8

Applying Artificial Intelligence to Predict the Performance of Data-dependent Applications

Paula Fritzsche, Dolores Rexachs and Emilio Luque
DACSO, University Autonoma of Barcelona
Spain

1. Introduction

Computational science (CS) is an emerging discipline that unites science and mathematics with disciplinary experience in biology, chemistry, physics, and other applied scientific fields. Within the scientific method, it is often referred to as the third science paradigm, complementing both theoretical and laboratory science (Miller & Boxer, 2005). In this work, CS involves the collaboration of the computing discipline, the mathematical support represented by the knowledge discovery process and by the models, and the computing parallel environment capability. Central to this computational science problem is the performance prediction of data-dependent applications, as shown on Figure 1. By the way, CS allows doing things that were previously too difficult to do due to the complexity of the mathematics, the large number of calculations involved, or a combination of both. Therefore, new challenges are continuously arising although CS is still at an early stage of development.

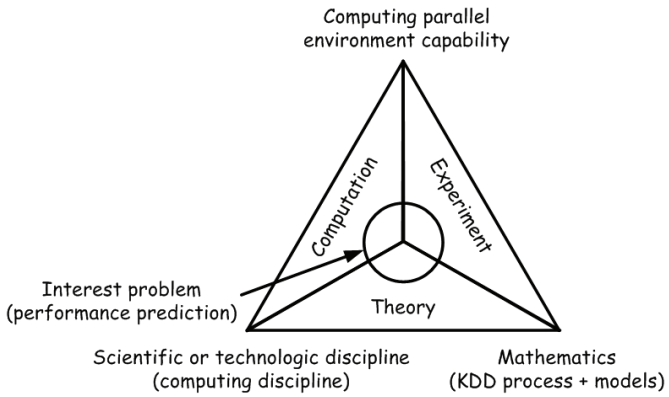


Figure 1. Computational science complementing both theoretical and laboratory science

Parallel computers provide an efficient and economical way to solve large-scale and/or time-constrained scientific, engineering, medicine, industry, and commerce problems. It is an alternative that makes easy to reach a solution in a fraction of the original time that would consume a single computer. Consequently, the research community, the computer

designers, the professional engineers, and the end-users of these systems have a vested interest in knowing and predicting the performance order of parallel algorithms. Although measuring the performance of a parallel algorithm for all possible input values would allow answering any question about how the algorithm will respond under any set of conditions, it is impossible to make it. The situation is even worse for data-dependent algorithms where similar input data sets may cause significant variability in execution times. For this kind of algorithms, the performance does not depend only on the number of processors used (P) and on the data size (N). Other parameters have to be taken into account, the values of which are data-dependent. Great examples of this type of programs are the sorting algorithms, the searching algorithms, the satisfiability problem, the graph partition, the knapsack problem, the bin packing, the motion planning, and the traveling salesman problem (TSP). Furthermore, there are important cases of practical problems that can be formulated as TSP problems and many other problems are generalizations of this problem.

The goal of this chapter is to present a general novel methodology to the problem of predicting the performance of data-dependent algorithms. This is a good starting point for understanding some facts related with the non-deterministic algorithms. Briefly, the methodology works as follows. It begins by designing a certain number of instances and measuring their execution times. A well-designed instance guides the experimenters in choosing what experiments actually need to be performed in order to provide a representative sample. A data mining process then explores the collected data in search of patterns and /or relationships detecting the main parameters that affect performance. These common properties are modelled numerically so as to generate an analytical formulation of the execution time, a multiple-linear-regression model. Finally, the regression equation allows predicting how the algorithm will perform when given new input data sets.

A global pruning algorithm (*GP-TSP*) is used to analyze the influence of indeterminism in performance prediction, and also to show the usefulness of the proposed methodology. It is a branch-and-bound algorithm which recursively searches all possible paths and prunes large parts of the search space by maintaining a global variable containing the length of the shortest path found so far. 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.

The *GP-TSP* execution time depends on the number of processors (P), number of cities (C), and other parameters. As a result of this investigation, right now the sum of the distances from one city to the 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.

The preliminary experimental results of predictions are quite promising. An important fact has been reached beyond was originally sought. Choosing the city which has minimum *SD* associated value, it is possible to obtain the exact TSP solution investing less amount of time.

This chapter is organized as follows. The next section presents the novel methodology to the problem of predicting the performance of data-dependent algorithms. Section 3 reviews the traveling salesman problem (TSP) and provides detailed coverage of a parallel TSP implementation called *GP-TSP*. Section 4 focuses on the discovering process carried out to find the significant input parameters for the *GP-TSP* algorithm. Section 5 explains how to build a prediction model and then the evaluation process in order to estimate times. Finally, Section 6 summarizes and draws the main conclusions of this chapter identifying challenging future research.

2. Entire approach

The general novel methodology attempts to estimate the performance order of a parallel algorithm that solves a data-dependent problem. The defined methodology consists of three main phases: the design and composition of experiments to obtain and fit the prediction model, the validation of the model, and the use of the model developed, see Figure 2.

