Defining asymptotic parallel time complexity of data-dependent algorithms

Paula FRITZSCHE and Dolores REXACHS and Emilio LUQUE

Computer Architecture and Operating Systems Department
University Autonoma of Barcelona, 08290 Barcelona, SPAIN

{Dolores.Rexachs,Emilio.Luque}@uab.es

Received 1 August 2012, 22 April 2013, 17 October 2013, 3 January 2014

Abstract The scientific research community has reached a stage of maturity where its strong need for high-performance computing has diffused into also everyday life of engineering and industry algorithms. In efforts to satisfy this need, parallel computers provide an efficient and economical way to solve large-scale and/or time-constrained problems. As a consequence, the end-users of these systems have a vested interest in defining the asymptotic time complexity of parallel algorithms to predict their performance on a particular parallel computer.

The asymptotic parallel time complexity of data-dependent algorithms depends on the number of processors, data size, and other parameters. Discovering the main other parameters is a challenging problem and the clue in obtaining a good estimate of performance order. Great examples of these types of applications are sorting algorithms, searching algorithms and solvers of the traveling salesman problem (*TSP*).

This article encompasses all the knowledge discovery aspects to the problem of defining the asymptotic parallel time complexity of data-dependent algorithms. The knowledge discovery methodology begins by designing a considerable number of experiments and measuring their execution times. Then, an interactive and iterative process explores data in search of patterns and/or relationships detecting some parameters that affect performance. Knowing the key parameters which characterise time complexity, it becomes possible to hypothesise to restart the process and to produce a subsequent improved time complexity model. Finally, the methodology predicts the performance order for new data sets on a particular parallel computer by replacing a numerical identification.

As a case of study, a global pruning traveling salesman problem implementation (*GP-TSP*) has been chosen to analyze the influence of indeterminism in performance prediction of data-dependent parallel algorithms, and also to show the usefulness of the defined knowledge discovery methodology. The subsequent hypotheses generated to define the asymptotic parallel time complexity of the *TSP* were corroborated one by one. The experimental results confirm the expected capability of the proposed methodology; the predictions of performance time order were rather good comparing with real execution time (in the order of 85%).

Keywords Complexity, Data-Dependent Algorithms, Parallel Computing, Performance Order

§1 Introduction

Computational science (*CS*) is often referred to as the third science, complementing both theoretical and laboratory science [29]. As shows Fig. 1,

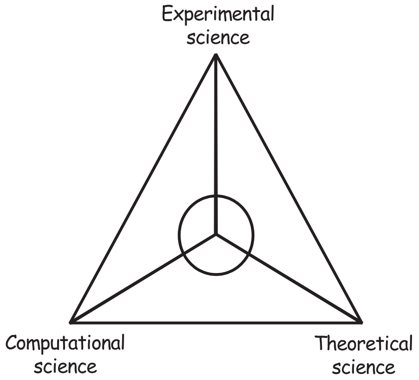


Fig. 1 Computational science as the third science.

there is a symbiotic relationship between the three sciences: theoretical findings guide the experimentalists, experimental data is used to build and validate computational research, and computational research provides the theorists with new directions and ideas. In particular, as is shown in Fig. 2, the components of *CS* are applications, algorithms, and architectures. *CS* is a scientific endeavor (an application) that is supported by concepts and skills of mathematics (algorithms) and computer science (architecture). Central to any computational science problem, there is a model of the problem. Building models for abstracting the components of a real problem let us to make predictions of what might happen. *CS* takes advantage of not only the improvements in computer hardware, but also improvements in computer algorithms and mathematical techniques. It allows resolving issues that were previously too difficult to do due to the intricacy of the mathematics, the large number of calculations involved, or a combination of both. Besides, it permits visualization, analysis, and interpretation of large data sets in ways that can inform some complex problems.

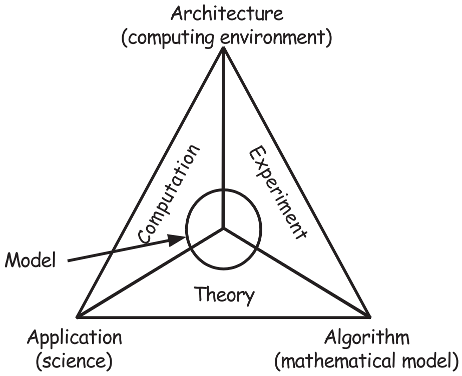


Fig. 2 Computational science components.

For a data-dependent parallel algorithm, similar input data sets may cause significant execution time variations. Measuring the time required to reach the solution (the fundamental performance metric) of any parallel algorithm for all possible input values would allow to answer any question about how the algorithm will respond under any set of conditions. Nevertheless, it is impossible to make all of these measurements.

The asymptotic parallel time complexity of a data-dependent algorithm does not depend only on the number of processors (P) and on the problem size (N) but also on unknown parameters. Unfortunately, the other parameters are unpredictable and seemingly dependent on the properties of the data. Discovering these properties is a challenging problem and the clue to obtain a good estimate of performance order of data-dependent parallel algorithms. Great examples of these types of problems are sorting algorithms, searching algorithms, solvers of the satisfiability problem, solvers of the graph partition, solvers of the knapsack problem, solvers of the bin packing, solvers of the motion planning, and solvers of the traveling salesman problem, amongst many others. There are a lot of interesting related works in the literature that deal with one implementation that solve a problem mentioned above and define its complexity. However, there are no studies showing how to evaluate the asymptotic time complexity of data-dependent algorithms.

The complexity of a sorting algorithm is established in terms of the size of a problem and in terms of the disorder of the given problem instance [12)]. As a consequence, different measures of disorder have been presented in these kinds of algorithms [9, 12, 35)]. The complexity of the problem of deciding whether a given propositional formula in conjunctive normal form is satisfiable and its variations has been widely studied [32, 2)]. The asymptotic complexity of the knapsack problem is defined by $O(nW)$ where n is the number of items and W the knapsack capacity. Many articles exist that discuss different implementations. Most of them talk about the partially-ordered knapsack [23)]. It is well-known that an exhausted search for a traveling salesman problem *TSP* has an exponential time complexity [36)]. An exact solution takes $O(n!)$ time, which is prohibitively long. For this reason, polynomial-time heuristic search approaches are proposed [21, 22)].

This paper provides a general methodology to cover all the issues to the problem of defining the asymptotic parallel time complexity of data-dependent algorithms, using a computational approach. This is a good starting point for understanding some facts related to the non-deterministic algorithms. Any minimum contribution in this sense represents a great advance in this subject.

The scientific experimental methodology begins by designing a considerable number of examples and measuring their execution times. A well-designed example involves the worked hypotheses at that time. It guides the experimenters in selecting what experiments actually need to be performed. A data-mining tool then explores the collected data in search of hidden patterns and/or relationships detecting the main parameters that affect performance. The patterns and/or relationships are modelled numerically in order to generate an analytical formulation of the execution time required. Knowing the key parameters which characterise time complexity, it becomes possible to hypothesise to restart the process and to produce a subsequent improved time complexity model. Finally, the methodology predicts how the algorithm will perform on a particular parallel computer when it is given new data sets. The entire process is neither more nor less than the scientific method, a way to ask and answer scientific questions by making observations and doing experiments [20)], computational experiments in this case.

