2008

Maximizing Correlation in the Presence of Missing Data

Xinfang Wang
Georgia Southern University, xfwang@georgiasouthern.edu

Applied Mathematical Sciences, Vol. 2, 2008, no. 54, 2653 - 2664

Maximizing Correlation in the Presence of Missing Data

Xinfang (Jocelyn) Wang

Department of Finance and Quantitative Analysis
College of Business Administration, P. O. Box 8151
Georgia Southern University
Statesboro, GA 30460-8151, USA
xinfang.jocelyn.wang@gmail.com

Abstract

In this paper we address the problem of maximizing the correlation between two vectors of time series data, when one of the vectors has missing data and the timing of the missing data is unknown. The motivation for this work comes from environmental monitoring where because of monitoring malfunction, some data are lost. We study the use of integer programming and a genetic algorithm (GA) for this problem.

Keywords: Integer programming; combinatorial optimization; genetic algorithm; missing data

1. Introduction

The problem we study concerns two vectors of time series data of two variables known to be strongly positively correlated. Due to one or more data collection errors, for example the random failure of a monitoring device, one of the vectors used has missing data, causing it to be "shorter" than the other vector. Knowing that the two vectors are strongly correlated, we seek to know where to insert zeros in the shorter vector, so as maximize the correlation with the longer vector. We need to insert zeros into the shorter vector so as to preserve the order of the elements, but the positions of the zeros must be made such that the dot product of the two vectors (and hence the correlation between the two vectors) is maximized.

The motivation for this work comes from environmental monitoring where because of monitoring malfunction, some data are lost. Our work is related to

signal matching in paleoclimate reconstructions where chronostratigraphic correlation among records needs to be recovered. This is often achieved by matching signals between climate proxies and orbital parameters or between multiple climate proxies [1]. A simplified example of signal matching can be stated as follows: n data points in a series A are matched to the m points in the series B so that the square of their differences can be minimized. Both series A and B are proxy records and continuously distributed. Provided that the sequence of points in both series is preserved, the points in series A are allowed to fall between points in series B and linear interpolation is applied in this case. Lisiecki and Lisiecki [1] applied the dynamic programming approach to solve the signal-matching problem. Each series of record is divided into several hundreds of intervals and a score, mainly determined by the sum square of the difference between two series, is calculated for all feasible alignments of these intervals. The dynamic program is designed to search for the optimal alignment which results in the lowest accumulative score. Our problem differs from the signal-matching problem in terms of the data type. Our problem has the discrete data and signal-matching problem has continuous data.

As an example of the problem we address, consider the situation where two signals are taken at constant time intervals. The second measuring device fails randomly and we retrieve from it a shorter vector of data. The long vector is of length 20, but the second vector, because of nine randomly missing data points, contains only 11 elements. The long vector and short vector are [0 1 0 3 7 4 6 5 9 6 7 4 3 9 5 9 7 1 6 2] and [3.34 7.47 4.72 6.84 5.32 9.19 9.89 5.91 9.66 7.55 2.41], respectively. The timing of the short vector is unknown, only the order of the data is preserved. The two series are known to be highly correlated and we would like to use this fact to impute zeros for the missing values into the short vector of data so as to ascertain the timing of the data values. In order to maximize the correlation, we must maximize the dot product of the two vectors. Ignoring the problem of missing data, that is, taking the first 11 elements and pairing them with the first 11 elements of the larger vector results in a dot product of 340.2 and a matching of the data streams as shown in Figure 1. The optimal insertion of zeros results in a dot product of 522.7 as shown in Figure 2. Our goal in this paper is to develop a systematic approach to solving this problem.

Figure 1: Series 1 has 20 elements, but series 2 is missing 9 elements.

