of parameters for the problem is as follows: $k_f = 0.5$, $\beta = 1.0$, $\delta = 0.00079$, $min = 100$ and $max = 800$. It is therefore concluded that the presented hybrid heuristic that combines Flex Deluge and Simulated Annealing is a good algorithm for the LOP, and in this case, it performed better than the Great Deluge, Flex Deluge, and Simulated Annealing in terms of time with the same quality of the solution.

As pointed out in the introduction, LOP has been extensively studied in the literature, from exact to heuristic methods. However, most of the tested instances come from a general library (LOLIB), which contains mostly randomly generated problems. Testing of more real-world problem instances may provide better insights into the problem. Working on problems in sports tournaments and machine translation are areas of further research as well. Furthermore, there are several variations of the LOP that have been introduced in the literature and require further study. These include LOP with cumulative costs (Bertacco et al., 2008), quadratic LOP(Buchheim et al.,

2010), space allocation (Simmons, 1969), checkpoint ordering(Hungerländer, 2017), facility layout (Kothari & Ghosh, 2012), and many others. These have interesting real-world applications that are worth further research.

Practical Implications

The Linear Ordering Problem has many applications, as explained in the Introduction section. One of the direct applications is the triangulation of the Input-Output tables in the Country's economy. This has been demonstrated in this paper by triangulating Tanzanian Input-Output tables. Since the problem is NP-hard, the design of algorithms that improve the performance of results has a significant impact on the analysis of economies.

References

Agrawal, R., Iranmanesh, E., & Krishnamurti, R. (2019). Primal heuristic for the linear ordering problem. *ICORES 2019 - Proceedings of the 8th International Conference on Operations Research and Enterprise Systems*, 151–156. https://doi.org/10.5220/0007406301510156

Baioletti, M., Milani, A., & Santucci, V. (2016). Linear Ordering Optimization with a Combinatorial Differential Evolution. *Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015*, *January 2017*, 2135–2140. https://doi.org/10.1109/SMC.2015.373

Belloni, A., & Lucena, A. (2003). Lagrangian Heuristics for the Linear Ordering Problem. In *Metaheuristics: Computer Decision-Making. Applied Optimization* (Issue January 2001, pp. 37–63). Springer. https://doi.org/10.1007/978-1-4757-4137-7_3

Benito-Marimon, M., Martí, R., & Martínez-Gavara, A. (2025). Multiple linear ordering problem. *European Journal of Operational Research*, *October*. https://doi.org/10.1016/j.ejor.2025.10.040

Bertacco, L., Brunetta, L., & Fischetti, M. (2008). The Linear Ordering Problem with cumulative costs. *European Journal of Operational Research*, *189*(3), 1345–1357. https://doi.org/10.1016/j.ejor.2006.03.071

Buchheim, C., Wiegele, A., & Zheng, L. (2010). Exact algorithms for the quadratic linear ordering problem. *INFORMS Journal on Computing*, *22*(1), 168–177. https://doi.org/10.1287/ijoc.1090.0318

Burke, E. K., & Yuri, B. (2006). Solving Exam Timetabling Problems with the Flex-Deluge Algorithm. *In Proceedings of the 6th International Conference Practice and Theory of Automated Timetabling VI (PATAT 2006), Brno, Czech Republic, 30 August–1 September 2006; Pp. 370–372. 62.*, *January 2006*, 9–11.

Cameron, T. R., Charmot, S., & Pulaj, J. (2021). On the linear ordering problem and the rankability of data. *Foundations of Data Science*, *3*(2), 133. https://doi.org/10.3934/fods.2021010

Campos, V., Laguna, M., & Martí, R. (1999). Scatter search for the linear ordering problem. In *New ideas in optimization*. McGraw-Hill.

Ceberio, J., Mendiburu, A., & Lozano, J. A. (2015). The Linear Ordering Problem Revisited. *European Journal of Operational Research*, *241*(3), 686–696.