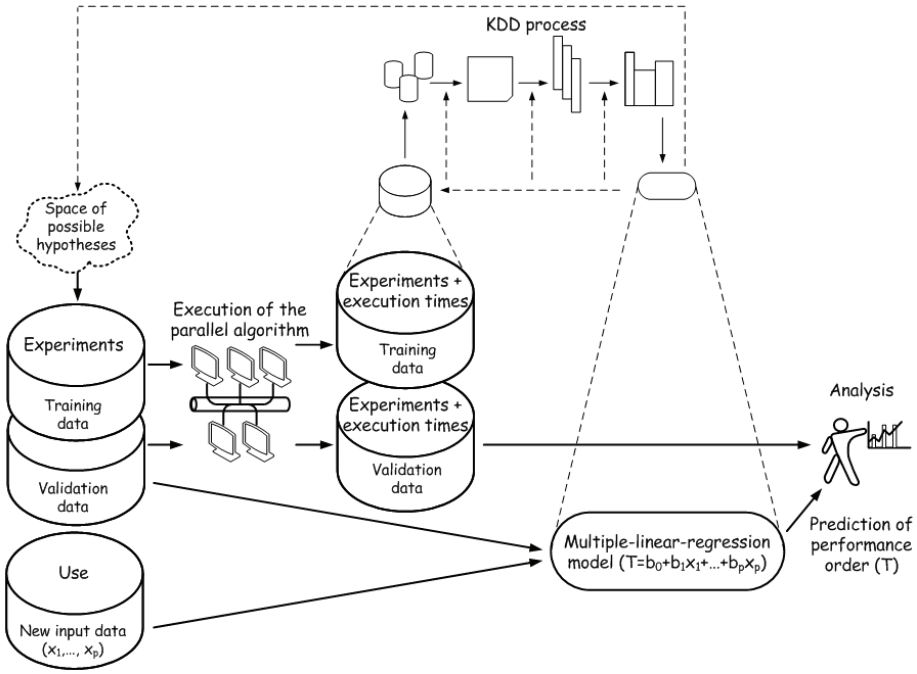


Figure 2. The performance prediction methodology

2.1 Design and composition of experiments to obtain and fit the prediction model

In principle, it is important to understand the application domain and the relevant prior knowledge, and to analyze their 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 algorithm studied. Discovering the proper set of parameters is the basis to obtain a good capacity of prediction. Including too many parameters may lead to an accurate but too complicated or even unsolvable model. Hence, great care should be taken in selecting parameters and a reasonable trade-off should be made.

Designing an 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. A well-designed instance guides the experimenters in choosing what experiments actually need to be assessed. Once a training data has been defined, the studied parallel algorithm reads and processes the experiments one by one obtaining their execution times.

A data mining application analyzes the quantitative measured values of the main parameters that affect performance and summarizes these into a useful multiple-linear-

regression model (MLR model, $T=b_0+b_1x_1+\ldots+ b_px_p$). It allows including the effects of several input variables that are all linearly related to a single output variable (T). This is a first approximation to deal with the problem. Figure 3 shows the knowledge construction model. Note that the instances must provide a representative sample (a training data set) first to obtain and fit the model and then to estimate the regression coefficients.

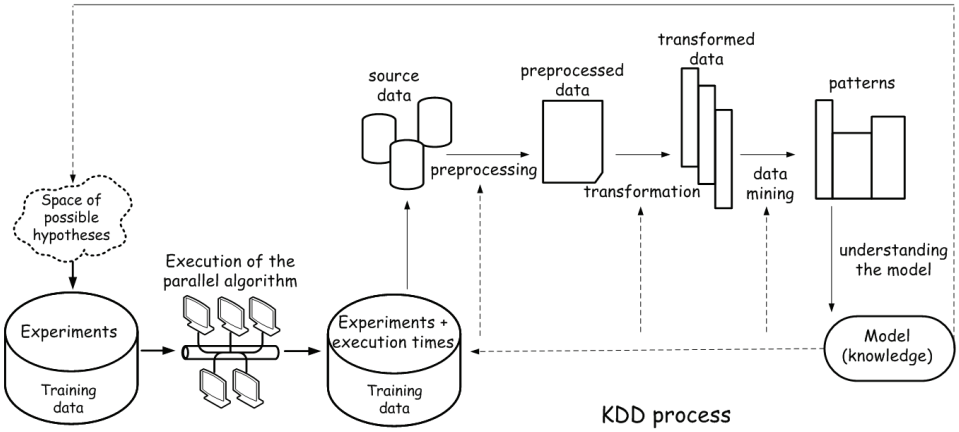


Figure 3. The knowledge construction model

2.2 Validation of the model

A new data set is used to be able to validate the created model. The validation data set constitutes a hold-out sample and is not used in building of the model. This enables to estimate the error in the predictions without having the assume that the execution times follow a particular distribution.

The training data set is used to estimate the regression coefficients ($b_0,\ldots,b_p$). These coefficients are used to make predictions for each case in the validation data. The quality analysis is a relevant issue in this stage and has to include interest measurements. The prediction for each case is then compared to the value of the dependent variable that was actually observed in the validation data obtaining the prediction error. The average of the square of this error enables to compare different models and to assess the accuracy of the model in making predictions. Figure 4 exhibits the model validation phase.

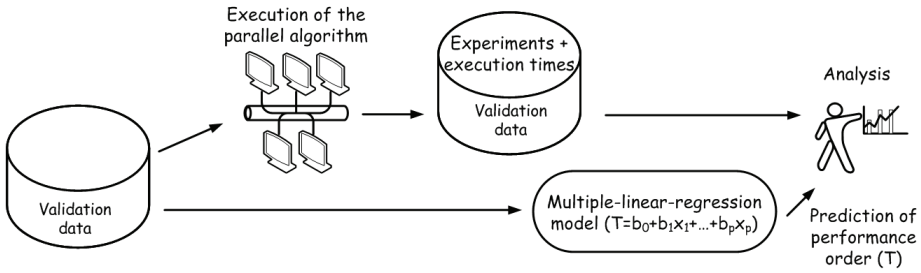


Figure 4. The validation of the model

Great care should be taken in analyzing the first approximations because it is difficult to know the degree of complexity of the relationship between the parameters and execution time. It is important to take in mind that the model aids in testing hypothesis and finding solution to performance prediction problems.
From the scientific point of view is essential to find confidence intervals for the regression parameters to provide some indication of how well they model the measured values. Taking this as a basis, it could determine the necessary number of elements in the sample.

2.3 Prediction of performance order

Once a MLR model has been fit, it is used to predict how the studied parallel algorithm will perform when given a new input data set. The $b_0,\ldots,b_p$ values are the estimated regression parameters. To predict the dependent value (T), it is necessary to replace the independent values $x_1,\ldots,x_p$ with known values.
At this point, it is necessary to emphasize that the MLR model provides a prediction framework easy to use and useful, see Figure 5.