The traveling salesman problem (*TSP*) has been chosen as a case study.

The *TSP* is of considerable importance not only from a theoretical point of view but also from a practical one. The drilling of printed circuits boards, the assignment of routes for planes of a specified fleet, the handling of materials in a warehouse are some examples. Therefore, there is a concrete need for defining *TSP* asymptotic time complexity. As a representative of these practical problems, a parallel global pruning traveling salesman problem implementation (*GP-TSP*) has been analyzed.

The goals of this research are both to examine the influence of indeterminism in performance prediction of data-dependent parallel algorithms and to show the usefulness of the defined scientific experimental process for this kind of algorithms. A significant number of experiments confirm the expected capability of the proposed methodology.

The remainder of this article is organized as follows. The next section presents a general "knowledge discovery" methodology to the problem of defining the asymptotic parallel time complexity of data-dependent algorithms. Section 3 describes the case study: the traveling salesman problem. Besides, it provides detailed coverage of a *TSP* implementation and their underlying ideas behind the discovery process of the significant input parameters. Section 4 summarizes and draws the main conclusions of this work.

§2 Knowledge discovery methodology

The scientific experimental knowledge discovery methodology attempts to define the asymptotic parallel time complexity of data-dependent algorithms. The process of knowledge discovery is certainly not new. It is typical of the experimental sciences. An experimental science is based on observation of performing repeated controlled experiments. Before computers were used to automate this process, math, physics or statistics researchers were using probability techniques to model historical data.

The following subsections describe how the scientific experimental methodology works.

2.1 Design of experiments to propose a model

First of all, it is important to understand the algorithm domain and the relevant prior knowledge, and to analyze its behavior step by step, in a deep way. It is a try-and-error method that requires specialists to manually or automatically identify the relevant parameters that can affect the execution time of the studied algorithm. Discovering the key parameters is the basis to obtain a good prediction capacity. Including too many parameters may lead to an accurate but too complex or even unsolvable model. Hence, great care should be taken in selecting parameters and a reasonable trade-off should be made. The *CS* takes advantage of not only the improvements in computer hardware, but also, probably more importantly, the improvements in computer data-mining algorithms and mathematical techniques.

Creating a well-designed experiment involves articulating a goal, choosing an output that characterizes an aspect of that goal, and specifying the data that will be used in the study taking into account the worked hypotheses at that time. The experiments must provide a good training data set first to measure the quality of the model / hypotheses and then to fit the model. After the necessary training data set has been defined, the studied parallel algorithm must process each training data obtaining their corresponding execution time, left side of the Fig. 3.

The term knowledge discovery in databases (*KDD*) refers to the process of analyzing data from different perspectives and summarizing it into useful information. Technically, *KDD* is the process of finding correlations or patterns among dozens of fields in large relational databases. A *KDD* process involves data preparation (cleaning, preprocessing, and transformation), defining a data-mining study, reading the valuable data and building a model, understanding the model, and finally predicting. Every step is shown in Fig. 3. Understanding previous models and the final model will help researches to improve their general knowledge about the interest problem. It is an interactive and iterative process, surrounding numerous steps with many decisions that the end-user carries out [18).]

The stage of defining a data-mining study encompassing both the decision of choosing a data-mining technique (classification, regression, clustering, dependency modeling, summarization of data, or change and deviation detection), and the selection of the particular technique and the algorithm to apply according to the chosen technique. It is important for these techniques to have the ability to identify relevant attributes and to disregard the superfluous ones.

Regarding the analysis of the problem, a clustering study could be performed to potentially identify groups. Current clustering techniques can be broadly classified into two categories: partitional and hierarchical. Given a set of records, partitional clustering obtains a partition of the records into k subsets (clusters), so that the data in each subset (ideally) share some common traits (often proximity according to some defined distance measure). Hierarchical clustering algorithms produce a nested sequence of partitions. These algorithms work on both bottom up and top down approaches.

Therefore, some important parameters values that affect performance for the studied algorithm together with their corresponding measured execution times are analyzed with a k-means clustering algorithm (partition algorithm) included in a data-mining tool in order to summarize these into a useful information (patterns). A more concise description of the result including only a handful of necessary attributes is more desirable than hundreds of attributes, see right side of the Fig. 3. Studying these attributes which characterise time complexity, it becomes possible to suspect new hypotheses to restart the process and to produce a subsequent improved time complexity model (dashed lines in Fig. 3).

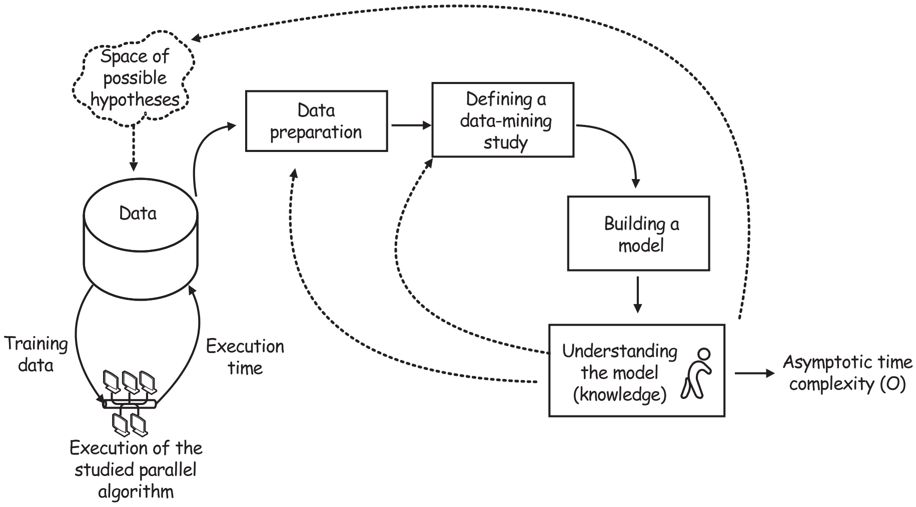

Space of possible hypotheses
Data
Data preparation
Defining a data-mining study
Building a model
Understanding the model (knowledge)
Asymptotic time complexity (O)
Training data
Execution time
Execution of the studied parallel algorithm

Fig. 3 Knowledge construction process.

In summary, Fig. 3 shows the high-level steps that are involved in the overall knowledge acquisition process which includes designing of the training data set, executing of the studied parallel algorithm, data preparation, defining a data-mining study and building a model, understanding the model, and finally predicting. There is no doubt that the design of experiments is directly related to the suspected hypotheses. The solid lines represent the compulsory path to follow in the methodology and the dashed lines represent paths of refinement.

2.2 Validation of the model

A new data set is proposed to be able to validate the created model (left side in Fig. 4). Although the validation data set constitutes a hold-out sample, it has not been considered in the building of the model. This enables to estimate the error in the predictions without having the assumption that the execution times follow a particular distribution.