Cergibozan, C., & Tasan, A. S. (2019). *Solving the Linear Ordering Problem Using a Genetic Algorithm with Local Search*. Springer International Publishing. https://doi.org/10.1007/978-3-319-93488-4_16

Charon, I., & Hudry, O. (2007). A survey on the linear ordering problem for weighted or unweighted tournaments. *4or*, *5*(1), 5–60. https://doi.org/10.1007/s10288-007-0036-6

Duarte, A., Laguna, M., & Martí, R. (2011). Tabu search for the linear ordering problem with cumulative costs. *Computational Optimization and Applications*, *48*(3), 697–715. https://doi.org/10.1007/s10589-009-9270-5

Elmohamed, M. A. S., Coddington, P., & Fox, G. (1998). A comparison of annealing techniques for academic course scheduling. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1408*, 92–112. https://doi.org/10.1007/bfb0055883

Garmendia, A. I., Ceberio, J., & Mendiburu, A. (2024). Applicability of Neural Combinatorial Optimization : A Critical View Applicability of Neural Combinatorial Optimization : *ACM Transactions on Evolutionary Learning and Optimization*, *4*(3), 1–26. https://doi.org/10.1145/3647644

Glover, F., Klastorin, T., & Klingman, D. (2020). *Optimal Weighted Ancestry Relationships* (Management, Vol. 20, Issue 8). INFROMS.

Groetschel, M., Juenger, M., & Reinelt, G. (1984). Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, *32*(6), 1195–1220. https://doi.org/10.1287/opre.32.6.1195

Hartmanis, J. (1982). Computers and Intractability: A Guide to the Theory of NP-Completeness (Michael R. Garey and David S. Johnson). *SIAM Review*, *24*(1), 90–91. https://doi.org/10.1137/1024022

Hungerländer, P. (2017). The checkpoint ordering problem. *Optimization*, *66*(10), 1699–1712. https://doi.org/10.1080/02331934.2017.1341507

Iranmanesh, E., & Krishnamurti, R. (2016). Mixed Integer Program heuristic for Linear Ordering Problem. *ICORES 2016 - Proceedings of the 5th International Conference on Operations Research and Enterprise Systems*, *Icores*, 152–156. https://doi.org/10.5220/0005710701520156

Kothari, R., & Ghosh, D. (2012). The single row facility layout problem: state of the art. *Opsearch*, *49*(4), 442–462. https://doi.org/10.1007/s12597-012-0091-4

Kwak, J., & Shroff, N. B. (2019). Simulated Annealing for Optimal Resource Allocation in Wireless Networks with Imperfect Communications. *2018 56th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2018*, 903–910. https://doi.org/10.1109/ALLERTON.2018.8635910

Landa-Silva, D., & Obit, J. H. (2008). Great deluge with non-linear decay rate for solving course timetabling problems. *2008 4th International IEEE Conference Intelligent Systems, IS 2008*, *1*, 811–818. https://doi.org/10.1109/IS.2008.4670447

Li, H. T., Zhang, Y. L., Zheng, C. H., & Wang, H. Q. (2014). Simulated annealing based algorithm for identifying mutated driver pathways in cancer. *BioMed Research International*, *2014*. https://doi.org/10.1155/2014/375980

Lugo, L., Segura, C., & Miranda, G. (2022). A diversity-aware memetic algorithm for the linear ordering Problem. *Memetic Computing*, *14*(4), 395–409. https://doi.org/10.1007/s12293-022-00378-5

Manuel V. (2024). A linear ordering problem with weighted rank. *Journal of Combinatorial Optimization*, *47*(2). https://doi.org/10.1007/s10878-024-01109-x

Martí, R., & Reinelt, G. (2011). The Linear Ordering Problem Exact and Heuristic Methods in Combinatorial Optimization. In *Applied Mathematical Sciences -- 175*. Springer Berlin Heidelberg.