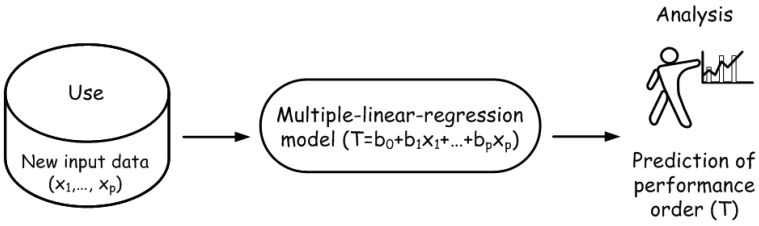


Figure 5. The prediction framework

3. Traveling salesman problem

The traveling salesman problem (TSP) is one of the most famous problems (and the best one perhaps studied) in the field of combinatorial optimization. In spite of the apparent simplicity of its formulation, the TSP is a complex data-dependent. Not only the complexity of its solution has been a continue challenge to the researchers but also the prediction of its performance due to there are many practical problems that can be formulated as TSP problems.

3.1 Problem statement

The TSP for C cities is the problem of finding a tour visiting all the cities exactly once and returning to the starting city such that the sum of the distances between consecutive cities is minimized (TSP, 2008). The requirement of returning to the starting city does not change the computational complexity of the problem.

3.2 TSP computational complexity

The TSP has been shown to be NP-hard (Karp, 1972). More precisely, it is complete for the complexity class (FP^{NP})[1], and the decision problem version is NP-complete. If an efficient

[1] The class NP is the set of decision problems that can be solved by a non-deterministic Turing machine in polynomial time. FP means function problems.

algorithm is found for the TSP problem, then efficient algorithms could be found for all other problems in the NP-complete class. Although it has been shown that, theoretically, the Euclidean TSP is equally hard with respect to the general TSP (Garey et al., 1976), it is known that there exists a sub exponential time algorithm for it.
The most direct solution for a TSP problem would be to calculate the number of different tours through *C* cities. Given a starting city, it has *C-1* choices for the second city, *C-2* choices for the third city, etc. Multiplying these together it gets *(C-1)!* for one city and *C!* for the *C* cities. Another solution is to try all the permutations (ordered combinations) and see which one is cheapest. At the end, the order is also factorial of the number of cities. Generally, the presented solutions are quite similar.

3.3 TSP practical problems

Besides the drilling of printed circuits boards (Duman, 2004), transportation and logistics areas (TSP, 2008), problems having the TSP structure occur in the analysis of the structure of crystals (Bland & Shallcross, 1989), in material handling in a warehouse (Ratliff & Rosenthal, 1983), in clustering of data arrays (Lenstra & Kan, 1975), in sequencing of jobs on a single machine (Gilmore & Gomory, 1964), in physical mapping problems (Alizadeh et al., 1993), in genome rearrangement (Sankoff & Blanchette, 1997), and in phylogenetic tree construction (Korostensky & Gonnet, 2000) among others. Related variations on the TSP include the resource constrained traveling salesman problem which has applications in scheduling with an aggregate deadline (Miller & Pekny, 1991). The prize collecting TSP (Balas, 1989) and the orienteering problem (Golden et al., 1987) are special cases of the resource constrained TSP. The problem of finding a tour of maximum length is the objective in MAX TSP (Barvinok et al., 2003). 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. It is motivated by applications in manufacturing and medical imaging (Arkin et al., 1996). Most importantly, the TSP often comes up as a subproblem in more complex combinatorial problems, the best known and important one of which is the vehicle routing problem, that is, 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 (Christofides, 1985).

3.4 *GP-TSP* algorithm

An implementation, called global pruning algorithm (*GP-TSP*), to obtain the exact TSP Euclidean solution in a parallel machine has been used. For simplicity of implementation, they were considered cities in R^2 instead of R^n. The most straightforward way of computing distances between cities in a two-dimensional space is to compute Euclidean distances. Anyway, the election of distance measure (Euclidean, Manhattan, Chebychev) is irrelevant. Also would be the same to work with an equivalent formulation in terms of graph theory. This is 'given a complete weighted graph (where the vertices would represent the cities, the edges would represent the roads, and the weights would be the cost or distance of that road), find a Hamiltonian circuit with the least weight' (Gutin & Punnen, 2006). Therefore, the ideas of this paper can be generalized.
The *GP-TSP* algorithm analyzes the influence of indeterminism in performance prediction. It is a branch-and-bound algorithm which recursively search all possible paths. It follows the Master-Worker programming paradigm (Fritzsche, 2007). Each city is represented by two

coordinates in the Euclidean plane. Considering C different cities, the Master defines a certain level L to divide the tasks. Tasks are 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 start-up the Master sends the cities coordinates to every Worker.
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 Figure 6. The Master can divide this problem into 1 task of level 0 or 4 tasks of level 1 or 12 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. The requirement of returning to the starting city is without detracting from the generality. In this closed cycle the salesman may begin and end in the city who wants.