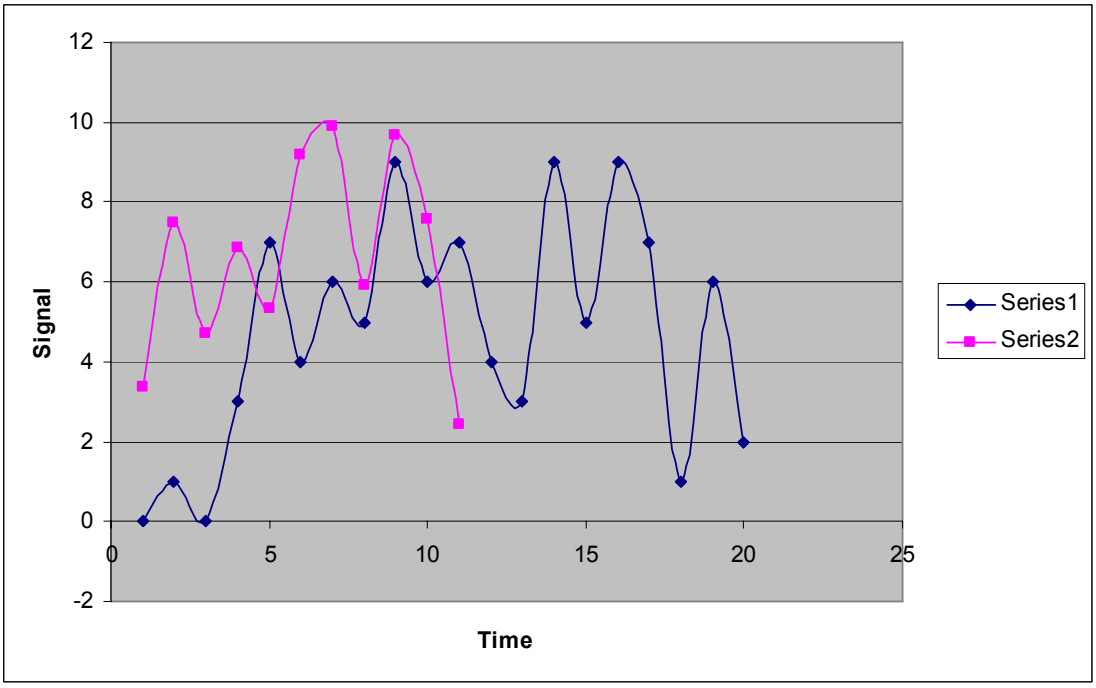
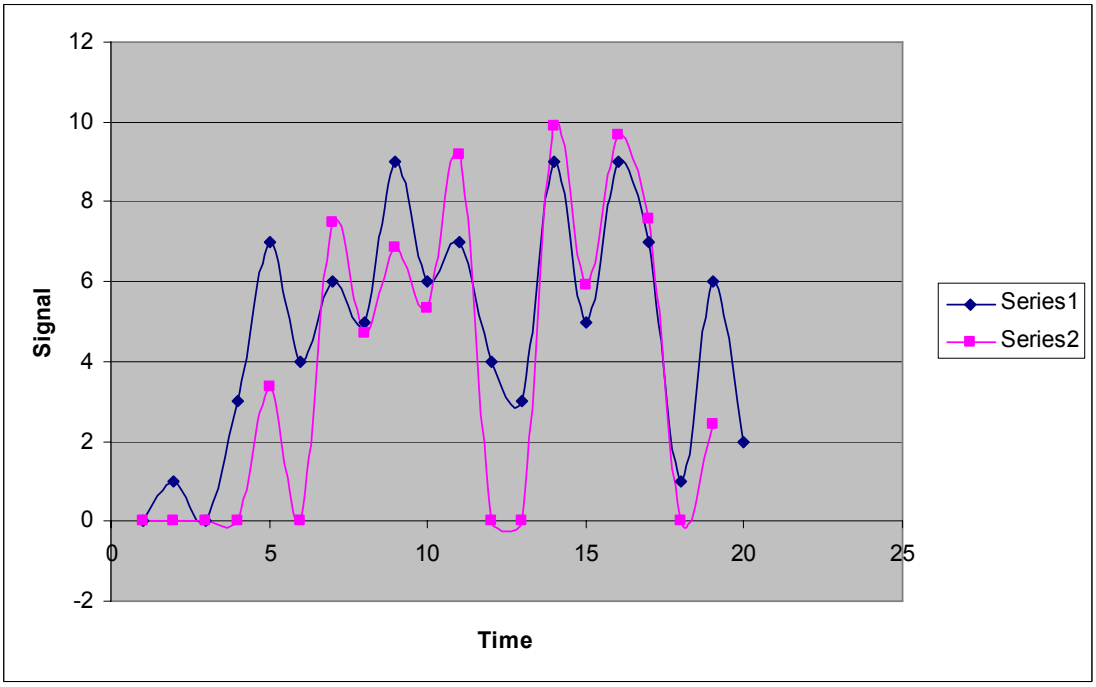


Figure 2 The optimal insertion of zeros into the shorter vector.



