I.J. Intelligent Systems and Applications, **2024, 4, 86-103**
Published Online on August 8, 2024 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2024.04.06

An Exploratory Study on Simulated Annealing for Feature Selection in Learning-to-rank

Mohd. Sayemul Haque
Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh
E-mail: sh.sayem.haque36@gmail.com
ORCID iD: https://orcid.org/0009-0002-6453-7440

Md. Fahim
Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh
E-mail: fahimcse381@gmail.com
ORCID iD: https://orcid.org/0009-0001-6139-5283

Muhammad Ibrahim*
Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh
E-mail: ibrahim313@du.ac.bd
ORCID iD: https://orcid.org/0000-0003-3284-8535
*Corresponding author

Received: 13 February 2024; Revised: 15 April 2024; Accepted: 28 June 2024; Published: 08 August 2024

Abstract: Learning-to-rank is an applied domain of supervised machine learning. As feature selection has been found to be effective for improving the accuracy of learning models in general, it is intriguing to investigate this process for learning-to-rank domain. In this study, we investigate the use of a popular meta-heuristic approach called simulated annealing for this task. Under the general framework of simulated annealing, we explore various neighborhood selection strategies and temperature cooling schemes. We further introduce a new hyper-parameter called the progress parameter that can effectively be used to traverse the search space. Our algorithms are evaluated on five publicly benchmark datasets of learning-to-rank. For a better validation, we also compare the simulated annealing-based feature selection algorithm with another effective meta-heuristic algorithm, namely local beam search. Extensive experimental results show the efficacy of our proposed models.

Index Terms: Learning-to-rank, Feature Selection, Meta-heuristics, Simulated Annealing, Local Beam Search.

1. Introduction

Information Retrieval (IR) [1] deals with the methods, process and different procedures of searching, locating, and retrieving any kind of meaningful and structured information. This information may come from the database, documents or any kind of source like web. Indeed, it is also considered as a science of searching information where the search queries can be based on searching for meta data (data about data), searching for ranking the information, document lists or the information from the databases or documents. In an IR-based system, a query is given to the system and the job of the system is to retrieve the information from the documents or database which are most relevant to the query.

Generally, an IR model is probabilistic or mathematical. IR usually deals with bigger datasets, such as a lot of documents like web pages. In this modern age of technology, the databases are becoming bigger and bigger and number of documents of any collection is increasing rapidly. Furthermore, variations of data are becoming available. The probabilistic or mathematical IR models are not sufficient to handle these large and complex datasets as these models are slow and less accurate. So, some automation techniques are needed in the IR system for doing the job quickly and accurately.

Machine Learning [2] is the process of automated learning from data. These algorithms learn from the historical data and improve themselves without requiring any manipulation by the user. As nowadays there is huge amount of data available, so it is easier to build a robust machine learning model. Learning-to-Rank (LtR) [3] is an application of the machine learning and information retrieval. It uses supervised machine learning to solve the ranking problem. The aim of LTR is to come up with optimal ordering of items given a query. The training data for an LTR model consist of a list of query-document pairs. The features of the training data are the output scores of various IR scoring functions of the documents (such as tf, idf, tf-idf, bm25 score etc.), and the labels are relevance scores which indicates how much relevant a document is with respect to a given query. These data are fed into a machine learning model. After the learning process is done based on the training data, the model is used to generate relevance scores for unseen query-document pairs.

Nowadays the size of data is increasing rapidly. Large amount of data is not only increasing the model complexity but also affecting the performance. In information retrieval, especially in web search, usually the data size is very large and thus training of ranking models is computationally costly. Besides, redundant and irrelevant features may lead to poor effectiveness of the models. So, selecting a good subset of features is quite important for LtR field.

Usually, practitioners garner all possible features of a domain for learning a ranking algorithm. This number often reaches hundreds. However, not all of these are conducive to learning a good ranking function. Handpicking a good subset of features is not feasible. In such scenarios, feature selection algorithms are quite useful in machine learning in general, and in information retrieval in particular.

1.1. Motivation

For building a robust LtR model, we need to handle the redundant features and prevent the overfitting problem. A proper subset of features is needed not only for building a better LtR model but also for faster training. There are several methods for feature selection like filter method, wrapper method, embedded method and meta-heuristic approach.

Meta-heuristic algorithms are used to solve different large and complex optimization problems such as NP-hard problems. These algorithms provide better solutions to incomplete optimization problems and are easily adaptable to different circumstances. They also perform well in solving the feature selection problem of machine learning. Among the meta- heuristic approaches, simulated annealing is a choice of many researchers for its ability to find good solutions quickly.

1.2. Research Question

The main objective of this investigation is to address the feature selection problem in learning-to-rank using an effective meta-heuristic algorithm, namely simulated annealing algorithm. In particular, the following research questions are addressed in this study:

- How do various neighbor generation strategies perform?
- Which cooling schemes are effective?
- How can the temperature be used to traverse the search space?

This study is expected to enrich the existing literature on the use of meta-heuristic algorithms for feature selection in rank-learning domain. It is expected to improve the performance of existing approaches that use simulated annealing-based feature selection.

1.3. Contributions

The following contributions are made in this research:

- We adapt the standard simulated annealing algorithm to select a good subset of features in learning-to-rank domain.
- We utilize two effective neighbourhood definitions for simulated annealing.
- We explore the effects of different cooling schemes in simulated annealing algorithm.
- We introduce a novel technique called the progress parameter in the basic simulated annealing algorithm to better traverse th search space.
- We compare the performance of all these settings of simulated annealing algorithm on five benchmark LtR datasets. To better validate our investigated techniques, we also compare these results with another meta-heuristic algorithm called local beam search algorithm.

The rest of the paper is organized as follows. In Section 2, we describe the related research works and identify the research gap. In Section 3 we propose a framework for feature selection problem in learning-to-rank domain using simulated annealing algorithm. Here we discuss how we define the feature selection problem, the state definition, neighbours structure, cooling schemes, and progress parameter. In Section 4 we discuss our experimental settings and analyze the results. Finally, in Section 5 we conclude the paper mentioning some directions for future work.

2. Background and Related Works

In this section, we briefly discuss the relevant existing research, thereby identifying the research gap in the existing literature.

2.1. Feature Selection in Supervised Machine Learning

The quality of the features in a dataset has a major impact on the performance of machine learning models. While building a machine learning model for a real life scenario, it is rare that all the features in the dataset are useful to make prediction. Including redundant features reduces the generalization capability of the model. It may also lead to overfitting problem if the number of instances is comparatively low. Thus, sometimes the redundant features may decrease the performance of the model. Feature selection techniques are used to choose a subset of the features from the original feature set by removing irrelevant, redundant and noisy features. Generally, careful use of feature selection methods leads to better performance of machine learning models [4]. These methods are briefly described below.

A. Traditional Methods

The most widely used non-heuristic feature selection approaches are 1. *Filter Method*, 2. *Wrapper Method*, 3. *Embedded Method*.

Filter methods pick up the intrinsic properties of the features measured using univariate statistics instead of cross-validation performance of learning models. As these methods do not use any learning model to evaluate performance and instead solely depends on statistical methods, they are computationally less expensive than other methods like wrapper and embedded methods. These methods are preferable when the feature space is too large [5]. Several techniques like SelectKBest using Chi Square test, Fisher Score's, Correlation Coefficient, Variance threshold, ReliefF [6] etc. are examples of such methods.

Wrapper methods employ a machine learning model to evaluate the potential subsets of features [7]. These potential subsets are generated using a search method. After selecting a subset, it is used to train the model that generates predictions for validation dataset. Based on this prediction, the subset is evaluated using performance metrics. The search methods help exploring the feature space efficiently. This is an iterative and computationally expensive process but it is usually more accurate than the filter methods [8]. There are few techniques used in wrapper method like forward feature selection, backward feature elimination, exhaustive feature selection, recursive feature elimination etc.

Embedded methods make a trade-off between computational cost and accuracy by combining the wrapper and filter methods, which often results in better accuracy while keeping the computational cost reasonable [9]. Embedded methods employ an iterative process in the sense that they take care of each iteration of the model's training process and carefully extracts those features which contribute the most to the training error for a particular iteration. Regularization methods are the most commonly used embedded methods which penalize a feature given a coefficient threshold. These methods are far less computationally intensive than wrapper methods [10].

B. Meta-Heuristic Approaches

Meta-heuristics are problem-independent techniques [11], i.e., they do not include problem specific features. They are very efficient to avoid the local optima problem in optimization problems since, unlike greedy or purely heuristic approaches, they may accept a worse solution while maneuvering in the search space. Meta-heuristic optimization algorithms are widely used to solve large-scale optimization problems [12]. The advantages of these algorithms over traditional optimization algorithms are their adaptability and the ability of dealing with complex problems [13].

Like many domains, meta-heuristic algorithms work well in the feature selection problem of machine learning. Although these approaches are slow, they are often able to select near-optimal feature subsets [14]. So many researchers have used different meta-heuristic approaches for solving the feature selection problem. Jihoon Yang and Vasant Honavar et al. [15] use one of the popular meta-heuristic algorithms called genetic algorithm in this regard. After finding best "parents" from the "population", new generations (a.k.a "child") are created by crossing over the parents. Then a mutation process is conducted to bring some new property in the child. Lucija Brezoˇcnik et al. [16] use Swarm Intelligence [17] techniques for solving feature selection problem. Here after initializing population, the agents are modified or updated based on a fitness function until it reaches a termination criterion. Another popular meta-heuristic algorithm is simulated annealing. Siedlecki et al. [18] use simulated annealing algorithm for automatic feature selection. Simulated annealing algorithm tries to find optimal solution by exploring almost the whole search space. This algorithm is used in some other feature selection technique like Hybrid Whale Optimization [19]. Alvi et al. [20] use simulated annealing algorithm for feature selection, but in a very naive setting and with a very few datasets. On another line of research, Ibrahim et al. [21] use simulated annealing in LtR domain, but not for feature selection, rather for designing a novel LtR algorithm.