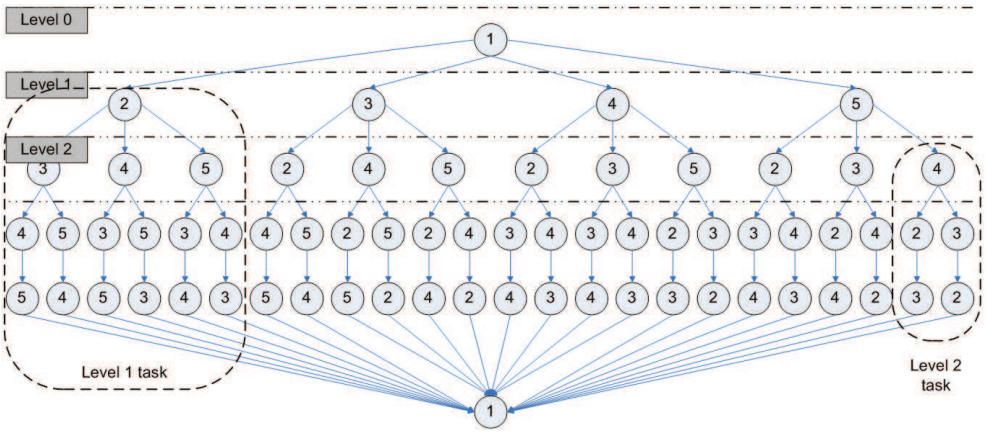


Figure 6. Possible paths for the salesman considering 5 cities

Workers are responsible for calculating the distance of the permutations left in the task and sending to the Master the best path and distance of these permutations. One of the characteristics of the TSP is that once the distance for a path is superior to the already computed minimum distance it is possible to prune this path tree.

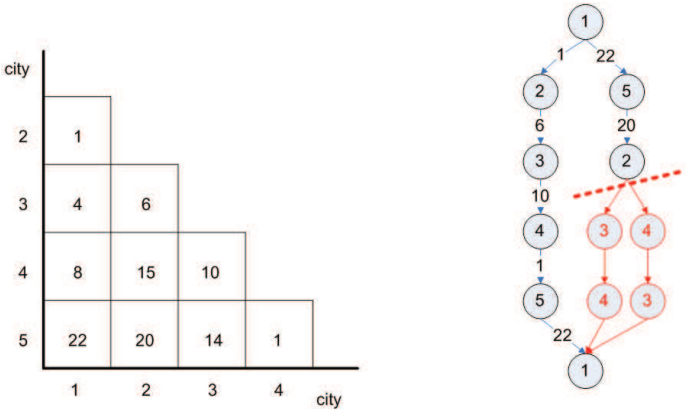


Figure 7. (a) Matrix of distances between cities (b) Pruning process in the *GP-TSP* algorithm

Figure 7(a) shows a strictly lower triangular matrix of distances; meanwhile Figure 7(b) exhibits the pruning process for the *GP-TSP* algorithm where each arrow has the distance between the two cities it connects. Analyzing Figure 7(b), the total distance for the first followed 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 algorithm to keep calculating distances from the city 2 on because it is impossible to reach a better distance for this branch.

4. Discovering the significant *GP-TSP* input parameters

It is clear that the *GP-TSP* execution time order depends on the number of processors (*P*), on the number of cities (*C*), and 'other parameters'. Discovering the 'other parameters' is the key to obtain a good or an acceptable prediction of performance order. Undoubtedly, the knowledge discovery in databases process (KDD process) has been one of the most profitable stages in the scientific examination. A huge amount of data sets was processed with the only goal of finding some common properties. First intuitions guided the different tests in order to determine the characteristics, the relationships, and the patterns between the data sets. 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 *SDs* values (*MDSD*) are the numerical parameters characterizing the different input data beyond the number of cities (*C*). But how these final parameters have been obtained? Next, it is described the followed way to discover the above mentioned dependencies (*SD* and *MDSD*), the construction of a model, and finally the evaluation of the obtained regression equation.

4.1 First hypothesis → location of the cities (geographical pattern)

For simplicity, only a particular training data set is analyzed and shown along different sections. It consists of five different geographical patterns of fifteen cities each one (named *G1* to *G5*). Figure 8 illustrates the five patterns handled at the beginning.

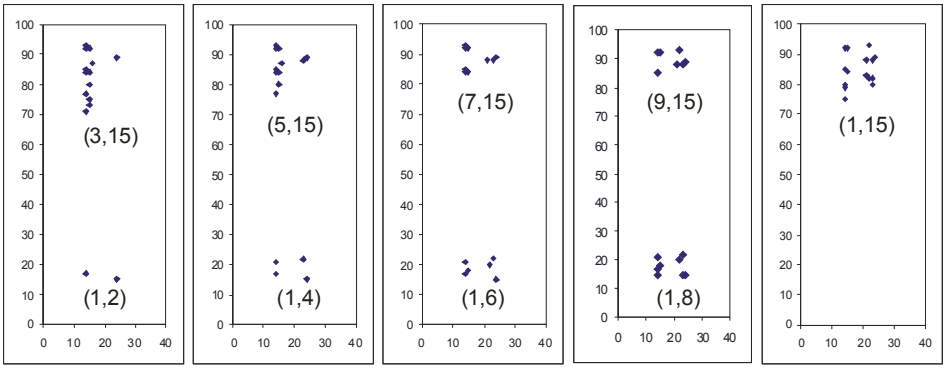


Figure 8. Five patterns defined for fifteen cities

The *GP-TSP* implementation receives the number of cities and their coordinates, and the level as input parameters. In order to find the shortest path, it proceeds recursively searching all possible paths and applying the global pruning strategy whenever it is feasible.

Hence before continuing, there are two important concepts to refresh. The main objective of data mining is finding useful patterns and knowledge in data. Clustering is one of the major data mining techniques, grouping objects together into clusters that exhibit internal cohesion (similar execution time) and external isolation.
As depicted in Figure 9, five clusters were found using a k-means algorithm (MacQueen, 1967) included in Cluster-Frame environment, see Appendix B for extra information. The idea was to obtain quite similar groups with respect to the groups (patterns) used at the beginning. The initial centroids were randomly selected by the clustering application and the squared error function, Equation (1), was the selected objective function

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

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 entire function is an indicator of the distance of the n data points from their respective cluster centroids.

