

Computational Efficiency of Structural Image Matching

Richard Chase, Boris Kovalerchuk
Department of Computer Science
Central Washington University
Ellensburg, WA, U.S.A

Abstract. *This paper addresses computational efficiency issues of a new algebraic method for imagery registration/conflation. An algebraic approach to conflation/registration of images does not depend on identifying common points. It uses the method of algebraic invariants to provide a common set of coordinates to images using chains of line segments formally described as polylines with possible gaps between segments. The original design of the method operates with matrixes and has complexity $O(n^3)$ for entire matrixes and complexity $O(n^5)$ for individual binary comparisons.*

This paper shows that this polynomial complexity can be significantly reduced by converting each matrix to the special linear structures with n elements. Generation of the special linear structure itself requires $n \log n$ binary comparisons for sorting, and the complexity of comparing two polylines overall is $O(n^3 \log n)$. The ability to perform these matching computations quickly is essential in correlating images containing large quantities of polylines.

Keywords: imagery conflation, algebraic invariants, structural polyline match, linear structure.

1. Introduction

The algebraic relational approach for geospatial feature correlation has been developed in [1,2,3], where it was shown that extracting a shared subsystem from linear features can be performed using algebraic invariants. The complexity of this process is $O(n^5)$ for comparing all subsystems [1,2]. In this paper, we present the exact formula

for this complexity, and show that the method from [1,2] can be modified to achieve a complexity of $O(n^4)$. In addition, we present a new method for identifying subsystems using a special linear structure (LS), that is of complexity $O(n^4)$, and can be modified to achieve $O(n^3 \log n)$.

2. Preliminaries

Let us begin by reviewing the main definitions and conflation algorithms from [1,2].

Definition. A pair $\mathbf{a} = \langle A, \Omega \rangle$ is called an algebraic system [4] if A is a set of elements, Ω is a set of predicates $\{P\}$ and operators $\{F\}$ on A and on its Cartesian products, where

$P: A \times A \times \dots \times A \rightarrow [0,1]$ and $F: A \times A \times \dots \times A \rightarrow A$.

Definition. A triple $\mathbf{a} = \langle A, R, \Omega \rangle$ is called a multisort algebraic system if A and R are sets of elements, Ω is a set of predicates $\{P\}$ and operators $\{F\}$ on A and R and on their Cartesian products, where $P: B_1 \times B_2 \times \dots \times B_n \rightarrow [0,1]$ and $F: B_1 \times B_2 \times \dots \times B_n \rightarrow B_{n+1}$, where each B_i is A or R .

A set of axioms can be associated with an algebraic system to generate a specific system such as a group, a field or a linear feature.

Definition. An algebraic system $\mathbf{a} = \langle A, R, \Omega_a \rangle$ is called a linear feature if R is a set of real numbers, Ω_a consists of two operators (functions) $D(a_i)$ and $L(a_i, a_j)$ and three predicates (linear order relations) $>_a, \geq_D, \geq_L$.

Thus $\Omega_a = \langle D(\cdot), L(\cdot, \cdot); >_a, \geq_D, \geq_L \rangle$ where:

1. $\forall a_i, a_j \in A: a_i >_a a_j$ or $a_j >_a a_i$ (All elements of A are totally ordered.)

2. $D: A \rightarrow [0, \infty)$ (An element a is called a linear interval and $D(a)$ is the *length* of a .)
3. $L: A \times A \rightarrow [0, 360]$ ($L(a_i, a_j)$ is called the *angle* between a_i and a_j .)
4. $a_i \geq_D a_j \Leftrightarrow D(a_i) \geq D(a_j)$. (This links \geq_D with $D(\cdot)$). We call elements a_i, a_j linear intervals and say that element a_i is no shorter than element a_j if $a_i \geq_D a_j$, that is, $D(a_i) \geq D(a_j)$).
5. $\forall a_i, a_j, a_k, a_m \in A: (a_i, a_j) \geq_L (a_k, a_m) \Leftrightarrow L(a_i, a_j) \geq L(a_k, a_m)$ (This links \geq_L with $L(a_i, a_j)$).

