

Algorithmic-Information Theory interpretation to the Traveling Salesman Problem

Matheus Sant' Ana Lima
Department of Computing
Federal University of Sao Carlos
Sao Paulo, Brazil

ABSTRACT

The Traveling Salesman Problem (TSP) is a important optimization problem in computer science, mathematics and logistics. It belongs to the class of NP-Hard problems and can be very time consuming to find solutions to large instances with guarantee optimality. As number of city-nodes in the graph increases, the amount of valid route tours also grows rapidly and thus requiring considerable time to evaluate and classify each permutation. The objective of the heuristic process is to search the solution space for the optimal solution while maximizing the attached utility-cost function (i.e. finding the shortest euclidean distance tour) and minimizing the computational time complexity of the algorithm.

Many complex real world scenarios can be reduced to a simulation of a salesman trying to find the shortest (length) Hamiltonian (cycle) route in a euclidean super-graph G^* . If each city-node is modeled as a input symbol in a communication channel represented by an output pair with consistent probabilities distribution thus an polynomial-time probabilistic algorithm can use this information to improve the solution quality at the same rate of transmission of information over the channel.

In this paper we explore an quantitative stochastic process based in Algorithm Information Theory and the Shannon-Kelly criterion to find valid near optimal solutions using a new growth-optimal strategy applied to the TSP problem that have statistically significant transmission rate even when no encoding scheme is available, regardless of time-complexity of the problem.

Previous heuristics such as 2 opt, Genetic Algorithms (GA) and Simulated Annealing (SA) approach's the TSP problem by relying on *a priori* knowledge about the data distribution in order to reduce the probability of error in finding the best candidate solution tour.

In this work we propose a method that models the solution space boundaries of the TSP problem as a communication channel by means of Information Theory. We describe a search algorithm that check for patterns (i.e information content) in the elements of a constrained solution space modeled as messages transmitted through communication systems. The boundaries of the search space are defined by the Kolmogorov complexity of the candidate solutions sequences. We conclude with an discussion about the quality of the results and implications for general decision problem in Turing machines.

Keywords: Information Theory, Algorithm Complexity, Turing Machine, Traveling Salesman Problem

1. Introduction

1.1. Hamiltonian Graphs, TSP Problem and Computational Complexity

The computational complexity of an algorithm show how much resources are required to apply an algorithm.[5] In other words how much time and memory are required by a Turing machine to complete execution and can be interpreted as a measure of difficulty of computing functions. [7] A measurement of computation complexity is big O notation. It can be defined as: Let two functions f and g such as $f(n)$ is $O(g(n))$ if there are positive numbers c and N such that

$$f(n) \leq cg(n)$$

for all $n \geq N$ and is used to estimate the function growth tax (asymptotic complexity). [5] [6]

ORCID(s):

The TSP problem is an important combinatorial optimization problem. As most of the decision problems, it is in the class of NP-hard problems.

Consider a salesman traveling from city to city and some of them are connected. His goal is to visit each city exactly once and go back to the first city when he finishes. The salesman can choose any path as long as it is valid (ie visit each city once and finish at the city it has started the tour) he also wants to minimize his cost by taking the shortest route. This problem can be described as a weighted graph G where each city is a node (or vertex) and is connected by a weighted edge only if the two cities are connected by any kind of road, and this road does not cross any other city. The weight value of the edge is defined as the distance of the tour (roads) between cities.

For symmetric TSP, as the number of nodes (or cities) increases in the graph G , the number of possible tours grows also increases and is exponential. If we consider N nodes, the function of the input size is $f_{size} = ((N - 1)!/2)$. This means that the number of elements an algorithm must evaluate to decide the problem is very large thus requiring considerable time and computational resources even for small instances of the problem.[3]

An strategy to address this limitation is to accept near-optimal solutions by setting constraints in the problem (using heuristics methods)[4]. Algorithms such as 2opt, GA and SA define *a priori* knowledge about the distribution of the solution space and then repetitively try to improve the quality. It works by following some heuristic function schema while trying to avoid a local minimal. As the heuristics (of TSP and NP problems) are a best effort strategy to find a (good) near-optimal solution (by enforcing space and time boundaries), it does not guarantee that the solution found is the best candidate to the problem and therefore an algorithm can never be sure that if by running more time the overall solution cost could be improved, unless the entire solution space to the problem is evaluated. This limitation is set by the definition of NP-hard class.

Problems in the NP class can be solved by a non-deterministic polynomial algorithms [1]. Any given class of algorithms such as P, NP, etc must have a lower bound that indexes the best performance any problem in the class can have. This bound can be described as the total amount of input items (our symbols) a machine must process and the respective output items produced [1] or as we use in this paper, an Probability Distribution Function. A strategy to find a solution to the decision problem is to find a function that *reduce* or transform a problem from a domain in which there is no known solution to a constrained domain with a known solution. This allows for an algorithm to search the solution space and decide if any solution is a valid (or yes-instances) or invalid (or no-instances) and its computable by a polynomial-time algorithm[1].

This strategy allows us to map instances of the Hamiltonian circle problem to a decision version of the Traveling Salesman Problem and can be described as a decision problem to determine *if exists an Hamiltonian circuit in a given complete graph with positive integer weights whose length is not greater than a given positive integer m*. [1] Each valid (yes-instance) in the TSP problem is mapped to a valid instance in the Hamiltonian problem space and this transformation can be done in polynomial time.[1]

An Hamiltonian cycle (or circuit) can be described as an "path" that contains all nodes and the elements in this set are not repeated, with exception to the final vertex. This means that a Hamiltonian cycle in G with start node v has all other nodes exactly once and then finishes at node v . An graph G is Hamiltonian if it has a Hamiltonian cycle. An Hamiltonian cycle with minimum weight is a optimal circuit and therefore is the shortest tour in the TSP Problem.

Although heuristics such as 2opt define special cases for the TSP problem and produces near-optimal solutions with short length (weight) Hamiltonian cycles it does not guarantee that the results are the shortest circuit possible. [2]

1.2. Information Theory, Relative Entropy and Bernoulli sequences (i.e. binary random words)

Information can be described by a quantity called entropy defined by the probability distribution function of a random variable X . It is a measure of uncertainty about this variable. Let X be discrete with alphabet L and probability mass function

$$p(x) = Pr\{X = x\}, x \in X$$

The entropy H is defined

$$H(X) = - \sum p(x) \log_2 p(x)$$

The expectation value E of a random variable $f(x)$ is defined as

$$E(x) = \sum f(x)p(x)$$

from a known probability mass function $p(x)$. Alternatively, entropy can also be defined as the expected value exp of the random variable

$$\log 1/p(X)$$

from X in a probability mass function $p(x)$.

This interpretation allows us to set constraints that the entropy function of a random variable must have.[13] In other words, entropy can be interpreted as the average code length of the shortest description of a random variable and is measured by taking the logarithm. Entropy describes the amount of information required on average to encode a random variable.[15]

The *joint entropy* of a pair of random variables X and Y can be defined as

$$H(X, Y) = -E \log_2 p(X, Y)$$

The conditional entropy of a random variable X given the conditional entropy of Y , thus

$$H(Y|X) = -E \log p(Y|X)$$

The *relative entropy* between distributions p and q is a measure of distance $\text{Dist}(p||q)$ and can be interpreted as a measure of performance (or inefficiency) when assuming distribution q but knowing p is the true distribution.[15] In other words, by assuming a code for distribution q , this measures the additional cost $D(p||q)$ required to describe $H(p)$. This means that on average we need

$$H(p) + D(p||q)$$

bits to describe the random variable. [13]

The *mutual information* $I(X, Y)$ measures the amount of information one random variable X has about another random variable Y with joint probability mass function $p(x, y)$ and marginal probability function $p(x)$ and $p(y)$. $I(X, Y)$ can be defined as the relative entropy between the joint distribution and the product distribution $p(x)p(y)$. This quantity measures the reduction in the uncertainty of X due to knowledge about the distribution of the random variable Y .

In other words by conditioning a random variable by another we can reduce entropy by adjusting the probability of error. This property makes it possible to estimate the expected value of a correlated random variable X by observing the random variable Y . Consider as an example that we choose a card from a deck with 52 cards. The probability of this event is $P(X) = 1/52$. Thus the entropy is

$$H(X) = -(1/52) \log_2(1/52) = 0.13 \text{ bits}$$

and we need this quantity of bits to describe the event of randomly choosing this card. Let *event* $e1$ on the card deck be to randomly pick a king of hearts (There are 4 kings in a 52 deck) and *event* $e2$ be the knowledge that the card is a heart (There are 13 hearts thus the probability is $1/4$). The entropy of $e1$ by knowing $e2$ is defined by the conditional entropy $H(Y = e2|X = e1)$. The uncertainty of the first event was reduced by knowing $e2$. Thus event $e2$ has mutual information with $e1$. [14]

A function $f(x)$ is convex over an interval (a, b) if it is positive and has a second derivative [13]. Examples of such convex functions are \log , x^2 and e^x . Let X be a random variable and E the expectation on variable X . For the discrete case there is a x in X such as

$$EX = \sum [p(x)x]$$

and

$$EX = \int [xf(x)dx]$$

in the continuous case. The Jensen inequality states that if $f(x)$ is convex and X is a random variable then this expectation is such that

$$Ef(X) \geq f(EX)$$

This implies that $X = EX$ with probability 1 if f is strictly convex. If we have two probability mass functions $p(x)$ and $q(x)$ that describes x in variable X then the distance $D(p||q)$ is only equal if and only if

$$p(x) = q(x) : \forall x \in X$$

. Thus if X and Y are independent variables then the entropy

$$H(X|Y) = H(X)$$

and

$$H(X|Y) \leq H(Y)$$

otherwise. This means that on average observing the random variable Y can only reduce uncertainty on the variable X .

This inequality properties between random variables X and Y can be used to measure the performance. [16] Let Z be a conditional distribution function that depends only on Y and is conditionally independent from X . We can form a Markov chain

$$X -> Y -> Z$$

only if the joint probability function can be defined as

$$p(x, y, z) = p(x)p(y|x)p(z|y)$$

The mutual information for this chain is

$$I(X, Y) \geq I(X, Z)$$

Therefore by observing the random variable Z (i.e. "index variable") its possible to evaluate the dependence of X and Y .

Let x be a sample of a variable X , an index value Θ from a distribution $p(X)$ and a function $T(X)$ of the sample. By inequality we know that

$$I(\Theta; T(X)) \leq I(\Theta; X)$$

A statistic T is sufficient for index Θ if it contains all information in X about that index. Thus function T preserves mutual information. The minimal sufficient statistically maximally compress the information about Θ , relative from a family of probability distribution functions that are indexed by Θ . [18]


Those inequalities between distributions enables us to divide the set of all possible sequences in two sets and thus we can use to predict the probability of the random sequence and find the shortest description of those sequences. A binary random sequence is a list produced by a Bernoulli process. Each element in the list is defined as a independent distributed random variable X with values 0 or 1. Let X be an independent identically distributed random variable X with probability mass function $p(x)$. [20] [19] All elements in the both sets are sorted by some index of the sequences. This encoding schema maps all sequences in x to a respective binary string. Although small there is a significant probability and this implies that some sequences have higher probability than others binary sequences.

Let random variables x in X form a stationary process (sequence). The *Entropy rate of a process* is the the entropy of such sequence defined as $H(x_1, x_2, x_3, \dots, x_n)$ and it grows linearly (asymptotically), with n at a rate $H(X)$ - . This rate is the limit of achievable compression of information. [13]

1.2.1. Kolmogorov Complexity and inference

The Shannon-Bernoulli definition of a random variable X with a probability mass function $p(x)$ has a entropy equal to the number of bits required to describe x in X by a Shannon code. Alternative in his work Kolmogorov define the algorithm complexity of an object. Kolmogorov defined that the descriptive complexity of an object to be the length of the shortest binary that describes the object.[25] [26] Each symbol in the language have a probability of occurring (an event in statistics) in any string or word, with a finite length. This complexity is independent of the machine interpreting the coding. [13]

For example lets consider $seq1 = 1010101$ and $seq2 = 101000$. In $seq2$ we have list of length 6 with 2 elements with binary symbol 1 and 4 elements with symbol 0. The first sequence appears to be more random than the second sequence since we have an more uniform distribution of events in which the string has elements (or variables) with values 0 and 1. The first sequence is more complex than $seq2$ and must require more bits to represent.[13]

Input strings	<ol style="list-style-type: none"> 1. abcbcbbcbcbcd 2. abd 3. abcbcbcbcd 4. abcbcbcbcbcbcbcbcbcbcbcbcbcbcbcbcbcbcd
Regular expression	<p>a(bc)^{0,13}bd</p> <p>more generalized: a(bc)*bd</p>
Automaton	 <pre> graph LR S(()) -- a --> Q1(()) Q1 -- b --> Q2(()) Q2 -- c --> Q1 Q2 -- d --> F((())) </pre>

The expected length of the shortest binary description of a Bernoulli variable is close to its Shannon $H(X)$ entropy. This descriptive program can be interpreted as a universal Shannon code to the Turing Machine for all probabilities.