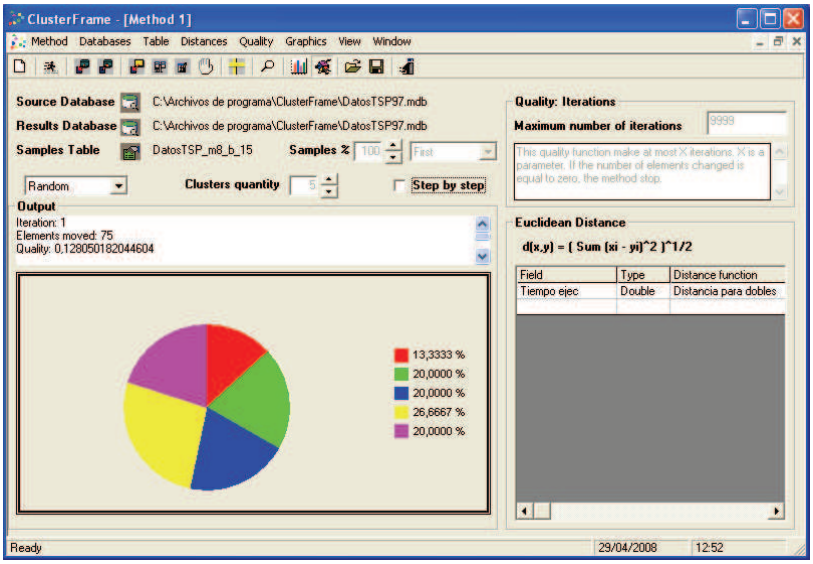


Figure 9. Cluster-Frame environment

Table 1 presents the obtained *GP-TSP* execution times (in sec.) by pattern (columns *G1* to *G5*) and starting city using 8 nodes of the parallel machine described in Appendix A. Columns *Cl1*,.., *Cl5* show the assigned cluster for each sample after running k-means algorithm. For the clusters 1 to 5, the final centroids values were 92.22 sec., 16.94 sec., 37.17 sec., 10.19 sec., and 7.94 sec., respectively. A simple remark derived from pattern columns is that the execution times belonging to a group are quite similar except for some cases.
The quality evaluation involves the validation of the above mentioned hypothesis. For each sample, the assigned cluster was confronted with the previously defined graphic pattern. The percentage of hits expresses the capacity of prediction. A simple observation is that the execution times were clustered in a similar way to patterns fixed at starting; the capacity of

prediction was of 75% for this example (56 hits on 75 possibilities). There was a close relationship between the patterns and the execution times.

Starting city	*Pattern*									
	G1	*Cl1*	*G2*	*Cl2*	*G3*	*Cl3*	*G4*	*Cl4*	*G5*	*Cl5*
1	216.17	1	36.50	3	15.34	2	10.51	4	8.03	5
2	214.44	1	36.82	3	15.19	2	10.49	4	7.82	5
3	77.25	1	38.09	3	15.57	2	10.02	4	7.71	5
4	72.64	1	37.29	3	15.02	2	10.30	4	7.91	5
5	70.94	1	18.54	2	15.84	2	10.41	4	7.83	5
6	74.21	1	17.83	2	15.24	2	10.24	4	7.71	5
7	75.59	1	18.16	2	10.31	4	10.36	4	7.93	5
8	73.72	1	18.03	2	10.34	4	10.26	4	7.87	5
9	69.47	1	17.79	2	10.27	4	9.98	4	8.14	5
10	74.96	1	17.48	2	10.23	4	9.88	4	8.22	5
11	75.89	1	17.07	2	10.24	4	9.85	4	8.04	5
12	70.17	1	17.39	2	10.28	4	9.87	4	8.12	5
13	73.73	1	18.10	2	10.36	4	9.88	4	7.98	5
14	70.87	1	17.37	2	10.17	4	9.95	4	8.02	5
15	73.30	1	18.00	2	10.32	4	9.97	4	7.78	5
Mean	92.23		22.97		12.32		10.14		7.94	

Table 1. Execution times (in sec.) and assigned cluster for the *GP-TSP* algorithm

Conclusions: The initial hypothesis for the *GP-TSP* was corroborated. At this stage, the asymptotic time complexity was defined as *O(C, P, pattern)*. The capacity of prediction was greater than 70% for the full range of experiments worked. This value gave 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.

4.2 Second hypothesis → location of the cities and starting city

Comparing Figure 8 with Table 1 it is easy to infer some important facts. The two far cities in Figure 8(a) correspond with the two higher time values of Table 1(G1). The four far cities in Figure 8(b) correspond with the four higher execution time values of Table 1(G2). The six far cities in Figure 8(c) correspond with the six higher time values of Table 1(G3). The cities in Figure 8(d) are distributed among two zones so, the times turn out to be enough similar, see Table 1(G4). Finally, the cities in Figure 8(e) are enough closed so, the times are quite similar, see Table 1(G5).

An additional important observation is that the mean of execution times by pattern decreases as the cities approach (92.23, …, 7.94).

Conclusions: Without doubt, the location of the cities and the starting city (C_1) play an important role in execution times; the hypothesis was corroborated. At this point, the asymptotic time complexity for the *GP-TSP* was redefined as $O(C, P, pattern, C_1)$. Anyway,

an open issue remained for discussion and resolution, how to relate a pattern (in general) with the value of the execution time. This relationship would be able to establish a numerical characterization of patterns. On this basis, a new original hypothesis was formulated.

4.3 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? Right now for each pattern, the sum of the distances from one city to the other cities (SD_j), as shown on Equation (2) and the mean deviation of *SDs* values (*MDSD*) are the worked inputs.

$$\forall j : 1 \le j \le C \quad SD_j = \sum_{i=1}^{C} \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2} \tag{2}$$

In the following sub sections, three different kinds of experimentations are done. One of these is useful to see the necessity to include the both *SD* and *MDSD* parameters in the complexity expression. Another one proves that a pattern is univariate regardless of their scale or position. The last one is a singular case where the cities are uniformly distributed in a circumference.

