Technical Report No. 2001-448

Computing Nearest Neighbors In Real Time*

Marius Nagy and Selim G. Akl
Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6
Canada
Email: {marius,akl}@cs.queensu.ca

July 4, 2001

Abstract

The nearest-neighbor method can successfully be applied to correct possible errors induced into bit strings transmitted over noisy communication channels or to classify samples into a predefined set of categories. These two applications are investigated under real-time constraints, when the deadlines imposed can dramatically alter the quality of the solution unless a parallel model of computation (in these cases, a linear array of processors) is used. We also study a class of real-time computations, referred to as reactive real-time systems, that are particularly sensitive to the first time constraint imposed.

1 Introduction

A traditional computational paradigm is based primarily on the following assumptions:

1. The entire data set that has to be processed in order to obtain the desired result is available at the outset.
2. Once the computation is initiated, no additional data is taken into account by the algorithm elaborating the solution to the problem at hand.
3. The output is expected to be produced as soon as the resources of the machine solving the problem allow it, with no precise deadlines specified.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

Probably, the majority of the computations carried out today fall into this category. However, an increasing number of computations do not fit into this paradigm. Time is in these cases a factor of capital importance. When the computation begins, only a fraction of the data to be processed is available to the algorithm in charge. The rest arrives as the computation proceeds, generating a multitude of possible real-time paradigms. The data arrival rate might be constant, increasing, decreasing or non-monotonic over time. Also, the data received at one given moment in time might represent an independent subproblem, or it might have to be incorporated into a global solution, influencing the subsequent computation.

Another feature of real-time paradigms, underlining the role of time in these computations, is the presence of deadlines. Deadlines can be imposed on when the input data should be processed and/or when the results are to be produced. The particular nature of some real-time paradigms makes a parallel machine the only viable alternative in the competition with their sequential counterparts. The presence of deadlines has been identified as a source of superunitary behavior with respect to the quality of the solution computed by parallel machines to real-time problems [1, 2, 3, 4, 5, 8, 9, 10].

This paper focuses on real-time paradigms that resort to the *nearest-neighbor* method in order to compute the desired result. Generally, this method implies computing distances between *objects* in a conveniently chosen metric. From the multitude of possible applications, two are selected in this paper to investigate the efficiency of a parallel implementation of the nearest-neighbor method. In the first case, the problem is one of error correction. *Codewords* from a predefined *code* are sent, in the form of bit strings, through a communication channel subjected to perturbations. At the other end, an algorithm using the nearest-neighbor method tries to recover, to the best possible extent, the original codewords sent.

The second example is taken from *pattern recognition*. Here, the nearest-neighbor method is used to classify a new sample as belonging to one of several existing categories. The real-time environment in this case is of special importance due to the following two characteristics:

1. The data arrival rate as well as the output rate are not constant, but slowly decrease over time

2. The output generated is actively involved in processing the new data received.

We call such a system a *reactive system*, as opposed to the case in which the output becomes inactive as soon as it is produced and does not influence the subsequent computation in any way.

In all the paradigms investigated, the use of a parallel model of computation brings an impressive gain in the quality of the solution computed.

The paper is organized as follows. Section 2 briefly describes the models of computation employed. The use of a unidirectional linear array of processors for correcting errors induced in bit strings (when transmitted over a noisy communication channel) is analyzed in Section 3. Section 4 deals with classification problems in pattern recognition. Both a simple real-time setting and a reactive system are investigated. Conclusions are formulated in Section 5.

We assume throughout the paper that the nearest-neighbor rule always gives the correct classification. Specifically, in no case does the nearest neighbor to an item belong to a class other than the proper class to which that item belongs.

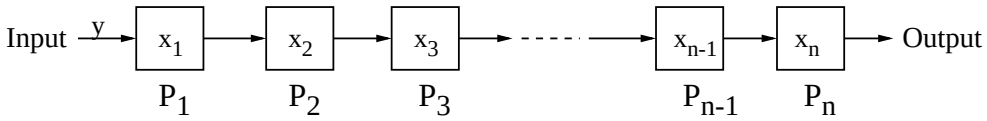

Figure 1: Unidirectional linear array of processors.