The Kolmogorov complexity gives a method to compare the complexity in the struct of such random sequence strings. For example, in a sequence created from a coin-flip, with n bits length, there are 2^n possible sequences and all are probable. The algorithm complexity of this random string (Bernoulli sequence) has the same size as the sequence itself. The intrinsic complexity (measured as length of the shortest program) is computer independent, up to limiting constrains. This constrain is defined as a additive constant to the complexity and is the length of the pre-program that allows the machine to compile another program.

Turing machines TM are the canonical model for computers. Universal computers mimic (or replicate) the actions of other computers. The Von-Neumann Architecture can be reduced to a Turing Machine. TM can be defined as a finite-state machine operating on a finite tape. At each time the machine read a symbol from the tape and may change its state, under some predefined decision rule. Each operation is traced by a finite list of transactions. [13] The figure 2 contains a basic automata representation of a Turing machine from Hopcroft et al. [42]

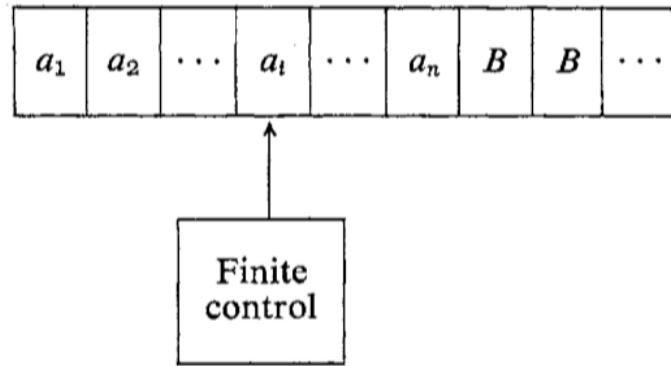


Figure 2: Basic Turing Machine

The machine TM reads symbols s from tape t from left to right and at each symbols it decides if to update the cell with another symbol. This machines never goes back and stops after reading the entire tape once. The programs for this machine form a prefix-free list. However is assumed that no program that produces a halting computation for TM can be prefix of another program. This constrain in the machine is defined by the Kolmogorov Complexity of the string that encode this program.

As the TM machines computes the cells from the tape it maps the elements from a set of finite length binary string to another set that than can have finite or infinite binary string length. If the computation does not halt it is *undefined*. [13]

1.2.3. Complexity of a string and shortest description length

Let U be a Universal computer. The Kolmogorov complexity $K(x)$ of a string x of a computer U is

$$K(x) = \minlength(p)$$

when

$$p : U(p) = x$$

It is the minimum length program p that output variable x and halts. Its the small possible program. Let C the another computer. If this complexity is universal there is a universal computer U that simulates C for any string x by a constant c on computer C . Thus $K(x)_{of U} \leq K(x)_{of C} + c$. [25] This constant does not depend on string x . [13] The upper bound on the Kolmogorov complexity is

$$K(x) \leq K(x|length(x)) + 2 \log_2 length(x) + c$$

Therefore there are few sequences with low complexity in the set. The lower bound states that are not many short programs in the set. If X is a Bernoulli sequence with probability $p(x) = 1/2$, this means there are no more than 2^k strings with complexity $K(x) < k$. [25]

Consider each program can produce only one possible output, the number of sequence with complexity $K(x) < k$ is less than 2^k . Thus the complexity will depend on the computer up to an additive constant. Lets consider fractals for example. They can produce complex 2d images with self-replicating patterns with many different scales. However the complexity to describe its rules have a Kolmogorov Complexity close to zero. [13]

The expected value of the Kolmogorov complexity of a random sequence is close to the Shannon entropy. This relationship between complexity and entropy can be described as a stochastic process drawn to a i.i.d on variable X following a probability mass function $p(x)$. The symbol x in variable X is defined by an finite alphabet. This expectation is

$$E(1/n)K(X^n|n) \Rightarrow H(X)$$

Therefore most of the sequences in the set do not have simple description. However there are some simple sequences.

The probability that a random sequence can be compressed by more than k bits is no greater than $2^{(-k)}$. Thus most Bernoulli sequences have a complexity close to their length.

The Kolmogorov complexity of an independent and identically distributed random binary variable created by a Bernoulli process is close to the entropy binary function.[29] The expected value of the complexity of the Bernoulli sequence converges to the entropy. Therefore we can bound the number of sequences with complexity that are significantly lower than entropy and classify each element between two sets (the typical sets and a non-typical set). [13]

Let U be a universal computer. The universal probability of a string is defined as

$$P_U(x) = \sum [2^{(-length(p))}] = Pr([U(p) = x])$$

This is the probability that a random program produces the string x as output. In other words it is the probability of observing the string in a set. This is based on the assumption that the output strings are not uniformly distributed and strings with simple description are more likely than complicated strings.[28] This universal probability can be interpreted as the general representation of all computable probability distributions. [29]

1.2.4. Halting problem

The halting problem states that there is no general algorithm to decide if any program halts or goes on forever, for all inputs in a machine. Turing proved that no such algorithm can exist for all programs. This problem is similar to the self-reference property of the Godel incompleteness theorem. The halting problems can be understood as a limitation of the class of problems that can be solved by any computer model.[13]

Therefore to find the shortest program we need to decide between all possible short programs. Since some programs may or may not halt there is no efficient procedure to find the shortest program that produces a given output string. The consequence is the *non-computability* of the Kolmogorov complexity. This means that the machine executing the pair of input and program may find the solution but never knows that no better short programs exist. However by observing the outputs produced we can estimate a distribution function that converges to the true Kolmogorov complexity in the long run.[13]

The Chaitin Ω describes a compact representation of information and randomness. It defines the probability that a universal computer U halts when the input is a random binary sequence produced by a Bernoulli process. Formally

$$\Omega = \sum [2^{(-length(p))}]$$

such that $p : U(p)$ halts. Let's consider that we can process all programs in parallel (at each iteration time unit / clock cycle in a computer) and assume (by the law of large numbers) that eventually all programs that halt will be found. Those programs that were found to halt are then placed in a list of instructions to track the execution tree. These elements in this list must be prefix-free. Therefore we can discover if a program with less than n bits exists and will halt. In other words this method *reduces any problem to a search of a counterexample*. In its general form this method assumes that it is possible to find a finite-proof for any theorem if such proof exists.[13] [22]

1.3. Universal Gambling device and Universal Probability

As an example let's consider a universal gambling machine that places bets on a binary sequence. The gambler device can place bets at each interaction and with net odds of 2-for-1 on each bit. This means the winning bet yields gains of 2 units for a 1 unit "invested". How much should the device bet at each interaction to avoid ruin? If you have the distributions of elements in the sequence you can use proportional betting as proposed by Kelly and achieve maximum optional-growth-rate possible. A gambler's bet can be interpreted to be his estimate of the probability distribution of the data. This distribution can be encoded. This method is similar to data compression since both are estimation of the true data distribution. If the estimation is good the wealth of the gambler will grow significantly more (with probability 1) and thus better data compression can be achieved. If the gambling is log optimal the data compression has the Shannon limit H . [13] The relationship between the expected value of the growth rate coefficient $G(f)$ versus the Kelly fraction f from wealth W is shown in figure 3 from Thorp et al. [40]

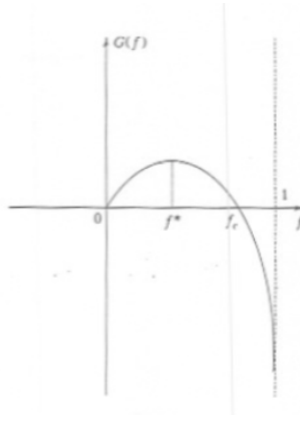


Figure 3: The growth rate ($G(f)$) of the bankroll W .

If we consider the Kolmogorov Complexity and the Chaitin Ω definition of minimum information description on strings we can conclude that strings with simple description are more likely than complex sequences. Therefore we can improve the method of proportional betting proposed by Kelly by using the universal probability of this sequence to decide how much to bet. [27] If this universal distribution is known then the gambler can place bets on the expectation of the next symbol of the string and grow his wealth $W(x)$ by a factor of $2^{length(x)}$. This betting method $bet(x)$ is called *universal gambling* and

$$W(x) = 2^{length(x)} bet(x)$$

The logarithm of the wealth on a random sequence and the complexity of the sequence is no smaller than the length of the sequence. Since 2^l is the most the device can win in betting on a random string, thus this scheme performs asymptotically similarly to any other scheme that has prior knowledge about the sequence distribution, under some confident levels. [13]

The universal probability of a string is defined by its Kolmogorov complexity. This states the probability of observing the symbol 1 or 0 in a sequence after observing the previous elements in the sequence. The conditional probability of finding the next symbol is the ratio of the probability of all sequences with this symbol as this initial segment to the probability of all elements with this prefix.

Sufficient statistics describes the distribution of a random variable by estimating the probability mass function of the observable data. [13] This estimate is very dependent of the observed sample and does not describe the structure of the underlying distribution. In most cases it is assumed that the data follow some fixed set of parametric distribution such as normal-Gauss distribution. [16]

Alternatively the Kolmogorov sufficient statistic provides a two-stage description of the data. In the first stage it describes the distribution function. In stage two we create a Shannon code to describe this distribution using $\log 1/[p(Xn)]$ bits (ie find the model with the shortest description of the input data and the model itself.)

The remainder of this paper is organized as follows: Section 2 presents an summary of the previous works and literature review. Section 3 presents our proposed algorithm to solve the instances of the TSP problem using Information Theory. Section 4 describes the results of the experiments and simulations performed. Section 5 summarizes the concluding remarks founded in our study.

2. Previous Work & Preliminaries

This section contains an overview of the most important heuristics to solve the TSP problem. Other algorithms such as Nearest Neighbor, Greedy, Insertion, Christofides, Lik-Kernighan, Branch & Bound and Ant Colony Optimization were not discussed in this paper. Those methods and its complexity considerations are studied in details by Nilson [4].

2.1. Heuristics for the TSP & Graph Theory

An algorithm that is searching for the smallest distance (cost or weight) between nodes v and u in a graph G must record the information about the intermediary distances to w . This information can be encoded as a label attached to the nodes where its value is defined as the distance between v and w . [6] For n nodes We can use an distance matrix

$$D = (d_{ij})_{n \times n}$$

where each element represents the distance between each city-nodes. Lets Π be the set that contains all possible permutations from node 1 to n . Zhan et al[12] in his work described that the goal of the TSP is to find a permutation π that minimizes the distance between nodes. For symmetric instances the distance between two nodes in the graph is the same in each direction, forming an undirected graph. For asymmetric instances the weights for the edges between nodes can be dynamic or non-existent.

2.1.1. 2-opt, k-opt

Croes [8] proposed the 2-opt algorithm, an simple local-search heuristic, to solve the optimization problem for the TSP. It works by removing two edges from the tour and reconnects the two paths created. The new path is a valid tour since there is only one way to reconnect the paths. [4] The algorithm continues removing and reconnecting until no further improvements can be found. k-opt implementations are instances of 2-opt function but with $k > 2$ and can lead to small improvements in solution quality. However as k increases so does the time to complete execution.

In his work Glover [RR] proposed the Tabu Search method and it can be used to improve the performance of several local-search heuristics such as 2opt. As neighborhood searches algorithms like 2opt can sometimes converge to a local optimum, the Tabu search keeps a list of illegal moves to prevent solutions that provide negative gain to be chosen frequently. In 2opt the two edges removed are inserted in the tabu list. If the same pair of edges are created again by the 2opt move, they are considered tabu. The pair is keep in the list until its pruned or it improves the best tour. [4] However using tabu searches increases computational complexity to $O(n^3)$, as additional computation is required to insert and evaluate the elements in the list.

The Figure 5 show the 2-opt moves from Zunic et. al [38].

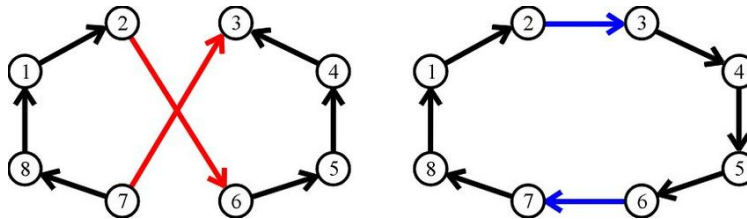


Figure 4: 2-opt moves.

Nilsson [4] compared several heuristic strategies for the TSP problem such as Greedy, Insertion, SA, GA, etc. He investigated the performance trade off between solution quality and computational time. He classify the heuristics in two class: Tour construction algorithms and Tour Improvement algorithms. All algorithms in the first group stops when a solution is found. In the second group, after a solution is found by some heuristics, it tries to improve that solution (up to certain computation and/or time constraints.)* He concluded by showing that the computational time required is proportional to the desired solution quality.

2.1.2. Genetic Algorithm (GA)

Genetic Algorithms was first introduced by Holland [9] based on natural selection theory, as an stochastic optimization method in random searches for good (near-optimal) solutions. This approach is analogous to the "survival of the fittest" principle presented by Darwin. This means that individuals that are fitter to the environment are more likely to survive and pass their genetic information features to the next generation.