4.3.1 Experimentation 1

Columns *SD1,.., SD5* in Table 2 show the values obtained by applying the Equation (2) to each pattern and starting city. If a particular city *j* is very remote of the others, its SD_j will be considerably greater to the rest and consequently the execution time will grow also.

Starting city	*Pattern*									
	G1	***SD1***	***G2***	***SD2***	***G3***	***SD3***	***G4***	***SD4***	***G5***	***SD5***
1	216.17	853.94	36.50	746.10	15.34	664.60	10.51	643.75	8.03	148.74
2	214.44	887.44	36.82	740.49	15.19	649.14	10.49	635.54	7.82	104.16
3	* **77.25**	* **315.51**	38.09	820.63	15.57	707.70	10.02	555.70	7.71	141.15
4	◊ **72.64**	◊ **230.11**	37.29	789.80	15.02	678.07	10.30	599.99	7.91	103.35
5	70.94	226.88	18.54	345.83	15.84	643.65	10.41	611.45	7.83	111.79
6	74.21	244.56	17.83	330.76	15.24	638.04	10.24	595.58	7.71	102.81
7	75.59	276.09	18.16	369.56	10.31	467.99	10.36	592.68	7.93	111.28
8	73.72	294.62	18.03	383.38	10.34	490.55	10.26	639.61	7.87	147.14
9	69.47	233.53	17.79	370.10	10.27	491.52	9.98	574.23	8.14	123.19
10	◊ **74.96**	◊ **234.84**	* **17.48**	* **323.12**	10.23	446.48	9.88	578.78	8.22	172.52
11	75.89	259.19	17.07	332.87	10.24	477.42	9.85	544.61	8.04	124.64
12	70.17	234.22	17.39	325.19	10.28	449.03	9.87	534.91	8.12	131.68
13	73.73	306.99	18.10	383.11	10.36	504.79	9.88	530.72	7.98	109.78
14	70.87	239.19	17.37	327.02	10.17	451.21	9.95	574.97	8.02	124.96
15	73.30	295.27	18.00	372.00	10.32	494.09	9.97	534.36	7.78	96.29
MDSD		140.94		165.47		90.60		31.56		16.78

Table 2. Execution times (in sec.) and sum of the distances for the *GP-TSP* algorithm

Why is it needed to consider *MDSD* in addition to *SD* as a significant parameter? Quite similar *SD* values from the same experiment (same column) of Table 2 imply similar execution times. The SD_1 values for the starting cities 4 and 10 are 230.11 and 234.84, respectively. Their execution times (*G1*) are similar 72.64 and 74.96 (labelled with the symbol ◊). Instead, this relation is not true considering similar *SD* values from different patterns (different columns). The SD_1 value for starting city 3 and the SD_2 value for the starting city 10 are similar (315.51 and 323.12, respectively) but the execution times are completely dissimilar (labelled with the symbol *). Therefore, the different $MDSD(SD_1)$ and $MDSD(SD_2)$ values explain the different execution times for similar SD_1 and SD_2 values.

4.3.2 Experimentation 2

Make geometric transformations (shifting, scaling, and rotation) to well-known patterns is a fundamental test. The idea is to prove that a given pattern is univariate regardless of their scale or position. Applying each one of the transformations to a data set, similar times are expected using the same algorithm.

The coordinates of a city shifted by Δx in the *x*-dimension and Δy in the *y*-dimension are given by

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

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 city is enlarged in size when S_x is greater than 1 and reduced in size when S_x is between 0 and 1) are given by

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

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

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

A data set consisting of fifteen cities is chosen from the historical database (Hist). The shifting and rotation transformations are obtained interchanging *x*-coordinate by *y*-coordinate (Sh+Rot), and the scaling transformation dividing by two both coordinates (Scaled). While Figure 10 shows these three patterns together, Table 3 exhibits a comparison of the execution times by pattern and starting city using 32 nodes of the parallel described in Appendix A. Analyzing the information by row, the historical execution times and the execution times of the geometric transformations for a sample are quite similar as it was to be expected. The mean deviations are smaller than 2%.

4.3.3 Experimentation 3

A singular case is to have the cities uniformly distributed in a circumference. The *MDSD* will be near to 0 so, similar times are expected applying any worked algorithm. Different patterns consisting of 15 to 24 cities have been studied. One of these circumferences which is composed of 24 cities is shown in Figure 11.

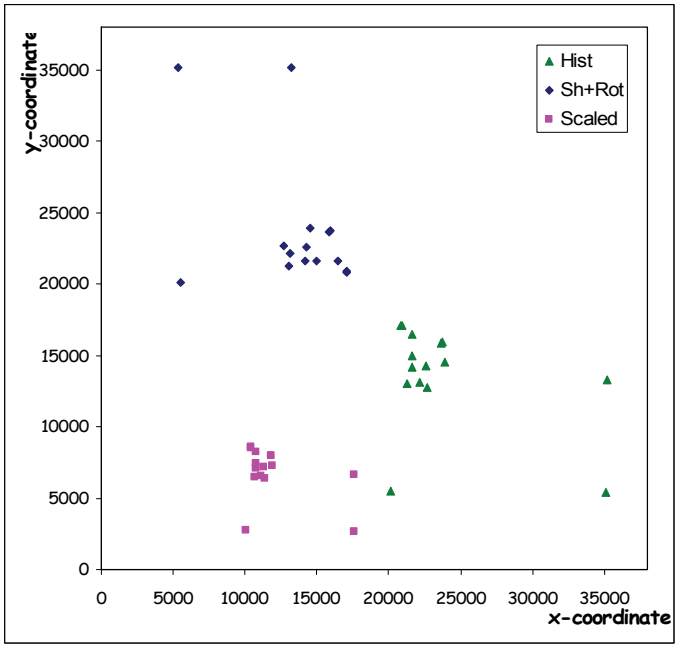


Figure 10. A historical pattern, a shifted and rotated historical pattern, and a scaled historical pattern consisting of fifteen cities