Properties:

$$\forall a_i, a_j \in A: a_i \geq_D a_j \text{ or } a_i \geq_D a_i$$

$$\forall a_i, a_j, a_k, a_m \in A: (a_i, a_j) \geq_L (a_k, a_m) \text{ or } (a_k, a_m) \geq_L (a_i, a_j)$$

Definition. An algebraic system \tilde{a} is called an *abstracted linear feature* of feature \mathbf{a} if $\Omega_{\tilde{a}}$ consists of three predicates (linear order relations) $>_a, \geq_D, \geq_L$, with $\Omega_a = \langle >_a, \geq_D, \geq_L \rangle$ from the linear feature \mathbf{a} .

Definition. An algebraic system $\mathbf{e} = \langle E, R, \Omega_e \rangle$ is a *subsystem* of an algebraic system $\mathbf{a} = \langle A, R, \Omega_a \rangle$ if $E \subseteq A$ and $\Omega_e = \Omega_a$. The subsystem relation is denoted by $\mathbf{e}_a \subseteq \mathbf{a}$.

Definition. An algebraic system $\mathbf{e} = \langle E, R, \Omega_e \rangle$ is a *shared subsystem* of algebraic systems $\mathbf{a} = \langle A, R, \Omega_a \rangle$ and $\mathbf{b} = \langle A, R, \Omega_b \rangle$ if $E \subseteq A, E \subseteq B$ and $\Omega_e = \Omega_a = \Omega_b$.

Consider now an example [2] for matching. Using a notation similar to that above, denote the angle between a_i and a_{i+1} as $L_i = L(a_i, a_{i+1})$. Then record in a matrix comparisons between all pairs of angles in a given feature. As above record the i, j entry as 0 iff $L_i < L_j$ and 1 iff $L_i \geq L_j$.

Note the resulting matrix is symmetric. Now when two such matrices are built for two features \mathbf{a} and \mathbf{b} , a more complex approach in determining a maximum co-reference is based on searching for the largest common part these matrices. This is done by “sliding down” the diagonals of the matrices. Consider for example the following Tables where the largest common parts are highlighted. We will refer to this Matrix Matching Algorithm [2, 3] as MMA1. Later we will also examine an improved Matrix Matching Algorithm, MMA2.

Table 1. Illustrative matrix for feature \mathbf{a} .

	L1	L2	L3	L4	L5	L6
L1	1	0	1	1	0	1
L2		1	0	1	0	0
L3			1	0	1	0
L4				1	1	0
L5					1	1
L6						1

Table 2. Illustrative matrix feature \mathbf{b} .

	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	1	0	1	1	0	1	1	1	0
L2		1	0	1	0	0	0	0	0
L3			1	0	1	0	1	0	0
L4				1	1	0	1	1	1
L5					1	1	0	0	0
L6						1	1	1	0
L7							1	0	0
L8								1	1
L9									1

Next we will define a concept used in representing new Linear Structure Algorithms which we will refer to as LS1 and LS2.

Definition. An algebraic system $LS = \langle I, R, \Omega_l \rangle$ is called a *linear structure* for feature \mathbf{A} if

- (1) I is the set of integer indexes of elements a_i from feature \mathbf{A} according to their order $>_a$ in \mathbf{A} . That is $I = \{1, 2, \dots, n\}$.
- (2) The signature $\Omega_l = \langle \geq_{cD}, >_{cL}, \varphi, \psi \rangle$ contains two relations $\geq_{cD}, >_{cL}$, and two sorting functions φ and ψ that are defined on I such that:

for every i, j from I

$$\varphi(i) \geq_{cD} \varphi(j) \Leftrightarrow D(a_i) \geq D(a_j)$$

and for every i, j, k, m from I

$$\psi(i, j) \geq_{cD} \psi(k, m) \Leftrightarrow L(a_i, a_j) \geq L(a_k, a_m).$$

The function φ shows the results of sorting intervals according their lengths and ψ shows the results of sorting angles between adjacent intervals according to their magnitude.