In TSP the chromosome that models a solution is represented by a "path" in the graph between cities. GA has three basic operations: Selection, Crossover and Mutation. In the Selection method the candidate individuals are chosen for the production of the next generation by following some fittest function. In the TSP This function can be defined as the length (weight) of the candidate solutions tour.

Next those individuals are chosen to mate (reproduction) to produce the new offspring. Individuals that produce better solutions are more fit and therefore have more chances of having offspring. However individuals that produces worst solutions should not be discarded since they have a probability to improve solution in the future. In other words, the heuristic accepts solutions with negative gain hoping that eventually it may lead to a better solution.

Several researches have studied the performance trade off of selection strategy and how the input parameters affects the quality of solution and the computational time. Julston studied the performance of rank-based and concluded that tournament selection presents better results than rank-based selections.[3] Razali et al. [3] in his work explores different selection strategies to solve the TSP and compare the performances quality and the number of generations required. It concludes that tournament selection is more appropriate for small instance problems and rank-based roulette wheel can be used to solve large size problems.

Goldenberg & Deb [11] compared the quality of the solution and the convergence time on many selection methods such as proportional, tournament and raking. They conclude that ranking and tournament have produced better results than proportional selection, under certain conditions to convergence. [3] In his work Zhong et al. [10] explored proportional roulette wheel and tournament. He concluded tournament selection is more efficient than proportional roulette selection.

The figure 5 contains the pseudo-code for a Genetic Algorithm from Ferentinos et. al [39]

```

generate an initial random population
while iteration <= maxiteration
    iteration = iteration + 1
    calculate the fitness of each individual
    select the individuals according to their fitness
    perform crossover with probability  $p_c$ 
    perform mutation with probability  $p_m$ 
    population = selected individuals after
                    crossover and mutation
end while

```

Figure 5: Basic genetic algorithm.

2.1.3. Simulated Annealing (SA)

Simulated Annealing are heuristics with explicit rules to avoid local minimal. It can be described as a local random search that temporarily accepts moves with negative gain(i.e were produced by solutions with worst quality than current). The probability of accepting a solution is set by a probability function of a temperature parameter variable. As the temperature decreases over time the probability change accordingly. The acceptance probability is defined as $p(x) = 1$ if $f(y) \leq f(x)$ and when otherwise

$$p(x) = e^{[-(f(y)-f(x))/tmp]}$$

where tmp is the input temperature. The SA algorithm specifies the neighborhood structure and the cooling function.

Other heuristics such as 2-opt, 3-opt, inverse, swap methods can be use to generate candidate solutions. Several researches have been made to study the performance of different SA operators to solve the TSP problem. [31][33][34][35] Zhan et. al [12] proposed a list-based SA algorithm using a list-based cooling method to dynamically adjust the temperature decreasing rate. This adaptive approach is more robust to changes in the input parameters.

The quality of the solution can be improved by allowing more time for the algorithm to run. Steiglitz and Weiner observed that the performance can be improved by keeping a list of the closest neighbors for each city-node and thus reducing the amount of solutions to search.

2.2. Information Theory

In his work Shannon was the first to formally define entropy and mutual information[15]. Relative entropy or cross entropy was defined by Kullback and Leibler [16] and is a important measure of distances (or divergence between distributions). The relationship between mutual information and channel capacity was introduced by Shannon. The channel capacity is the upper bound that can be achieved reliably on the rate of transmission over a communication channel. In his work Shannon defined the Continuous entropy and discrete entropy.

2.2.1. Inductive Inference, Compression & Prefix-free Grammar Discovery

Lehmann and Scheffé defined the the concept of minimum sufficient statistics.[30] Kulback introduced the connection between mutual information and sufficiency [18]. This statistics is the most efficient representation that provides all possible information. Shannon[15] has also defined asymptotic equipartition property to a sequence of variables draw by an random Bernoulli-process. This property states the probability of an outcome of a random sequence. This allows the partition of the elements in the set between two sets: The typical set and its complement. McMillan and Breinnan has introduced the proof of this property for the stationary ergodic process in finite alphabet sources [19] [20]. This model for information implies that the random process will not have its proprieties such as mean and standard deviation) changing over time and can be deduced from a sample.

2.2.2. Growth-optimal Strategy: Maximum Transmission Rate & Minimum equivocation

The entropy rate of a stochastic process and the possible sequences generated by this process on a alphabet was introduced by Shannon.[15] Proportional gambling and log-optimal growth rate is defined by Kelly and is know as the Kelly-criterion. [27] He proposed an model in which a gambler can place bets on signals transmitted over a communication channel and concluded that the maximum exponential rate of growth is equal to the rate of transmission. The applications of this allocation strategy in the stock market and black-jack were introduced by Thorp [21]. Shannon and Thorp have worked in one of the first wearable devices used to exploit patterns in the probabilities of outcomes in games like roulette.

2.2.3. Algorithmic Information Theory, Kolmogrov Complexity & Boundaries in Formal Systems

The relationship between relative entropy and information in random sequences was introduced by Kolmogorov. [25] [26] The complexity of an object defines the length of the shortest program that can be processed by a turing machine(ie a computer) and creates this object as output. This interpretation is equivalent to the entropy rate of random variables as introduced by Shannon. Martin-Lof expanded Kolmogorov complexity and defined the idea of algorithmic random sequences and tests for randomness. [28] In his work Levin and Zvonkin[29] defined the complexity for random strings generated by a universal probability. Chaitin explored the relationship between algorithmic complexity and mathematical proofs. [22][23][24]

The van Lambalgen study provides a critical discussion on the interpretation of Chaitin's algorithmic information theory and the relationship between the concept of randomness (in simply defined sequences) and recursive theory.[37] In his work Solomoff investigated the applications of Algorithmic Probability Theory as a method for inference and discuss the completeness, incomputability, diversity and subjectivity of the theory.[36]

2.3. Implementation Methods & Probabilistic Algorithm for the TSP

2.3.1. Algorithm Information Solution to the TSP Problem

Let L be a finite language with n symbols in a set such as $L : \{NODE_A, NODE_B, NODE_C \dots\}n$. Each item is a identification label for the elements in the set under some probability distribution function $pdf(x)$. This is the likelihood of choosing a given symbol x in L . This can be interpreted as an random variable x in L that can assume any item in the set by following the know distribution $pdf(x)$. Each symbol represents a city with a 1-1 mapping to a respective node in a super-graph G^* . Therefore each node is a point with x_n and y_n coordinates in a 2D euclidean space.

For example consider the alphabet L that describes cities encoded by symbols in the set $L : \{SP-BR(SaoPaulo)(x1, y1), NY-US(NewYork)(x2, y2), LD-GB(London)(x3, y3)\}$. Thus each element is a node that have two variables that stores the x and y axis coordinates accordingly.

A valid sequence s^* can be produced by combining elements of L . Permutations with duplicated elements are invalid. S^* is the set with all possible permutations of this sequence. The neighbor of each node at position i in the sequence is the next connected city (i.e item at position $i+1$). This sequence describes a path between nodes and is a tour starting from the first node until the last, and returning to the first node when finished.

Examples of sequences that form valid paths are $path1 = ['SP - BR', 'NY - US', 'LD - GB']$, $path2 = ['SP - BR', 'LD - GB', 'NY - US']$, $path3 = ['LD - GB', 'SP - BR', 'NY - US']$. In the first sequence we start at $SP - BR$ and go to the next node ($NY - US$). From there we go to the final city $LD - GB$ and then go back to the starting node $SP - BR$. This walk in the graph is a tuple $[('SP - BR', 'NY - US'), ('NY - US', 'LD - GB'), ('LD - GB', 'SP - BR')]$.

The cost between the nodes is defined as the 2D euclidean distance $d(x, y)$ between points such that $Node1 := (x1, y1)$ and $Node2 := (x2, y2)$ is calculated as:

$$dist((x1, y1), (x2, y2)) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

The content of Figure 6 is a graph representation of the cities and the corresponding distance matrix that stores the euclidean distance between nodes.

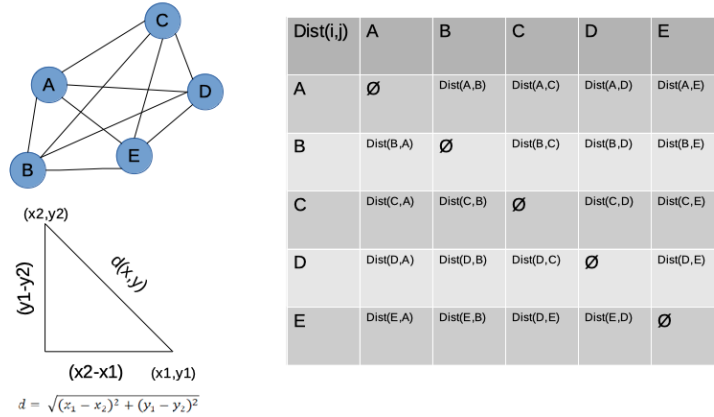


Figure 6: A sample graph for the traveling salesman problem and the corresponding cost matrix.

Let $L : \{'SP - BR', 'NY - US', 'LD - GB'\}$ and consider the coordinate vector $[(1, 5), (100, 80), (30, 40)]$ for route

$$path3 = [('LD - GB', 'SP - BR'), ('SP - BR', 'NY - US'), ('NY - US', 'LD - GB')]$$

The distance between nodes $'LD - GB'$ and $'SP - BR'$ with points

$$(X1, Y1) = (1, 5); (X2, Y2) = (100, 80)$$

is

$$d(LD - GB, SP - BR) = 124.201449$$

The distance between nodes $'SP - BR'$ and $'NY - US'$ where

$$(X1, Y1) = (100, 80); (X2, Y2) = (30, 40)$$

is

$$d(SP - BR, NY - US) = 80.622577$$

. In this sequence, the final walk in the tour is from the last node $'NY - US'$ to the first node $'LD - GB'$ and this distance is $d(NY - US, LD - GB) = 45.453273$.

The total distance of the route is the sum of all intermediary paths between the nodes. For this example the total distance

$$td(path3) = 124.20 + 80.62 + 45.45 = 250.269$$

The distance can be interpreted as the cost (or weight) of the edges between vertex(nodes). Therefore the cost vector for *path3* is [124.20, 80.62, 45.45].

Consider a salesman that needs to visit n cities in L . He wants to optimize the route between nodes by choosing the shortest tour through all cities. Each city is visited only once and the salesman travel back to the city where it has started the tour. We can use a matrix $D(i, j)$ with size $n \times n$ to store the relative distance for all pairs of nodes. The objective is to find the permutation that minimizes the euclidean distances of the tour. Solutions that produce short distances maximizes the perceived value of a given utility function used by the salesman.

In other words, the salesman is trying to interactively optimize his current solution. Therefore he must decide if there is a tour with a cost less than some value v , between all possible solution in the set. This the optimal tour length. As the number of cities increases the number of permutations that need to be sequentially evaluated also increase and thus requires more time to be computed.

The sequences *path1*, *path2*, *path3*, ...*path_n* can be interpreted as elements that are searched when traversing a tree. Each element in this tree is a possible solution. The number of permutations grows exponentially with respect to n . In this example there are $3! = 6$ solutions ($n!$) that needs to be evaluated by the decision rule.

To address the space and time complexities required to compute all elements in the set we can use heuristics such as 2-opt, GA and SA algorithms. This approach assumes some predefined structure to encode the sequences and the decision-rules. 2-opt splits and reconnects the graph, SA generate random sequences that are accepted under some decaying temperature probability and GA mimics biologic systems to optimize the quality of the solution through natural selection. For each method we can infer the probability distribution function for the pair of output and input strings (in a large sample of executions).

In this paper we propose a algorithm information theory (AIT) approach to deal with problems that have such large search space. Consider a function Z defined as a binary function. This function accepts as inputs random sequences created from L .

```
# Boolean function that evaluate candidate solution cost
# against parameter value. Total cost is the sum of all
# intermediary distances w* etween nodes u* and v*,
# returning to u* at end.

fun_Z(solution , val):
    if solution.total_cost < val:
        return 1
    else:
        return 0
```

In Figure 8 we can see the automata representation of function Z implemented as a Turing Machine with two macro operation states: Addition and Comparison. The operators receive two unary integers and writes the sum of them (the Unary addition) or the comparison between the numbers(the Unary Comparison). The unary system repeats a symbol(like 0 or 1) in order to represent an decimal number. As an example the number 4 ca be defined as 0000 with symbol 0 or 1111 with symbol 1.

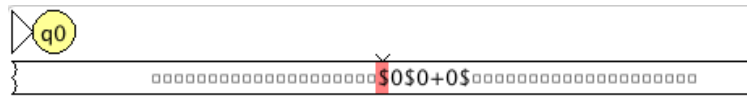


Figure 7: Sample tape accepted as a valid input in the Turing Machine.

For this machine the symbol 0 is used. The tape visualization is demonstrated in FIGURE 7 using the input string $*0\$0+0\#$. The tape has the string with the values to be processed. The symbol $*$ marks the start of the string in the tape. A valid sequence has the format: $*val\$cost0 + cost1\#$. The number val will be compared to determine if its less than the sum of the sequence of intermediary costs $cost_n$. The symbol $\$$ splits the string between the cost numbers to be added (delimited by the symbol $\#$) and the number val . This is used to determine if val is small than the sum of

the sequence (total cost). The machine uses the symbol X to mark the evaluated cells according to the state transitions at each interaction. The blank symbol is defined as \square .