In section 2 we provide a binary integer programming formulation of the missing data problem. In section 3, we develop a genetic algorithm for large instances. Section 4 shows computational results comparing the integer program and genetic algorithm. Section 5 is our summary and conclusion.

2. Model formulation

First, we model the discrete vector missing data problem previously described using an integer program.

Let *m* be the number of elements in long vector and the set M= {1,2,….*m*}

n be the number of elements in short vector (m > n) and the set N={1,2,..*n*}

τ_j be the j^{th} element in long vector, $j \in M$

α_i be the i^{th} element in short vector, $i \in N$

$W_{ij} = \alpha_i * \tau_j \quad i \in N, j \in M$

We also introduce the following sets, which concern the feasible pairings of elements in the short vector with elements in the long vector:

$R_i = \{i, i+1, \ldots (m + n - 1)\} \quad i \in N$

$L_j = \{1,2,\ldots..j\}$ for $j < n$

$\{1,2,\ldots.n\}$ for $n \le j \le m - n + 1$

$\{(n - m + j), \ldots n\}$ for $j > m - n + 1 \quad j \in M$

Note that R_i contains the set of all feasible slots for which the ith element of the short vector can be paired in the long vector. Likewise, L_j contains the set of all possible assignments of short vector elements for element j in the long vector. In the integer program that follows, we create variables only over these feasible sets.

Let $X_{ij} = 1$ if the i^{th} element of the short vector is paired with the j^{th} element of the long vector, 0 if not, $i \in N$, and $j \in R_i$. The objective function and constraints are as follows:

$$\text{Max} \quad \sum_{i \in N} \sum_{j \in R_i} W_{ij} X_{ij} \qquad (1)$$

s.t.

$$\sum_{j \in R_i} X_{ij} = 1 \qquad i \in N \qquad (2)$$

$$\sum_{i \in L_j} X_{ij} \le 1 \qquad j \in M \qquad (3)$$

$$X_{ij} + \sum_{k=j}^{m-n+i-1} X_{i-1,k} \le 1 \quad i = 2,...n \quad j = i,...(m - n + i - 1) \qquad (4)$$

The objective function (1) is to maximize the dot product. Constraint set (2) specifies that each one of the elements in the short vector must be paired with exactly one of the elements in the long vector. Constraint set (3) specifies each one of the elements in the long vector can be paired with at most one element in the short vector. Note that by construction, the missing values of data in the short vector are assumed to have no value. Their places in the vector are recognized by the fact (3) is nonbinding, that is, no element from the short vector was assigned to the spot in the long vector. Constraint set (4) forces the sequence of the original elements of the short vector to be maintained. In summary, the program (1)-(4) pairs the elements of the short vector with elements of the long vector,

preserving the order of elements so as to maximize the dot product of the two vectors.

Attempts to solve (1) - (4) with off-the-shelf code indicated that the IP can be very difficult to solve for larger problems. We therefore also test a genetic algorithm as detailed in the next section.

3. A Genetic Algorithm

Genetic algorithm (GA) was first proposed by Holland [2]. GA is a popular meta-heuristic that has been successfully applied to many hard problems determined to be too difficult for the traditional mathematical programming method [3]. The fundamental steps followed in a GA are similar to the genetic evolution of a species [4]. A GA starts with an initial population of solutions (in this case a population of feasible dot product solutions) which are generated either randomly or by some simple heuristic. By applying genetic operators (reproduction, crossover and mutation) to this population, a child generation of solutions is created and added to the original population. The fitness of each member of the resulting population is evaluated. Based on the fitness score, various selection rules can be implemented to select members for carrying over to the next generation. The entire cycle is repeated until a pre-specific stopping criterion (e.g., a certain number of iterations) is reached. Next, we present a genetic algorithm for the problem by means of pseudocode and discuss the appropriate coding scheme and necessary modifications to the standard operators for application to the problem.

Step 0: Input a dot product problem with a long vector (β) of length m and a short vector (α) of length n

Step 1: $i \leftarrow 0$

Step 2: Generate initial population pool $PopPool(i)$ randomly

Step 3: Apply guided mutation operator to each string in $PopPool(i)$

Step 4: $i \leftarrow i+1$

Step 5: Select 30 fittest strings from $PopPool(i-1)$ to create $PopPool(i)$

Step 6: If $i = floor(m/3)$, stop. Otherwise go to Step 3.