Example: The sequence of lengths of intervals a_1, a_2, a_3, a_4 is 0.1m, 4.3m, 7.6m and 0.5m. Their indexes can be written as follows $\{1, 2, 3, 4\}$ then by sorting the lengths in descending order, the structure’s sequence can be represented by the reordering of the original indices thus: $\{3, 2, 4, 1\}$.

In formal terms, this means that we have a substitution (permutation) φ for indexes of lengths such that

$$\varphi = \begin{pmatrix} 1234 \\ 3241 \end{pmatrix}$$

A similar index substitution function is produced for angles (i.e. for pairs of linear segments with indexes [i, j] and [k, m]).

3. The Matrix Matching Algorithms

In this section, we begin by presenting a detailed analysis of MMA1 [2, 3] summarized above. We then analyze an improved Matrix Matching Algorithm MMA2.

3.1. An exact complexity formula for MMA1

In [2] it was shown that an upper bound on the complexity of MMA1 is $O(n^5)$ but the exact formula was not provided. The general bound is provided in [2] is:

$$n^2 1^2 + (n-1)^2 2^2 + (n-2)^2 3^2 + \dots + 2^2 (n-1)^2 + 1^2 n^2$$

We will now demonstrate an exact representation for the complexity of MMA1.

Theorem 1: The complexity of MMA1 when applied to $n \times n$ matrices is given by

$$\frac{1}{120} [2n^5 + 5n^4 - 5n^2 - 2n].$$

Proof.

Let S1 and S2 be $n \times n$ comparison matrices that share a common subsystem.

As mentioned in [2, 3], the matrixes being considered are anti-symmetric so that only upper triangular comparisons are required. Thus,

$$(n-k) + (n-k-1) + \dots + 3 + 2 + 1 = (n-k+1)(n-k)/2$$

binary comparisons need to be made for each subsystem. Thus, the first terms of each item in the sum can be replaced by $(n-k+1)(n-k)/2$, which leads to the following representation for the full system:

$$(n(n-1)/2) * 1^2 + ((n-1)(n-2)/2) * 2^2 + \dots + 1 * 0 * (n)^2.$$

We can represent and then manipulate the equation as follows.

$$\begin{aligned} & \sum_{k=1}^n k^2 (n-k+1)(n-k)/2 = \\ & \frac{1}{2} \sum_{k=1}^n (k^4 - (2n+1)k^3 + (n^2+n)k^2) = \\ & \frac{1}{2} \left(\sum_{k=1}^n k^4 - (2n+1) \sum_{k=1}^n k^3 + (n^2+n) \sum_{k=1}^n k^2 \right) = \\ & \frac{1}{2} \left[\left(\frac{1}{5} n^5 + \frac{1}{2} n^4 + \frac{1}{3} n^3 - \frac{1}{30} n \right) - \right. \\ & (2n+1) \left(\frac{1}{4} n^4 + \frac{1}{2} n^3 + \frac{1}{4} n^2 \right) + \\ & \left. (n^2+n) \left(\frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n \right) \right] = \\ & \frac{1}{120} (2n^5 + 5n^4 - 5n^2 - 2n). \quad \square \end{aligned}$$

Thus, we have shown this is indeed $O(n^5)$. For comparison purposes as we proceed, note that for $n = 10$ this is 2079.

It is interesting to note that for small n ($n \leq 60$) this equation behaves more like $O(n^4)$ due to the small leading coefficient. However, it is not generally unusual to have $n > 60$ segments in a feature.

3.2 The improved algorithm MMA2

The performance of MMA1 can be improved further using the principle of monotonicity. In this method, the largest potentially matching matrices are searched first. If the matrix match in progress should fail at a given location, we then continue from where the match was last successful. Thus, we are able to continue looking for a smaller match originating from the same starting positions without repeating work already performed.