2.2. Feature Selection in Learning-to-rank

The number of features determines the time required to extract features from documents, both in the development phase and testing phase. Also, inclusion of noisy features reduces the accuracy of model. Some researches have been performed on feature selection for LtR task. In [22], the authors propose a filter-based method, namely Greedy Search Algorithm (GAS) where the problem is treated as an optimization problem. The authors try to find features with maximum total importance scores and minimum total similarity scores. In [23], the authors propose three filter-based feature selection methods, namely NGAS, XGAS and HCAS. In NGAS, the authors first add the most relevant feature to the set. After that goes on a loop adding features which minimize similarity and maximizes relevance. XGAS is an upgraded version of NGAS that compares the feature to be added to a subset of features. HCAS performs a hierarchical agglomerative clustering. But this type of feature selection does not ensure the optimal feature selection as argued in [24]. In [25], the authors propose a wrapper method which is a single multi objective criteria for feature selection. In [26], authors propose an evolutionary algorithm-based strategy that reduces the search space by eliminating weak features during the traversing process. Although this type of strategy is adaptive and ensures near-optimal solutions, wrapper methods are time consuming. FSMRank [27] is an embedded strategy which simultaneously reduces the ranking error along with feature selection. In [28], a deep neural network-based architecture is used to automatically select features that uses $l1$ regularization.

2.3. Research Gap

As we have discussed above, a lot of research has been done to investigate the impact of feature selection on learning-to-rank domain which mostly consists of filter methods. However, apart from the genetic algorithm, there has not been any significant work on the impact of different meta-heuristic algorithms. In the current research, we intend to investigate the efficacy of a popular meta-heuristic algorithm, namely simulated annealing, to select a useful and informative set of features in the LtR domain.

2.4. Simulated Annealing Technique

Since our work is based on simulated annealing, in this sub-section we briefly describe this procedure.

Simulated annealing [29] is one of the most popular and widely used optimization algorithms. This algorithm is used to solve different real life and interesting problems like travelling salesman problem, scheduling problems, task allocations, graph coloring and partitioning, non-linear function optimizations and so on [30]. It uses a probabilistic technique for approximating the global optimum of a given function. Simulated annealing gets its motivation from the process of slow cooling of metals. It performs better than bare greedy algorithms because of its ability to overcome the local optimum problem. This process is very useful for situations where there are a lot of local minima [31].

Algorithm 1: Generic Simulated Annealing Algorithm [33]

```
Result: Returns a solution state
Input: problem, a problem;
Input: schedule, a mapping from time to temperature;
current ← MAKE-NODE (problem.InitialState);
for t = 1 to ∞ do
    T ← schedule(t);
    if T = 0 then
        return current;
    end
    next ← a randomly selected successor, i.e., neighbor of current;
    $\Delta E$ ← next.Value − current.Value;
    if $\Delta E > 0$ then
        current ← next
    else
        prob = $e^{\Delta E/T}$;
        if prob > rand(0, 1) then
            current ← next
        end
    end
end
```

Simulated annealing algorithm takes the idea of the physical annealing process where a hot metal is shaped by gradually reducing the temperature. The algorithm uses this temperature cooling scheme for optimizing parameters in a model [32]. Simulated annealing algorithm performs well because it does not only explore the better states but also traverses some worse states (aka bad moves). If we only look for the better states (like traditional hill climbing method), we may get stuck in a local optimum. In contrast, if we allow to traverse some apparently bad moves, we are able to explore a larger search space which helps in finding the global optima. However, if we allow an unrestrained number of bad moves, the solution may get worsened. Thus, simulated annealing algorithm trades off between these two extremes. Before a bad move is about to be taken place, an acceptance probability is calculated. If the acceptance probability is less than some random threshold value, the next state (worse state) is not explored. Otherwise, the bad move is explored. The acceptance probability is usually calculated as $e^{-\Delta E/T}$ where T is the temperature and ΔE the difference of solution qualities between the two states in question. We see that when the higher the temperature, the higher the acceptance probability, so the algorithm is more likely to accept a bad state. This means that the algorithm is more likely to explore bad moves when the temperature is higher. Thus, the temperature has a significant effect on performance of simulated annealing algorithm. The temperature is controlled by annealing or cooling schedule which helps to reduce the value of temperature gradually from a higher value to the lower one.

The pseudo code ([33]) of generic simulated annealing algorithm is given in Algorithm 1. Note that this generic algorithm must be adapted differently to different problem domain.

3. Proposed Framework

In this section we elaborately discuss our proposed framework of simulated annealing-based feature selection for learning-to-rank which includes discussions on the definition of solution state, evaluation of a solution, neighborhood definition, traversing strategy, and annealing or cooling schemes. Below in Algorithm 2 we provide a generic framework as to how the simulated annealing algorithm can be used to solve feature selection problem in learning-to-rank [34]. This generic version will later be modified followed by our discussions on specific details of the algorithm.

Algorithm 2: Simulated Annealing-Based Framework for Feature Selection in Learning-to-Rank

```
Result: Returns best subset of features of size $k$. ($k$ is provided as a hyper-parameter.)
create an initial random subset of features of size $k$, current;
for $t \leftarrow 1$ to $\infty$ do
    T $\leftarrow$ schedule(t);
    if $T = 0$ then
        return current;
    end
    a new feature subset, next $\leftarrow$ get_neighbour(current);
    fit model with next features and estimate performance of learnt model;
    $\Delta E \leftarrow$ performance(next) - performance(current);
    if $\Delta E > 0$ then
        accept next as the new current subset, i.e., current $\leftarrow$ next;
    else
        probability $\leftarrow e^{\Delta E/T}$;
        if probability > rand(0,1) then
            accept next as the new current subset, i.e., current $\leftarrow$ next;
        else
            reject the new subset, i.e., no change in current;
        end
    end
end
```

3.1. *Adapting Generic Simulated Annealing Algorithm to Feature Selection in Learning-to-rank*

In order to use simulated annealing for feature selection in learning-to-rank scenario, we need to decide on several aspects of the algorithm which are as follows:

- How to define a state?
- How to evaluate the quality of a state?
- How to define the neighborhood relationship among the states?
- How to traverse the search space?
- How to decide on the annealing or cooling scheme?

Below we elaborately discuss these aspects which leads to our proposed framework.

A. Definition of a State

In the feature selection problem of learning-to-rank, we intend to search for the subset of features that gives us the best performance in terms of IR metrics like NDCG or MAP. So a state in our setting represents a subset of features among the available ones. To implement this idea, we take a bit array whose length is equal to number of total available features. Each bit represents a feature, and 1 in *i*-th index means that we include *i*-th feature for measuring IR performance, and 0 means otherwise. We search for the best subset among the states with k number of features, where k ranges from 1 to $n-1$, where n is the total number of features. For each subset of features for each k, we invoke simulated annealing and assign annealing steps in proportion to the total number of states and the number of neighbors of a state. The reason behind this approach is to get a better view of the impact of increasing number of features.

B. Quality of a State

To measure the quality of a state in simulated annealing, we need to consider the task at hand, which is learning-to-rank. In this domain, the quality of the list of documents/elements ranked by an algorithm is measured against some IR metrics like NDCG, MAP etc. Therefore, for each state, i.e., a subset of features among the available ones, we train the model learnt with only the features that are included in the feature subset at hand. We then measure the performance of the learnt model using NDCG and MAP.

C. Definition of Neighbourhood

One of the most important things for building a robust simulated annealing algorithm is to define the neighbouring states. We borrow two effective neighbourhood definitions from existing literature. These two neighbour selection techniques have been used in the greedy hybrid operator proposed by Zhen et al. [35] and Wang et al. [36] to produce candidate solutions from the current state. We use a slightly modified version of these techniques. The two neighbours selection techniques are: 0's and 1's Swapping and Insertion.

- ***0's and 1's Swapping***: Here a state's neighbors are defined as those having one feature different from the current state. Specifically, considering two sets, where one holds the indexes of the bit-array (current state) with 1's and the later with 0's. From each set we randomly choose an index and flip the bits (swap) of those indexes to generate neighbors. Suppose, our current state is ϕ. Two indexes i and j are chosen randomly from the sets of 1's and 0's respectively. After that, by performing $flip(\phi[i])$ and $flip(\phi[j])$ we generate a neighbour state.
- ***Insertion***: Here we choose two random positions i and j from the current state. We then move the element in position j to position i and shift all values from i by 1 position to the right. In case of $i < j$, if our current state is ϕ then we generate a neighbour state ϕ' where $\phi'[i + 1] = \phi[i]$ for all i up to j, and $\phi'[i] = \phi[j]$. The other case, i.e., $i > j$, works similarly.

Fig. 1 pictorially demonstrates the two neighboring schemes.

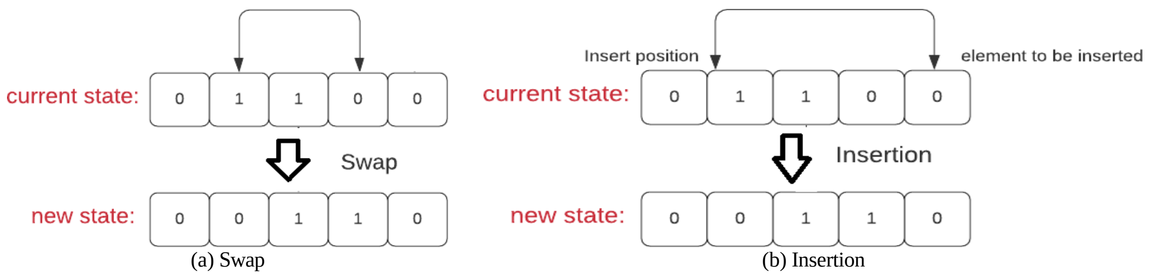


Fig.1. Two neighborhood definitions for simulated annealing scheme

D. Traversing the Neighborhood: Progress Parameter

The problem with selecting a neighbor randomly from the set of neighbors is that we may miss potentially good states. So, Connolly [37] proposes to evaluate neighbors in a sequential order. However, in our problem the number of neighbors is large. So instead of visiting every neighbor sequentially, we introduce a novel parameter which we call the "progress" parameter. This parameter keeps track of the number of iterations of the algorithm that has not seen update on the current best solution. When the progress parameter reaches a threshold value, we restart from the current best solution. This way it is likely to help in avoiding the local optima. It also ensures exploring the current best solution's neighborhood space thoroughly in lower temperatures so that it, at least, ensures the local optima of that neighborhood space if not the global optima.