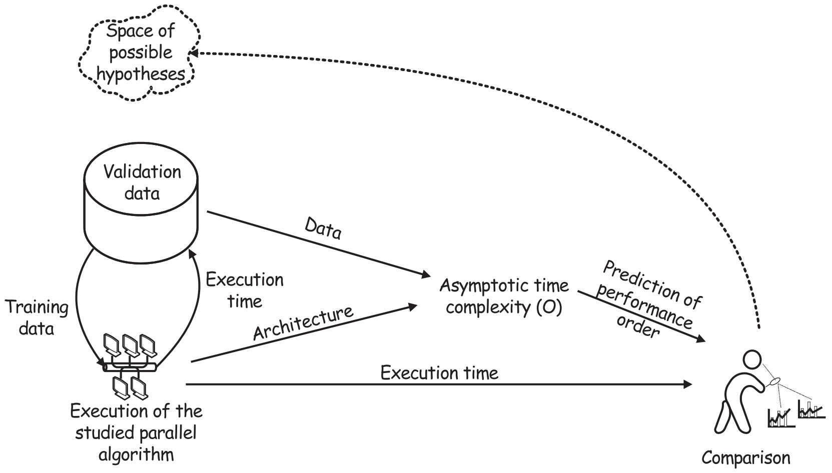

Space of possible hypotheses
Validation data
Data
Execution time
Training data
Architecture
Asymptotic time complexity (O)
Prediction of performance order
Execution time
Execution of the studied parallel algorithm
Comparison

Fig. 4 Model validation phase.

Instantiating the analytical formulation (called asymptotic time complexity in Fig. 4) with validation data and particular values of a parallel computer architecture, it is possible to make predictions of performance order. The quality analysis is an important issue in this stage and has to include relevant measurements. Each prediction is then compared to the execution time obtaining the prediction error, right side in Fig. 4. The average of the square of this prediction error enables to compare different models and to assess the accuracy of the model in making predictions.

2.3 Predicting the performance order

The objective of predicting modeling is always to build a model which best predicts outcomes for future data. In this study, building a model means giving a formulation of complexity order. Then, every prediction of performance order for new input data in a particular parallel computer is obtained instantiating the asymptotic time complexity.

2.4 The entire methodology

The entire methodology is shown in Fig. 5. The higher part of the picture describes how is built and fit the model in order to define the final asymptotic time complexity formulation, sections 2.1 and 2.2. The lower part of the picture shows the prediction framework, section 2.3.

Every stage in the defined methodology can implicate a backward motion to previous steps (dashed lines in Fig. 5) in order to obtain extra or more precise information.

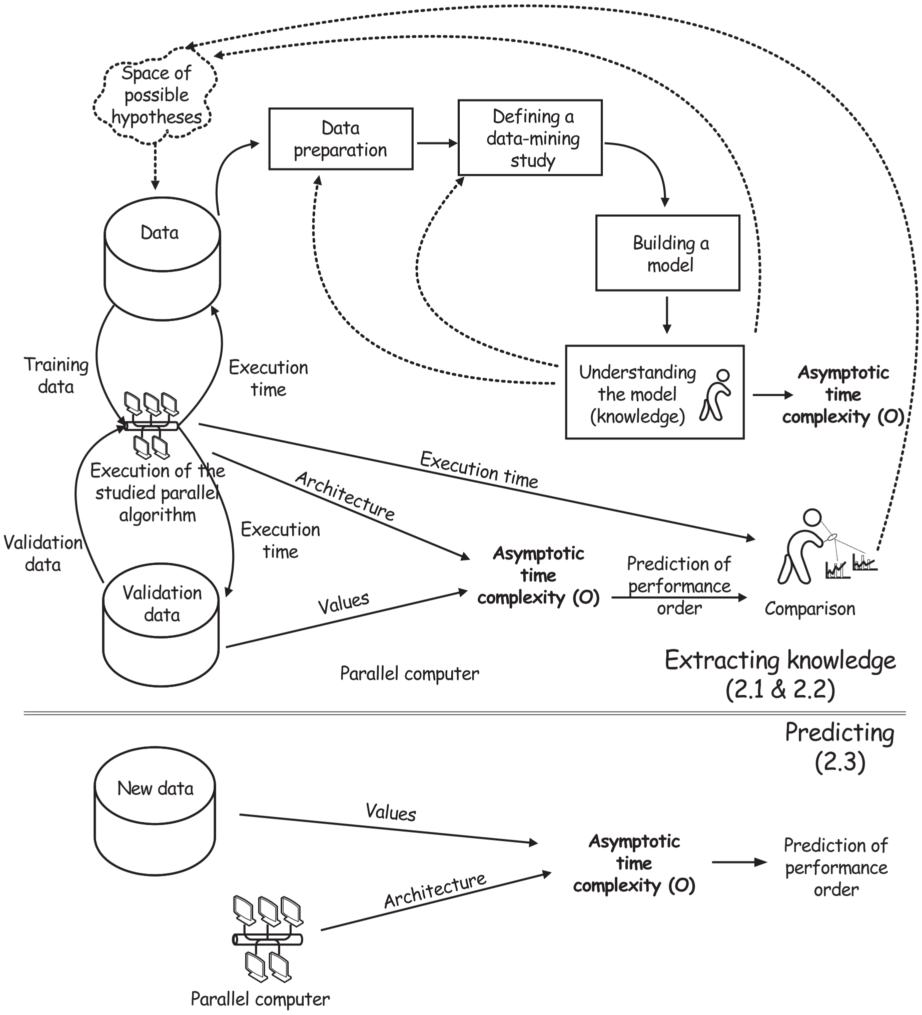

Space of possible hypotheses
Data preparation
Defining a data-mining study
Data
Building a model
Training data
Execution time
Understanding the model (knowledge)
Asymptotic time complexity (O)
Execution time
Architecture
Execution of the studied parallel algorithm
Execution time
Validation data
Validation data
Asymptotic time complexity (O)
Prediction of performance order
Values
Comparison
Parallel computer
Extracting knowledge (2.1 & 2.2)
Predicting (2.3)
New data
Values
Asymptotic time complexity (O)
Prediction of performance order
Architecture
Parallel computer

Fig. 5 The entire novel methodology.

§3 A case of study: the traveling salesman problem

The traveling salesman problem (*TSP*) is one of the most famous issues in the field of the combinatorial optimization. In spite of the apparent simplicity of their formulation, the *TSP* is a complex solving data-dependent problem. Not only the complexity of its solution has been a continue challenge to many researchers [7, 33, 36, 3] but also the prediction of its performance to reach the solution.

This section gives an overview of the existing *TSP* problems as well as many practical issues that can be formulated as *TSP* problems. In addition a parallel Euclidean *TSP* implementation is presented. The objective is to show the usefulness of the general knowledge discovery process.