We employ the standard binary string representation of the dot product solutions. Suppose one is interested in a dot product problem with a long vector of size 5 and a short vector of size 3. A string can be represented as $[11100]$, which means the first three elements from the long vector are selected in their original order to pair with the 3 elements in the short vector. The resulting dot product is an evaluation of the fitness of the string. In general, a string has m bits, of which $m-n$ are zeros and n are ones.

Operator crossover destroys the feasibility of candidate solutions, and is left out of the GA. All 30 strings in the population are subject to mutation in step 3. Without crossover, each individual population element explores the solution space without interaction. We have created a guided mutation operator to

produce candidate solutions with better fitness. A guided mutation on string C is done as follows: let the set $J_0 = \{j \mid C[j] = 0,\ j \in M\}$ be the bits in a string C that have a value of 0 and the set $J_1 = \{j \mid C[j] = 1,\ j \in M\}$ be the remaining bits that have a value of 1. Furthermore, we define set $A \in J_0$ that has the $ceiling((m-n)/3)$ largest elements in β_j and set $B \in J_1$ that has the $ceiling(n/3)$ smallest elements in β_j. We flip two bits namely $C[a]$ and $C[b]$ at a time, where $a \in A$ and $b \in B$. After the guided mutation, each string C in the population pool generates additional $ceiling((m-n)/3) * ceiling(n/3)$ strings. Only the best-fitted 30 strings out of the original population and newly created ones are carried over the next generation in Step 5. The stopping rule is a pre-specified fixed number of $m/3$ generations in step 6.

4. Computational results

In this section, we test CPLEX applied to (1) – (4) and the GA described in the previous section. We consider nine different problem sizes in terms of lengths of the two vectors, as shown in Table 1. We make the conjecture that the variance of vector values might also affect the complexity of the problem. Therefore, for each problem size, we consider two sets of data, drawing from uniform distribution [1, 9] and [1, 99] respectively. We have a total of eighteen problem instances. For each of the eighteen instances, we consider five replications. Therefore, we have a total of ninety test problems.

Table 1. Nine sets of test problems.

Problem	Length of long vector	Length of short vector
1	50	10
2	50	25
3	50	40
4	100	20
5	100	50
6	100	80
7	200	40
8	200	100
9	200	160

All computational work was carried on a Dell Dimension 8100 machine with memory of 512MB. The IP solver used is CPLEX 8.0. The genetic algorithm is coded in C++. A time limit of 10,080 seconds (3 hours) was imposed on both methods in our test. The computational times for the IP model may be possibly reduced further by using preprocessing directives available in CPLEX solver.

We tried different preprocessing options available and found that adding the probe option to CPLEX is the most effective one. There are three levels of probing available in CPLEX and based on pre-testing, we set the 'Probe' level to 2. The computational results from solving ninety test problems using CPLEX and GA are presented in Table 2.

Table 2. Performance comparison of IP and GA.

Problem	Set	Rep.	IP Time	IP Solution	GA Time	GA Solution	GA % of Optimal
1	1	1	1.4	463	0.1	463	100.0%
1	1	2	1.2	432	0.1	432	100.0%
1	1	3	1.2	422	0.1	422	100.0%
1	1	4	1.2	321	0.1	321	100.0%
1	1	5	1.2	505	0.1	505	100.0%
1	2	1	1.3	46389	0.1	46389	100.0%
1	2	2	1.2	36889	0.1	36889	100.0%
1	2	3	1.2	37888	0.1	37888	100.0%
1	2	4	1.2	48736	0.1	48736	100.0%
1	2	5	1.2	41713	0.1	41713	100.0%
2	1	1	1.7	980	0.1	980	100.0%
2	1	2	1.7	984	0.1	984	100.0%
2	1	3	1.7	809	0.1	809	100.0%
2	1	4	1.7	881	0.1	881	100.0%
2	1	5	1.7	1013	0.1	1013	100.0%
2	2	1	2	103280	0.1	103280	100.0%
2	2	2	1.8	74953	0.1	74917	100.0%
2	2	3	1.8	91154	0.1	91154	100.0%
2	2	4	1.8	100037	0.1	100037	100.0%
2	2	5	1.8	101118	0.1	101118	100.0%
3	1	1	0.6	1257	0.1	1251	99.5%
3	1	2	0.6	1333	0.1	1333	100.0%
3	1	3	0.4	1211	0.1	1211	100.0%
3	1	4	0.5	1353	0.1	1346	99.5%
3	1	5	0.6	1308	0.1	1308	100.0%
3	2	1	0.5	134686	0.1	134686	100.0%
3	2	2	0.4	106767	0.1	106767	100.0%
3	2	3	0.4	124479	0.1	124479	100.0%
3	2	4	0.6	135992	0.1	134983	99.3%
3	2	5	0.4	143669	0.1	143195	99.7%
4	1	1	40.9	793	0.9	793	100.0%
4	1	2	41.1	883	0.9	883	100.0%
4	1	3	41.2	922	0.9	922	100.0%
4	1	4	41	717	0.9	717	100.0%
4	1	5	41.4	809	0.9	809	100.0%
4	2	1	41.8	100101	0.9	100101	100.0%
4	2	2	42	93166	0.9	93166	100.0%
4	2	3	41.7	86591	0.9	86591	100.0%
4	2	4	42.1	94678	1	94678	100.0%
4	2	5	42.3	112987	0.9	112987	100.0%
5	1	1	69.1	2033	1.5	2027	99.7%
5	1	2	70.5	1868	2.1	1865	99.8%
5	1	3	70.6	1919	1.7	1907	99.4%
5	1	4	69.4	1659	1.6	1656	99.8%
5	1	5	70.5	2028	1.6	2016	99.4%
5	2	1	71.1	209293	1.5	208892	99.8%
5	2	2	70.7	212459	1.5	210879	99.3%
5	2	3	70.6	215405	1.5	213373	99.1%
5	2	4	71.1	213500	1.5	212826	99.7%
5	2	5	72.6	232743	1.5	232471	99.9%