E. Annealing Scheme

An important aspect of simulated annealing algorithm is the temperature [30]. We not only traverse the better states but also some worse states to find a good global solution. Traditionally, the probability score based on which a bad move is accepted or not is measured by function called *metropolice*, which is calculated as $e^{\Delta E/T}$, where ΔE = $next.Value - current.Value$ [33]. We see that traversing a worst state directly depends on the temperature – if the temperature is high, so is the probability score, and hence there is a better chance for it to be accepted, and vice versa.

The algorithm starts with a bigger value of temperature ($T_{initial}$) and then slightly cools down as per a cooling schedule. As a result, in the early iterations of the algorithm, we allow more bad moves, and this allowance is gradually restricted as the algorithm matures. Thus, the performance of simulated directly depends on the cooling schedule. There are several cooling schemes (aka annealing schemes) in the existing literature. Among them, we employ three schemes which are popular and widely used. These schemes are described below:

- *Geometric Annealing Scheme*: A popular yet simplest cooling scheme is geometric annealing scheme [38]. This scheme uses the following rule to update the temperature: $T_{t+1} = \alpha * T_t$, where t is the time step, T_t and T_{t+1} are the current and updated temperatures, respectively, and α is a controlling parameter. Generally, α is in (0,1]. For our experiments, we set $\alpha = 0.9$.
- *Logarithmic Annealing Scheme*: Another effective cooling scheme is logarithmic annealing scheme. Though this scheme is slow, it is reported to perform better than some other schemes in the smaller problems [38]. Here

the temperature update rule is: $T_t = logT_0(t+t_0)$, where t is the time step, T_0 is the initial temperature, T_t is the updated temperature at t-th time step and t_0 is a fixed value to prevent division by zero. For our experiments, we use $t_0 = 10$.

- *Fast Annealing Scheme*: Harold Szu et al. [39] introduce a Fast Simulated Algorithm (FSA) which is claimed to be much faster than the classical simulated annealing. In this paper a new temperature update rule is introduced, which is: $T_t = T_0(t+t_0)$, where t is the time step, T_0 is the initial temperature, T_t is the updated temperature at t-th time step.

F. Temperature Length

The temperature length parameter is the number of steps after which the temperature is updated using the cooling scheme. States are evaluated at certain temperatures until it is reduced to a terminating value by the cooling schedule. There are some options such as fixed temperature length, adaptive method and variable length method. We choose the adaptive method proposed by Abramson [40] where the temperature is updated depending on the search progress. When the number of accepted moves crosses a certain value, the temperature is updated, otherwise, the same temperature is used in the next iteration.

3.2. Implementation of Simulated Annealing Algorithm for Feature Selection in Learning-to-rank

Based on all the ideas and concepts discussed above, we devise the algorithm for solving the feature selection problem in learning-to-rank which is given in Algorithm 3. Here our contributions over the traditional simulated annealing are in four places: neighborhood generation, cooling scheme, temperature length, and restart traversing based on local history of state updates (using progress parameter).

3.3. Local Beam Search

Although in this research our main objective is to investigate efficacy of various settings of simulated annealing algorithm for selecting a useful feature subset in LtR task, for the sake of better validation of our proposed framework, we adapt another meta-heuristic algorithm called local beam search to the same problem, and compare the experimental results. The authors in [41, 42] show the use of LBS algorithm for feature selection in machine learning.

Local Beam Search (LBS) is a popular meta-heuristic algorithm [43] which is well-known for its efficient memory usage [44]. Whereas the heuristic algorithms (like hill climbing) consider only one successor of the states at a time, LBS keeps track on more than one successor at a time. At every level of search tree, it considers β number of successors where β is called the beam-width. For large search spaces where there is insufficient amount of memory to store the fully grown search tree, the LBS algorithm thus offers help by keeping track a small number of successors. The algorithm uses breadth-first search while traversing a search tree. While building the tree, the algorithm generates all the successors of the states at each level. It then sorts the successors in an increasing order of their heuristic values. It then stores only the top β successors. The memory requirement is bounded by the value of β to perform the search. At the end of the search, the algorithm returns the best successor as a solution from the last β successors. Algorithm 4 shows the pseudocode of the algorithm we have devised for solving feature selection problem in learning-to-rank using local beam search algorithm.

4. Experiments and Result Analysis

In this section we discuss the datasets and experimental setup, and analyze the results.

4.1. Datasets

For our experiments we use five benchmark LETOR datasets as described below:

- MQ2008: MQ2008 [45] is a part of Microsoft LETOR 4.0 package, which was released in July, 2009. There are 784 queries. Each query-document pair is labeled using a relevance score (0, 1 or 2). Each row consists of 46 features, along with query ID, a comment about the document-query pair, document ID and the relevance score.
- MQ2007: MQ2007 [45] is also a part of the LETOR 4.0 package. The dataset's structure is similar to the MQ2008 (46 features) with 1692 queries.
- OHSUMED: Oregon Health Sciences University's MEDLINE data collection is also known as OHSUMED [46]. It is a subset of MEDLINE dataset. It consists of 45 features and a total of 106 queries, and has multiple labels (0, 1 or 2).
- TD2004: Topic distillation 2004 (TD2004) [47] is one of the seven datasets of the LETOR 3.0 collection. It has 44 features and a total of 107 queries.
- MSLR-10K: Microsoft released a large dataset named MSLR-WEB10K [48]. The dataset has 136 features and 10000 queries. It assigns relevance score to each query-document pair from a scale of one to five (0, 1, 2, 3 or 4).

Algorithm 3: Proposed Simulated Annealing Algorithm for Feature Selection in Learning-to-Rank

```
Result: Returns best subset of features of size $k$. ($k$ is provided as a hyper-parameter.)
current-state ← select-initial-state()//a subset of features of size $k$;
$T \leftarrow T_{initial}$;
best-state ← current-state;
for $t \leftarrow 1$ up to no. of iterations do
    if $T = 0$ then
        return current-state;
    end
    next-state ← get_neighbour(current-state) ;
    Fit LtR model on training data using features expressed by next-state;
    //Evaluating performance on test data;
    $\Delta E \leftarrow$ Evaluation(next-state) − Evaluation(current-state);
    if $\Delta E > 0$ then
        current-state ← next-state;
        if Evaluation(next-state) > Evaluation(best-state) then
            Update best-state using next-state
        end
    else
        probability = metropolice($\Delta E$);
        if probability > rand(0,1) then
            current-state ← next-state;
        else
            Reject next-state, i.e., no change in current-state;
        end
    end
    if Temperature update condition is met then
        $T \leftarrow$ cooling-scheme($t$) ;
    end
    if Progressing condition is not met then
        Restart from the best-state
    end
end
```

Algorithm 4: Proposed Local Beam Search-Based Algorithm for Feature Selection in Learning-to-Rank

```
Result: Returns best subset of features of size $k$. ($k$ is provided as a hyper-parameter.)
$q \leftarrow$ beam-width;
current-top-q-best-states ← select-$q$-initial-states()//subsets of size $k$;
for $t = 1$ to $q$ do
    state ← current-top-q-best-states[t] ;
    Fit LtR model on training data with the feature set represented by state;
    //Evaluate performance on test data;
    performance-scores[t] ← Evaluation(state);
end
best-state ← $-\infty$ ;
for $t = 1$ to no. of steps do
    current-top-q-best-states ← sort-descending-order(performance-scores);
    for $t = 1$ to $q$ do
        current-state ← current-top-q-best-states[t];
        new-state ← Neighbours(current-state);
        Fit LtR model on training data with the feature set represented by new-state;
        //Evaluate performance on test data;
        if Evaluation(new-state ) > Evaluation(current-state) then
            Update(current-top-q-best-states, performance-scores)
        end
    end
end
current-top-q-best-states ← sort-descending-order(performance-scores);
best-state ← current-top-q-best-states[1];
```

4.2. *Rank-learning Model*

In order to evaluate the ranking performance of the model trained on the selected feature subsets, we employ a widely-used and state-of-the-art rank-learning algorithm called LambdaMART [49] which is a tree-ensemble machine learning model. This algorithm is based on gradient-boosting framework that optimizes a surrogate ranking loss function. We set the hyper-parameters of this algorithm as follows: learning rate = 0.1, number of trees = 50, subsample size per tree = 0.5, and maximum number of allowable leaves per tree = 5.

4.3. *Evaluation Metrics*

We evaluate the ranking efficacy of a model using two metrics: Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP). NDCG is the ratio of Discounted Cumulative Gain (DCG) score of the ranking of items predicted by the model to that of ideal ranking. MAP is the mean of all the average precision (AP) scores for each query.

4.4. *Algorithm Setup*

For simulated annealing, in Table 1, the two neighboring strategies are noted by **n1** and **n2** and the three cooling schemes are noted by **s1**, **s2** and **s3**. So, in total, there are six settings. For local beam search algorithm, we set *beam-width* to be 10.