3.1 TSP problem statement

The *TSP* for C cities is the problem of finding a tour visiting all the cities only once and returning to the starting city so that the sum of the distances between consecutive cities is minimized. The requirement of returning to the starting city does not change the computational complexity of the problem [13].

3.2 TSP practical problems

The *TSP* is of considerable importance not only from a theoretical point of view but also from a practical one. Issues having the *TSP* structure occur in the analysis of the structure of crystals [8], in handling materials in a warehouse [30], in clustering of arrays data [25], in sequencing jobs on a single machine [15], in physical mapping problems [1], in genome rearrangement [31], and in phylogenetic tree construction [24] amongst many others. Related variations on the *TSP* include the resource constrained traveling salesman problem which has applications in scheduling with an aggregate deadline [28]. The prize collecting traveling salesman problem [5] and the orienteering problem [17] are special cases of this. The max *TSP* objective is finding a tour of maximum length [6]. The maximum scatter *TSP* is the problem of computing a path on a set of points in order to maximize the minimum edge length in the path. This kind of algorithm is motivated by applications in manufacturing and medical imaging [4]. Most importantly, the *TSP* often emerges as a subproblem in more complex combinatorial problems. The problem of determining for a fleet of vehicles which customers should be served by each vehicle and in what order each vehicle should visit the customers assigned to it is the best known combinatorial problem [11].

But why is it so important to define the asymptotic time complexity in any of the above examples? For instance, a model for the vehicle routing problem can provide answers to questions such as 'How much computing time is needed to finish daily scheduling?'; 'What happens to the complexity if data and machine sizes are modified?'; 'How many machines are needed to finish the work within a given time order?' Besides, the purpose of parallel complexity theory is to explore the extent to which efficient and fast parallel computation is possible [10].

3.3 GP-TSP implementation

An Euclidean global pruning *TSP*[*1] implementation (called *GP-TSP*) is presented to obtain the exact *TSP* solution in a parallel machine. It is used to analyze the influence of indeterminism in defining the asymptotic time complexity and also to show the usefulness of the methodology. The implementation is a branch-and-bound algorithm which recursively searches all possible Hamiltonian paths and prunes large parts of the search space by maintaining a global variable containing the length of the shortest path found up to that moment. The *GP-TSP* follows the Master-Worker (MW) programming paradigm [16].

Each city is represented by two coordinates in the Euclidean plane then, for each city pair exists a symmetrical distance. Considering C different cities, the Master defines a certain level L to divide the tasks. Tasks are the evaluations of the possible permutations of $C-1$ cities in L elements. The granularity G of a task is the number of cities that defines the task sub-tree: $G = C - L$. At the execution, the Master sends tasks with a variable containing the length of the shortest path found until that moment.

A diagram of the possible permutations for 5 cities, considering the salesman starts and ends his trip at the city 1, can be seen in Fig. 6. The Master can divide this problem into one task of level 0 or four tasks of level 1 or twelve tasks of level 2 for example. The tasks of the first level would be represented by the cities 1 and 2 for the first task, 1 and 3 for the second, followed by 1 and 4 and 1 and 5.

[*1] Since any distance measure (Euclidean, Manhattan, Chebychev, ...) would be equivalent, it was decided to use the Euclidean. Besides, for simplicity, it was considered cities in R^2 instead of R^n. Finding an optimal Hamilton Circuit with the least weight in a complete weighted graph (where the vertices, the edges, and the weights represent the cities, the roads, and the cost or distance of that road respectively) would be an equivalent formulation. Consequently, the ideas of this article can be generalized.

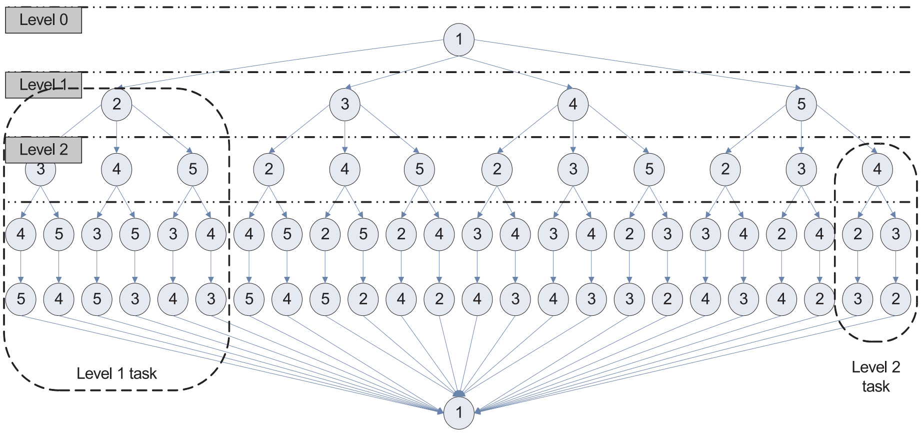


Fig. 6 Possible paths for the salesman considering 5 cities.

Workers are responsible for calculating the distance of the permutations left in the task and sending the minimum path and distance of these permutations to the Master. Since the performance of the *GP-TSP* is not only *I/O*, but also *CPU*-bound, it is very important to minimize the number of distance computations. Pruning conditions must be applied not only to avoid accessing irrelevant sets of cities, but also to minimize the number of distances computed. If the length of a partial path is bigger than the current minimal length, this path is not expanded further and a part of the search space is pruned in the *GP-TSP*. Hence, the estimate of times becomes a complex function.

The *GP-TSP* algorithm internally works on a symmetric matrix of Euclidean distances. Fig. 7(a) shows an example of a strictly lower triangular matrix of Euclidean distances; meanwhile, Fig. 7(b) shows a pruning process. Analyzing Fig. 7(b), each arrow has an associated distance coefficient between the two cities it connects. The total distance in the first path (in the left) is of 40 units. The distance between 1 and 2 on the second path (in the right) is already of 42 units. It is then not necessary for the *GP-TSP* algorithm to continue calculating distances from the city 2 on because it is impossible to improve the distance for this branch.

3.4 Discovering the significant GP-TSP input parameters

Using simple experiments, varying one or two values at a time, it is possible to infer that time required for the parallel *GP-TSP* algorithm depends