2 Models of computation

Using a parallel machine when trying to solve a computational problem in a real-time environment often brings a remarkable improvement in the quality of the solution computed, with respect to the best sequential solution for the problem at hand. The results obtained in [3, 8, 9, 10] for some graph problems support this claim. In order to achieve those results, different variants of the PRAM [3, 8, 9] and RMBM [10] models of parallel computation have been used. Although the power that the PRAM or RMBM brings into play is essential for dealing with the problems addressed therein, similar results can be obtained on weaker models of parallel computation (see [1, 2, 4, 5]). In this paper we focus on (arguably) the weakest model of parallel computation, namely the linear array (Figure 1).

The model consists of n processors $P_1, P_2, \cdots, P_n$ arranged into an array such that each processor P_i is connected to P_{i-1} and P_{i+1}, except for P_1 (which is only connected to P_2) and P_n (which is only connected to P_{n-1}). We restrict the communication links between the processors to be unidirectional. Consequently, all inputs are read by P_1. Data then flows into the linear array from P_1 towards P_n, which is the only processor in charge of the output to the outside world. Each processor is assumed to have access to a local memory of size $O(n)$ bits. During one time unit each processor can:

1. Read (receive) a constant number of fixed-size data,
2. Perform a fixed number of constant-time operations involving two operands (such as adding two numbers or comparing two numbers), and
3. Write (transmit) a constant number of fixed-size data.

In order to successfully face the constraints imposed by the second real-time paradigm investigated in the paper, a bidirectional linear array of processors is assumed as a model of parallel computation. In this model, there are two communication links between any two neighboring processors P_i and P_{i+1}: one from P_i to P_{i+1} and the other from P_{i+1} back to P_i. In this way, data can flow into the linear array in both directions simultaneously.

The sequential model of computation assumed throughout this paper is the random access machine (RAM) with a processing unit identical to those forming the linear array.

3 Error correction

The following subsection covers some basic definitions and explains those notions that are necessary in order to understand the problem addressed in this section.

3.1 Background

Coding theory, the study of codes, including error detecting and error correcting codes, has been studied extensively for the past forty years. It has become increasingly important with the development of new technologies for data communications and data storage. When information, in the form of bit strings, is manipulated, errors may occur due to various reasons. Messages transmitted over a communication channel may be altered by electrical interference or equipment error. Similarly, static information may be modified unintentionally; for example, errors are introduced into data stored over a long period of time on magnetic tape as the tape deteriorates. To guarantee reliable transmission or to recover degraded data, techniques from coding theory are used.

A bit string encoding a certain message and traveling through a communication channel is called a *codeword*. A set of codewords forms a *code*. Some codes incorporate redundant information in their codewords in order to detect and/or correct errors (a parity check bit attached to a bit string is a common way of detecting errors). Usually, a more redundant code allows for a larger number of errors to be detected and/or corrected.

However, efficient codes exist that do not use redundancy. Special care must be taken when constructing such codes and the methods used for error correction are generally more involved. One such method is called *nearest-neighbor decoding* and is based on the *Hamming distance* between two bit strings.

Definition 1 *The Hamming distance $d(a, b)$ between the bit strings $a = a_1a_2 \cdots a_n$ and $b = b_1b_2 \cdots b_n$ is the number of positions in which these strings differ, that is, the number of $i (i = 1, 2, \cdots, n)$ for which $a_i \neq b_i$.*

Example
The Hamming distance between the bit strings 01110 and 11011 is $d(01110, 11011) = 3$.

Suppose now that when a codeword x from a code C is sent, the bit string y is received. If the transmission was error-free, then y would be the same as x. But if errors were introduced by the transmission, then y is not the same as x. Suppose that $y \notin C$. The nearest-neighbor method of decoding y is to compute the Hamming distance between y and each of the codewords in C. Then we take the codeword of minimum Hamming distance to y. If the distance between the closest pair of codewords in C (called the *minimum distance* of C, according to the definition below) is large enough and if sufficiently few errors were made in transmission, this codeword should be x, the codeword sent.

Definition 2 *The minimum distance of a binary code C is the smallest distance between two distinct codewords, that is,*

$$d(C) = min\{d(x, z) | x, z \in C, x \neq z\}.$$

The following result ([7]) establishes the relation between the minimum distance of a code C and the number of errors it can correct.

Theorem 1 *A binary code C can correct up to k errors in any codeword if and only if $d(C) \geq 2k + 1$.*

If the transmission channel is a binary symmetric channel, that is, each bit sent has the same probability p of being received incorrectly and $p < 1/2$, then nearest-neighbor decoding produces the most likely codeword that was sent from a binary string that was received.