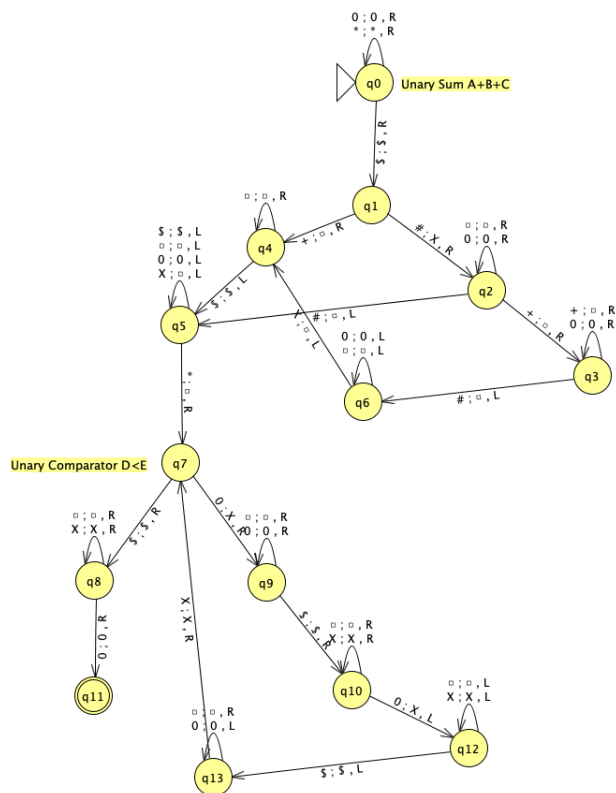


Figure 8: Turing Machine to compare if the sum of the cost (i.e total distance) is greater than then a given parameter value $* val\$ \# cost_{n-1} + cont_n \#$

Examples of valid input strings are:

- * $0\$ \# 0 + 0 \#$
- * $\$ \# 0 + \#$
- * $\$ \# 0 \#$
- * $00000000\$ \# 0 + 000 + 0000 + 0000000 + 0000000000 \#$
- * $00\$ \# 0 + 0 + 00 + \#$

Example of invalid input strings:

- * $000000000000\$ \# 0 + 0 + 00 + 0 \#$
- * $00000\$ \# 0 + 0 + 00 \#$
- * $0\$ \# 0 + \#$
- * $0\$ \# 0 \#$

The matrix representation for the states and transitions is show bellow:

from , to , read , write , move
 $0, 1, \$, \$, R$
 $6, 4, X, , L$
 $7, 8, \$, \$, R$

13,7,X,X,R
 8,11,0,0,R
 1,4,+, ,R
 7,9,0,X,R
 2,5,#, ,L
 3,6,#, ,L
 10,12,0,X,L
 1,2,#,X,R
 5,5,X, ,L
 12,13,\$,\$,L
 5,7,* , ,R
 12,12,X,X,L
 12,12, , ,L
 13,13,0,0,L
 5,5,0,0,L
 5,5, , ,L
 5,5,\$,\$,L
 6,6, , ,L
 6,6,0,0,L
 9,10,\$,\$,R
 0,0,* , ,R
 9,9,0,0,R
 13,13, , ,R
 4,4, , ,R
 3,3,0,0,R
 10,10,X,X,R
 2,2,0,0,R
 9,9, , ,R
 2,2, , ,R
 10,10, , ,R
 8,8,X,X,R
 8,8, , ,R
 0,0,0,0,R
 2,3,+, ,R
 3,3,+, ,R
 4,5,\$,\$,L

Lets assume that the distribution of the output of function Z is unequal. Therefore some sequences are more likely than others and we can use this knowledge to classify each element between two sets (1-group, 0-group). If the input symbol in a communication channel models the likelihood of observing an symbol in a string (or sequence) with consistent probabilities, thus a algorithm can use this information to change the state of the current know solution. At each execution the dependent function Z reveals more bits of information about the underling structure of the input string. In FIGURE 9 is show the relationship between the input and output domains from fun_Z . The function receives an positive number (the cost value) and returns an Boolean value draw according to the occurrence of another positive number val following a probability distribution function $pdf(val)$.

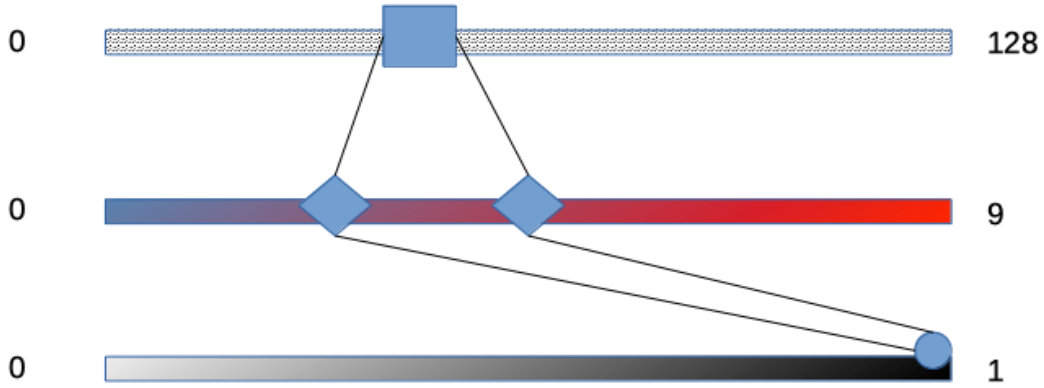


Figure 9: Relationship between input-output domains from the boolean mapping function Z .

Lets consider a computer that can process solutions in parallel at each time unit. The algorithm must evaluate the solution space of candidate solutions and decide which has the best quality. In other words search for the Hamiltonian circuit trough an large set of valid permutations. Although there are lot of complex solutions with longer sequences in set S^* there is a small but significant probability of finding an near-optimal shortest solution with less complexity.

Traditional Heuristics algorithms requires proper encoding to produce results with better quality (less errors) and assumes all outputs sequences have equal probabilities. We know from Shannon the maximum rate of transmission of symbols in a channel can be achieved trough adjusting the error rate by encoding the strings to reduce this probability. Therefore heuristics can be interpreted as a special case of a communication channel in which symbols are transmitted by a source and the decoding algorithm must decide based in a predefined criterion.

From the algorithm perspective the symbols in the sequence can be random variables. However, once a symbol is sent it can not be duplicated and thus the number of possible sequences that can be generated are reduced. We know from Kelly that by tracking the occurrence of symbols in a channel (with a probability distribution function) we can use this information to quantify the optimal amount of resource in the channel to allocate with each symbol. This strategy produces solutions with improved quality without prior-knowledge about the system. We can use this information to dynamically adjust the probability of finding any symbol in the sequence as the algorithm iterate through the solution tree.

We know from Kolmogorov and Chaitin that it may exist a small program that produces any binary sequence. This program must encode all required information for the algorithm to compute and halt, with the least amount of bits possible. An algorithm to solve the TSP can be interpreted as a program that produces the shortest route with the least amounts of bits encoded.

2.4. Overview

If L is a finite-language with symbol $\{c_1, c_2, c_3, \dots, c_n\}$ with a $1 - 1$ map to a city for each symbol. A route is a sequence connecting the cities from left to right. Consider two groups α and β . Let $\alpha = [route_1, route_2, route_3, \dots, route_n]$ be a sample with size n of randomly created valid paths between cities. Each element in this list is a independent candidate solution to the TSP problem. Let $\beta = [route_1, route_2, route_3, \dots, route_m]$ a list of random routes with size m . Thus α and β are samples with random strings created by the same random ergotic process.

Consider a algorithm that process both groups and decide which one has the near-optimal solution on average with the least amount of bits possible to represent this information. Suppose this is a universal algorithm that reads symbols from a channel, measures the data and writes a index about the pair of input-output symbols, using a tuple t . At each interaction it classify the data. There are 3 classification steps: *Statistical Significance sig **, *Error Rate err ** and *Information Divergence div **. We define

$$t = (sig *, err *, div *)$$

as the label for the class. The α group is our model sample in which we want to test against an alternative dynamic sample β . This alternative group is created randomly at each interaction of the algorithm, while α is keep static. The number of interactions or trials performed by this algorithm is a positive parameter max_inter .

2.4.1. Statistical Significance: sig*

In Statistical Significance we need to measure how much different are the samples. We can use a test statistic such as the chi square statistic to quantify this. Hypothesis test methods provides an signal-to-noise procedure to encode/decode the data sample, when the null hypothesis is true. Its a function of observed and expected counts constructed with a Contingency Table. This determines if there is significant difference between observed and expected frequencies.

Let $vet_cost_\alpha = [cost1, cost2, cost3, ..., cost_n]$ be a list where each element is the total cost for each candidate solution in the sample. The cost is defined as a random variable Y , dependent on another variable X , with a probability density function $pdf(y)$. It is measured as sum of all intermediary distances in a given path. Let vet_cost_β be the equivalent to group β .

Let $E(\alpha)$ be the expected value(or average) of group The max_α is the maximum value in the cost vector and min_α is the minimum value for the α sample. The same variables are available for group β .

We can use a contingency table $CT_{\alpha:E(\alpha)}$ to count the frequencies of elements in α that are greater or equal(Category 0) and small(Category 1) than the $E(\alpha)$. Let $CT_{\beta:E(\alpha)}$ be the frequencies of elements in group β that are in category 0 and in category 1 also for $E(\alpha)$.

We can combine $CT_{\alpha:E(\alpha)}$ and $CT_{\beta:E(\alpha)}$ to create a matrix $CT_{\alpha,\beta:E(\alpha)}$ to tabulate the frequency distribution between groups and categories against $E(\alpha)$.

Table 1

Contingency Table from sets α and β against $E(\alpha)$

group\category	greater or equal than $E(\alpha)$: 0_ $E(\alpha)$	small than $E(\alpha)$: 1_ $E(\alpha)$
$CT_{\alpha,E(\alpha)}$	count itens with cost $\geq E(\alpha)$ in α	count itens with cost $< E(\alpha)$ in α
$CT_{\beta,E(\alpha)}$	count itens with cost $\geq E(\alpha)$ in β	count itens with cost $< E(\alpha)$ in β

The chi square test for $CT_{\alpha,\beta:E(\alpha)}$ assumes the hypothesis H_0, H_1 :

H_0 (null hypothesis): There is no significant difference in the number of candidates solutions with cost less than $E(\alpha)$ between the group α, β . Both groups α, β are independent.

H_1 (alternative hypothesis): There is significant difference in the sample observations of costs between α and β for $E(\alpha)$.

If p-value p^* from the chi-square test to the $CT_{\alpha,\beta:E(\alpha)}$ is less than the significance level (0.05) we can reject the null hypothesis. Therefore we can conclude there is statistical significant difference in the costs between the two groups.

Consider an alphabet L^* with symbols 0,1. Let 1 bet the number of items with $cost < E(\alpha)$ in α and β . Similarly 0 is the number of items with $cost \geq E(\alpha)$ in α and β If for example $vector_cost_\alpha = [100, 30, 50, 10, 40, 80, 99, 10]$ and $vector_cost_\beta = [80, 35, 65, 30, 50, 90, 13, 80]$ and $E(\alpha) = 52, E(\beta) = 55$ thus $CT_{\alpha,\beta:E(\alpha)}$ is

Table 2

Contingency Table $CT_{\alpha,\beta:E(\alpha)}$

group\category $E(\alpha)$	0	1
$CT_{\alpha,E(\alpha)}$	3	5
$CT_{\beta,E(\alpha)}$	4	4

The chi-square statistic is 0.254. The p-value is .614295. The result is not significant at $p < .05$. Therefore we fail to reject the null hypothesis. Alternatively we could also calculate the p-value from chi-square test for $CT_{\alpha:E(\alpha)}$ and $CT_{\beta:E(\alpha)}$.

2.4.2. Error Rate: err^*

Consider a channel in which an algorithm receives encoded signals from a source. Each symbol in this channel has a probability of occurring following a probability distribution. The error rate of a channel is the rate of equivocation or misinformation when transmitting symbols through a noisy communication channel. By adjusting the code to adapt to this rate we can transmit information regardless of the error.

The Kelly criterion is a strategy to maximize the expectation of the logarithm of wealth (using fractional betting sizes), instead of the expected gain for each trial. In the long run this strategy produces better returns than any other strategy. This method is used by gambling and investing to maximize the expected geometric growth rate. We will use this formula to discover the optimal channel allocation capacity for a given error rate. We use this value to measure the likelihood of finding elements in groups α and β that have candidate solutions that improve costs.