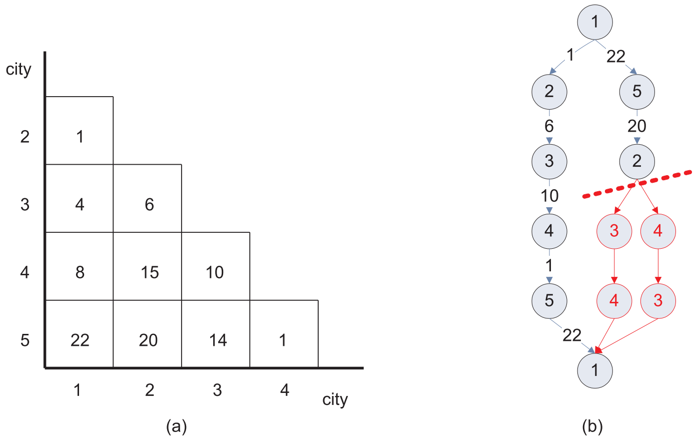


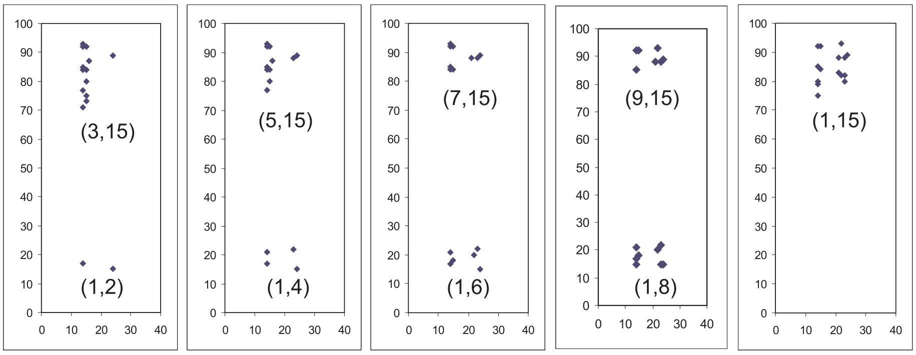
Fig. 7 (a) Matrix of distances (b) and a pruning process in the *GP-TSP* algorithm.

on the number of processors (P), on the number of cities (C), and on other parameters. The value of these other parameters are data-dependent. As a result of the investigation, right now the sum of the distances from one city to other cities (SD) and the mean deviation of SDs values ($MDSD$) are the numerical parameters characterizing the different input data beyond the number of cities. But, how have these final parameters been obtained? Next, it is described the way to discover the above mentioned dependencies (SD and $MDSD$).

Specification of the parallel machine: The executions have been reached with a 32 node homogeneous PC Cluster (Pentium IV 3.0GHz., 1Gb DDR-DSRAM 400Mhz., Gigabit Ethernet) at the Computer Architecture and Operating Systems Department, University Autonoma of Barcelona. All the communications have been accomplished using a switched network with a mean distance between two communication end-points of two hops. The switches enable dynamic routes in order to overlap communication.

[1] First hypothesis: location of the cities (geographical pattern)

Given a number of cities with its pattern of distribution, the initial experiments have provided evidence that times required for the completion of the algorithm are dissimilar. In order to understand the general process and to show its progress and results, it has been chosen an example data-set to follow along this section. It consists of 5 different geographical patterns (named $G1$ to $G5$) of 15 cities each ($C1$ to $C15$). Fig. 8 shows the five patterns used for the 15 cities.



(a) *G1* (b) *G2* (c) *G3* (d) *G4* (e) *G5*

Fig. 8 Five patterns for 15 cities.

The *GP-TSP* algorithm receives the number of cities (C) and their coordinates, the level (L), and the number of processors (P). It proceeds recursively searching all possible paths, applying the pruning strategy whenever possible and, finally, generating the minimal path and the time spent.

Table 1 *GP-TSP* execution times (ET, in sec.) by pattern ($G1...G5$) and starting city (SC, $C1...C15$) and assigned cluster (Cl).

	Pattern									
	$G1$		$G2$		$G3$		$G4$		$G5$	
SC	ET	Cl	ET	Cl	ET	Cl	ET	Cl	ET	Cl
$C1$	216.17	1	36.50	3	15.34	2	10.51	4	8.03	5
$C2$	214.44	1	36.82	3	15.19	2	10.49	4	7.82	5
$C3$	77.25	1	38.09	3	15.57	2	10.02	4	7.71	5
$C4$	72.64	1	37.29	3	15.02	2	10.30	4	7.91	5
$C5$	70.94	1	18.54	2	15.84	2	10.41	4	7.83	5
$C6$	74.21	1	17.83	2	15.24	2	10.24	4	7.71	5
$C7$	75.59	1	18.16	2	10.31	4	10.36	4	7.93	5
$C8$	73.72	1	18.03	2	10.34	4	10.26	4	7.87	5
$C9$	69.47	1	17.79	2	10.27	4	9.98	4	8.14	5
$C10$	74.96	1	17.48	2	10.23	4	9.88	4	8.22	5
$C11$	75.89	1	17.07	2	10.24	4	9.85	4	8.04	5
$C12$	70.17	1	17.39	2	10.28	4	9.87	4	8.12	5
$C13$	73.73	1	18.10	2	10.36	4	9.88	4	7.98	5
$C14$	70.87	1	17.37	2	10.17	4	9.95	4	8.02	5
$C15$	73.30	1	18.00	2	10.32	4	9.97	4	7.78	5
	Mean		Mean		Mean		Mean		Mean	
	92.23		22.97		12.32		10.14		7.94	

Table 1 shows the execution times (ET, in sec.) by pattern (columns $G1$ to $G5$) and starting city (SC, $C1...C15$) for the *GP-TSP* using only 8 nodes of the parallel machine. It is important to observe the dispersion of times while maintaining constant the number of processors and the number of cities.

Clustering, a data mining technique, has been applied to discover the internal information of these values and then to decrease the data-dependence. Its divides the data set into distinctive data clusters, so that the data in each cluster share some common trait (similar execution time). The general action has been done using the well-known k-means clustering algorithm [26] included in Cluster-Frame [16]. Cluster-Frame is a dynamic and open environment of clustering.

To obtain quite similar groups with respect to the patterns used at the beginning, k has been fixed in 5 (k is the number of searched clusters). The initial centroids (one for each cluster) have been randomly selected by the clustering application. The k-means algorithm aims at minimizing a squared error function. The objective function with n data points and k disjoint subsets is:

$$\sum_{j=1}^{k}\sum_{i=1}^{n}|x_i^{(j)} - c_j|^2 \tag{1}$$

where $|x_i^{(j)} - c_j|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centroid c_j. The Eq. 1 is an indicator of the distance of the n data points from their respective cluster centroids.

Columns *Cl* in Table 1 show the assigned cluster for each sample. For the clusters 1 to 5, the centroids values have been 92.23, 16.94, 37.17, 10.19, and 7.94 seconds, respectively.

The quality evaluation involves the validation of the above mentioned hypothesis. For each experiment, the assigned cluster has been confronted with the defined graphic pattern previously. The percentage of hits expresses the capacity of prediction. A simple observation is that the execution times have been clustered in a similar way to patterns fixed at starting (Fig. 8). In this example, the capacity of prediction has been of 65% for the *GP-TSP*. This means that there is a close relationship between the patterns and the run times.

Conclusions: The initial hypothesis for the *GP-TSP* has been corroborated; the capacity of prediction has been greater than 60% for the full range of experiments tested. The expressed percentage has given evidence of the existence of other significant parameters. Therefore, a deep analysis of results revealed an open issue remained for discussion and resolution, the singular execution times by pattern. Another major hypothesis was formulated. At this stage, the asymptotic parallel time complexity was defined as *O(P, C, pattern)*.