Martí, R., Reinelt, G., & Duarte, A. (2012). A benchmark library and a comparison of heuristic

methods for the linear ordering problem. *Computational Optimization and Applications*, *51*(3), 1297–1317. https://doi.org/10.1007/s10589-010-9384-9

Mathias, A., & Mushi, A. R. (2015). Great Deluge Algorithm for the Linear Ordering Problem: The Case of Tanzanian Input-Output Table. *International Journal of Information Technology and Computer Science*, *7*(7), 28–34. https://doi.org/10.5815/ijitcs.2015.07.04

Mitchell, J. E., & Borchers, B. (1996). Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, *62*(1), 253–276. https://doi.org/10.1007/bf02206819

Mushi, A. (2010). The linear ordering problem: an algorithm for the optimal solution. *African Journal of Science and Technology*, *6*(1), 51–64. https://doi.org/10.4314/ajst.v6i1.55167

Mushi, A. (2020). Simulated Annealing Algorithm for the Linear Ordering Problem : The Case of Tanzania Input Output Tables. *Tanzania Journal of Science*, *46*(2), 281–289.

Palmqvist, T., Dos Santos Matias, L., Marthinsen, A. B. L., Sundset, M., Wanderås, A. D., Danielsen, S., & Toma-Dasu, I. (2015). Radiobiological treatment planning evaluation of inverse planning simulated annealing for cervical cancer high-dose-rate brachytherapy. *Anticancer Research*, *35*(2), 935–939.

Pérez Lugo, L. J., Segura, C., & Miranda, G. (2025). Future trends in the design of memetic algorithms: the case of the linear ordering problem. *Neural Computing and Applications*, *37*(18), 12471–12485. https://doi.org/10.1007/s00521-025-11171-z

Pop, P. C., & Matei, O. (2012). A genetic programming approach for solving the linear ordering problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7209 LNAI*(PART 2), 331–338. https://doi.org/10.1007/978-3-642-28931-6_32

Raa, T. Ten. (2009). Input-output economics: Theory and applications: Featuring Asian economies. *Input-Output Economics: Theory and Applications: Featuring Asian Economies*, *October 2009*, 1–549. https://doi.org/10.1142/6968

Reeves C. (1993). *Modern heuristic techniques for Combinatorial Problems* (1st ed.). John Wiley & Sons. http://www.gbv.de/dms/ilmenau/toc/214312879.PDF

Sakuraba, C. S., & Yagiura, M. (2010). Efficient local search algorithms for the linear ordering problem. *International Transactions in Operational Research*, *17*(6), 711–737. https://doi.org/10.1111/j.1475-3995.2010.00778.x

Schiavinotto, T., & Stützle, T. (2003). Search space analysis of the linear ordering problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2611*, 322–333. https://doi.org/10.1007/3-540-36605-9_30

Simmons, D. M. (1969). One-Dimensional Space Allocation: An Ordering Algorithm. *Operations Research*, *17*(5), 812–826. https://doi.org/10.1287/opre.17.5.812

Tromble, R., & Eisner, J. (2009). Learning Linear Ordering Problems for better translation. *EMNLP 2009 - Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: A Meeting of SIGDAT, a Special Interest Group of ACL, Held in Conjunction with ACL-IJCNLP 2009*, *August*, 1007–1016. https://doi.org/10.3115/1699571.1699644

Valdez, G. C., & Medina, S. S. B. (2012). Iterated Local Search for the Linear Ordering Problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *3*(1), 12–20.

Welsh, D. J. A. (1989). Simulated Annealing: Theory and Applications. In *Bulletin of the London*

Mathematical Society (Vol. 21, Issue 2). https://doi.org/10.1112/blms/21.2.204b

Ye, T., Wang, T., & Zhipeng, L. (2014). A Multi-parent Memetic Algorithm for the Linear Ordering Problem. *ArXiv*, 26.

Zomaya, A. Y., Smith, A., & Seredynski, F. (2003). The use of the simulated annealing algorithm for channel allocation in mobile computing. *Wireless Communications and Mobile Computing*, *3*(2), 239–253. https://doi.org/10.1002/wcm.114