Table2(Continued).

Problem	Set	Rep.	IP Time	IP Solution	GA Time	GA Solution	GA % of Optimal
6	1	1	7.2	2696	1	2687	99.7%
6	1	2	12.7	2494	1	2482	99.5%
6	1	3	7.6	2596	1	2581	99.4%
6	1	4	8.2	2364	1	2352	99.5%
6	1	5	10.5	2750	1	2729	99.2%
6	2	1	9.6	285640	1	281889	98.7%
6	2	2	9.2	250877	1	248215	98.9%
6	2	3	8.6	279490	1	276349	98.9%
6	2	4	9.1	278705	1	277315	99.5%
6	2	5	7.1	300950	1	300108	99.7%
7	1	1	1432	1741	17.3	1738	99.8%
7	1	2	1703.5	1832	17.4	1827	99.7%
7	1	3	1793.1	1756	16.3	1754	99.9%
7	1	4	1781.1	1702	16.2	1702	100.0%
7	1	5	1808	1431	16.7	1427	99.7%
7	2	1	1860	182005	17.5	181831	99.9%
7	2	2	1837.8	196216	16.4	195918	99.8%
7	2	3	1813.4	181227	16.6	180894	99.8%
7	2	4	2106.2	193235	17.4	192926	99.8%
7	2	5	1709.5	195020	20.8	194593	99.8%
8	1	1	2820.7	3925	27.9	3909	99.6%
8	1	2	2622.2	3809	27.7	3749	98.4%
8	1	3	2181	4023	27.7	4005	99.6%
8	1	4	2352.2	3502	28.3	3493	99.7%
8	1	5	2292.3	3595	41.5	3567	99.2%
8	2	1	2310.4	389377	30.9	388043	99.7%
8	2	2	2319.8	408589	31.9	401460	98.3%
8	2	3	2259.9	380120	34.6	374702	98.6%
8	2	4	2183.2	419711	28.3	414625	98.8%
8	2	5	2221.5	405990	31.9	402594	99.2%
9	1	1	443	5190	17.4	5135	98.9%
9	1	2	461.6	5204	18.1	5166	99.3%
9	1	3	444.1	5293	18.2	5233	98.9%
9	1	4	441.1	4826	17.5	4730	98.0%
9	1	5	440.9	5244	17.8	5209	99.3%
9	2	1	463.9	533016	18.7	527409	98.9%
9	2	2	459	552942	17.4	547113	98.9%
9	2	3	458.9	505802	17.2	495152	97.9%
9	2	4	462	582159	17.7	569619	97.8%
9	2	5	454	594118	17.5	591083	99.5%

While the IP model has the capability of solving all problem instances within the time limit, the average solution times for problems 7 and 8 are long. GA performs well on all problems in terms of solution time and quality as shown in Table 3 (results are averaged over 10 instances for each problem set). It is especially effective for solving the "hard" problems 7 and 8 by reducing the time down to tens of seconds and providing solution within 1% of the optima.