In [2], the issue of *monotonicity* is presented by noting that the worst-case is often not what occurs. That is, if no common matrices of size $s \times s$ are found, then there is no reason to search for larger common submatrices. We have chosen to

now take a reverse viewpoint, while still utilizing the same basic principle. We shall attempt to verify the largest subsystems first from a given set of starting positions in the matrices. If during comparisons, elements are found that do not match, the entire match is not abandoned, the match is just scaled down to include the last matching elements. In this way, the principle of monotonicity can be combined in a dynamic programming approach that reduces the complexity involved.

This can be represented by the equation

$$\begin{aligned} & \sum_{k=1}^n k(n-k)(n-k+1)/2 = \\ & \frac{1}{2}[(n^2+n)\sum_{k=1}^n k - (2n+1)\sum_{k=1}^n k^2 + \sum_{k=1}^n k^3] = \\ & \frac{1}{2}[(n^2+n)(\frac{1}{2}n^2 + \frac{1}{2}n) \\ & - (2n+1)(\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n) \\ & + (\frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2)] = \\ & (n^4 + 2n^3 - n^2 - 2n)/24. \end{aligned}$$

We have shown the following:

Theorem 2: The complexity of MMA2 when applied to $n \times n$ matrices is $O(n^4)$, and is given by

$$(n^4 + 2n^3 - n^2 - 2n)/24. \quad \square$$

Again as an example, consider $n = 10$ which yields 495.

4. The Linear Structure Algorithms

In this section, we present two versions of the Linear Structure algorithm LS1 and LS2.

4.1 The Linear Structure algorithm, LS1

When setting up a comparison matrix, the values of the table reflect whether the value of element x is greater than or equal to or less than the value at element $x+1$, $x+2$, $x+3$, and so on. In deriving a program to model all possible combinations of values that also satisfy these same

relationships, we found the best method was to first sort the values, keeping track of the indices from which they came, set up a new group of values from lowest to highest, then distribute them back into the proper order as per the original indices. That is, the order of the original indices after sorting fully describes the comparison matrix. Given this order, it is a simple matter to reconstruct the matrix. In fact, this order itself can be used to compare two features in a way in which you do not need the comparison matrices at all.

This is what we call the *linear structure's* sequence. The following example illustrates this concept.

Let us use the length measurements for feature S1: {5, 2, 8, 9, 4}, which have the comparison matrix shown in Table 3. The values 0 and 1 are used to represent $x < y$ and $x \geq y$ respectively, where x is a row value and y is a column value.

Table 3. Comparison of lengths for feature S1

Length	5	3	8	9	4
5	-	1	0	0	1
2		-	0	0	0
8			-	0	1
9				-	1
4					-

Now, we also have S2: {8, 9, 4, 6, 7} (see Table 4). The matching segment is highlighted in both Table 3 and Table 4.

Table 4. Comparison of lengths for feature S2

Length	8	9	4	6	7
8	-	0	1	1	1
9		-	1	1	1
4			-	0	0
6				-	0
7					-

Let's build the index sequence for Table 3. This table depicts the set of lengths of intervals {5,3,8,9,4} and their relationships. The structural sequence for this table will begin with the value 2, which is the index of the smallest element (3). The next value 5 indicates the second smallest element

is from index 5 whose value was 4. This process continues and the final sequence of the indices in S1 is {2, 5, 1, 3, 4}. The sequence of the values by index in S2 is {3, 4, 5, 1, 2}, and is derived from table 4 in a manner similar to S1. At first it seems that there is no similarity, but when the first two elements of S1 have been stripped off and the values ranked again, we find the S1 LS-sequence to be {3,1,2}. Further, when the last two elements of S2 have been stripped off, we find the S2 LS-sequence to be {3,1,2} also. We have found the matching segments. All that is necessary is to delineate all sequential element segments, convert them to linear structures then compare these sequences with other LS-sequences starting with the longest segments first until we find an exact match. Each sequence will require sorting which can be done in $O(n \log n)$ using, say, a mergesort algorithm [5], and the overall total will involve the addition of all combinations.