Starting city	*Pattern*			*MDev*
	Hist	*Sh+Rot*	*Scaled*	
1	46.25	48.52	47.30	0.78
2	100.30	105.60	102.77	1.81
3	73.48	76.34	74.52	1.04
4	32.92	34.52	33.75	0.54
5	30.83	31.96	31.35	0.39
6	30.49	31.92	31.22	0.48
7	31.77	33.00	32.21	0.45
8	30.10	31.06	30.43	0.35
9	31.08	32.13	31.92	0.42
10	30.98	32.24	31.60	0.42
11	29.94	31.09	30.36	0.42
12	30.33	31.53	30.85	0.42
13	31.45	32.82	32.14	0.46
14	32.67	33.44	32.53	0.37
15	32.49	33.49	32.89	0.36

Table 3. Comparison of execution times (in sec.) for the three patterns using 32 nodes

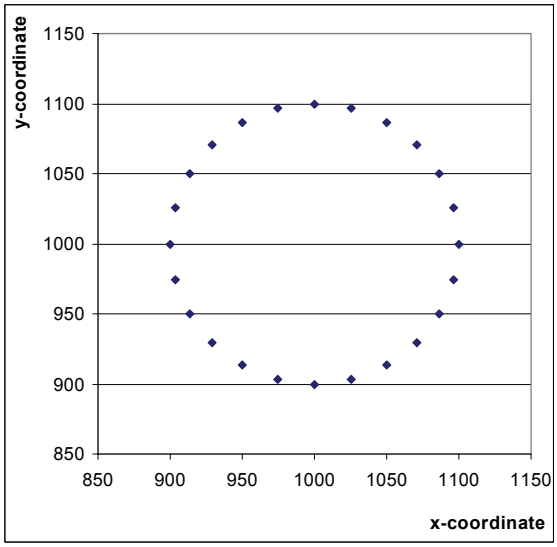


Figure 11. A circumference pattern composed of 24 uniformly distributed cities

Table 4 shows a comparative study of behaviour of different circumference patterns applying the *GP-TSP* algorithm. As it can be appreciated in Table 4, there is a minimum progressive increase in the times. It is remarkable that in every case, the mean deviations of execution times were smaller than 1%.

#Cities	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*
Mean	12.71	17.47	23.42	32.93	42.95	54.94	68.67	129.53	367.29	1085.57
Mean deviation	0.03	0.04	0.08	0.08	0.07	0.10	0.10	0.11	0.30	2.12

Table 4. Mean and mean deviation of execution times (in sec.) by number of cities for the *GP-TSP* algorithm using 32 nodes

Conclusions: It is important to emphasize that the *GP-TSP* algorithm obtains good results of prediction. Their asymptotic time complexity should be defined as *O(C, P, SD, MDSD)*. Another important fact has been reached beyond what was originally sought. Choosing the city which has minimum *SD* associated value, it is possible to obtain the exact TSP solution investing less amount of time.

5. Predicting the *GP-TSP* execution time

The *GP-TSP* algorithm has been executed for a great amount of training patterns in order to take enough experimental data to validate this experimental approach. At this point, the methodology views the algorithm being study as a black box in which the normalized measured values for the input variables *(C, P, SD, MDSD)* arrive, are processed, and then produce a MLR model. A desired normalization converts values to a common basis for comparison. It is important to take in mind that the MLR model is a first approximation to deal with the performance prediction problem.

5.1 Building a MLR model for the *GP-TSP* algorithm

There are four independent input variables (*C, P, SD, MDSD*) and the basis form of the four-dimensional regression model for the execution time (*T*) is

$$T = b_0 + b_1 C + b_2 P + b_3 SD + b_4 MDSD \tag{6}$$

where b_0, b_1, b_2, b_3, and b_4 are the regression parameters to estimate. There exist m measurements of the output T for various combinations of the inputs C, P, SD, and $MDSD$. Each measurement can be expressed as

$$T_i = b_0 + b_1 C_i + b_2 P_i + b_3 SD_i + b_4 MDSD_i + e_i \tag{7}$$

where e_i is the residual for the data (C_i, P_i, SD_i, $MDSD_i$, T_i).

To find the regression parameters, it is necessary to minimize the sum of squares of the residuals, denoted *SSE*.

$$SSE = \sum_{i=1}^{m} e_i^2 = \sum_{i=1}^{m} (T_i - b_0 - b_1 C_i - b_2 P_i - b_3 SD_i - b_4 MDSD_i)^2 \tag{8}$$

The Equation (8) takes on its minimum value when the partial derivatives of *SSE* with respect to b_0, b_1, b_2, b_3, and b_4 are all set to zero. This procedure then leads to a system of five equations. The solution could be found by using any of the standard methods for solving systems of equations, or using any available software package designed for this purpose (Lilja, 2000).

5.2 Evaluating the regression equation

Finally, the regression equation is used to predict how the *GP-TSP* algorithm will perform when given new input data sets. Replacing *C*, *P*, *SD*, and *MDSM* with real values in Equation (6), it is possible to estimate the time required (*T*) to find the shortest path for this master-worker global pruning TSP algorithm.

6. Conclusions

This chapter introduces a general novel methodology to estimate the performance order of data-dependent parallel algorithms. It is important to understand that the parallel performance achieved depends on several factors, including the application, the parallel computer, the data distribution, and also the methods used for partitioning the application and mapping its components onto the architecture.

Briefly, the general methodology works as follows. It begins by designing a certain number of instances and collecting their execution-time data. A well-designed instance guides the experimenters in choosing what experiments actually need to be performed in order to provide a representative sample. A data-mining process then explores these collected data in search of patterns and/or relationships detecting the main parameters that affect performance. These common properties are modelled numerically so as to generate an analytical formulation of the execution time. The methodology views the algorithm being study as a black box in which the measured values for this limited number of inputs arrive, are processed, and then produce a multiple-linear-regression model. Finally, the regression equation allows for predicting how the algorithm will perform when given new input data sets.