Table 3. Average Computational Time for various problem sizes.

Problem	M	N	IP Time	GA Time	GA Solution / Optimal IP
1	50	10	1.2	0.1	100%
2	50	25	1.8	0.1	100%
3	50	40	0.5	0.1	99.80%
4	100	20	41.5	0.9	100%
5	100	50	70.6	1.6	99.60%
6	100	80	9.0	1.0	99.30%
7	200	40	1784.5	17.3	99.80%
8	200	100	2356.3	31.1	99.10%
9	200	160	452.9	17.7	98.80%

As we mentioned earlier, the 'Probe' option in CPLEX can lead to reductions in problem size, but require more computer time and therefore total solving time may increase or decrease. We ran the IP model without using the "Probe" option to test its effectiveness on different test problems. The resultant model decreases the total solution time for those "easy" problems (1, 2, 3, 4, and 6) slightly. This is because the probing time spent in reducing the problem size cannot be justified by the time saved in solving the reduced problem. On the other hand, probing contributes to a significant reduction in solution time for problems 5, 7, 8, and 9. The time spent on probing pays off because the reduction in problem size leads to a significant reduction in the search space.

As shown in Table 4, the variance of the vector elements (Set 1 versus Set 2) does not appear to significantly affect the solving time of either method or the solution quality.

Table 4. Average computational results for low variance (Set 1) and high variance (Set 2) for different problems sizes.

Problem	Set	IP Time	GA Time	GA % Optimal
1	1	1.2	0.1	100%
	2	1.2	0.1	100%
2	1	1.7	0.1	100%
	2	1.8	0.1	100.0%
3	1	0.6	0.1	99.8%
	2	0.5	0.1	99.8%
4	1	41.1	0.9	100%
	2	42.0	0.9	100%
5	1	70.0	1.7	99.6%
	2	71.2	1.5	99.5%
6	1	9.2	1.0	99.5%
	2	8.7	1.0	99.1%
7	1	1703.6	16.8	99.8%
	2	1865.4	17.7	99.8%
8	1	2453.7	30.6	99.3%
	2	2259.0	31.5	98.9%
9	1	446.1	17.8	98.9%
	2	459.6	17.7	98.6%

The performance of the IP model with CPLEX varies in response to the problem complexity. As seen in Table 3, for any given length of the long vector, the problem is hardest for CPLEX when the short vector is half the size of the long vector. In our test problems, given the size of 50, 100, and 200 for the long vector, it took the IP model the longest to solve when the lengths of short vectors are 25, 50, and 100 respectively. This makes sense in that the maximum number of feasible solutions of the IP model is $\binom{m}{m-n}$ and it is the biggest when $m/2 = m-n$, that is $n = m/2$.

An experiment is designed to analyze CPLEX solution time and the results are summarized in Table 5. The dependent variable is the solution time required by CPLEX to solve the problem and the independent variables used are m (length of the long vector), ratio (the ratio of the short to the long vector, that is, n/m), the

variance of the data elements and cross terms. The results confirm that the vector size and the ratio of the number of elements in the short vector to the number of elements in the long vector are the important factors. Variance is not significant.

Table 5. Results of the factor analysis on the 90 test problems.

R^2 = 0.99; F = 582.18; p<.0001

Parameter	p - Value
Intercept	0.9879
M	<.0001
Ratio	<.0001
Variance	0.9076
M*Ratio	<.0001
M*Variance	0.9827
Ratio*Variance	0.0213
M*Ratio*Variance	0.0049

5. Conclusion

We provided two approaches to solving the problem of maximizing correlation between two streams of data measured over time, when one of the streams of data has missing values. We developed an integer programming model of the problem and used CPLEX with its probing option. We also developed a genetic algorithm and compared its performance to the integer programming approach on ninety test problems. The IP approach worked well, that is, provides the provably optimal solution in a reasonable amount of time for most problems. The most difficult problems to solve are those for which the percentage of missing data points approaches one half. The genetic algorithm is much faster than IP and although it cannot guarantee optimality, provided near optimal solutions for the test bank of ninety problems used in this study.

References

[1] Lisiecki, L.E., Lisiecki, P.A., Application of dynamic programming to the correlation of paleoclimate records, *Paleoceanography* 2002 **17** (4).

[2] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[3] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[4] Pirlot, M., General local search methods, *European Journal of Operational Research* 1996 **92** (3), 493-511.

Received: March 26, 2008