Table 1. Various settings of simulated annealing

Abbreviation	**Description**
n1s1	neighborhood selection strategy: swap (n1), cooling scheme: geometric (s1) with a cool-down factor of 0.9
n1s2	neighborhood selection strategy: swap (n1), cooling scheme: logarithmic (s2)
n1s3	neighborhood selection strategy: swap (n1), cooling scheme: fast annealing (s3)
n2s1	neighborhood selection strategy: insertion (n2), cooling scheme: geometric (s1) with a cool-down factor of 0.9
n2s2	neighborhood selection strategy: insertion (n2), cooling scheme: logarithmic (s2)
n2s3	neighborhood selection strategy: insertion (n2), cooling scheme: fast annealing (s3)

4.5. *Experimental Results*

In what follows, we demonstrate the following experimental results for each dataset:

- We first compare different cooling strategies for each neighboring technique for simulated annealing.
- We then compare the ranking performance with progress parameter with ranking performance without progress parameter for simulated annealing.
- We then show the ranking performance of local beam search and compare it with that of simulated annealing.

In each case, we report NDCG@10 and MAP scores. As our algorithms are non-deterministic, we take the average of these scores over ten iterations. Since extensive experiments on the (large) MSLR-WEB10K is computationally highly expensive, we discuss its experiments separately at a later stage of our analysis.

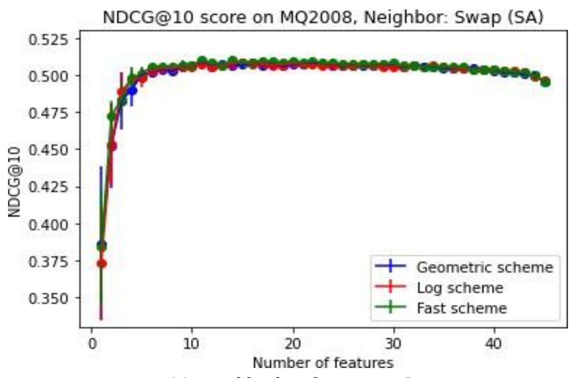
A. *Combinations of Neighborhood Strategies and Cooling Schemes*

MQ2008 Data

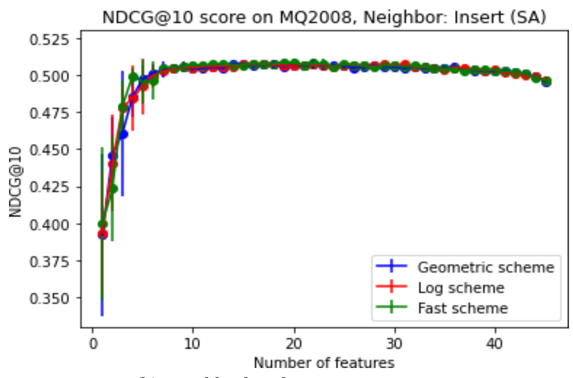
Fig. 2 plots the feature selection performance of simulated annealing evaluated using NDCG and MAP metrics for all the six settings. In these graphs, *x*-axis represents the number of features and the *y*-axis representsthe corresponding NDCG@10 scores. For swapping, fast annealing reaches the highest average NDCG score of 0.50970 among the cooling strategies. For insertion also, fast annealing reaches a peak of 0.5082. For both graphs the standard error is small enough for the results to be accepted after number of selected features is more than 8. The maximum MAP score 0.3599 is achieved by fast annealing for swapping. For insertion, a maximum MAP score of 0.3605 is achieved by fast annealing.

MQ2007 Data

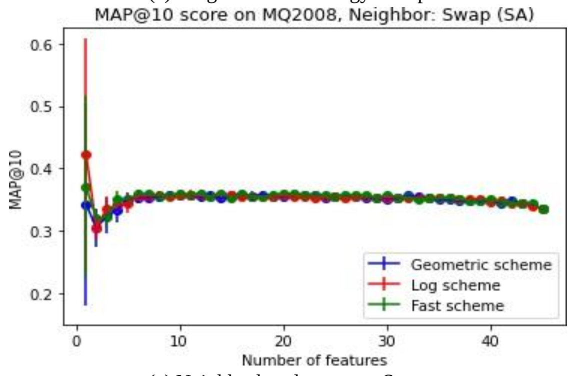
Fig. 3 plots the feature selection performance of simulated annealing evaluated using NDCG score on MQ2007 for all the six settings. For swapping, fast annealing reaches the highest average NDCG score of 0.4829 among the cooling strategies. For insertion, fast annealing also reaches a peak of 0.4828. Both neighbor selection strategies give almost similar result in this case. Swapping works slightly better compared to insertion. For both graphs the standard error is small enough for the result to be accepted after number of selected features is more than 6. The maximum MAP score of 0.2478 is achieved by fast annealing for swapping. For insertion, a maximum map score of 0.2486 is also achieved by fast annealing.



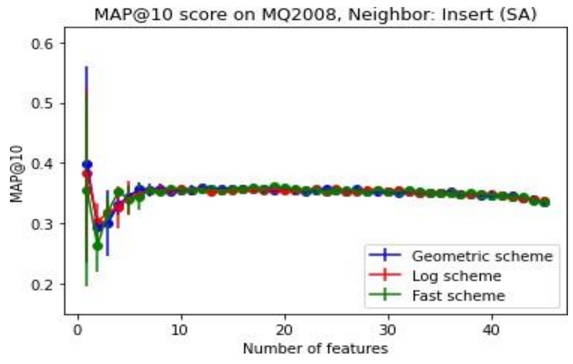
(a) Neighborhood strategy: Swap



(b) Neighborhood strategy: Insertion

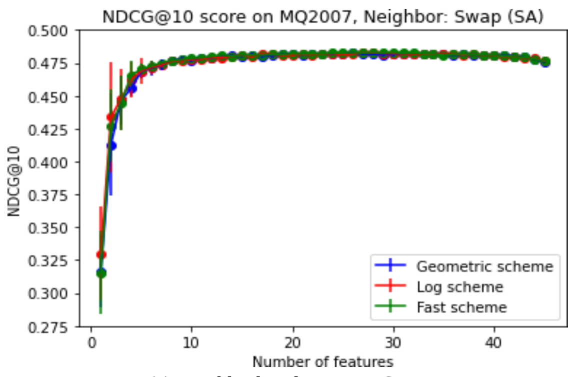


(c) Neighborhood strategy: Swap

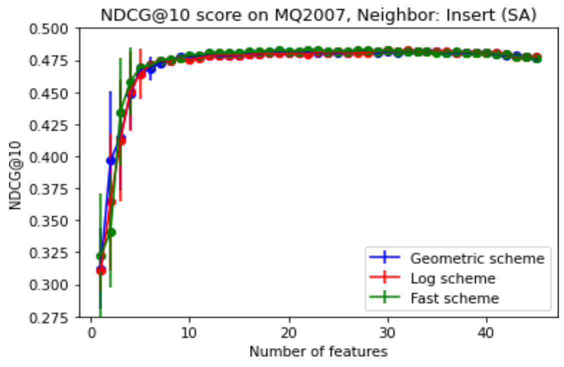


(d) Neighborhood strategy: Insertion

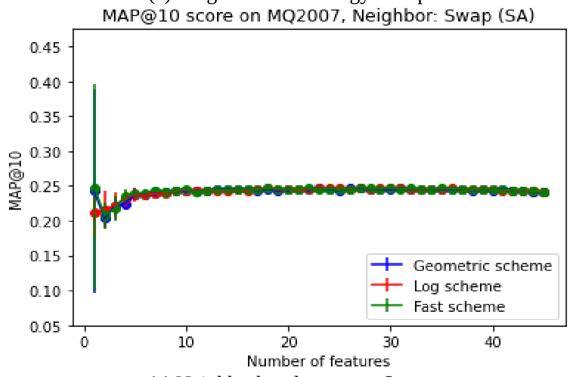
Fig.2. NDCG@10 and MAP scores on MQ2008 dataset for simulated annealing



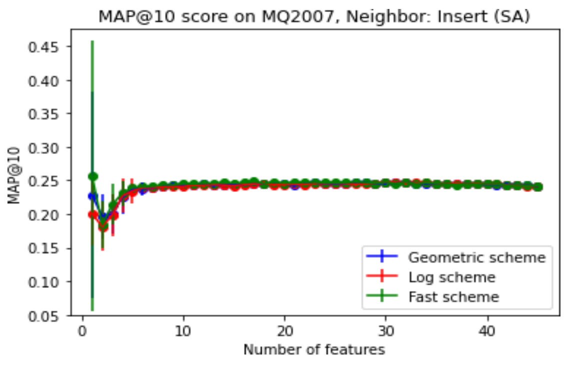
(a) Neighborhood strategy: Swap



(b) Neighborhood strategy: Insertion



(c) Neighborhood strategy: Swap

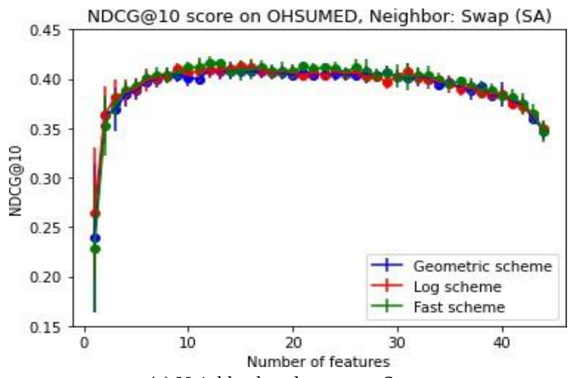


(d) Neighborhood strategy: Insertion

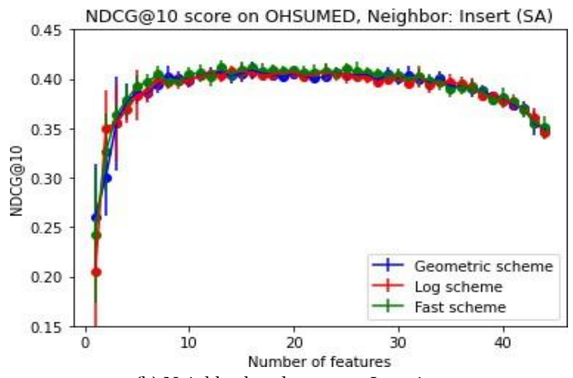
Fig.3. NDCG@10 and MAP scores on MQ2007 dataset for simulated annealing