3.2 Real-time setting

We now describe the real-time environment in which the performances of both the sequential and parallel machines are to be evaluated. Let C be a code composed of n codewords, with each codeword consisting of n bits. At the beginning of the computation, the n codewords are stored in the memory of the machine in charge of solving the problem at hand. This means that each of the processors of the linear array will store one codeword in its local memory, while the sequential computer will store all n codewords in its memory.

A stream of messages is sent over a binary symmetric channel to be received by the error recovering machine at the other end. Each bit string received is decoded using the nearest-neighbor method in order to produce the original codeword sent. The transmission rate is of one bit per time unit, with one time unit break between two consecutive messages. Evidently, the task of the receiver is to recover (to the best possible extent) the original codeword transmitted from the bit string received, knowing that the message could have been altered during transmission.

The output stream, composed of the recovered codewords is subjected to the same constraints as the input stream. More precisely, one bit, part of a decoded message, has to be produced each time unit, with a one time unit break between two consecutive decoded messages. However, we allow an initial delay of at most $2n$ time units before the first bit of the first recovered codeword has to be produced.

3.3 Performance comparison

We analyze in this section the performances of both (sequential and parallel) machines in the real-time environment described above. A measure of the performance will be the accuracy achieved in decoding a message in the worst case. To be more specific, if the Hamming distance between the recovered codeword y and the actual codeword sent x is $d(x, y)$ then

$$accuracy = n - d(x, y).$$

Note that in the given real-time paradigm, storage of input data in memory for later processing is not prohibited. Furthermore, a capacity limit for the memory of the sequential machine is not even specified. The sequential computer is free to store in its virtual unlimited memory all the data that cannot be dealt with in the current time unit or that is needed for some subsequent computation.

Storage for later processing might seem a good idea in the case of the first message received, due to the delay allowed for its decoding. This delay increases the time available for decoding the first message to $2n$ time units. Unfortunately, after the initial delay has elapsed, a new bit has to be output every time unit. Therefore, each of the subsequent messages must be decoded in no more than n time units. To the advantage of the sequential

machine, we analyze in the following the accuracy of the decoding operation performed on the first message received, for which we have the largest amount of processing time available.

In order to compute the Hamming distance between the received bit string and all the n codewords in C, the sequential computer would need n^2 time units. Since the admitted delay is of only $2n$ time units, a result has to be produced without knowing the Hamming distances to all the n codewords. In fact, the available time is just enough to compute the distance to two of the n codewords.

In the worst case, this decision based on partial information has dramatic effects on the result produced. For example, consider that the following four bit strings are the codewords of C: 0000, 0011, 1100, and 1111. Suppose now that the codeword $x = 1111$ is transmitted over the binary symmetric communication channel and that the message received by the sequential machine is $y = 1110$. In this example, a number of $k = 1$ errors affected the bit string transmitted. The sequential computer will only have time to compute the Hamming distances from y to the first two codewords: $d(1110, 0000) = 3$, and $d(1110, 0011) = 3$. Based only on these two pieces of information it will now have to decide which was the original codeword sent. Since the two computed Hamming distances are equal, one of the two corresponding codewords is arbitrarily chosen as the original codeword. In the worst case, the first codeword (0000) will be output as the decoded message. The accuracy achieved in this case is

$$sequential\ accuracy = 0.$$

In general, if $k(k \geq 1)$ errors are induced during transmission, the same worst-case scenario can happen for codes C with the minimum distance $d(C) \leq 2k$.

How can the linear array of Figure 1 deal with this problem? We recall that each of the n processors stores one codeword in its local memory. Processor P_1 is in charge of receiving the stream of bits arranged in messages, while P_n has to produce the output stream, containing the recovered codewords. For a certain message y, processor P_i is in charge of computing the Hamming distance between y and the codeword x_i stored in its local memory. This is done as follows. Initially, all processors set $d(y, x_i) = 0$, $i = 1, 2, \cdots, n$. Once the first bit of y is received, P_1 compares it with the first bit of x_1 and updates $d(y, x_1)$ accordingly. P_1 then passes this first bit along to P_2. In the next step, while P_1 deals with the second bit of y, P_2 performs the same operation between the first bit of y and the first bit of x_2. In this way, after n steps (when y is completely received), the Hamming distances between y and x_i, $i = 1, 2, \cdots, n$ are at different stages of computation in the processors of the linear array. For instance, the computation of $d(y, x_1)$ is completed, while P_n has just started to work on $d(y, x_n)$.