A TSP parallel implementation has been studied. The *GP-TSP* algorithm analyzes the influence of indeterminism in performance prediction, and also shows the usefulness and the profits of the methodology. Their execution time depends on the number of cities (*C*), the number of processors (*P*), 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 *SDs* values (*MDSD*) are the numerical parameters characterizing the different input data beyond the number of cities (*C*). The followed way to discover these proper set of parameters has been exhaustively described. Finally, their asymptotic time complexity has been defined *O(C, P, SD, MDSD)*.

Building a MLR model with the four independent input variables *(C, P, SD, MDSD)* and, then, using the regression equation, a prediction of performance order for a new data set it is possible to give. Another important fact has been reached beyond what was originally sought. Choosing the city which has minimum *SD* associated value, it is possible to obtain the exact TSP solution investing less amount of time.

This work has raised certain issues that it would be interesting to address. The utility, applicability and implementation of the methodology to other data-dependent problem still remain to be studied. Another issue concerns the problem of the obtained performance model. The existence of more or less parameters that affect performance may suggest strategies to fit the final model. Last but not least, how to provide automatic useful feedback in order to asses more studies and experiments.

Appendix

A. Specification of the parallel machine

The execution were 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.

B. Specification of Cluster-Frame environment

Cluster-Frame is a dynamic and open environment of clustering (Fritzsche, 2007). It permits the evaluation of clustering methods such as K-Means, K-Prototypes, K-Modes, K-Medoid, K-Means^{+}, K-Means^{++} for the same data set. Using Cluster-Frame, the results reached applying different methods and using several parameters can be analyzed and compared.

6. References

Alizadeh, F.; Karp, R.; Newberg, L. & Weisser, D. (1993). Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Symposium on Discrete Algorithms,* pp. 371-381, ACM Press.

Arkin, E.; Chiang, Y. ; Mitchell, J.; Skiena, S. & Yang, T. (1996). On the Maximum Scatter TSP, *In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 97)*, pp. 211-220, ACM New York.

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

Barvinok, A.; Tamir, A.; Fekete, S.; Woeginger, G; Johnson, D. & Woodroofe, R. (2003). The Geometric Maximum Traveling Salesman Problem. *Journal of the ACM,* Vol.50, No.5, pp. 641-664.

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

Christofides, N. (1985). Vehicle Routing. *N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, Combinatorial Optimization,* pp. 315-338, Wiley, Chichester, UK.

Duman, E. & Or, I. (2004). Precedence constrained TSP arising in printed circuit board assembly. *International Journal of Production Research,* Vol.42, No.1, pp. 67-78, 1 January 2004, Taylor and Francis Ltd.

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

Garey, M.; Graham, R. & Johnson, D. (1976). Some NP-complete geometric problems, STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing, pp. 10-22, Hershey, Pennsylvania, United States, ACM, New York, NY, USA.

Gilmore, P. & Gomory, R. (1964). Sequencing a One-State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. Operations Research, Vol.12, No.5, pp. 655-679.

Golden, B.; Levy, L. & Vohra, R. (1987). The Orienteering Problem. *Naval Research Logistics,* Vol.34, pp. 307-318.

Gutin, G. & Punnen, P. (2006). *The Traveling Salesman Problem and Its Variations,* Springer, 0-387-44459-9, New York.

Karp, R. (1972). Reducibility among combinatorial problems: In Complexity of Computer Computations. *Plenum Press,* pp. 85-103. New York.

Korostensky, C. & Gonnet, G. (2000). Using traveling salesman problem algorithms for evolutionary tree construction. *BIOINF: Bioinformatics,* Vol.16, No.7, pp. 619-627.

Lenstra, J. & Kan, A. (1975). Some simple applications of the Travelling Salesman Problem. Operations Research Quarterly, Vol.26, No.4, pp. 717-732.

Lilja, D. (2000). *Measuring computer performance: a practitioner's guide,* Cambridge University Press, ISBN: 0-521-64105-5, New York, NY, USA.

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

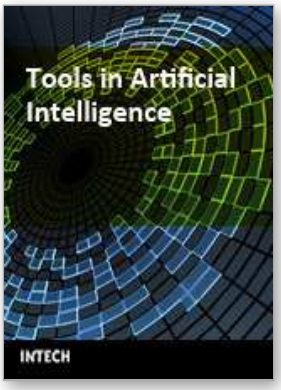
Miller, D. & Pekny, J. (1991). Exact Solution of Large Asymmetric Traveling Salesman Problems. *Science,* Vol.251, pp. 754-761.

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

Ratliff, H. & Rosenthal, A. (1983). Order-Picking in a Rectangular Warehouse: A Solvable Case for the Traveling Salesman Problem. Operations Research, Vol.31, No.3, pp. 507-521.

Sankoff, D. & Blanchette, M. (1997). The median problem for breakpoints in comparative genomics, *Proceedings of the 3rd Annual International Conference on Computing and Combinatorics (COCOON'97)*, Vol.1276, pp. 251-264, New York.

TSP page (2008). http://www.tsp.gatech.edu/history/.



Tools in Artificial Intelligence

Edited by Paula Fritzsche

ISBN 978-953-7619-03-9
Hard cover, 488 pages
Publisher InTech
Published online 01, August, 2008
Published in print edition August, 2008

This book offers in 27 chapters a collection of all the technical aspects of specifying, developing, and evaluating the theoretical underpinnings and applied mechanisms of AI tools. Topics covered include neural networks, fuzzy controls, decision trees, rule-based systems, data mining, genetic algorithm and agent systems, among many others. The goal of this book is to show some potential applications and give a partial picture of the current state-of-the-art of AI. Also, it is useful to inspire some future research ideas by identifying potential research directions. It is dedicated to students, researchers and practitioners in this area or in related fields.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821