OHSUMED Data

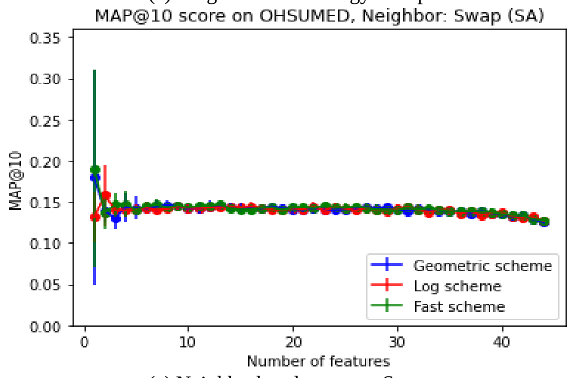
Fig. 4 plots the feature selection performance of simulated annealing evaluated using NDCG score on OHSUMED dataset for all the six settings. For swapping, fast annealing reaches the highest average NDCG score of 0.4165 among the cooling strategies. For insertion, fast annealing also reaches a peak of 0.4121. Swapping is found to be more effective compared to insertion. For both graphs the standard error is small enough for the result to be accepted after number of selected features is more than 8. The maximum MAP score of 0.1455 is achieved by fast annealing for swapping. For insertion, a maximum map score of 0.1448 is achieved by logarithmic scheme.



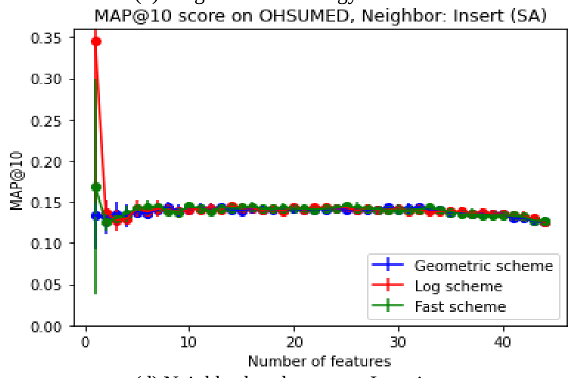
(a) Neighborhood strategy: Swap



(b) Neighborhood strategy: Insertion

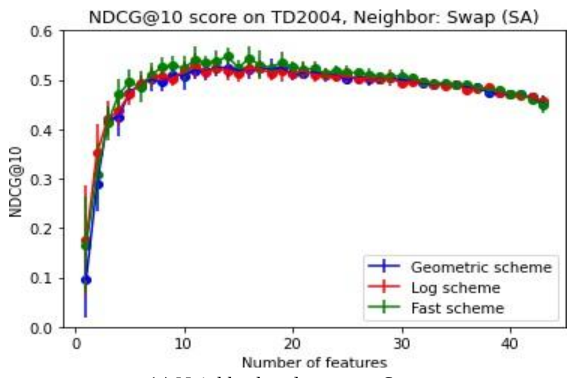


(c) Neighborhood strategy: Swap

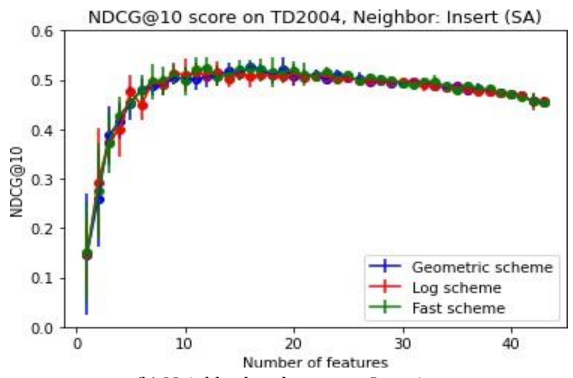


(d) Neighborhood strategy: Insertion

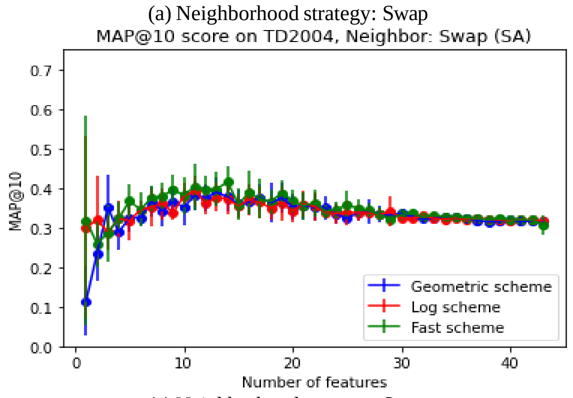
Fig.4. NDCG@10 and MAP scores on OHSUMED dataset for simulated annealing



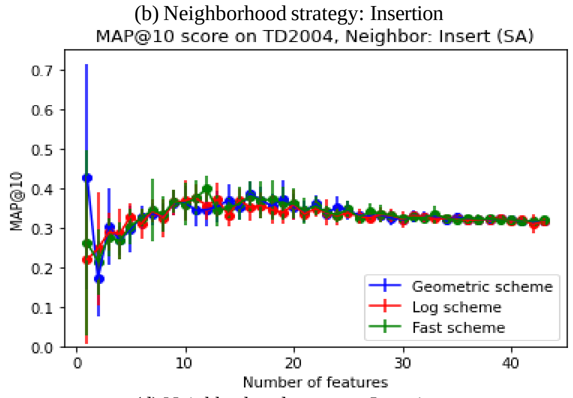
(a) Neighborhood strategy: Swap



(b) Neighborhood strategy: Insertion



(c) Neighborhood strategy: Swap



(d) Neighborhood strategy: Insertion

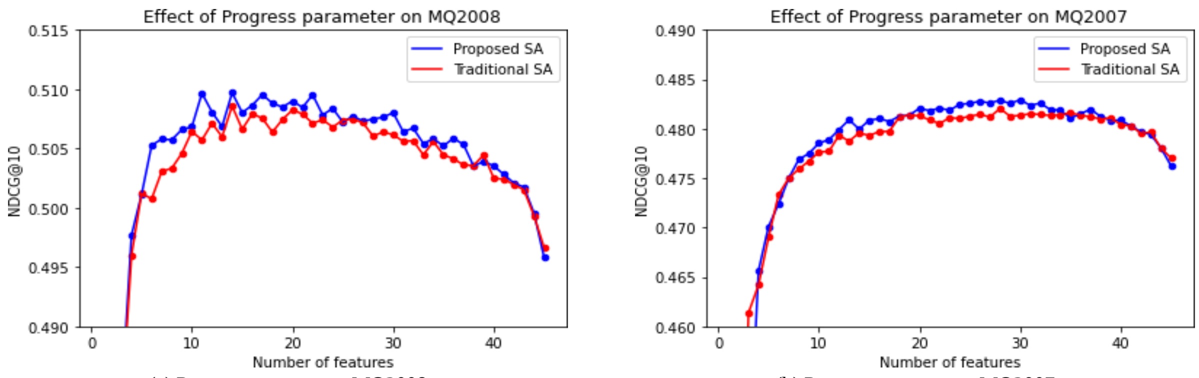
Fig.5. NDCG@10 and MAP scores on TD2004 dataset for simulated annealing

TD2004 Data

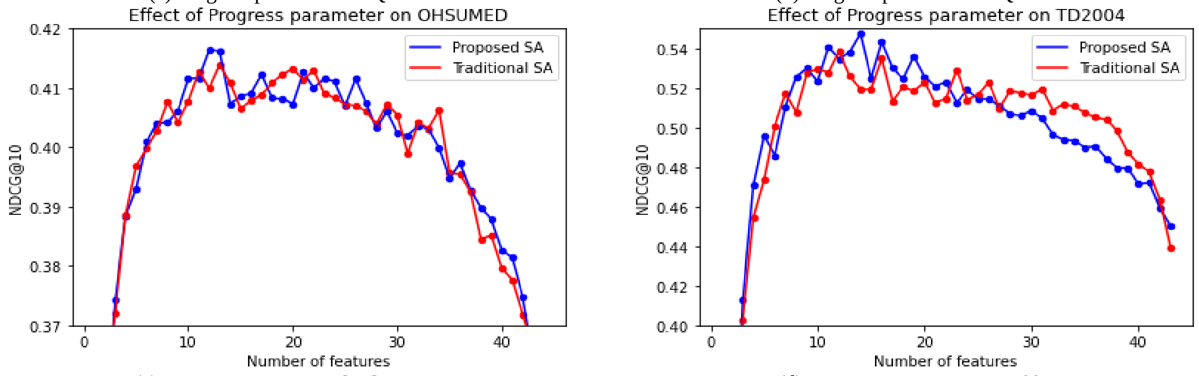
Fig. 5 plots the feature selection performance of simulated annealing evaluated using NDCG score on TD2004 dataset for all the six settings. For swapping, fast annealing reaches the highest average NDCG score of 0.5474 among the cooling strategies. For insertion, fast annealing also reaches a peak of 0.5227. Swapping is more effective compared to insertion. For both graphs the standard error is small enough for the result to be accepted after number of selected features is more than 6. The maximum MAP score of 0.4160 is achieved by fast annealing for swapping. For insertion, a maximum MAP score of 0.3977 is achieved by fast annealing.