[2] Second hypothesis: location of the cities and starting city

The data-set is the same previously used. Comparing each chart of Fig. 8 with its corresponding column in Table 1, it is easy to infer some important facts:

- The two furthest cities (1, 2) in Fig. 8(a) correspond with the two higher time values of starting city $C1$ and $C2$ in Table 1 ($G1$).
- The four furthest cities (1, 4) in Fig. 8(b) correspond with the four higher execution time values of starting city $C1$ to $C4$ in Table 1 ($G2$).
- The six furthest cities in Fig. 8(c) correspond with the six higher time values of Table 1 ($G3$).
- The cities in Fig. 8(d) are distributed among two zones therefore, the times turn out to be similar enough, see Table 1 ($G4$).
- Finally, the cities in Fig. 8(e) are close enough, in consequence, their times are quite similar, see Table 1 ($G5$).

Another important observation is that the mean of execution times by pattern decreases as the cities approach, see again Table 1.

Conclusions: Sampling demonstrates the location of the cities and the starting city (SC) play an important role in run times; the hypothesis has been corroborated. However, an open issue remains for discussion and solution: how to relate a pattern (in general) with a numerical value which means execution time. This relationship establishes a numerical characterization of patterns. On this basis, an original hypothesis was formulated. At this point, the asymptotic time complexity for the *GP-TSP* was redefined as $O(P, C, pattern, SC)$.

[3] Third hypothesis: sum of distances and mean deviation of sum of distances

What parameters could be used to quantitatively characterize different geographical patterns in the distribution of cities? In graph theory, the distance of a vertex p, $d(p)$, of such a connected graph G is defined by $d(p) = \sum d(p, q)$ where $d(p, q)$ is the distance between p and q and the summation extends over all vertices q of G. This measure is an inverse measure of centrality. At this stage, the worked inputs are the sum of the distances from one city (x, y) to the other cities (SD, as shown on Eq. 2), and the mean deviation of SDs values ($MDSD$).

$$SD = \sum_{i=1}^{C} \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{2}$$

The greater is the sum of the distances, the lower is the centrality. If a particular city is very far from the others, its SD will be considerably greater than the rest and consequently the execution time will also increase. This can be observed in Table 2. Therefore, the SD value is an index time.

Table 2 *GP-TSP* execution times (ET, in sec.) and sum of the distances (SD) by pattern ($G1...G5$) and starting city (SC, $C1...C15$).

	Pattern									
	$G1$		$G2$		$G3$		$G4$		$G5$	
SC	ET	SD	ET	SD	ET	SD	ET	SD	ET	SD
$C1$	216.17	853.94	36.50	746.10	15.34	664.60	10.51	643.75	8.03	148.74
$C2$	214.44	887.44	36.82	740.49	15.19	649.14	10.49	635.54	7.82	104.16
$C3$	77.25	315.51	38.09	820.63	15.57	707.70	10.02	555.70	7.71	141.15
$C4$	72.64	230.11	37.29	789.80	15.02	678.07	10.30	599.99	7.91	103.35
$C5$	70.94	226.88	18.54	345.83	15.84	643.65	10.41	611.45	7.83	111.79
$C6$	74.21	244.56	17.83	330.76	15.24	638.04	10.24	595.58	7.71	102.81
$C7$	75.59	276.09	18.16	369.56	10.31	467.99	10.36	592.68	7.93	111.28
$C8$	73.72	294.62	18.03	383.38	10.34	490.55	10.26	639.61	7.87	147.14
$C9$	69.47	233.53	17.79	370.10	10.27	491.52	9.98	574.23	8.14	123.19
$C10$	74.96	234.84	17.48	323.12	10.23	446.48	9.88	578.78	8.22	172.52
$C11$	75.89	259.19	17.07	332.87	10.24	477.42	9.85	544.61	8.04	124.64
$C12$	70.17	234.22	17.39	325.19	10.28	449.03	9.87	534.91	8.12	131.68
$C13$	73.73	306.99	18.10	383.11	10.36	504.79	9.88	530.72	7.98	109.78
$C14$	70.87	239.19	17.37	327.02	10.17	451.21	9.95	574.97	8.02	124.96
$C15$	73.30	295.27	18.00	372.00	10.32	494.09	9.97	534.36	7.78	96.29
		$MDSD$		$MDSD$		$MDSD$		$MDSD$		$MDSD$
		140.94		165.47		90.60		31.56		16.78

Why is it needed to consider *MDSD* in addition to SD as a significant parameter? Quite similar SD values from the same pattern (same column) of Table 2 imply similar execution times. In $G1$, the SD values for starting city $C4$ and $C10$ are 230.11 and 234.84, respectively. Their execution times (ET) are similar, 72.64 and 74.96 seconds respectively. But, this relation is not true considering similar SD values from different patterns (different experiments). The SD value for $G1$ and starting city $C3$, and the SD value for $G2$ and starting city $C10$ are similar (315.51 and 323.12, respectively) but the execution times are considerably different. The distinct values of *MDSD* for $G1$ and $G2$ explain the variation of execution times for these similar *SD* values.

Conclusions: asymptotic parallel time complexity for the *GP-TSP* algorithm should be defined as $O(P, C, SD, MDSD)$. A new fact was discover in the process. By choosing the city which has minimum SD associated value, it is possible to obtain the exact *TSP* solution investing less amount of time. Much better results it would be reached if the algorithm begins considering the closer L cities to that city.

3.5 Prediction of GP-TSP performance order

To start, the concepts of geometric pattern matching under Euclidean motion were applied to measure the mismatch between a new data set and historical patterns. The Hausdorff distance as a function of relative position was the distance used [19]. Then, the comparison module was definitely replaced by an identification number in order to make the prediction. Comparing graphics patterns provides a higher level of uncertainty respect of using a good analytical expression.

The redefinition of the asymptotic parallel time complexity for the non-deterministic *TSP* algorithm represents a significant qualitative change. At this moment, the *GP-TSP* has a time complexity of $O(P, C, SD, MDSD)$. The analytical formulation allows making predictions for a new data set on a particular parallel computer. Fig. 9 shows the prediction framework.

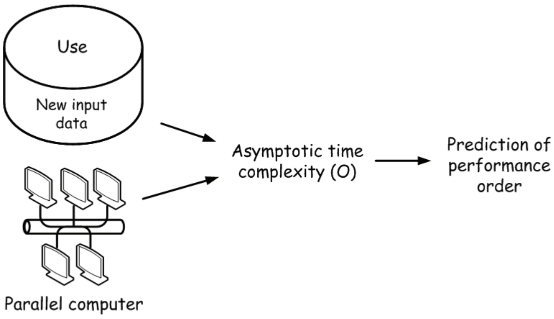

Use
New input data
Parallel computer
Asymptotic time complexity (O)
Prediction of performance order

Fig. 9 The prediction of performance order framework.

3.6 Two relevant GP-TSP experiments

Additional *TSP* experiments have been tested to prove certain hypotheses. Tests show how important the geographical patterns of cities are in comparison to knowing only their coordinates. Two groups of experiments which follow a specific pattern each one have helped to confirm the strong compliance of our hypotheses. Both groups are presented to illustrate this section.