P_1 now sends $d(y, x_1)$ together with the first bit of x_1 to P_2 and takes a one time unit break before starting to work on the next transmitted message. However, this is more of a computational break because P_1 continues to send the codeword x_1 further to P_2, one bit per time unit, until completely transmitted. When P_2 receives $d(y, x_1)$ and the first bit of x_1, it compares $d(y, x_1)$ with $d(y, x_2)$ (whose computation has just been completed) and sends the minimum to P_3. If $d(y, x_1)$ was found smaller than $d(y, x_2)$, then P_2 will forward the first bit of x_1 to P_3 and, in the following steps, all the remaining bits of x_1 received from P_1. Otherwise, it will ignore these bits and start sending x_2 to P_3, also at a rate of one bit

per time unit, until fully transmitted. Following this path, $2n-1$ steps after the first bit of y was received by P_1, P_n is able to start outputting the codeword corresponding to the global minimum Hamming distance. The same sequence of steps is repeated for each of the messages received by P_1.

In order to evaluate the performance of this algorithm we consider the same case as for the sequential machine, in which k errors have been generated during the transmission of a particular message y, and the minimum distance of C is $d(C) \leq 2k$. It is not difficult to see that this is the worst possible case for the parallel machine too, since a code C with $d(C) \geq 2k+1$ ensures the correction of all k errors.

Since the distance between y and the original codeword sent x is k, x is competing for the place of recovered codeword only with those codewords that are at a Hamming distance of at most k from y. In the worst case, if a different codeword than x is chosen as the recovered codeword, this can only be at a distance of at most $2k$ from x. Therefore, the accuracy achieved by the linear array in this case is

$$parallel\ accuracy = n - 2k.$$

If the communication channel is a relatively reliable one, that is, the number k of errors generated during transmission is small when compared with the size n of the message, then the parallel accuracy is $\theta(n)$.

The ratio of the parallel accuracy to the sequential accuracy is therefore

$$accuracy\ ratio = \frac{n-2k}{0} = \infty.$$

This result can be interpreted in the sense that even if the amount of resources available and the case analyzed were in favor of the sequential computer, the improvement in accuracy brought about by the parallel computer is unbounded. We observe that the parallel model of computation used (namely, the unidirectional linear array) is the simplest possible among those assuming some kind of communication among processors. Further, the amount of memory available for the sequential machine was not limited in any way. Finally, the performance of the sequential machine was analyzed with respect to the first message decoded, for which time resources were the most generous.

4 Classification

The error-correcting codes analyzed above use the nearest-neighbor method in order to recover the original codeword sent through a noisy communication channel. This method, however, is not specific only to error-correction applications. Other areas, such as pattern recognition, benefit from using the nearest-neighbor method or some other related techniques (such as the k-nearest-neighbors rule) [6]. These methods are particularly helpful in clustering and classification algorithms. Labeling a given set of samples as belonging to distinct groups, based on the relative distance between them in a predefined metric, is called *clustering*. When the clusters are already identified and we want to determine to which of the given categories a new sample belongs, the problem is a matter of *classification*.

We show in what follows that when a real-time solution is required, parallel processing is as important in classification as it is in error correction.

4.1 Simple setting

Imagine that, after an initial scene analysis procedure involving clustering, n samples are grouped into c distinct clusters. Suppose now, that at regular intervals, a new sample is acquired by the sensors monitoring the scene, and has to be classified into one of the c existing categories. Following the nearest-neighbor method, the distance from the new sample to each of the n labeled samples has to be computed according to the metric at hand. The label of the nearest sample found will dictate the category into which the new sample is placed.

Let us define the time interval between two consecutive sample acquisitions as a time unit. The same deadline imposed for the error-correcting problem is set here too. Specifically, after an initial delay of n time units, a newly acquired sample must be labeled at the end of each time unit. The model of parallel computation used remains the unidirectional linear array. Each processor P_i stores a pair (A_i, c_i), where A_i is a vector describing the i-th initial sample and c_i denotes its classification.