B. Effect of Progress Parameter

Fig. 6 shows the comparison of our algorithm with progress parameter with the traditional one where the progress parameter is not included. As the swapping and fast annealing combination yielded the best results in our previous experiments, here the comparison is made using this setting. We can see that in most of the cases the traditional algorithm (i.e., without progress parameter) is outperformed by our proposed technique.



(a) Progress parameter: MQ2008 (b) Progress parameter: MQ2007

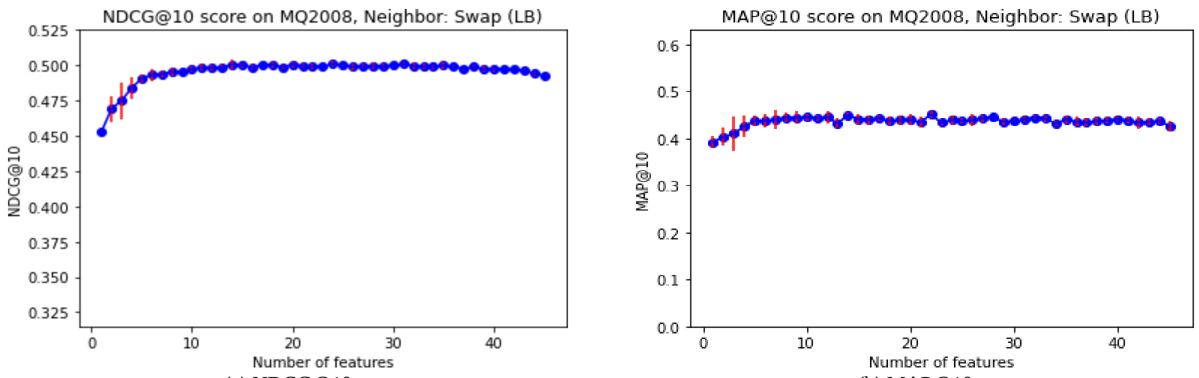


(c) Progress parameter: OHSUMED (d) Progress parameter: TD2004

Fig. 6. Effect of progress parameter on performance of Simulated Annealing.

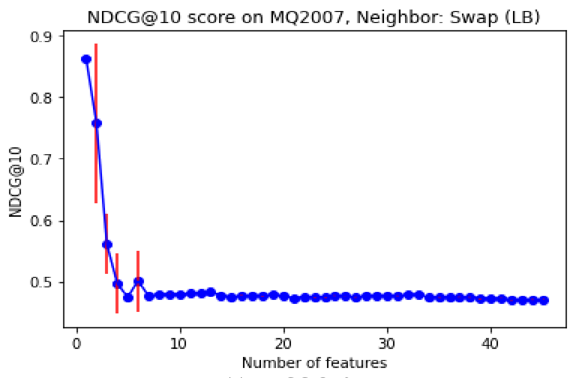
C. Local Beam Search Compared to Simulated Annealing

Now we discuss the feature selection performance of local beam search, which is followed by the comparison between local beam search and simulated annealing.

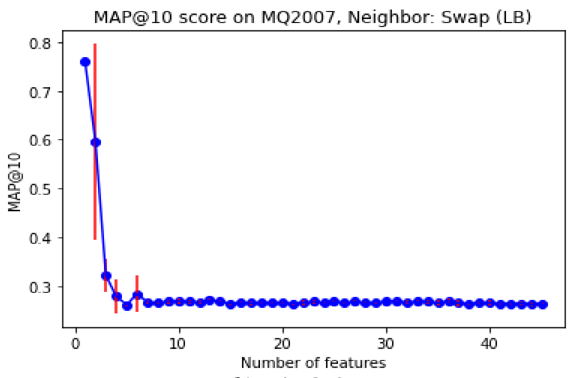


(a) NDCG@10 (b) MAP@10

Fig.7. Ranking scores on MQ2008 dataset for local beam search



(a) NDCG@10

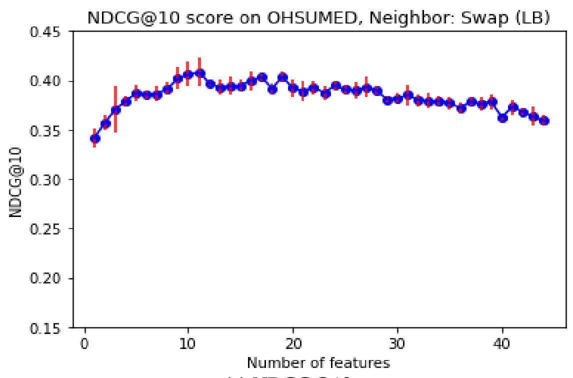


(b) MAP@10

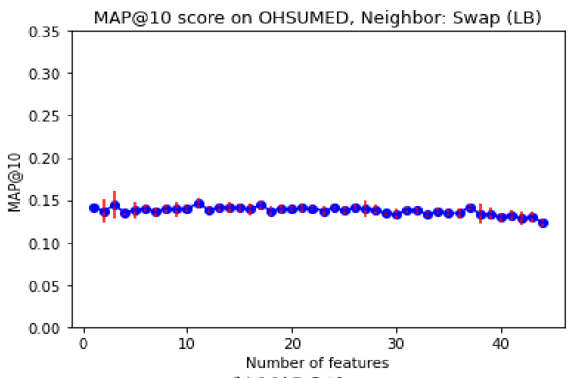
Fig.8. Ranking scores on MQ2007 dataset for local beam search

For MQ2008 dataset (Fig. 7), the max NDCG score reached by local beam search is 0.5009 and MAP score of 0.4503. For MQ2007 dataset (Fig. 8), the maximum NDCG and MAP scores are 0.4827 and 0.2703 respectively. For OHSUMED dataset (Fig. 9), the maximum NDCG score is 0.4081 and MAP score is 0.1459. For TD2004 dataset (Fig. 10), the max NDCG score is 0.5398 and MAP score of 0.4399.

Fig. 11 compares simulated annealing (neighbor selection: swap, cooling scheme: fast annealing) and local beam search for all four datasets. For MQ2008 dataset, although at very early stages, performance is better for local beam search, simulated annealing performs better as we increase the number of features. For MQ2007 dataset, both performs quite similarly, with simulated annealing having slightly better results. For OHSUMED and TD2004 datasets, simulated annealing performs much better in finding better feature subsets than local beam search.

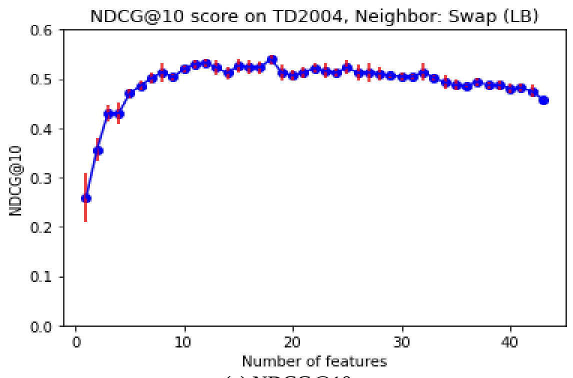


(a) NDCG@10

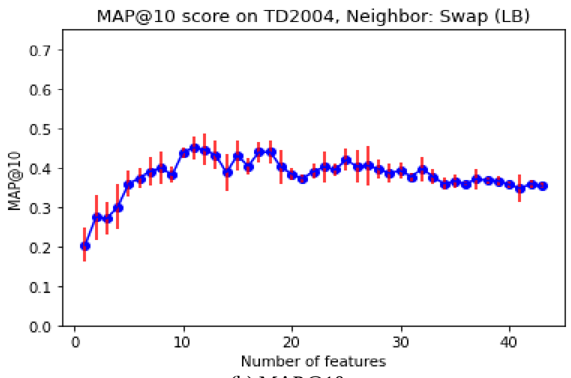


(b) MAP@10

Fig.9. Ranking scores on OHSUMED dataset for local beam search



(a) NDCG@10

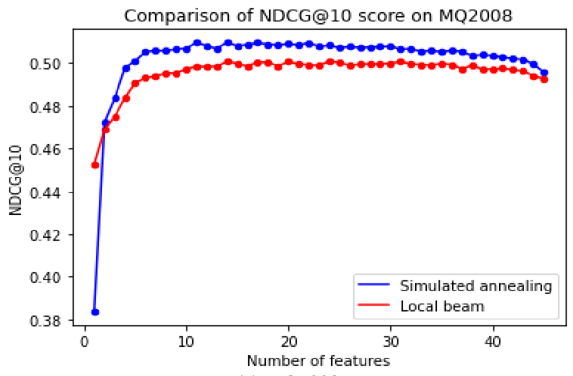


(b) MAP@10

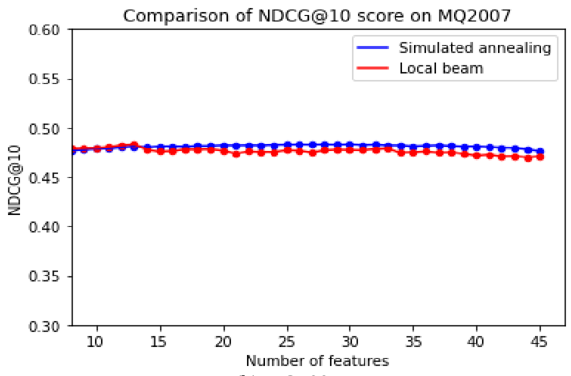
Fig.10. Ranking scores on TD2004 dataset for local beam search

D. Experiments on Big Data

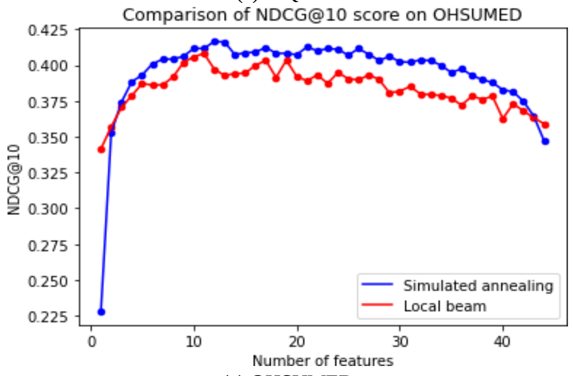
The MSLR-WEB10K dataset is quite large compared to other datasets we have experimented with. So for this dataset we consider the best variant (neighbor selection: swap, cooling scheme: fast annealing) of simulated annealing found on the other four datasets. Also, for this dataset we conduct only one run of the algorithms instead of 10 runs. Fig. 12 plots the feature selection performance of simulated annealing. The maximum NDCG score achieved by simulated annealing is 0.4729 and MAP score of 0.1387. Fig. 13 plots the feature selection performance of local beam search. The maximum NDCG score reached by local beam search is 0.4725 and MAP score of 0.1400. Fig. 14 compares between simulated annealing and local beam search. For the MSLR-WEB10K dataset, simulated annealing and local beam search both perform quite similarly.