This can be represented by the formula:

$$\sum_{k=1}^n (n+1-k)k \log(k)$$

and since $\log k \leq \log n$ we have

$$\begin{aligned} \sum_{k=1}^n (n+1-k)k \log(n) &= \\ n \log(n) \sum_{k=1}^n k + \log(n) \sum_{k=1}^n k - \log(n) \sum_{k=1}^n k^2 &= \\ n \log(n) \left(\frac{1}{2} n^2 + \frac{1}{2} n \right) + \log(n) \left(\frac{1}{2} n^2 + \frac{1}{2} n \right) & \\ - \log(n) \left(\frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n \right) &= \\ \frac{2}{3} n^3 \log n + n^2 \log n + \frac{1}{3} n \log n . & \end{aligned}$$

We have shown the following:

Lemma 1: The complexity of generating LS-sequences, which is $O(n^3 \log n)$, is given by

$$\frac{2}{3} n^3 \log n + n^2 \log n + \frac{1}{3} n \log n . \square$$

This process of generating sequences can be performed for each feature before any comparisons between features are made because it requires no information from any other feature. It

needs to be noted that mergesort or another stable sorting method is preferred to a sort such as quicksort here because of the importance of preserving the correct order in the event that two equal values are found.

We can now analyze the complexity of the actual comparisons between features.

- For each sequence of length $n-k+1$ a comparison must be exact for the entire matching section. This requires $n-k+1$ comparisons. Thus, this is a simple $O(n)$ operation.
- The number of subsequences of length $n-k+1$ produced from a sequence of length n is k .
- Each of k subsequences of length $n-k+1$ generated in one feature must be cross referenced with each sequence of the same length in feature 2, which will be $O(k^2)$.

This can be represented by the equation:

$$\begin{aligned} \sum_{k=1}^n k^2 (n-k+1) &= (n+1) \sum_{k=1}^n k^2 - \sum_{k=1}^n k^3 = \\ (n+1) \left(\frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n \right) & \\ - \left(\frac{1}{4} n^4 + \frac{1}{2} n^3 + \frac{1}{4} n^2 \right) &= \\ (n^4 + 4n^3 + 5n^2 + 2n) / 12 . & \end{aligned}$$

We have shown the following:

Theorem 3: The complexity of LS1 when applied to $n \times n$ matrices is $O(n^4)$ and given by

$$(n^4 + 4n^3 + 5n^2 + 2n) / 12 . \square$$

As noted, this is $O(n^4)$ which is similar in complexity to the MMA2 method. However, it is not quite as efficient as the MMA2 method. Consider for example $n = 10$. Here the value is 1210.

4.2 The Linear Structure algorithm LS2

We can improve the performance of the LS1 method by incorporating some additional storage and sorting of the linear structures created. If linear structures of same length sequences are stored in a sorted order (e.g. a binary tree) then

binary searching of this information for the exact match would require only $O(\log n)$ time instead of $O(n)$. The ordering does take extra time, amounting to $O(n^4 \lg(n))$, but it can be done off-line, i.e. before comparing two features. This modifies the previous analysis for LS1 method. The LS1 method has a smaller “off-line” part. Below we estimate the complexity of the “on-line” part of the method LS2. We define the “on-line” part the method that is applied for comparison of two features from two images.

We have k sequences of length $n-k+1$ in each feature F1 and F2 to be compared. Let’s take an arbitrary sequence, say (1,2,5,4,...) generated for feature F2 with $n-k+1$ elements. To find a match for this sequence in an ordered set of sequences for feature F1, we need $\log k$ comparisons of sequences, because we have k such sequences.

Next we have $n-k+1$ elements in each sequence, and we need to compare them with the same $n-k+1$ elements in another sequence element by element for each position, that is $n-k+1$. Thus, for a single sequence, say (1,2,5,4,...) we need $(n-k+1)\log k$ individual comparisons of values (indices).