[1] Importance of the geographical pattern

Make geometric transformations (shifting, scaling, and rotation) to patterns is without a doubt a trivial test. This is an excellent case study to understand the importance of geographical patterns. Applying each one of the transformations to a cities set, similar times are expected using the same algorithm. This leading to conclude, the time required to reach the solution of the *GP-TSP* algorithm is constant to certain transformations into the geographical patterns.

- The coordinates of a city shifted by $\triangle x$ in the x-dimension and $\triangle y$ in the y-dimension are given by

$$x' = x + \triangle x \qquad y' = y + \triangle y \tag{3}$$

where x and y are the original and x' and y' are the new coordinates.

- The coordinates of a city scaled by a factor S_x in the x-direction and y-direction (the distances among cities are enlarged when S_x is greater than 1 and reduced when S_x is between 0 and 1) are given by

$$x' = xS_x \qquad y' = yS_x \tag{4}$$

- The coordinates of a city rotated through an angle θ about the origin of the coordinate system are given by

$$x' = x\cos\theta + y\sin\theta \qquad y' = -x\sin\theta + y\cos\theta \tag{5}$$

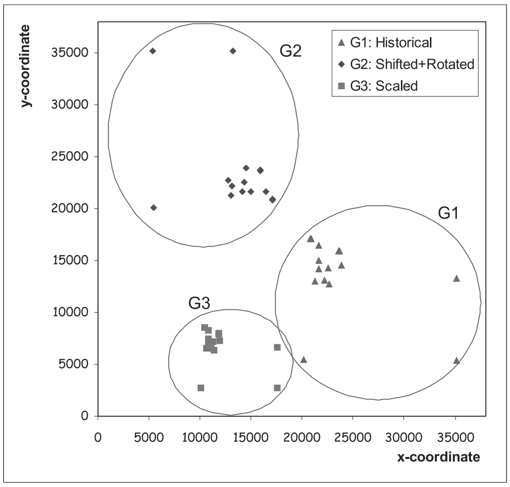


Fig. 10 A historical pattern ($G1$) consisting of 15 cities. Besides, the same pattern shifted and rotated ($G2$) and then the pattern scaled ($G3$).

An example set consisting of 15 cities is chosen from the historical database ($G1$). The shifting and rotation transformations are obtained interchanging x-coordinate by y-coordinate ($G2$), and the scaling transformation dividing by 2 both coordinates ($G3$), see Fig 10.

Table 3 shows the *GP-TSP* execution times using 32 nodes of the parallel computer for each pattern and starting city. Analyzing the values by row, the historical execution times and the execution times of the geometric transformations for a sample (row) are quite similar as it was to be expected. For all cases, the mean deviation of execution times is smaller than 2%.

Table 3 *GP-TSP* execution times (ET, in sec.) by patterns ($G1$, $G2$, $G3$) and starting city (SC, $C1...C15$).

	Pattern			
	$G1$	$G2$	$G3$	Mean
SC	ET	ET	ET	deviation
$C1$	46.25	48.52	47.30	0.78
$C2$	100.30	105.60	102.77	1.81
$C3$	73.48	76.34	74.52	1.04
$C4$	32.92	34.52	33.75	0.54
$C5$	30.83	31.96	31.35	0.39
$C6$	30.49	31.92	31.22	0.48
$C7$	31.77	33.00	32.21	0.45
$C8$	30.10	31.06	30.43	0.35
$C9$	31.08	32.13	31.92	0.42
$C10$	30.98	32.24	31.60	0.42
$C11$	29.94	31.09	30.36	0.42
$C12$	30.33	31.53	30.85	0.42
$C13$	31.45	32.82	32.14	0.46
$C14$	32.67	33.44	32.53	0.37
$C15$	32.49	33.49	32.89	0.36

[2] Limit cases

A singular case is to have the cities uniformly distributed in a circumference. As the *MDSD* value will be close to 0, similar execution times are expected. The idea is considering a limit case in order to confirm the hypothesis with respect to the *MDSD* value and its pattern.

Scaling circumference patterns where cities are uniformly distributed imply obtaining similar execution times. Table 4 shows a comparative behavior study of different circumference patterns ($G1$, ..., $G10$) and starting city ($C1...C16$, $C1...C17$,..., $C1...C25$) applying the *GP-TSP* algorithm using 32 nodes. The difference between each pattern is the number of cities uniformly distributed; from 16 ($C1...C16$ for $G1$) to 25 ($C1...C25$ for $G10$). As presented in Table 4, there is a progressive increase in the mean execution times. For every pattern and its derived geometric patterns, the execution times were quite similar starting by each city. The mean deviations of execution times were smaller than 1%.

Table 4 Mean and mean deviation of execution times (in sec.) by pattern and starting cities.

	Pattern				
	$G1$ ($C1...C16$)	$G2$ ($C1...C17$)	$G3$ ($C1...C18$)	$G4$ ($C1...C19$)	$G5$ ($C1...C20$)
Mean	17.47	23.42	31.87	42.95	54.94
MDev	0.04	0.08	0.08	0.07	0.10

	Pattern				
	$G6$ ($C1...C21$)	$G7$ ($C1...C22$)	$G8$ ($C1...C23$)	$G9$ ($C1...C24$)	$G10$ ($C1...C25$)
Mean	68.67	129.53	367.29	1085.57	2957.15
MDev	0.10	0.11	0.30	2.12	3.03

§4 Conclusion

This article shows that a knowledge discovery methodology based on simplicity, including only the major factors in performance, can define the asymptotic parallel time complexity of data-dependent algorithms with useful accuracy. Nevertheless, it is important to understand that the performance of a parallel application is resultant from at least factors of algorithm, implementation, underlying processor architecture, and interconnected technologies.

In short, the general knowledge discovery methodology begins by designing a considerable number of experiments and measuring their execution times. A well-designed test guides the researcher in choosing what experiments actually need to be performed in order to provide a representative sample. A data-mining tool then explores these collected data in search of patterns and/or relationships detecting the main parameters that affect performance. Knowing the key parameters which characterise performance, it becomes possible to hypothesise to restart the process and to produce a subsequent improved time complexity model. Finally, the methodology predicts the performance order for new data sets on a particular parallel computer by replacing a numerical identification.

As a case of study, a global pruning *TSP* (named *GP-TSP*) algorithm has been examined. It has been used to analyze the influence of indeterminism in performance prediction and also to show the practicality and the advantages of the methodology. The asymptotic time complexity for the parallel *GP-TSP* algorithm depends on the number of processors (P), the number of cities (C), and other parameters. As a result of the investigation, right now the sum of the distances from one city to the other cities (*SD*) and the mean deviation of these

SD values (*MDSD*) are the numerical parameters characterizing the different input data beyond the number of cities (C). The followed to discover these significant input parameters has been exhaustively described.

This work encourages further research and testing, as well as how to provide automatic useful feedback in order to assess more studies and experiments. The existence of more parameters that affect the performance of a data-dependent algorithm may suggest strategies to fit their asymptotic time complexity.