If $vector_cost_\alpha = [100, 30, 50, 10, 40, 80, 99, 10]$ and $vector_cost_\beta = [80, 35, 65, 30, 50, 90, 13, 80]$ and $E(\alpha) = 52$, $E(\beta) = 55$. Then $\Delta_vector_cost_{\alpha, E(\alpha)} = [(52 - 100), (52 - 30), (52 - 50), (52 - 10), (52 - 40), (52 - 80), (52 - 99), (52 - 10)] = [-48, 22, 2, 42, 12, -28, -47, 42]$. Similarly $\Delta_vector_cost_{\beta, E(\alpha)} = [(52 - 80), (52 - 35), (52 - 65), (52 - 30), (52 - 50), (52 - 90), (52 - 13), (52 - 80)] = [-28, 17, -13, 22, 2, -38, 39, -28]$

The positive variables in the cost vector can be interpreted as the quantity of improvement, or gain for each candidate solution. This measure is the decrease in cost from the expected value $E(x)$ for each element in the list. Respectively the negative variables are the units of decline in quality and is measured as a increase in observed cost from expected value $E(x)$.

Considering previous example, thus $+\Delta_vector_cost_{\alpha, E(\alpha)} = [22, 2, 42, 12, 42]$, $-\Delta_vector_cost_{\alpha, E(\alpha)} = [-48, -28, -47]$, $+\Delta_vector_cost_{\beta, E(\alpha)} = [17, 22, 2, 39]$, $-\Delta_vector_cost_{\beta, E(\alpha)} = [-28, -13, -38, -28]$.

The algorithm can use this information to decide how favorable are the routes in groups α and β . If f_{kelly} is negative than is very unlikely β has elements with costs that are less than $E(\alpha)$. The risk of "losing" is too high for this iteration. However $f_{kelly} > 0$ there are elements in α or β that reduces costs significantly and thus optimize the current solution quality. Similarly a gambler use probabilities to decide the optimal amount of his wealth to bet at each trial based on the symbols he receive from a channel.

From $CT_{\alpha, \beta: E(\alpha)}$ we know there are $(5 + 4) = 9$ items smaller than $E(x)$ and $(3 + 4) = 7$ items whose cost is higher than $E(x)$. There are in total 16 candidate solutions from samples α, β . Therefore class 1 has $9/16 = 0.5625$ or 56% of the elements and class 0 has $(7/16) = 0.43$ or 43%.

Let the average improvement gain be defined as

$$avg_+ + \Delta_cost_{\alpha, \beta: E(\alpha)} = [(avg_+ + \Delta_cost_{\alpha, E(\alpha)} + avg_+ + \Delta_cost_{\beta, E(\alpha)})/2]$$

and the average loss

$$avg_ - \Delta_cost_{\alpha, \beta: E(\alpha)} = [(avg_ - \Delta_cost_{\alpha, E(\alpha)} + avg_ - \Delta_cost_{\beta, E(\alpha)})/2]$$

From previous example we have:

$$avg_+ + \Delta_cost_{\alpha, E(\alpha)} = (22 + 2 + 42 + 12 + 42)/5 = 24$$

$$avg_+ + \Delta_cost_{\beta, E(\alpha)} = (17 + 22 + 2 + 39)/4 = 20$$

$$avg_ - \Delta_cost_{\alpha, E(\alpha)} = (-48 - 28 - 47)/3 = -41$$

$$avg_ - \Delta_cost_{\beta, E(\alpha)} = (-28 - 13 - 38 - 28)/4 = -26.75$$

Then the modified $CT_{\alpha, \beta: E(\alpha)}$ relative to the expected costs improvement and worsening.

Let the average for both groups be $avg_+ + \Delta_cost_{\alpha, \beta: E(\alpha)}$ (average gain) and $avg_ - \Delta_cost_{\alpha, \beta: E(\alpha)}$ (average loss).

$$avg_+ + \Delta_cost_{\alpha, \beta: E(\alpha)} = (24 + 20)/2 = 22$$

$$avg_ - \Delta_cost_{\alpha, \beta: E(\alpha)} = (-41 - 26.75)/2 = -33.87$$

Table 3Contingency Table $CT_{\alpha,\beta:E(\alpha)}$ for the expected cost

group\category $E(\alpha) = 52$	0	1
$CT_{\alpha,E(\alpha)}$	(-41)	(+24)
$CT_{\beta,E(\alpha)}$	(-26.75)	(+20)

From $CT_{\alpha,\beta:E(\alpha)}$ we have $Prob_1_{\alpha,\beta:E(\alpha)} = 0.56$, $Prob_0_{\alpha,\beta:E(\alpha)} = 0.44$ and let the ratio of improvement (ie average gain of positive solution by average loss of the negative solution) be

$$1_0_Ratio = avg_+ \Delta_cost_{\alpha,\beta:E(\alpha)} / |avg_ - \Delta_cost_{\alpha,\beta:E(\alpha)}|$$

thus for this example $1_0_Ratio = (22/33.87) = 0.64$. The Kelly formula with these parameter is:

$$f_kelly_{\alpha,\beta:E(\alpha)} = Prob_1_{\alpha,\beta:E(\alpha)} - [(1 - Prob_1_{\alpha,\beta:E(\alpha)}) / 1_0_Ratio_{\alpha,\beta:E(\alpha)}]$$

$$f_kelly_{\alpha,\beta:E(\alpha)} = 0.56 - (0.44/0.64) = (0.56 - 0.68) = -0.12$$

The Kelly fraction is $f_kelly = -0.12$ and since $f_kelly < 0$ the error rate is too high on the channel. Alternatively instead of calculating from the union of set α and β we could also calculate the fraction to each set separately. Therefore from the $CT_{\alpha,\beta:E(\alpha)}$ we have:

$$avg_+ \Delta_cost_{\alpha,E(\alpha)} = 24$$

$$avg_+ \Delta_cost_{\beta,E(\alpha)} = 20$$

$$avg_ - \Delta_cost_{\alpha,E(\alpha)} = -41$$

$$avg_ - \Delta_cost_{\beta,E(\alpha)} = -26.75$$

$$Prob_0_{\alpha,E(\alpha)} = 3/8 = 0.375$$

$$Prob_1_{\alpha,E(\alpha)} = 5/8 = 0.625$$

$$Prob_0_{\beta,E(\alpha)} = 4/8 = 0.5$$

$$Prob_1_{\beta,E(\alpha)} = 4/8 = 0.5$$

$$1_0_Ratio_{\alpha,E(\alpha)} = avg_+ \Delta_cost_{\alpha,E(\alpha)} / |avg_ - \Delta_cost_{\alpha,E(\alpha)}| = 24 / |-41| = 0.58$$

$$1_0_Ratio_{\beta,E(\alpha)} = avg_+ \Delta_cost_{\beta,E(\alpha)} / |avg_ - \Delta_cost_{\beta,E(\alpha)}| = 20 / |-26.75| = 0.747$$

$$f_kelly_{\alpha,E(\alpha)} = Prob_1_{\alpha,E(\alpha)} - [(1 - Prob_1_{\alpha,E(\alpha)}) / 1_0_Ratio_{\alpha,E(\alpha)}]$$

$$f_kelly_{\alpha,E(\alpha)} = 0.625 - (0.375/0.58) = -0.021$$

$$f_kelly_{\beta,E(\alpha)} = Prob_1_{\beta,E(\alpha)} - [(1 - Prob_1_{\beta,E(\alpha)}) / 1_0_Ratio_{\beta,E(\alpha)}]$$

$$f_kelly_{\beta,E(\alpha)} = 0.5 - (0.5/0.747) = -0.169/]$$

2.4.3. Information Divergence(div*)

Each set have an amount of information intrinsic to itself and this can be measure in bits using the entropy formula. We can also measure the amount information that is similar between distributions. This allow us to quantify the amount of information relative to α from β and to β from α . It can be interpreted as the amount of information lost by assuming an alternative distribution to represent the true distribution.

Let the language L *: 0, 1 with two probabilities distributions functions $pdf(x)$ for sets α and β . This functions describes the frequencies of symbols 0 and 1 in each sample. This probabilities are defined by the occurrence of candidate solutions in either set that are higher or smaller than the expected value $E(\alpha)$, measured from our model sample α . From the entropy formula by Shannon we can measure the amount of bits required to represent the symbols in L * for both distributions $pdf(\alpha)_{E(\alpha)}$ and $pdf(\beta)_{E(\alpha)}$.

From the example we have $CT_{\alpha,\beta:E(\alpha)}$ and therefore:

$$Prob_0_{\alpha,E(\alpha)} = 0.375$$

$$Prob_1_{\alpha,E(\alpha)} = 0.625$$

$$Prob_0_{\beta,E(\alpha)} = 0.5$$

$$Prob_1_{\beta,E(\alpha)} = 0.5$$

The entropy for the α distribution relative $E(\alpha)$ is $H(\alpha)_{E(\alpha)}$:

$$H(\alpha)_{E(\alpha)} = (0.375 * \log_2(0.375)) + (0.625 * \log_2(0.625)) = -(0.375 * -1.4150374992788437) + -(0.625 * -0.678071905112)$$

$$H(\alpha)_{E(\alpha)} = 0.954$$

The entropy for the β distribution relative $E(\alpha)$ is $H(\beta)_{E(\alpha)}$:

$$H(\beta)_{E(\alpha)} = (0.5 * \log_2(0.5)) + (0.5 * \log_2(0.5)) = -(0.5 * -1) + -(0.5 * -1)$$

$$H(\beta)_{E(\alpha)} = 1$$

Thus the we need 1 full bit to represent the distribution $H(\alpha)_{E(\alpha)}$ and exactly 1 bit to $H(\beta)_{E(\alpha)}$.

To compare the distributions we can measure the divergence between the entropy for two or more probability distribution. We can use the *Jensen-Shannon divergence* JS_div to quantify the similarity between distributions. This method is based in the *Kullback-Leibler* divergence (or relative entropy). If L *: 0, 1, $pdf(\alpha)_{E(\alpha)} = [0.375, 0.625]$, $pdf(\beta)_{E(\alpha)} = [0.5, 0.5]$ than $JS_div = 0.0114$. Thus the difference between the distributions requires less than 1 bit to be described and is close to zero.

This is the expectation of the logarithmic difference between the probabilities $pdf(\alpha)_{E(\alpha)}$ and $pdf(\beta)_{E(\alpha)}$ for sequences produced by a binary function with alphabet L *: 0, 1. This divergence is the mutual information between a random variable X related to the probability distributions from α and β and an binary function classifier used to map elements between α and β .

2.5. Algorithm & Data Structure

In FIGURE 10 is shown the flowchart for the algorithm proposed in this paper.

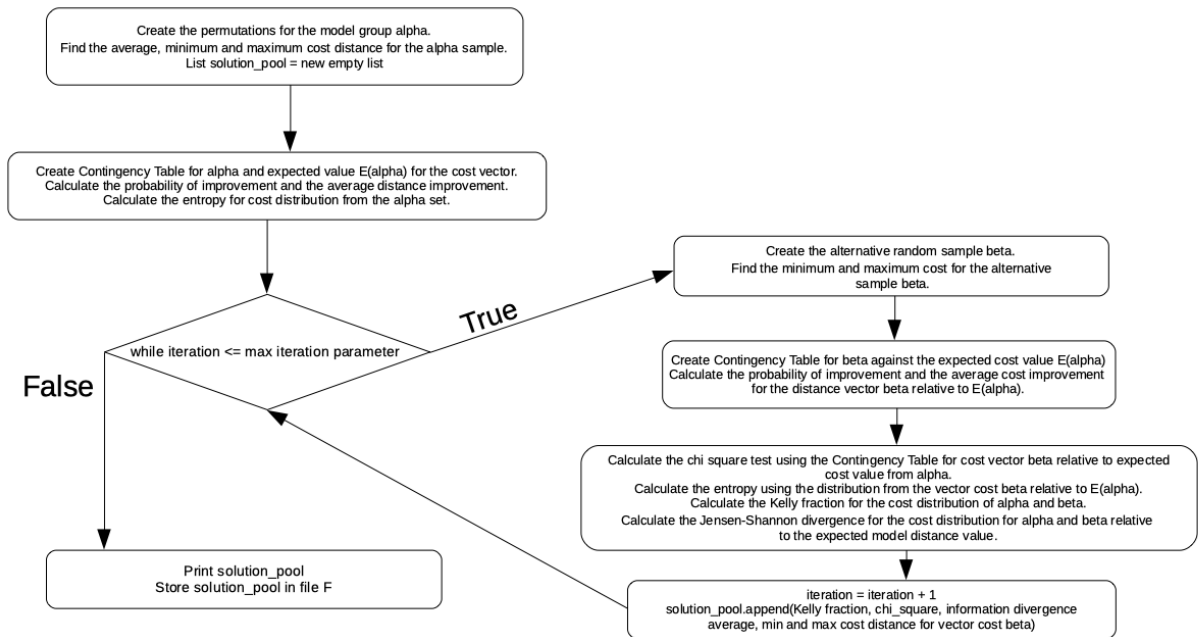


Figure 10: Flowchart of the algorithmic-information method for the salesman problem.