Vector X, corresponding to the new sample acquired, enters the array at P_1 and then travels towards P_n, reaching a new processor each time unit. Processor P_i computes the distance between vectors X and A_i, compares it with the current minimum received from P_{i-1} and transmits further to P_{i+1} three elements: the vector X, the updated minimum and the label of the sample achieving that minimum. All of these (computation and communication) steps are carried out by processor P_i during one time unit.

Computing the distance between two samples is the most time-consuming operation, and we base our analysis on the assumption that only one such distance can be determined during one time unit. In this way, n time units after X has been received by P_1, P_n is able to output the label associated with the sample represented by X. Furthermore, from that moment on, subsequent samples are accurately labeled at a rate of one sample per time unit, due to the pipeline capabilities of the linear array of processors.

The sequential processor has just enough time to determine the nearest neighbor only for the first sample acquired. Starting with the second one, the time available to classify the current sample is of one time unit. Since only the distance to one of the n existing samples can be computed before the deadline, in the majority of cases the sequential machine will fail to correctly classify the newly acquired sample. Once more, parallelism makes the difference between failure and success.

We notice that this conclusion is true, even if the sequential processor is faster than a processing element of the linear array. Indeed, in the worst case, even if the computation of up to $n-1$ distances can be successfully completed during one time unit, the single sample left over might be the nearest neighbor of the sample being classified. This can lead to an incorrect classification, as shown in Figure 2. Sample X is incorrectly classified as a *square* because the distance from X to A was not computed due to the limited time. The sequential machine has falsely declared B as the nearest neighbor of X.

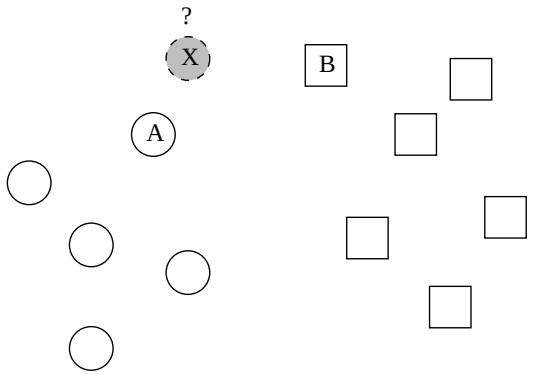

Figure 2: Misclassification by the sequential machine.

4.2 Reactive real-time system

The real-time paradigm constructed for the classification problem has some features that might justify an argument characterizing it as simplistic. Input data arrive at a constant rate and the same is true for the results produced. The computational effort spent in order to produce a result is the same for any of the outputs. Finally, once a result is output, there is no way in which it can influence subsequent computation. In the following, we define and analyze a real-time paradigm for classification, based on the nearest-neighbor method, that is more complex, at least in the aspects mentioned above.

The most important element of change is the fact that each new sample classified is used together with the rest of the labeled samples to classify the subsequent samples acquired. This change has two major consequences. First, the real-time paradigm becomes a *data-accumulating paradigm*. At any given step, the number of samples that have to be processed in order to avoid a misclassification is one more than in the preceding step. Secondly, the decision taken in one step depends on the decisions taken in the previous steps, thus making the classifier a *reactive* system. This can have a major impact on the correctness of the classifier's results, especially when time constraints are imposed.

The real-time computation starts with the acquisition of the first new sample, at the beginning of the first time unit. Only one sample can be acquired during one time unit, but not every time unit is characterized by the acquisition of a new sample from the scene being monitored. The frequency of these *gaps* in the stream of newly acquired samples is described by the following function:

$$f_{in}(i) = \lfloor \frac{i-1}{n} \rfloor, \text{ for } i = 1, 2, 3, \cdots$$

This means that after the i-th new sample has been acquired, at the beginning of the current time unit, a gap of $f_{in}(i)$ time units will follow, that is, no new sample is received during the next $\lfloor \frac{i-1}{n} \rfloor$ time units. In other words, at a global scale, the data-arrival rate slightly declines over time.