References

1) Alizadeh, F., Karp, R. M., Newberg, L. A. and Weisser, D. K., "Physical Mapping of Chromosomes: a Combinatorial Problem in Molecular Biology", in *SODA '93: Proc. of the 4th Annual ACM-SIAM Symposium on Discrete algorithms*, isbn: 0-89871-313-7, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 371-381, 1993.

2) Allender, E., Baul, M., Immerman, N., Schnoor, H. and Vollmer, H., "The Complexity of Satisfiability Problems: Refining Schaefer's Theorem", *J. Comput. Sys. Sci, 75*, pp. 245-254, 2009.

3) Applegate, D. L., Bixby, R. E., Chvatal, V. and Cook, W. J., "The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)" *Princeton University Press*, 2007.

4) Arkin, E. M., Chiang, Y., Mitchell, J. S. B., Skiena, S. S. and Yang, T., "On the Maximum Scatter TSP", in *SODA '97: Proc. of the 8th Annual ACM-SIAM Symposium on Discrete algorithms*, isbn: 0-89871-390-0, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 211-220, 1997.

5) Balas, E., "The Prize Collecting Traveling Salesman Problem", *Networks, 19*, pp. 621-636, 1989.

6) Barvinok, A., Fekete, S., Johnson, D., Tamir, A., Woeginger, G. and Woodroofe, R., "The Geometric Maximum Traveling Salesman Problem", *Journal of the ACM, New York, NY, USA, 50, 5*, pp. 641-664, 2003.

7) Biggs, N., Lloyd, E. K. and Wilson, R. J., "Graph Theory, 1736-1936", isbn: 0-198-53916-9, Clarendon Press, New York, NY, USA, 1986.

8) Bland, R. E. and Shallcross, D. F., "Large Traveling Salesman Problems Arising from Experiments in X-ray Crystallography: a Preliminary Report on Computation", *Operations Research Letters, 8*, pp. 125-128, 1989.

9) Burge, W. H., "Sorting Trees and Measures of Order", *Information and Control, 1*, pp. 181-197, 1958.

10) Chin, D. A., "Complexity Issues in General Purpose Parallel Computing", *University of Oxford*, 1991.

11) Christofides, N., "Vehicle Routing", The Traveling Salesman Problem, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., John Wiley, pp. 431-448, 1985.

12) Estivill-Castro, V. and Wood, D., "A Survey of Adaptive Sorting Algorithms", *ACM Comput. Surv., 24, 4*, 1992.

13) Garey, M. R., Graham, R. L. and Johnson, D. S., "Some NP-complete Geometric Problems", in *STOC '76: Proceedings of the 8th Annual ACM Symposium on Theory of computing*, pp. 10-22, 1976.

14) Geraerts, R. and Overmars, M. H., "Reachability Analysis of Sampling Based Planners", in *IEEE International Conference on Robotics and Automation*, John Wiley & Sons, Inc., pp. 406-412, 2005.

15) Gilmore, P. C. and Gomory, R. E., "Sequencing a One-State-Variable Machine: A Solvable Case of the Traveling Salesman Problem", *Operations Research, 12, 5*, pp. 655-679, 1964.

16) Fritzsche, P., "¿Podemos Predecir en Algoritmos Paralelos No Deterministas?", Computer Architecture and Operating Systems Department, University Autonoma of Barcelona, Spain, `http://caos.uab.es/`, 2007.

17) Golden, B. L., Levy, L. and Vohra, R., "The Orienteering Problem", *Naval Research Logistics, 34*, pp. 307-318, 1987.

18) Groth, R., "Data Mining: a Hands-on Approach for Business Professionals", Prentice Hall PTR, 1998.

19) Huttenlocher, D., Klanderman, G. and Rucklidge, W., "Comparing Images Using the Hausdorff Distance", *IEEE Transactions on Pattern Analysis and Machine Intelligence, 15, 9*, pp. 850-863, 1993.

20) Jarrard, R. D., "Scientific Methods", an online book, *Dept. of Geology and Geophysics, University of Utah*, 2001.

21) Johnson, D. S. and McGeoch, L. A., "The Traveling Salesman Problem: A Case Study in Local Optimization", in *Local Search in Combinatorial Optimization*, pp. 215-310, 1997.

22) Johnson, D. S. and McGeoch, L. A., "Experimental Analysis of Heuristics for the STSP", isbn: 978-1-4020-0664-7, in *The Traveling Salesman Problem and Its Variations, 12*, pp. 369-443, 2004.

23) Kolliopoulos, S. G. and Steiner, G., "Partially Ordered Knapsack and Applications to Scheduling", *Discrete Applied Mathematics, 155, 8*, pp. 889-897, 2007.

24) Korostensky, C. and Gonnet, G. H., "Using Traveling Salesman Problem Algorithms for Evolutionary Tree Construction", *BIOINF: Bioinformatics, 16, 7*, pp. 619-627, 2000.

25) Lenstra, J. K. and Kan, A. R., "Some Simple Applications of the Travelling Salesman Problem", *Operations Research Quarterly, 26, 4*, pp. 717-732, 1975.

26) MacQueen, J. B., "Some Methods for Classification and Analysis of MultiVariate Observations", in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. Le Cam and J. Neyman, University of California Press, 1, pp. 281-297, 1967.

27) Martello, S. and Toth, P., "Knapsack Problems: Algorithms and Computer Implementations", isbn: 0-471-92420-2, John Wiley & Sons, Inc., New York, NY, USA, 1990.

28) Miller, D. and Pekny, J., "Exact Solution of Large Asymmetric Traveling Salesman Problems", *Science, 251*, pp. 754-761, 1991.

29) Miller, R. and Boxer, L., "Algorithms Sequential and Parallel: A Unified Approach", isbn: 1-58450-412-9, Charles River Media, Computer Engineering Series, 2005.

30) Ratliff, H. D. and Rosenthal, A. S., "Order-Picking in a Rectangular Warehouse: A Solvable Case for the Traveling Salesman Problem", *Operations Research, 31, 3*, pp. 507-521, 1983.

31) Sankoff, D. and Blanchette, M., "The Median Problem for Breakpoints in Comparative Genomics", in *COCOON '97: Proc. of the 3rd Annual International Conference on Computing and Combinatorics*, isbn: 3-540-63357-X, *Springer-Verlag*, London, UK, pp. 251-264, 1997.

32) Schaefer, T. J., "The Complexity of Satisfiability Problems", *STOC*, pp. 216-226, 1978.

33) Schrijver, A., "On the History of Combinatorial Optimization (till 1960)", CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, 1990.

34) Skiena, S. S., "The Algorithm Design Manual", isbn: 0-387-94860-0, Springer-Verlag New York, Inc., New York, NY, USA, 1998.

35) Takaoka, T., "A New Measure of Disorder in Sorting - Entropy", CATS, pp. 441-476, 1998.

36) TSP Page, `http://www.tsp.gatech.edu/history/`, 2008.