The pseudo-code for the method described is bellow:

```

# create a valid random permutation of the input string – no duplicates.
# output: string, the sequence is a path between nodes v ad w in the graph g*.
func create_random_solution_sample_list(L, const_sample_size).

    sample_set = []
    for i in 0 -> const_sample_size-1:
        sample_set.append(createValidRandomPermutation(L))

    return sample_set

# language L with city-node symbols.
L: {c1(x1,y1), c2(x2,y2), c3(x3,y3), c4(x4,y4)...}

# language L* with binary symbols produced by a binary function classifier
# whose input is a random senqeence produced using L.
L*: {0,1}

# number of elements in each sample
const_sample_size = m
# number of trials
const_max_inter = mi*

random_model_sequence <- createValidRandomPermutation(L)
model_vector_cost <- get_total_distance_for_routes([random_model_sequence])

# alpha: model set.
# beta: alternative set.
alpha <- create_random_solution_sample_list(random_model_sequence, const_sample_size)
  
```

```

vector_cost_alpha <- get_total_distance_for_routes(alpha)

# calculate expected cost in the set
expected_value_alpha <- get_average(vector_cost_alpha)

# get the solution with the highest cost relative to expected model cost
max_solution_cost_alpha = max(vector_cost_alpha)
# get the solution with the minimum cost relative to the expected model cost.
min_solution_cost_alpha = min(vector_cost_alpha)

# calculate delta difference between the cost of each candidate solution
# to the expected model cost.
Delta_cost_vector_alpha <- get_difference_cost(vector_cost_alpha, \
    expected_value_alpha)

# get the average improvement cost.
+Delta_vector_cost_alpha <- get_positive_elem(Delta_cost_vector_alpha)

# get the average worsening on cost relative to expected cost.
-Delta_vector_cost_alpha <- get_negative_elem(Delta_cost_vector_alpha)

avg_+Delta_cost_alpha <- calc_average(+Delta_vector_cost_alpha)
avg_-Delta_cost_alpha <- calc_average(-Delta_vector_cost_alpha)

# create the contingency table of occurrence of candidate solutions
# relative to expected cost.
CT_alpha <- [count_negative_elem(Delta_cost_vector_alpha), \
    count_positive_elem(Delta_cost_vector_alpha)]

# get the likelihood of category 1 (candidate cost less than expected cost)
Prob_1_alpha <- get_probability_class_1(CT_alpha)
# get the likelihood of category 0 (candidate cost greater or equal than expected cost)
Prob_0_alpha <- get_probability_class_0(CT_alpha)

# create the probability distribution function to the sample.
pdf_alpha <- [Prob_1_alpha, Prob_0_alpha]
# calculate the number of bits required to represent the sample distribution.
H_alpha <- calc_entropy(pdf_alpha)

# track solution output.
label_list = []

for i from 0 -> const_max_inter:

    # create random sample and calculate the distances between cities(vector cost)
    beta <- create_random_solution_sample_list(random_model_sequence, \
        const_sample_size)
    vector_cost_beta <- get_total_distance_for_routes(beta)

    # calculate the difference between distance from candidate solutions
    # in the set with the expected value of the model set.
    Delta_cost_vector_beta <- get_difference_cost(vector_cost_beta, \

```



```

        expected_value_alpha)

max_solution_cost_beta = max(vector_cost_beta)
min_solution_cost_beta = min(vector_cost_beta)

# get the candidate solutions that improve solution quality
# (reduce expected solution cost).
+Delta_vector_cost_beta <- get_positive_elem(Delta_cost_vector_beta)
# get solutions that are worst than expected solution cost.
-Delta_vector_cost_beta <- get_negative_elem(Delta_cost_vector_beta)

# get the average improvement relative to the expected cost value.
avg_+Delta_cost_beta <- calc_average(+Delta_vector_cost_beta)

# get the average worsening relative to the expected cost value.
avg_-Delta_cost_beta <- calc_average(-Delta_vector_cost_beta)

# contingency table for beta
CT_beta <- [count_negative_elem(Delta_cost_vector_beta), \
            count_positive_elem(Delta_cost_vector_beta)]

# calculate entropy for distribution beta
Prob_1_beta <- get_probability_class_1(CT_beta)
Prob_0_beta <- get_probability_class_0(CT_beta)

# probability distribution function of finding candidate solutions with cost less
# than expected cost (category 1) or greater than expected cost (category 0).
pdf_beta <- [Prob_1_beta, Prob_0_beta]

# the number of bits required to represent (encode) the set is defined as entropy
# binary function H(beta).
H_beta <- calc_entropy(pdf_beta)

# chi-square statistic between alpha and beta
CT_alpha_beta = [CT_alpha, CT_beta]

# Is significant difference between the cost distributions in the samples sets?
p_value_chi2 = chi2_test(CT_alpha_beta)

# Kelly-criterion, error rate

# average improvement gain from both sets - beta (alternative)
# set and model set(alpha).
avg_+Delta_cost_alpha_beta = (avg_+Delta_cost_alpha + avg_+Delta_cost_beta)/2
avg_-Delta_cost_alpha_beta = (avg_-Delta_cost_alpha + avg_-Delta_cost_beta)/2

# probability of choosing a candidate solution that improves current expected cost
Prob_1_alpha_beta = get_probability_class_1(CT_alpha_beta)

# probability of choosing a candidate solution at random that is worse the cost
# quality relative to the expected cost.
Prob_0_alpha_beta = get_probability_class_0(CT_alpha_beta)

```

```

# the ratio of improvement between category 1 and category 0.
l_0_Ratio_alpha , beta <- avg_+Delta_cost_alpha , beta / |avg_-Delta_cost_alpha , beta|

# get kelly fraction proportional to the channel capacity .
f_kelly_alpha , beta <- Prob_1_alpha , beta - [(1-Prob_1_alpha , beta)/l_0_Ratio_alpha , beta]

# information divergence between distributions .
js_div_alpha , beta <- calc_jensen_shannon_divergence(L*,pdf_alpha , pdf_beta)

# create label
t <- (p_value_chi2 , f_kelly_alpha , beta , js_div_alpha , beta)
costs <- (min_cost_alpha , min_cost_beta , max_cost_alpha , max_cost_beta)

label_list[i] <- (t , costs)

# interpreted the tuple by some criterion and decide if halt or not .
# constrains are defined by program parameterers .

# statistical significant: p-value < 0.005
# likelihood/risk of improving current cost: f_kelly > 1
# number of bits to represent the distance between sets: js_div > 0.1

if decide_to_halt(label_list[i]) == true:
    break

return label_list

```

The time complexity for this algorithm is defined as the time required to:

Create the permutations for the model group α : $O(n)$ complexity for a list with n elements.
Find the average, minimum and maximum cost distance for the sample: $O(n)$
Contingency Table: Count the occurrence of elements small or greater than the average from the model sample α : $O(n)$
Calculate the probability of improvement in the distance and the average gain or loss for the α group: $O(2)$
Calculate the entropy for the distribution from α : $O(1)$

Iterate the algorithm an finite number of times m^* and at each trial: $O(m^*)$

Create the alternative sample β : $O(n)$
Find the minimum and maximum cost for the alternative sample β : $O(n)$
Contingency Table: Count the occurrence in β of elements small or greater than the expected cost from the model sample α : $O(n)$
Calculate the probability of improvement in the distance and the average gain or loss for the β group: $O(2)$

Calculate the Kelly fraction from α and β cost distributions: $O(1)$
Calculate the chi square test for the Contingency Table for β relative to expected cost value from α Calculate the divergence between distance distributions α and β : $O(1)$
Store solution state in list: $O(1)$

The upper bound is

$$O((3n + c1) + m * (3n + c2)) = O(n + m * n)$$

The space complexity is defined as:

List to store the sequences from the model group α : $O(n)$
 Variables to store the average, minimum and max values from α : $O(3)$
 Matrix to store the Contingency Table for the set: $O(4)$
 Variable to store the average gain or loss for the set: $O(2)$
 Variable to store the entropy for the distribution, represented by the Contingency Table frequencies: $O(1)$

List to store the sequences from the alternative model β : $O(n)$
 Store average, minimum and maximum values from β relative to expected value for α : $O(3)$
 Matrix to count frequencies for set β relative to α : $O(4)$
 Variable to store average gain or loss from solutions in set β relative to α : $O(2)$
 Variable to store the entropy for the distribution frequencies for β relative to α : $O(1)$
 Variable to store the similarity(or divergence) between the distributions α and β : $O(1)$

Additional list to store the classification tuple $t=(sig^*, err^*, div^*)$: $O(m^*)$

The upper bound is

$$O((n + c1) + (n + c2) + m) = O(n + m)$$

3. Computational Experiments & Results

3.1. Experimental Setup

In this section we will focus on the computational experiment carried to observe and analyses the strategy of using algorithmic information theory to detect patterns in strings and use this schema to obtain near-optimal solutions to the TSP problem. The algorithms are implemented in Python 3 with libraries dit, matplotlib and numpy. The performance of the method is tested against 3 instances of the problem with randomly chosen cities with 10, 30 and 50 nodes. At each interaction the program create a new alternative hypothesis set, compares it against the model distribution and classify the sample with a label tuple $t=(sig^*, err^*, div^*)$. This label is later used to sort and classify the sample distributions.

The node coordinates in the euclidean space are the input symbols used to generate paths to the salesman. This tour is encoded as a sequence of nodes. A valid permutation is a tour defined as a list with no duplicate elements. The solution space is the set with all valid permutation or candidate solutions.

```
DIMENSION: 50
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 288 149
2 288 129
3 270 133
4 256 141
5 256 157
6 246 157
7 236 169
8 228 169
9 228 161
10 220 169
11 212 169
12 204 169
13 196 169
14 188 169
15 196 161
16 188 145
17 172 145
```

18 164 145
 19 156 145
 20 148 145
 21 140 145
 22 148 169
 23 164 169
 24 172 169
 25 156 169
 26 140 169
 27 132 169
 28 124 169
 29 116 161
 30 104 153
 31 104 161
 32 104 169
 33 90 165
 34 80 157
 35 64 157
 36 64 165
 37 56 169
 38 56 161
 39 56 153
 40 56 145
 41 56 137
 42 56 129
 43 56 121
 44 40 121
 45 40 129
 46 40 137
 47 40 145
 48 40 153
 49 40 161
 50 40 169

The maximum number of iteration is defined as a input parameter $m \gg 0$. As the number of nodes in the sample increases the time and space required to evaluate and create valid permutations also increases. The number of permutations for each sample is a positive parameter $sample_size = 300$, with a total of 600 elements in total. The algorithm terminates when the maximum number of execution is reached. For this paper we have used an iteration time of 300 for all experiments.

The objective of the method is to measure the amount of information in each sample and the mutual information between groups. At each step the candidate solutions are evaluated against the expected value cost relative to the model distribution and this count is used to decide if the samples means are significantly difference using the chi-square test ($p < 0.05$). We also measure the amount of bits required to represent each sample using the entropy function and the similarity between samples is measured with the Johasen-Shannon Divergence. In this paper we are measuring the quantity of bits required to represent this knowledge. Thus the entropy and the divergence must be positive integers. The error rate for the sample distributions are measured as the likelihood of improvement in a communication channel with the Kelly criterion. In this experiment we have set the threshold for the fraction as $f_kelly > 0$.

At the end of each interaction the tuple $t=(sig^*, err^*, div^*)$, the minimum and maximum values for the route costs are inserted in a buffer list. This list is than used to search for solutions distributions that have chi-square test with $p - value < 0.05$ and $f_kelly > 0$.

After the trials are finished (i.e. maximum number of iteration is reached) We sort the results in descending order by using the number of divergent bits between the sample distributions.

3.2. Experimental Results

The algorithm results are filtered by $t=(sig^*, err^*, div^*)$ with condition ($sig^* < 0.005$ and $err^* > 0$) in descending order of div^* for three instance of the TSP with 10, 30 and 50 nodes. The performance of the best results from the three instances are shown in TABLE [TPP1], TABLE [TPP2] and TABLE [TPP3] respectively.

Table 4

TABLE [TPP1] 10-city a):

inter	a_min_cost	b_min_cost	a_max_cost	b_max_cost	a_avg_cost	b_avg_cost
293	199.8452	226.4190	464.2929	474.5521	358.1339	346.9520

Table 5

TABLE [TPP1] 10-city b):

inter	a_b_chi2_CT	b_pval	a_pval	a_b_chi2_pval
293	[[148, 152], [123, 177]]	0.001822	0.81736	0.04028

Table 6

TABLE [TPP1] 10-city c):

inter	a_kelly_k	b_kelly_k	a_b_kelly_k
293	2.220446049250313e-16	0.24496	0.12800

Table 7

TABLE [TPP1] 10-city d):

inter	a_b_info_js_div	a_info_ent	b_info_ent
293	0.00506	0.99987	0.97650

Table 8

TABLE [TPP2] 30-city a):

inter	a_min_cost	b_min_cost	a_max_cost	b_max_cost	a_avg_cost	b_avg_cost
72	1409.12405	1308.8004	2414.9828	2336.10197	1905.01199	1893.9638
196	1409.124054	1369.2457	2414.9828	2300.17248	1905.01199	1889.11683
184	1409.12405	1465.18982	2414.9828	2494.663	1905.01199	1896.68915

For all instances of the problem we can see from the tables that the expected cost for β is less than the average of α . Additionally the number of solutions in α with smaller costs than $E(\alpha)$ is less than the number of solutions in β with expected value smaller than $E(\alpha)$. This can be observed in the Contingency Tables $a_b_chi2_CT$ from TABLE [TPP1], TABLE [TPP2] and TABLE [TPP3]. If we compare the $p - value$ of the chi-square test we can see that in all cases there is a significance difference between the sample means.