The rate at which the outputs must be produced is restricted by a very similar law. An interval of n time units is allowed for the first new sample to be classified. Afterwards, the discontinuities in the output stream of labeled samples have to appear with a frequency described by the following function:

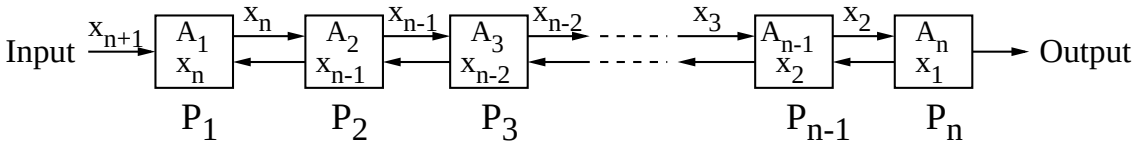

Figure 3: Synchronous storing step in the bidirectional linear array.

$$f_{out}(i) = \lceil \frac{i}{n} \rceil, \text{ for } i = 1, 2, 3, \cdots$$

According to this function, if the i-th sample acquired is output along with its computed label at the end of the current time unit, then the class to which the $(i+1)$st sample belongs must become known $f_{out}(i)$ time units later. Equivalently, the gap in the output stream, between the ith and the $(i+1)$st labeled samples should not exceed $\lceil \frac{i}{n} \rceil$ time units.

The computational steps performed for the correct classification of the first sample received by P_1 are much the same as in the case of the simpler real-time paradigm described in the previous section. Indeed, the distances between X_1 (the vector representing the first new sample acquired) and A_1 to A_n (the vectors stored in the processors of the linear array before the beginning of the computation) are successively computed by P_1 to P_n, keeping track of the current minimum. At the end of the n-th time unit, the vector X_1 along with the label of its nearest neighbor are output by P_n. In the same time unit, X_2 has reached P_{n-1}, while the distance between X_n and A_1 has been computed by P_1. At this moment, each processor stores in its local memory a copy of the new sample they currently hold. The situation is depicted in Figure 3.

During the following n time units, each processor P_i has to compute two distances for each new sample it receives, one to A_i and the other to X_{n-i+1}. Since this has to be done sequentially, and the assumption that only one such distance can be computed during one time unit is still valid, it results that a new sample is processed for two time units in each element of the linear array. This will generate a one time unit gap between any two consecutive labeled samples of the first $n+1$ output.

After X_{n+1} and its corresponding label are output by P_n, each processor stores a third sample in its local memory. The three vectors in the memory of P_n are now A_n, X_1 and X_{n+1}, while P_1 possesses a copy of A_1, X_n and X_{2n}. The immediate consequence is that the length of the gaps in the output stream increases to two time units. As more and more new samples are acquired, this length will continue to increase by one time unit after each group of n samples received. We note in passing that the gaps in the input stream allow for an immediate processing of the new samples upon arriving, without storing them for a later processing stage.

We end the presentation of the parallel algorithm which solves the classification problem in the new real-time environment with a technical detail. When the current minimum is passed along to the next processor in the linear array, the class identifier of the sample realizing this minimum distance has to be transmitted also. On the other hand, when a new set of n samples are stored in the local memories of the n processors composing the linear array, only one of them (namely the one stored by P_n) is already labeled. The classification

of the others is not yet known, at the time they are stored. This may lead to the situation in which the current minimum is achieved by an unlabeled sample, therefore making impossible the transmission of the class identifier to the next processor in the array.

In such cases, only the index i of the vector X_i representing the current nearest neighbor is transmitted further. The label of X_i will become known only after X_i is completely processed by P_n. At this moment, after it is produced as output, the computed label (paired with the index i) is also transmitted backwards through the array, down to the processor storing X_i. When such a pair $(i, label\_of\_X_i)$ encounters at some point in the array an index i traveling in the opposite direction, the index i is substituted with the class identifier $label\_of\_X_i$ and sent forwards.

Following the procedure highlighted above, the parallel machine is able to handle the input stream and produce the requested output within the given deadlines. Each newly acquired sample is correctly classified while the specifications of the real-time paradigm are fully respected. Theoretically, the parallel machine can accurately classify an infinite number of samples, acquired at a rate dictated by f_{in}. The only limitation comes from the size of the memory available to each of the processors in the bidirectional array.

By contrast, the sequential machine can guarantee a correct classification only for the first sample received, taking advantage of the relaxed deadline for the first output. After that, it simply cannot cope with the high output rate imposed by f_{out}. The natural consequence is a high probability of misclassification for any of the subsequent samples received. This is true despite the fact that output deadlines become more relaxed over time. The data-accumulation nature of the paradigm demands an increasing computational effort in order to produce a correct classification result.