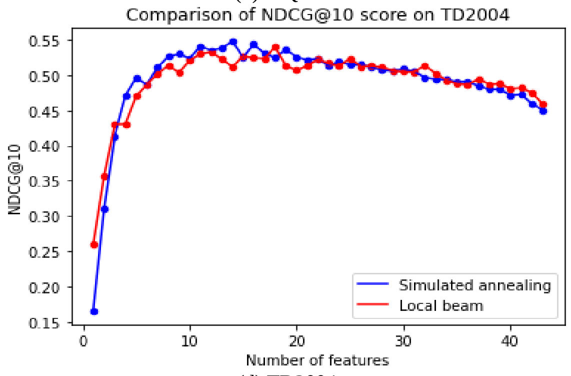
(a) MQ2008



(b) MQ2007

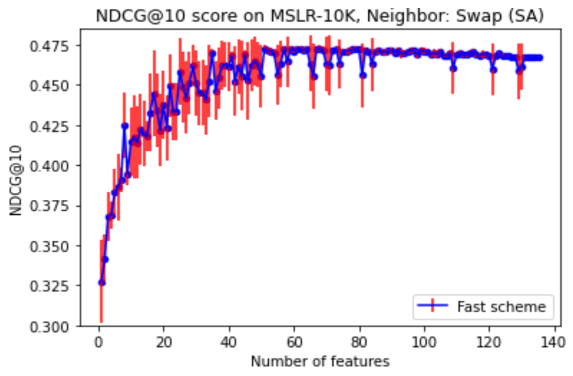


(c) OHSUMED

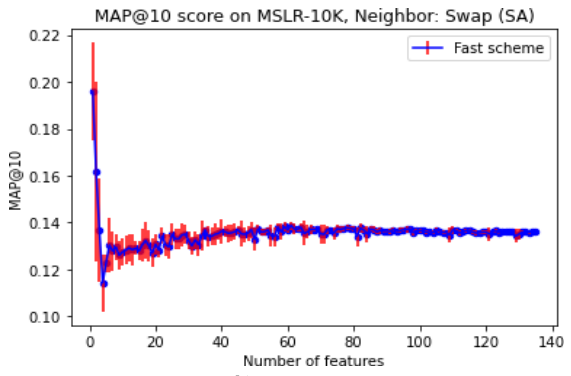


(d) TD2004

Fig.11. Simulated annealing vs local beam search for all datasets

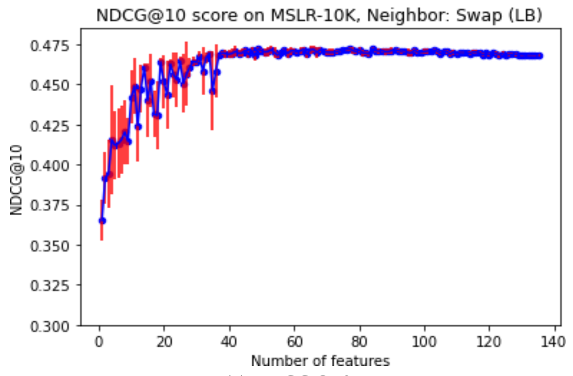


(a) NDCG@10

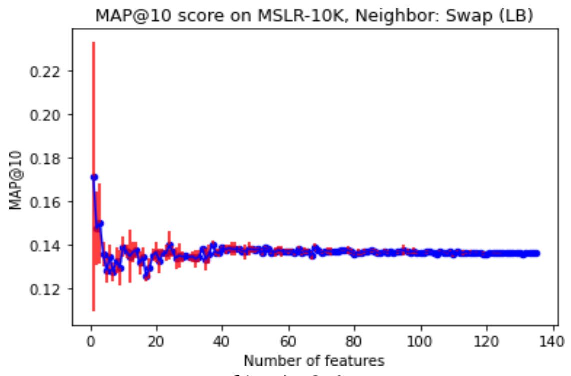


(b) MAP@10

Fig.12. Ranking scores on MSLR-WEB10K dataset for simulated annealing



(a) NDCG@10



(b) MAP@10

Fig.13. Ranking scores on MSLR-WEB10K dataset for local beam search

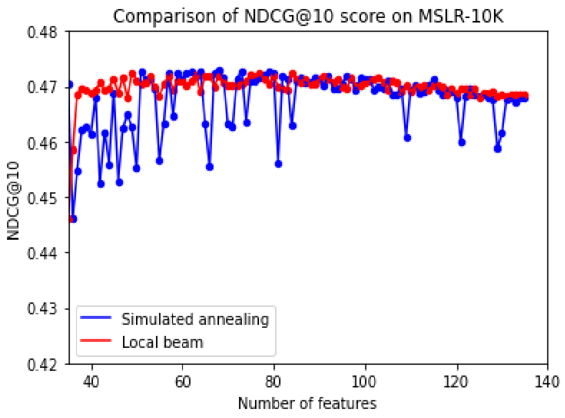


Fig.14. Simulated annealing vs local beam search on MSLR-WEB10K dataset

Table 2 demonstrates performance comparison between local beam search and simulated annealing algorithm across all five datasets.

Table 2. Numerical comparison of performance metrics between local beam search and simulated annealing across all five datasets.

Dataset	**Local Beam**		**Simulated Annealing**	
	NDCG	MAP	NDCG	MAP
MQ2008	0.5009	0.4503	0.5097	0.3605
MQ2007	0.4827	0.2703	0.4829	0.2486
OHSUMED	0.4081	0.1459	0.4165	0.1455
TD2004	0.5398	0.4399	0.5474	0.4160
MSLR-WEB10K	0.4725	0.1400	0.4729	0.1387

4.6. Training Time Comparison

Fig.15 shows the training time comparison between simulated annealing and local beam search. Here the *x* and *y* axes represent the datasets and the required time (in hours) respectively. We see that local beam search is found to be computationally more expensive than simulated annealing for all five datasets.

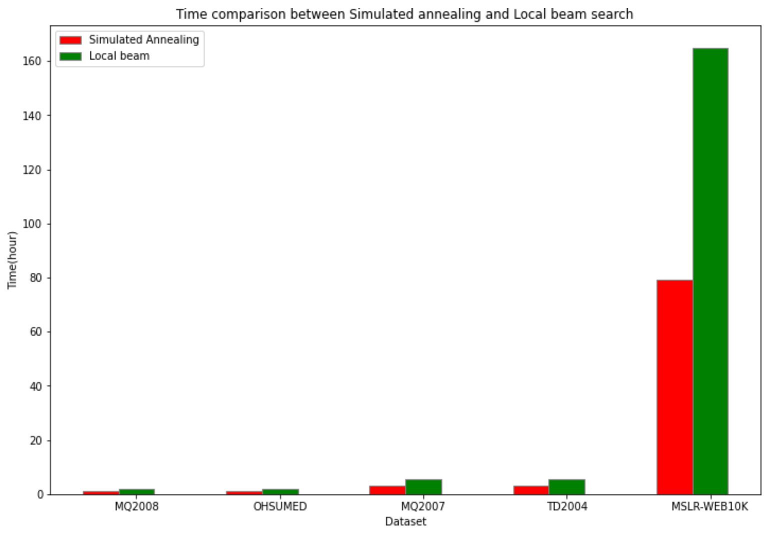


Fig.15. Time comparison between simulated annealing and local beam search

4.7. Discussion

After analyzing the experimental results of the benchmark datasets, we observe the following patterns:

- Feature selection improves the overall ranking accuracy of a model, and also reduces training time to some extent. This is because adding too many features may result in overfitting (especially if the number of training instances is relatively less) and increased training time. In all the datasets of our experiments, when the number of features was too low, the ranking performance was found to be unacceptable due to suffering from underfitting problem. As we kept on adding features, the ranking error and standard error started to go down up to a certain point. After that, adding more features did not improve the performance much. Instead, in some cases it resulted in lower performance, which, we conjecture, was due to overfitting problem or due to irrelevant or noisy features. The plots of NDCG@10 of OHSUMED and TD2004 datasets showed that adding too many features may even deteriorate the performance.