The rate of improvement between distributions is measured by the Kelly fraction. We can compare the performance using TABLE [TPP1], TABLE [TPP2] and TABLE [TPP3]. We can see from the tables that this rate is small and close to zero. If the number of nodes increases the error rate in the channel also increases. Thus as the number of

Table 9

TABLE [TPP2] 30-city b):

inter	a_b_chi2_CT	b_pval	a_pval	a_b_chi2_pval
72	[[159, 141], [133, 167]]	0.049647	0.298697	0.033699
196	[[159, 141], [133, 167]]	0.049647	0.298697	0.033699
184	[[159, 141], [134, 166]]	0.064671	0.298697	0.0411716

Table 10

TABLE [TPP2] 30-city c):

inter	a_kelly_k	b_kelly_k	a_b_kelly_k
72	2.220446049250313e-16	0.0791686	0.03744395
196	2.220446049250313e-16	0.11312	0.055189232
184	2.220446049250313e-16	0.05989	0.0278014806

Table 11

TABLE [TPP2] 30-city d):

inter	a_b_info_js_div	a_info_ent	b_info_ent
72	0.0054288	0.9974015	0.990714
196	0.0054288	0.9974015	0.990714
184	0.00501791	0.9974015	0.991777

Table 12

TABLE [TPP3] 50-city a):

inter	a_min_cost	b_min_cost	a_max_cost	b_max_cost	a_avg_cost	b_avg_cost
210	3687.880596704705	3745.695347	5749.646374	5642.41536	4666.26576	4604.589729
194	3687.880596704705	3332.52242	5749.646374	5590.93659	4666.26576	4630.9972754

Table 13

TABLE [TPP3] 50-city b):

inter	a_b_chi2_CT	b_pval	a_pval	a_b_chi2_pval
210	[[156, 144], [131, 169]]	0.02824036	0.4884223	0.04103653
194	[[156, 144], [132, 168]]	0.03766692	0.48842231	0.04986020

Table 14

TABLE [TPP3] 50-city c):

inter	a_kelly_k	b_kelly_k	a_b_kelly_k
210	1.1102230246251565e-16	0.1936353	0.09844197
194	1.1102230246251565e-16	0.1129841	0.05482081

candidate solutions increases than the risk of randomly choosing a element with a worst quality tour also increases. This value can be interpreted as a context for the chi square test. For example from TABLE [TPP2] 30-city a) we can see that the p-value at iterations 72 and 196 are the same at 0.033699 (since they have the same Contingency Table).

Table 15

TABLE [TPP3] 50-city d):

inter	a_b_info_js_div	a_info_ent	b_info_ent
210	0.00502466	0.99884553	0.98839523
194	0.00462900	0.99884553	0.98958752

However the Kelly fraction for iteration 196 is better than at 72 and thus the misinterpretation risk is smaller at 196 with 0.055189232.

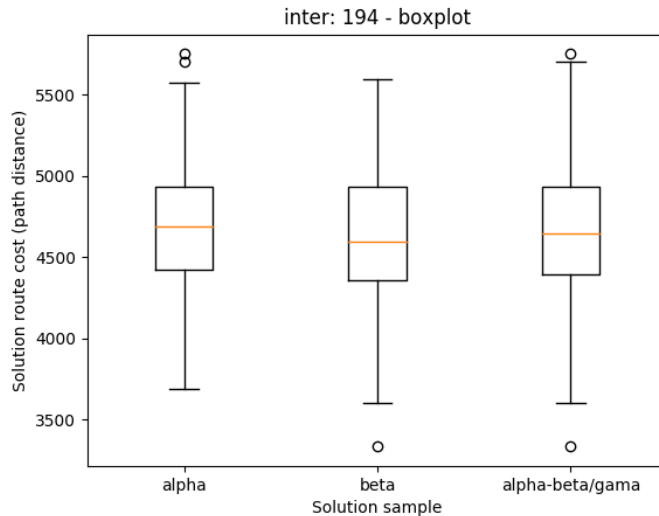
From the TABLE [TPP1] we can see that the average gain provided by the alternative distribution is greater for the 10-city instance. From the results we see that although distribution β can have in average more elements with expected cost less than α , it's still possible that the minimum tour length be in α . As shown in TABLE [TPP2] 30-city b) at iteration 184 the minimum cost at α is 1409.124 and at β is 1465.189. However the average cost for α is 1905.011 against 1896.689 from β .

As can be seen in the tables the similarity between the distributions is less than 0.005 bits of information. The entropy for the distributions α and β for each instance is approximately 1 bit per iteration.

In FIGURE 11a, 11b, 11c, 11d we have the graph for the maximum and minimum cost for iteration 194 from instance with 50 cities.

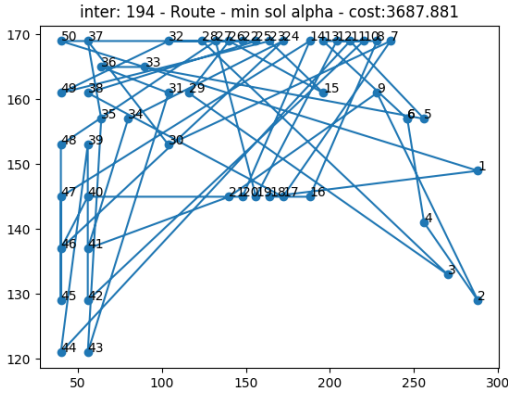
The graph shows the bounds in performance in each sample. The minimum costs strings have less complexity than the maximum cost sequences because they have less crossing edges between vertex in the 2d graph. The distance in the graphs reduce towards a near-optimal solution as the algorithm iterate. In FIGURE 12a, 12b, 12c we can see the histogram with 5 bins for distributions α, β and $\alpha \cup \beta$, for iteration 194 from 50-city instance. This measures the frequencies ranges of the cost samples. The data follow a normal distribution.

The FIGURE 13 show the distribution of data for samples α, β and the union of $\alpha \cup \beta$ using a boxplot. The graph show the maximum, minimum, average and quartiles from the cost distributions. Data points above the standard deviation and the quartiles are considered outliers.

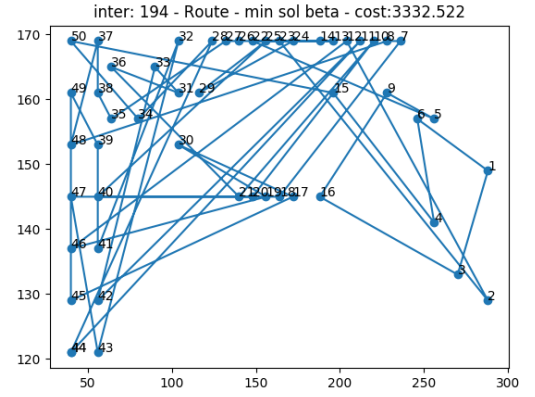
**Figure 13:** Boxplot from α, β and $\alpha \cup \beta$

From box-plot in FIGURE 13, histograms in FIGURE 12a and FIGURE 12b we can see that the β distribution has more candidate solutions with cost less than average cost for α ,

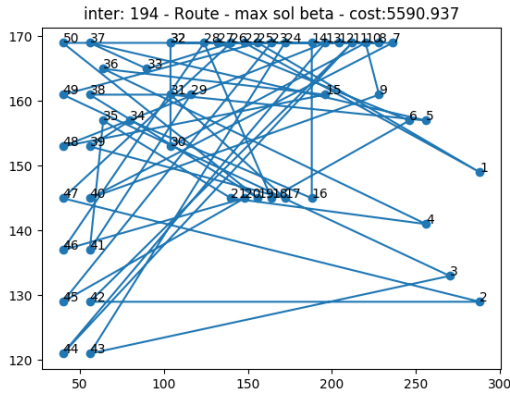
In FIGURE 14 We can see the relationship between the kelly fraction and the p-value for the chi-square test, draw from iteration 194 for the 50-city instance. To help visualization, each point size is set by multiplying the number of



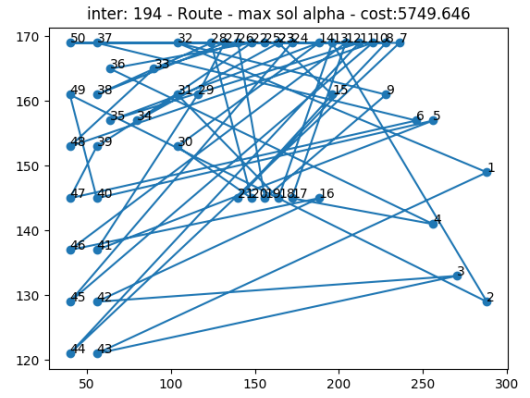
(a) Minimum cost in α



(b) Minimum cost in β



(c) Maximum cost in β

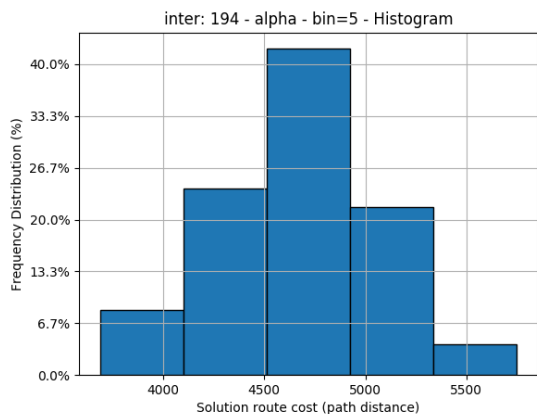


(d) Maximum cost in α

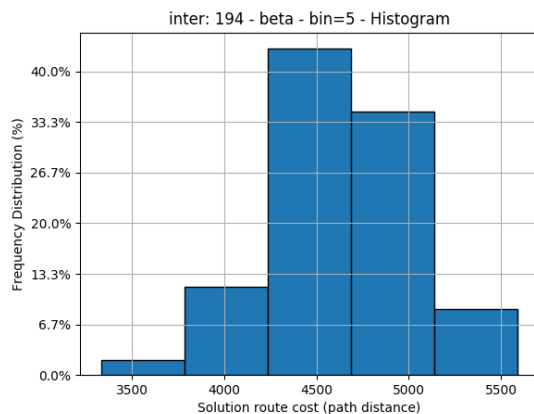
Figure 11: Maximum and minimum paths in graph

bits for distributions α (color red), β (color green) and the divergence between α and β (color blue) by a constant factor $c = 10000$.

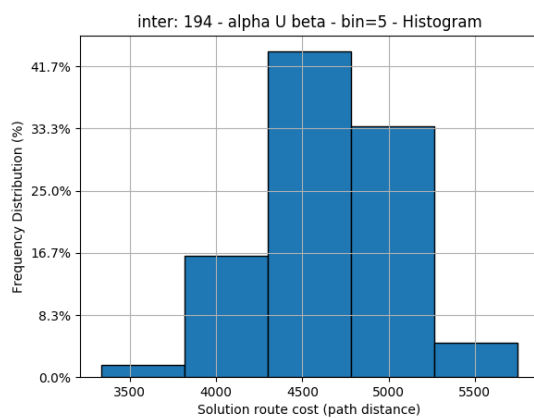
As show in FIGURE 14, if distribution β produces in average better solutions than α than the green group is closer to the left upper corner and the red group is shifted to the right bottom. The blue group is the subset with the divergence between α and β . If β is a improvement from α than this subset (represented by the blue group) is closer to the green group. Therefore we can conclude that at iteration 194 the distribution β produces in average solutions with better quality than distribution α (i.e. with smaller cost than expected) at a significant level and the likelihood of finding a near-optimal solution is positive.



(a) Histogram from α



(b) Histogram from β



(c) Histogram from α, β

Figure 12: Histogram from sets α and β

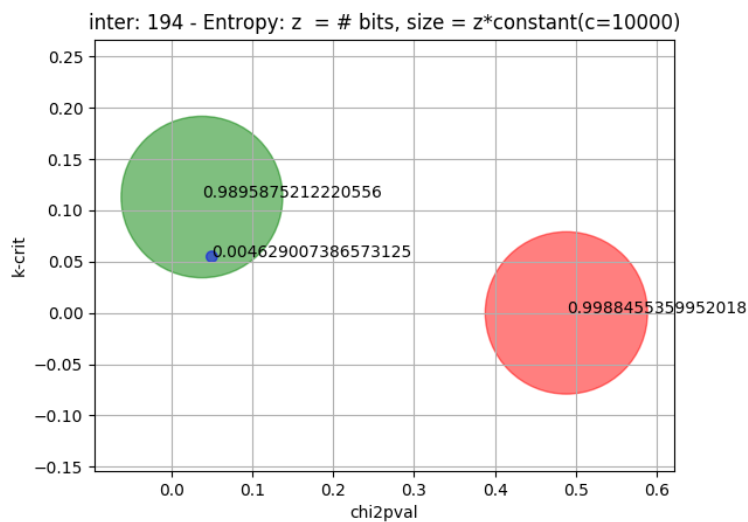


Figure 14: Significance vs Error Rate from α, β and $\alpha \cup \beta$

4. Conclusion

In this paper we have described a 3 stage classification algorithm to solve the TSP and compare the performance of the solution distributions in terms of statistical significance, error rate and information complexity. From the result of the experiments for three simulations with 10,30 and 50 city-nodes respectively it can be concluded that the expected cost between the model distribution α and the alternative distribution β is reduced to a near-optimal solution with smaller distance. We have found near-optimal solutions with p-value with less than 0.5% deviation for all instances of the problem.