The performances of the sequential and parallel machines in the complex real-time environment analyzed can be measured from at least two points of view. On one hand, we can compare the correctness of the classification for a single (arbitrarily chosen) sample in the worst possible case. In this respect, we are advancing from a total misclassification in the sequential case to a perfect classification achieved by the parallel model. Another alternative, and probably a more realistic one, would be to monitor the number of correct classifications produced by each of the algorithms over a certain period of time. It is not difficult to see that, in the worst case, this number equals 1 for the sequential machine, regardless of the length of the time interval monitored. For the bidirectional array of processors, the number of correct classifications tends to infinity, when the length of the time interval taken as reference approaches infinity. If we take this number as a measure of the quality of the solution produced by the respective algorithm, we can conclude that the quality improvement provided by the parallel algorithm is unbounded at the cost of an infinitely long time interval elapsed.

5 Conclusion

We have analyzed in this paper the impact of time constraints on computational processes employing the nearest-neighbor method. In both domains investigated, the sequential solution was compared to the one arrived at using a parallel model of computation, namely, the linear array of processors. The results obtained prove that in a real-time environment

parallelism can change the outcome of a computation from failure to success, or provide an unbounded improvement in quality. The paper also confirms that even weak models of parallel computations are capable of remarkable improvements in the quality of the solution computed, when compared with a sequential machine.

However, we believe that the most interesting results were obtained in the study of reactive real-time systems. To date, in all theoretical analyses of real-time computations, improvements in the quality of the solution, brought about by parallelism, were due to the firm deadlines imposed on when the solution has to be produced. To our knowledge, this is the first real-time paradigm investigated (from an algorithmic point of view), in which the a priori deadlines, establishing when the output is needed, are not all equally important. To be more specific, it suffices to set as firm only the first two deadlines and relax all the others so that the sequential machine has enough time to compute the distances to all of the existing samples. In the worst case, a single error in the classification of the second new sample acquired will generate a theoretically infinite series of misclassifications for the subsequent samples received.

Although limited time resources are directly responsible for the first misclassification, it is the reactive nature of the system that triggers the subsequent classification errors. This observation seems to support the idea that a reactive system placed in a real-time environment with deadlines set for producing the output is extremely sensitive to these deadlines (especially the first). If only one such deadline is too tight for the machine's capabilities, the effect on all of the outputs generated afterwards might be dramatic. In other words, a reactive system with insufficient computational resources may never recover from an error generated in its output. This suggests that the behavior of reactive systems in real-time conditions and the influence of parallelism on them are worthy of further study.

References

[1] S.G. Akl, Superlinear performance in real-time parallel computation, *Proceedings of the Thirteenth Conference on Parallel and Distributed Computing and Systems*, Anaheim, California, August 2001.

[2] S.G. Akl, Nonlinearity, maximization, and parallel real-time computation, *Proceedings of the Twelfth Conference on Parallel and Distributed Computing and Systems*, Las Vegas, Nevada, November 2000, pp. 31–36.

[3] S.G. Akl and S.D. Bruda, Parallel real-time optimization: beyond speedup, *Parallel Processing Letters*, 9, 1999, pp. 499–509.

[4] S.G. Akl and S.D. Bruda, Parallel real-time cryptography: Beyond speedup II, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 2000, pp. 1283–1289.

[5] S.G. Akl and S.D. Bruda, Parallel real-time numerical computation: Beyond speedup III, *International Journal of Computers and their Applications, Special Issue on High Performance Computing Systems*, 7, 2000, pp. 31–38.

[6] R.O. Duda and P.E. Hart, *Pattern classification and scene analysis*, John Wiley & Sons, 1973.

[7] J.G. Michaels and K.H. Rosen, *Applications of Discrete Mathematics*, McGraw-Hill, 1991.

[8] M. Nagy and S.G. Akl, Real-time minimum vertex cover for two-terminal series-parallel graphs, *Proceedings of the Thirteenth Conference on Parallel and Distributed Computing and Systems*, Anaheim, California, August 2001.

[9] M. Nagy and S.G. Akl, Locating the median of a tree in real time, Technical Report No. 2001-445, Department of Computing and Information Science, Queen's University, Kingston, Ontario, May 2001.

[10] N. Nagy and S.G. Akl, The maximum flow problem: A real-time approach, *Proceedings of the Thirteenth Conference on Parallel and Distributed Computing and Systems*, Anaheim, California, August 2001.