- In most scenarios we observe that the swap neighboring scheme seems to perform better than the insertion technique for simulated annealing. This is probably due to the fact that the insertion neighboring scheme is quite random in nature as inserting from one position to another shifts the whole set of selected features. Due to its randomness, it might give better result than swapping when cooling time is small compared to the problem. Whereas, swapping only exchanges one feature for another. As it may swap a feature that is contributing less to the performance of the model for a feature contributing more, this definition of neighbourhood enables us to find better solutions in a relatively controlled manner given sufficient cooling time, as it takes small steps towards a better solution.
- Among the cooling strategies, fast annealing is found to work better. Fast annealing converges faster than the other two cooling schemes. As a result, when in the early iterations of simulated annealing, fast annealing scheme tends to give better result. However, in the very early iterations, fast annealing works similarly as the other two strategies.
- The progress parameter is found to improve the ranking accuracy across all datasets. This is because it helps the algorithm escape local optima as well as ensuring a better solution state among the neighbors of the current best state.
- Local beam search is, in general, found to be slightly less effective than simulated annealing. We conjecture that the reason behind this is, local beam search always considers better solutions than the current one, thereby being more prone to getting stuck at a local optimum. Also, its time requirement is larger as it keep a list of states of the beam length, and in each iteration, it needs to find the best state from the list.
- While the techniques investigated in this work have been found to be promising in experimental results, the scalability still remains an issue to be addressed. Selecting the good subset of features from the available ones is itself a procedure that may require considerable computational time. Another limitation of this study is that if all the features are effective in some domain, the importance of feature selection becomes less.
- A particular challenge during implementation was how to store so many training sets generated for each potential feature subset. We tackled this problem by not storing the training sets in disks, rather by storing them in memory.

5. Conclusions

In applications where a large number of instances are not available, feature selection may significantly improve ranking performance of learning-to-rank algorithms. In this research, we have adapted an efficient meta-heuristic algorithm called simulated annealing to select a better subset of features for learning-to-rank problem. We have investigated two neighbour selection approaches, three temperature cooling schemes, and have incorporated a novel parameter for better traversal of the search space. We compared the performance of our simulated annealing algorithms with another meta-heuristic algorithm called local beam search.

Several research directions have emerged from this research. Other meta-heuristic approaches may be investigated for solving the feature selection in learning-to-rank. It is necessary to conduct further experiments on larger datasets. Further experimentation may be conducted to find the best settings of hyper-parameters to yield better ranking performance. Finding ways to reduce training time by carefully narrowing down the search space can also be an interesting research direction.

Funding Declaration

We declare that we receive no funding from anywhere for this research.

Competing Interest

We declare that we have no known competing interests regarding this research.

References

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[2] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[3] Liu, T. Y. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, *3*(3), 225-331.

[4] Isabelle Guyon and Andre´ Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[5] Mohammed A Ambusaidi, Xiangjian He, Priyadarsi Nanda, and Zhiyuan Tan. Building an intrusion detection system using a filter-

based feature selection algorithm. *IEEE transactions on computers*, 65(10):2986–2998, 2016.

[6] Noelia Sa´nchez-Marono, Amparo Alonso-Betanzos, and Mar´ıa Tombilla-Sanroma´n. Filter methods for feature selection–a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learn- ing*, pages 178–187. Springer, 2007.

[7] Ron Kohavi and Dan Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *KDD*, pages 192–197, 1995.

[8] Sebastia´n Maldonado and Richard Weber. A wrapper method for feature selection using support vector machines. *Information Sciences*, 179(13):2208–2217, 2009.

[9] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[10] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 17(4):491–502, 2005.

[11] Manik Sharma and Prableen Kaur. A comprehensive analysis of nature-inspired meta-heuristic techniques for feature selection problem. *Archives of Computational Methods in Engineering*, pages 1–25, 2020.

[12] Jitendra Rajpurohit, Tarun Kumar Sharma, Ajith Abraham, and A Vaishali. Glossary of metaheuristic algorithms. *Int. J. Comput. Inf. Syst. Ind. Manag. Appl*, 9:181–205, 2017.

[13] Xin-She Yang. Harmony search as a metaheuristic algorithm. In *Music-inspired harmony search algorithm*, pages 1–14. Springer, 2009.

[14] Shih-Wei Lin, Zne-Jung Lee, Shih-Chieh Chen, and Tsung-Yuan Tseng. Parameter determination of support vector machine and feature selection using simulated annealing approach. *Applied soft computing*, 8(4):1505–1512, 2008.

[15] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, con- struction and selection*, pages 117–136. Springer, 1998.

[16] Lucija Brezocˇnik, Iztok Fister, and Vili Podgorelec. Swarm intelligence algorithms for feature selection: a review. *Applied Sciences*, 8(9):1521, 2018.

[17] Paolo Dario, Giulio Sandini, and Patrick Aebischer. *Robots and Biological Systems: Towards a New Bionics?: Proceedings of the NATO Advanced Workshop on Robots and Biological Systems, held at II Ciocco, Toscana, Italy, June 26–30, 1989*, volume 102. Springer Science & Business Media, 2012.

[18] Wojciech Siedlecki and Jack Sklansky. On automatic feature selection. In *Handbook of pattern recognition and computer vision*, pages 63–87. World Scientific, 1993.

[19] Majdi M Mafarja and Seyedali Mirjalili. Hybrid whale optimization algorithm with simulated annealing for feature selection. *Neurocomputing*, 260:302–312, 2017.

[20] Mustafa Wasif Allvi, Mahamudul Hasan, Lazim Rayan, Mohammad Shahabuddin, Md Mosaddek Khan, and Muhammad Ibrahim. Feature selection for learning-to-rank using simulated annealing. *International Journal of Advanced Computer Science and Applications*, 11(3):699–705, 2020.

[21] Ibrahim, Osman Ali Sadek and Younis, Eman MG , Hybrid online--offline learning to rank using simulated annealing strategy based on dependent click model, *Knowledge and Information Systems,* 64(10), pp. 2833 – 2847, 2022, Springer.

[22] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 407–414, 2007.

[23] Andrea Gigli, Claudio Lucchese, Franco Maria Nardini, and Raffaele Perego. Fast feature selection for learning to rank. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, pages 167–170, 2016.

[24] Sanmay Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Icml*, volume 1, pages 74–81. Citeseer, 2001.

[25] Daniel Xavier Sousa, Se´rgio Canuto, Marcos Andre´ Gonc¸alves, Thierson Couto Rosa, and Wellington Santos Mar- tins. Risk-sensitive learning to rank with evolutionary multi-objective feature selection. *ACM Transactions on Information Systems (TOIS)*, 37(2):1–34, 2019.

[26] Feng Pan, Tim Converse, David Ahn, Franco Salvetti, and Gianluca Donato. Greedy and randomized feature selec- tion for web search ranking. In *2011 IEEE 11th International Conference on Computer and Information Technology*, pages 436–442. IEEE, 2011.

[27] Han-Jiang Lai, Yan Pan, Yong Tang, and Rong Yu. Fsmrank: Feature selection algorithm for learning to rank. *IEEE transactions on neural networks and learning systems*, 24(6):940–952, 2013.

[28] Ashwini Rahangdale and Shital Raut. Deep neural network regularization for feature selection in learning-to-rank. *IEEE Access*, 7:53988–54006, 2019.

[29] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applica- tions*, pages 7–15. Springer, 1987.

[30] Rene´ VV Vidal. *Applied simulated annealing*, volume 396. Springer, 1993.

[31] Dimitris Bertsimas, John Tsitsiklis, et al. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[32] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[33] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[34] Max Kuhn and Kjell Johnson. *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019.

[35] Noelia Sa´nchez-Marono, Amparo Alonso-Betanzos, and Mar´ıa Tombilla-Sanroma´n. Filter methods for feature selection–a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learn- ing*, pages 178–187. Springer, 2007.

[36] ChangYing Wang, Min Lin, YiWen Zhong, and Hui Zhang. Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling. *International Journal of Computing Science and Mathematics*, 6(4):336–353, 2015.

[37] David T Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, 46(1):93–100, 1990.

[38] Harry Cohn and Mark Fielding. Simulated annealing: searching for an optimal temperature schedule. *SIAM Journal on Optimization*, 9(3):779–802, 1999.

[39] Harold Szu and Ralph Hartley. Fast simulated annealing. *Physics letters A*, 122(3-4):157–162, 1987.

[40] David Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management science*, 37(1):98–113, 1991.

[41] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(1-4):131–156, 1997.

[42] David W Aha and Richard L Bankert. A comparative evaluation of sequential feature selection algorithms. In *Learning from data*, pages 199–206. Springer, 1996.

[43] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[44] Peng Si Ow and Thomas E Morton. Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[45] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

[46] William Hersh, Chris Buckley, TJ Leone, and David Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *SIGIR'94*, pages 192–201. Springer, 1994.

[47] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[48] https://www.microsoft.com/en-us/research/project/mslr/

[49] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

Authors' Profiles

Mohd. Sayemul Haque attained his Bachelor of Science degree in Computer Science from the University of Dhaka. Presently, he holds the position of Senior Software Engineer at Enosis Solutions, Bangladesh. His research interest lies in Data Science and the learning-to-rank domain. In addition to his involvement in various challenging projects, he has made significant contributions to the learning-to-rank domain through his role as the primary author in a journal publication.

Md Fahim received his B.Sc Degree in Computer Science from the University of Dhaka. He is currently serving as a Research Assistant at the Center for Computation and Data Sciences (CCDS Lab) at Independent University, Bangladesh. His research interests primarily lie in the fields of NLP, explainability, and low resource languages along with learning-to-rank domain. With a focus on advancing these areas, he has contributed significantly to the field, having authored five first-author conference publications.

Muhammad Ibrahim achieved his Ph.D. degree in Information Technology from Monash University, Australia. Prior to this, he achieved his M.Sc. and B.Sc. (Hons.) degrees in Computer Science and Engineering from University of Dhaka, Bangladesh where he is currently working as an Associate Professor. He has published 13 journal articles in peer-reviewed indexed journals and many conference papers. His area of research interest includes information retrieval, applied machine learning, meta-heuristic algorithms, human computer interaction.

How to cite this paper: Mohd. Sayemul Haque, Md. Fahim, Muhammad Ibrahim, "An Exploratory Study on Simulated Annealing for Feature Selection in Learning-to-rank", International Journal of Intelligent Systems and Applications(IJISA), Vol.16, No.4, pp.86-103, 2024. DOI:10.5815/ijisa.2024.04.06