Now we notice that there are maximum k different sequences of length $n-k+1$, that is the total number of comparisons is: $(n-k+1)*k*\log k$. Now, we derive the formula for all k :

$$\sum_{k=1}^n (n-k+1)k * \log(k) =$$

$$\sum_{k=1}^n ((n+1)k - k^2) * \log(k) =$$

$$(n+1) \sum_{k=1}^n k \log(k) - \sum_{k=1}^n k^2 \log(k) =$$

Since always $k \leq n$, we can substitute $\log(n)$ for $\log(k)$, which yields:

$$(n+1) \sum_{k=1}^n k \log(n) - \sum_{k=1}^n k^2 \log(n) =$$

$$(n+1) \log(n) \sum_{k=1}^n k - \log(n) \sum_{k=1}^n k^2 =$$

$$(n+1) \log(n) \left(\frac{1}{2} n^2 + \frac{1}{2} n \right)$$

$$- \log(n) \left(\frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n \right) =$$

$$\log(n) * (n^3 + 3n^2 + 2n) / 6$$

Thus, we have an upper bound for our analysis of

$$\frac{1}{6} n^3 \log(n) + \frac{1}{2} n^2 \log(n) + \frac{1}{3} n \log(n)$$

Now we have demonstrated

Theorem 4: The complexity of the “on-line” part of LS2 method when applied to two linear features with $n \times n$ matrices is $O(n^3 \log n)$. \square

Table 5. Comparison of complexities of methods

N	10	20	40	80
MMA1	2079	59983	1813266	56319732
MMA2	495	7315	111930	1749060
LS1	1210	16170	235340	3586680
LS2	731	6656	61096	559870

Table 5 shows comparison of complexities of four methods presented above for some typical n values used in linear feature encoding.

For cross-referencing more than two features, this LS2 matching process would need to be extended. We can build this extension using the same method of storing all linear structures with equal lengths sequences in a sorted order.

For $n = 10$ we have $1*10+2*9+3*8+4*7+5*6+6*5+7*4+8*3+9*2+10*1 = 165$ segments. These segments can then be found in $(n^3 - n)/6$ or $O(n^3)$ time.

If we let m be the total number of features, and we assume for this case that all features are of length n , then to compare to all other features we will have $O(mn^3 \log n)$.

The complexity of the LS2 has shown to be the best for processing efficiency. We have not, however, taken into account the *storage* requirements for maintaining the list of sequences which is $O(n^2)$.

An image with 1000 polylines of 100 points each would require $1000 \cdot 100 \cdot 101/2$ storage locations. Since all the indexes in this case are < 256 we could use a single byte of storage apiece, resulting in a storage requirement of 5,050,000 or 5 MB, which is quite reasonable.

5. Conclusion

Of the methods investigated, we have found the modified linear structure method LS2 provides the best efficiency with $O(n^3 \log n)$ followed by the modified matrix matching algorithm MMA2 with complexity $O(n^4)$.

The new method presented in this paper shows a new way to approach the matrix matching type of comparison algorithms that is more efficient than methods previously developed.

By converting each matrix to the special linear structure, the polynomial complexity of matching subset extraction is significantly reduced.

6. Acknowledgements

This research has been supported by the University Research Initiative grant from the National Imagery and Mapping Agency.

References

- [1] B. Kovalerchuk, J. Schwing, Algebraic relational approach for geospatial feature correlation, In: *Proceedings of International Conference on Imaging Science, Systems, and Technology* (CISST'2002, June 24-27, 2002), Las Vegas, pp. 115-121, 2002, www.cwu.edu/~borisk/pub/CWUMethod.pdf
- [2] B. Kovalerchuk, W. Sumner, Algebraic relational approach to conflating images, In: *Proc. of SPIE International Conference Aerosense' 2003*, Orlando, FL, 21-25 April, 2003, SPIE, 2003 (in print).
- [3] B. Kovalerchuk, W. Sumner, J. Schwing, Image Registration and Conflation Based on Structural Characteristics. In: *Proceedings of International Conference on Imaging Science, Systems, and Technology* (CISST'2003, June 23-26, 2003), Las Vegas, 2003 (this volume).
- [4] A. I. Mal'cev, *Algebraic Systems*, Springer-Verlag, New York, 1973.
- [5] R. Neapolitan, K. Naimipour, *Foundations of Algorithms Using C++ Pseudocode*, Jones and Bartlett Publ., 1998.