The algorithm is an approximation on the expected value of a symbol in a sequence represented by two probability distributions function $p(x)$ and $q(x)$ against another indexing function f_z and although it may not find the optimal solution reliably, it can produce near-optimal solutions with predictable time and space growth rate. In this paper we have modeled the salesman dilemma to a problem of compressing information in a communication channel that have symbols proportional to some true unknown distribution from a random variable. The maximum exponential rate of improvement of the algorithm is equal to the rate of transmission (and computation) of the information in this channel. The channel is the total of inside information available to the algorithm. As the number of elements n in the sample distributions increases the time and space required to produce valid permutation also increases proportional to n .

The algorithm executes a finite number of iterations until it finishes when it reaches a predefined program parameter constant. This behavior makes the method demand less resources to find the minimum total distance to the problem than other strategies. The amount of time and memory required is linearly proportional to the distribution of input-output pair symbols in a communication channel. Therefore the complexity to transmit and compute a sequence of symbols is depend on the program up to a additive constant. The index function $f_z(x)$ is used to search the solution space for permutations with less traveling distance in the tour against a measured expectation cost-distance value. This is based on the assumption that sequences with less complexity(short length) are more likely than output strings with greater complexity.

Assuming by the law of large number that eventually all possible candidate solutions can be generated we conclude that the quality of the solution could be improved by allowing more execution time. This approach can be used to solve small and large instances of the problem. The expectation and variance between the data distributions is adaptive to the topology of the solution space under some boundaries. This has the advantage of being resistant to changes in the program parameter values under some significance level and error probability, thus sample sizes should be set accordingly.

The interpretation proposed in this paper is different from classical methods to solve the TSP problem. At every interaction it searches the solution space with the goal of maximizing the expected value of the logarithm of the current channel capacity instead of the value function defined by the euclidean distance. The logarithm is used because its additive and independent of past events. While the algorithm is trying to maximize the value function it also has to decide if the alternative hypothesis(β) is different from the null hypothesis(α) and the number of bits required to encode this information.

If the samples mean in a given iteration are different and the likelihood of improving the cost distance is positive than the algorithm set a flag to *True* and record this data in a filtered list. This can be interpreted as the classification label for the sample data at each execution for the machine. The strategy can be implemented using a Shannon code using $H(X)$ bits with finite symbols and computed by any Turing machine T with finite tape draw according to a Universal Computation Probability U . The Kolmogorov complexity is approximated by the length of the shortest program(i.e. group of data structures and instructions) that produces the tour distribution with the shortest expected total distance as output and halts.

Therefore we can conclude that the salesman problem can be modeled as problem of optimizing the transmission rate of information in a noisy communication channel. Consider as a example a salesman that receives a list of cities that it needs to visit from a channel source and he wants to find the shortest tour between cities to minimize his cost. There is a probability density function associated with the occurrence of each symbol sent proportional to the distribution of input-output pairs. The salesman must compares his current total tour distance against the new signal received. At each interaction he wants to reduce the rate of misinformation or the risk of choosing a path that would lead to a route with a potential cost higher than the current know best solution. This interpretation requires two Turing machines with two finite tapes defined as:

Rule 1: Turing Machines T (Source Encoder & Channel Encoder) and τ (Channel Decoder & Source Decoder) with two tapes.

Rule 2: The tapes A and B are the communication channel C between machines T and τ .
Rule 3: Machines T and τ agree on same finite alphabet L to encode or decode symbols.
Macro state A: T writes encoded symbols in the tape A using finite alphabet L
Macro state B: τ reads the received symbols sent by T from C and decode each sign.
Macro state C: τ writes a symbol using another alphabet L^* in its own tape B .

In figure 15 we see an example representation of the finite state control unites.

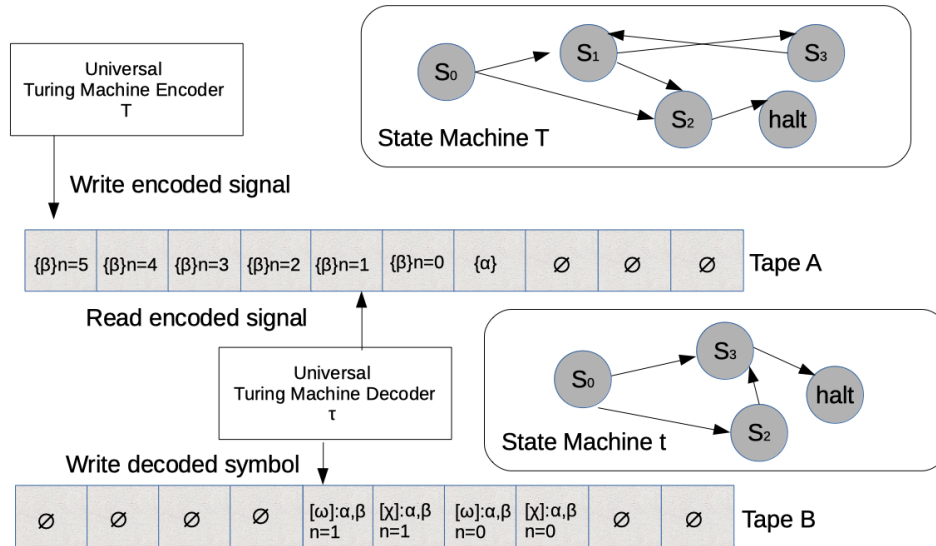


Figure 15: Universal Algorithmic-Information Turing Machines.

Future work could be to compare the benchmark between the Algorithmic Information Theory (AIT) based method and other heuristics such as 2-opt and SA. Additionally to improve performance we could use an extra buffer to store illegal moves (such as in the taboo list) to avoid frequently exploring solutions with worst expected distance (negative gain) between nodes. After expiration time units is reached the element can be purged from this neighborhood list. This method could improve performance by reducing the valid solutions that are evaluated but has the downside of requiring additional steps to search and operate the additional list structure. Alternatively the valid permutations can be created with 2-opt moves instead of random permutation. Future work could also explore setting constrains in the precedence/prefix of random strings using the feedback from previous complete executions of the AIT algorithm to dynamically update the current best know state.

References

- [1] Levitin, Anany. Introduction To Design And Analysis Of Algorithms, 2/E Pearson Education, 2008. Pag 397-399, p, NP and NP-complete Problems.
- [2] NICOLETTI, Maria do Carmo, and E. R. Hruschka Jr. Fundamentos da teoria dos grafos para computação. Série Apontamentos. EdUFSCar, Sao Carlos, ed. rev. edition (2006). page 151
- [3] Mohd Razali, Noraini & Geraghty, John (2011). Genetic Algorithm Performance with Different Selection Strategies in Solving TSP.
- [4] Nilsson, Christian (2003). Heuristics for the Traveling Salesman Problem.
- [5] Bachmann, Paul (1894). Analytische Zahlentheorie [Analytic Number Theory] (in German). 2. Leipzig: Teubner.
- [6] Drozdek, Adam. Data Structures and algorithms in C++. Cengage Learning, 2012, p49
- [7] Juris Hartmanis. An Overview of the Theory of Computational Complexity
- [8] G. A. CROES (1958). A method for solving traveling salesman problems. Operations Res. 6 (1958) , pp., 791-812.
- [9] H Holland, John. (1975). Adaptation In Natural And Artificial Systems. University of Michigan Press.
- [10] Zhong, Jinghui & Hu, Xiaomin & Zhang, Jun & Gu, Min. (2005). Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms.

- [11] E. Goldberg, David & Deb, Kalyanmoy. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. 51. 10.1016/B978-0-08-050684-5.50008-2.
- [12] Zhan, Shi-hua & Lin, Sannuo & Zhang, Ze-jun & Zhong, Yiwen. (2016). List-Based Simulated Annealing Algorithm for Traveling Salesman Problem. Computational Intelligence and Neuroscience. 2016. 1-12. 10.1155/2016/1712630.
- [13] Cover, Thomas M., and Joy A. Thomas. Elements of information theory. John Wiley & Sons, 2012.
- [14] Uro, M. 2002. Basic concepts in information theory. <http://www-public.imtbs-tsp.eu/~uro/cours-pdf/poly.pdf>
- [15] C. E. Shannon. A mathematical theory of communication. Bell Syst. Tech. J., 27:379 – 423, 623 – 656, 1948
- [16] S. Kullback and R. A. Leibler. On information and sufficiency. Ann. Math. Stat., 22:79 – 86, 1951.
- [17] R. A. Fisher. Theory of statistical estimation. Proc. Cambridge Philos. Soc., 22:700 – 725, 1925.
- [18] S. Kullback. Information Theory and Statistics. Wiley, New York, 1959.
- [19] B. McMillan. The basic theorems of information theory. Ann. Math. Stat., 24:196 – 219, 1953.
- [20] L. Breiman. The individual ergodic theorems of information theory. Ann. Math. Stat., 28:809 – 811, 1957. With correction made in 31:809-810
- [21] Thorp, Edward. (2008). The Kelly Criterion in Blackjack, Sports Betting, and the Stock Market. Handbook of Asset and Liability Management. 1. 10.1016/B978-0-444-53248-0.50015-0.
- [22] G. J. Chaitin. On the length of programs for computing binary sequences. J. ACM, 13:547 – 569, 1966
- [23] G. J. Chaitin. Information theoretical limitations of formal systems. J. ACM, 21:403 – 424, 1974
- [24] G. J. Chaitin. Randomness and mathematical proof. Sci. Am., 232(5):47 – 52, May 1975.
- [25] A. N. Kolmogorov. Three approaches to the quantitative definition of information. Probl. Inf. Transm. (USSR), 1:4 – 7, 1965.
- [26] A. N. Kolmogorov. Logical basis for information theory and probability theory. IEEE Trans. Inf. Theory, IT-14:662 – 664, 1968.
- [27] J. Kelly. A new interpretation of information rate. Bell Syst. Tech. J., 35:917 – 926, July 1956.
- [28] P. Martin-Lof. The definition of random sequences. Inf. Control, 9:602 – 619, 1966.
- [29] L. A. Levin and A. K. Zvonkin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. Russ. Math. Surv., 25/6:83 – 124, 19
- [30] E. L. Lehmann and H. Scheffe. Completeness, similar regions and unbiased estimation. Sankhya, 10:305 – 340, 1950.
- [31] E. H. Aarts, J. H. Korst, and P. J. van Laarhoven, A quantitative analysis of the simulated annealing algorithm: a case study for the traveling salesman problem," Journal of Statistical Physics, vol. 50, no. 1-2, pp. 187–206, 1988.
- [32] J. R. A. Allwright and D. B. Carpenter "A distributed implementation of simulated annealing for the travelling salesman problem," Parallel Computing, vol. 10, no. 3, pp. 335–338, 1989.
- [33] C.S. Jeong and M.H. Kim, Fast parallel simulated annealing for traveling salesman problem on SIMD machines with linear interconnections," Parallel Computing, vol. 17, no. 2-3, pp. 221– 228, 1991.
- [34] M. Hasegawa "Verification and rectification of the physical analogy of simulated annealing for the solution of the traveling salesman problem," Physical Review E, vol. 83, Article ID 036708, 2011.
- [35] P. Tian and Z. Yang, "An improved simulated annealing algorithm with genetic characteristics and the traveling salesman problem," Journal of Information and Optimization Sciences, vol. 14, no. 3, pp. 241–255, 1993.
- [36] Solomonoff, Ray J.. "Algorithmic Probability: Theory and Applications." (2009).
- [37] Michiel van Lambalgen. Algorithmic Information Theory - The Journal of Symbolic Logic. Vol. 54, No. 4 (Dec., 1989), pp. 1389-1400
- [38] Zunic, Emir & Besirevic, Admir & Skrobo, Rijad & Hasic, Haris & Hodzic, Kerim & Djedovic, Almir. (2017). Design of Optimization System for Warehouse Order Picking in Real Environment. 10.1109/ICAT.2017.8171630.
- [39] Ferentinos, K.P. & Arvanitis, Kostas & Sigrimis, N. (2002). Heuristic optimization methods for motion planning of autonomous agricultural vehicles. Journal of Global Optimization. 23. 155-170. 10.1023/A:1015527207828.
- [40] M. Rotando, Louis & Thorp, Edward. (1992). The Kelly Criterion and the Stock Market. American Mathematical Monthly. 99. 922-931. 10.2307/2324484.
- [41] Maier, Alexander. (2015). Identification of Timed Behavior Models for Diagnosis in Production Systems.
- [42] Hopcroft, John E. and Ullman, Jeffrey D. (1969) Formal languages and their relation to automata. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA