Java Conflation Suite

Technical Report

DRAFT

VIVID
SOLUTIONS

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	BRIEF DESCRIPTION OF CHANGE
Draft		Martin Davis	

Table of Contents

> **con·flate**
>
> 1. To bring together, meld or fuse.
>
> 2. To combine (two variant texts, for example) into one whole.
>
> *TheAmericanHeritage®DictionaryoftheEnglishLa nguage,FourthEdition*
> *Copyright©2000byHoughtonMifflinCompany.*

1. INTRODUCTION

This document details the design approach and algorithms investigated during the Java Conflation Suite project. As well as documenting in detail the algorithms used in the delivered software, it includes algorithms evaluated but not ultimately used.

JCS relies on other two other suites of software for performing core spatial processing. These are:

- The JUMP Unified Mapping Platform, which provides a GUI, spatial data I/O, and other functions
- The Java Topology Suite (JTS), which provides a geometry model and basic spatial operations on it
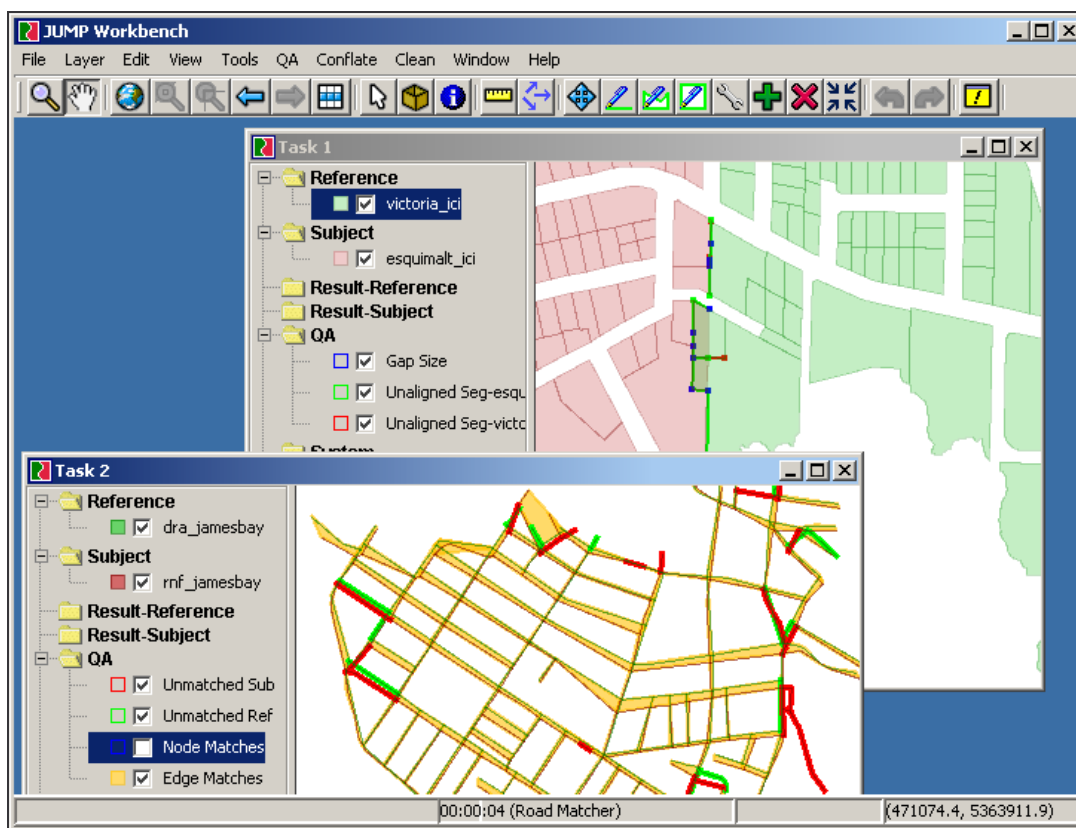
Figure 1- JCS running in the JUMP Workbench

1.1 RELATED DOCUMENTS

- JCS User Guide
- JCS Developer Guide
- JUMP Technical Report
- JUMP User Guide
- JUMP Developer Guide
- Java Topology Suite (JTS) documentation (http://www.vividsolutions.com/jts/jtshome.htm)

2.   DISCUSSION OF CONFLATION

2.1 OVERVIEW OF CONFLATION

Conflation is a process carried out during the integration of spatial datasets. It generally refers to a process that changes or adds to the geometric description of a given dataset to make it match another reference dataset according to some kind of criteria.

One of the main challenges in the JCS project is defining exactly what is meant by the term "conflation", in the context of the project. Conflation is a very general term that has no generally agreed-upon meaning in the GIS community. Moreover it is used to apply to several quite different concepts:

- It can refer to the activity of carrying out the integration of spatial datasets. We will denote this as "the conflation activity"
- It can refer to any or all of many different computer algorithms for performing a detection, matching and update on spatial data sets. We will denote these as "conflation algorithms", and will use more specific names to identify particular algorithms

2.2 CONFLATION TERMINOLOGY

Up to now conflation has been a relatively little studied field. As such it does not have a well-established terminology. A standard terminology is essential for both purposes of communication and for placing the field upon a more formal basis. During the course of this project we have attempted to identify terms that capture some of the important aspects of conflation. In using them we have been able to crystallize their definitions somewhat. Wherever possible we have tried to use existing terminology, but there are very few terms that can be considered to be standard or have clear, accepted definitions.

Term	Meaning
adjustment	The process of changing the location of a geometry (in particular a point or line segment) to bring it into **alignment** with another geometry. Also: the magnitude or delta vector associated with such a change.
alignment	The state of a geometry or its components being identical to another geometry or components. Also: the action of **adjusting** a geometry to bring it into alignment with other geometries. (Also known as "discrepancy elimination" elsewhere in the literature)
attribute transfer	The process of updating or adding attribute values to a feature from a matching feature.
component (of a geometry)	In the geometric model used by JCS (which is the model defined by **JTS**), geometric objects are typically built of simpler elements. For example, linestrings are built up of line segments, and polygons are build up of closed linestrings (rings). A component of a geometry is a subset of the points of a geometry,

	which form a geometric element in their own right.
conflation	The process of merging or combining information from two spatial datasets to create a new dataset with improved accuracy and/or consistency.
feature	The fundamental way of representing geographic or spatial objects. A feature has a set of named attributes (or properties). In the most general definition (for instance, as defined by the OGC), one or more of the attributes is usually, but not always, of a geometric type. In JCS, a feature always has exactly one attribute of type geometry.
feature class	The set of all features that represent objects of the same real-world type. The members of a feature class have the same **feature schema**.
feature collection	A set of features all belonging to the same feature class.
feature schema	An ordered list of {name, type} pairs, which define the names and types of the attributes of a feature class or feature collection.
horizontal conflation	Conflating two datasets that occupy separate areas. Examples are boundary alignment between two adjacent coverages, and edge matching between adjacent networks.
internal conflation	Conflating the geometries within a single dataset. An example is coverage cleaning, the process of ensuring that a dataset of polygons does not contain any overlaps or gaps less than a given tolerance.
JTS	The Java Topology Suite, an API that both defines a geometric model and provides fundamental operations on that model. JCS makes extensive use of JTS to represent and manipulate geometry
match	The process of comparing two geometric objects to determine how similar they are in terms of location or shape. The matching process may result in a match value being determined, which is a numerical quantity that measures the degree of similarity. A match may be constrained by a given **tolerance value**; if the match value is greater than this tolerance then no match is assigned. Also: the resulting association made between the two geometries.
reference dataset	A dataset that is considered to be relatively fixed during the conflation process; the dataset the **subject dataset** is aligned to. This is usually the dataset considered to be more accurate or recent. In certain kinds of conflation the reference dataset may be modified, since for certain kinds of topology it is not possible to add features without changing the dataset they are added to (e.g. adding edges to a network may required introducing new nodes; adding polygons to a coverage may require introducing new nodes along the common edges).
subject dataset	A dataset that contains information, which needs to be added to, or adjusted to fit a **reference dataset**.
tolerance value	A value that controls how close geometric components must be in order to match or to be adjusted.
topology	The properties of a geometry or relationships between

VIVID
S O L U T I O N S

	geometries which remain unchanged by homeomorphic transformations (e.g. continuous deformations of the plane).
vertical conflation	The conflation of two datasets that occupy the same area (or overlap). Examples would be matching two different road networks for the same area, and aligning polygon boundaries in one coverage to edges in a different dataset for the same area.

2.3 DISTINCTIONS BETWEEN CONFLATION AND OTHER SPATIAL PROCESSES

Conflation is distinct from generalization. Generalization tends to reduce the total number of vertices present in a dataset, to reduce the size and complexity of the data.
In conflation, vertices are usually maintained or added in order to make two datasets match at every common point. Also, in conflation a primary goal is to minimize the changes to the input datasets (for instance, by keeping as many vertex locations as possible unchanged.)

2.4 CLASSIFICATION OF CONFLATION PROBLEMS

Yuan and Tao [Yua99] classify conflation problems into two types: horizontal and vertical. We add a third type: internal.

2.4.1 Horizontal Conflation

Horizontal conflation is the process of eliminating discrepancies along the common boundary of datasets which lie adjacent to one another. Typically the datasets contain data from the same feature classes. Examples are aligning the boundaries of adjacent coverages, or edge-matching neighbouring networks.

2.4.2 Vertical Conflation

Vertical Conflation involves matching and/or eliminating discrepancies between datasets which occupy the same area in space. Two important kinds of Vertical conflation are **Version Matching** and **Feature Alignment**.

Version Matching. In this case the input datasets consist of different versions of the same features. The conflation process is intended to identify matching features. Attributes may be transferred between matched features, and unmatched features may be transferred in their entirety.   An example of this is matching different versions of road networks for the same geographical area.

Geometric Alignment. In this case the input data consists of features from two or more different feature classes which intended to bear some defined relationship to each other. The conflation process is intended to remove discrepancies between the datasets that causes this relationship to fail to hold. A common relationship is that of geometric alignment. An example of this is aligning the boundaries of different kinds of feature classes such as municipal districts and lot parcels.

2.4.3 Internal Conflation

Internal Conflation can be thought of as a special case of horizontal conflation. It involves eliminating discrepancies which are internal to a single dataset. An example would be the common operation of coverage cleaning.

2.5 CONFLATION WORKFLOW

The process of conflating spatial datasets can be broken down into various common subtasks. Depending on the nature and quality of the data in a specific conflation problem some of these subtasks may be trivial or not required.

2.5.1 Data Preprocessing

This step normalizes the input datasets to ensure that they are compatible. For instance, they must have the same coordinate system. This may also involve format translation and any other basic preparation of the datasets.

2.5.2 Data Quality Assurance

Some conflation tasks require that datasets have a given level of internal consistency. For instance, coverage alignment algorithms require that the input datasets are in fact a clean coverage. During this step the internal consistency of the datasets is verifed and if necessary improved.

2.5.3 Dataset Alignment

In some cases datasets are sufficiently misaligned that an initial alignment process is required to allow more precise conflation to be carried out. This alignment is typically coarse grained in nature, not descending to the level of aligning individual features. This process may be either manual or automatic.

2.5.4 Feature Matching

During this step common features between the datasets are matched. This may be done either in an automated fashion using one or more conflation algorithms, or via manually determined matches.

After this phase has been performed the discrepancies between the datasets will have been identified. It is often useful to provide statistical summaries of data quality, or to visualize the discrepancies.

2.5.5 Geometry Alignment and/or Information Transfer

Once features have been matched the match information can be used to improve the quality of one or both of the input datasets.

Geometry Alignment removes discrepancies between geometries.

Information Transfer involves updating one dataset with information from the other. This information can be either attributes or geometry to be added to an existing feature, or entire features to be added to the dataset.
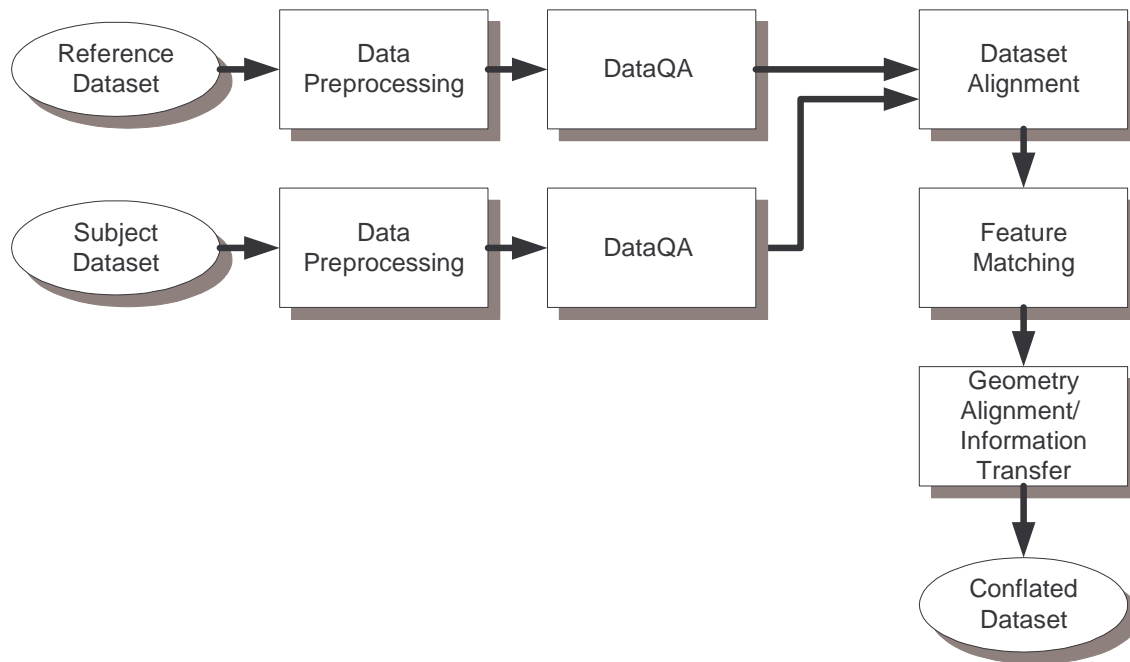
Figure 2 - Conflation Workflow

2.6 HUMAN-ASSISTED CONFLATION

Obviously the goal is to automate the solution of conflation problems entirely. However, this may not always be possible, for two reasons:

- **Algorithm Limitations**. This is the case if there is no currently known algorithm which is able to correctly identify all matches, or compute appropriate adjustments to eliminate discrepancies. In this case human input at different stages of the algorithm can increase the percentage of problems which can be fixed.
- **Data Ambiguity**. In this case there is not enough information contained in the input datasets to correctly identify matching features. It may not even be possible for a human operator to determine a correct course of action. Additional data attributes may need to be obtained in order to resolve the discrepancy.

The first limitation may be overcome by providing for human input into the conflation process. This input can occur at several places in the conflation workflow, including Dataset Alignment, Feature Matching, and Geometry Alignment.

Manual input into Dataset Alignment could include manually warping or shifting the data in order to reduce discrepancies to a level at which the available automated algorithm becomes effective.

Manual Feature Matching requires that human be able to create or delete matches between features. Ideally this can be carried out in conjunction with automated matching in a iterative process. It is important that an appropriate method of visualizing both the input data and current set of matches be provided.

Manual Geometry Alignment can take the form of simple geometry editing tools, as well as more specific tools such as Vertex Snapping. The User Interface should provide some method of determining that manual edits have correctly eliminated discrepancies.

For all of the above situations, the effectiveness of human assistance is directly proportional to the quality and richness of the user interface. This includes factors such as execution speed, ease-of-use, variety of feedback, ability to visualize spatial data items, and the presence of tools such as transaction recorders, multi-level undo, etc. The design of an effective user interface for human-assisted conflation is as much of a technical challenge as the design of the automated algorithms themselves.

3. PROJECT SCOPE

The focus of the JCS project is to solve conflation problems. To facilitate this, some aspects of spatial data processing and geomatics are not handled in JCS. These features do not impact the ability to perform conflation and maintain fidelity to the original data.

3.1.1 Coordinate Systems

For simplicity JCS assumes that all input data is in the same coordinate system. As a result, JCS does not provide facilities for specifying coordinate systems.

All internal calculations in JCS are carried out on a Cartesian plane. This does not preclude carrying out conflation operations on data in non-planar coordinate systems (such as geodetic coordinates), since most internal computation is combinatorial in nature, and the numerical computations are relatively insensitive to numerical accuracy. However, it could be important to be aware of this when specifying tolerance values.

3.1.2 Precision Model Issues

JCS assumes a floating precision model. Since JCS uses JTS to provided geometry representation, it uses IEE794 double precision floating point coordinates. This precision model is assumed to provide enough precision to represent all input data, as well as enough to provide extra precision to avoid robustness problems.

During conflation processes new points may be created. These points will be created with the full available precision. No attempt will be made to round the points to a lower precision.

4. JCS ARCHITECTURE

The JCS Architecture has been designed to be flexible, modular and reusable. Core functionality (conflation algorithms and other spatial algorithms) is exposed as a Java API. This allows JCS functionality to be incorporated in batch processes, and easily reused in other applications.

The following diagram illustrates the major building blocks of JCS. Each package is discussed in more detail below.
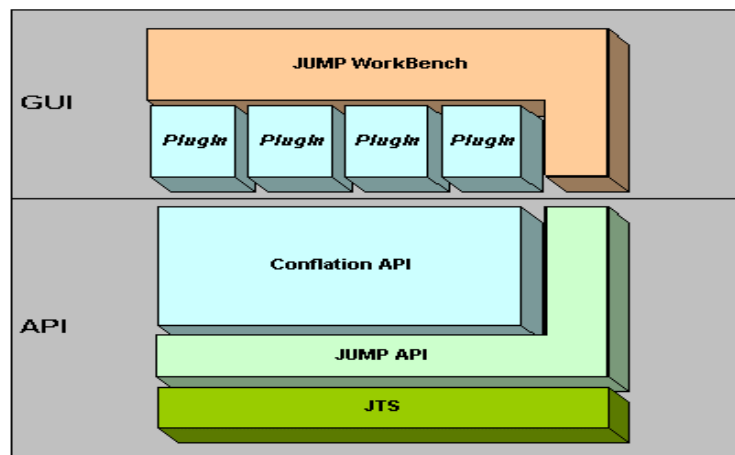

Figure 3 - JCS Architecture

4.1 JAVA TOPOLOGY SUITE (JTS)

JCS relies on the Java Topology Suite (JTS) to provide its geometry model and basic geometric operations. JTS is not considered to be part of JCS, but is used by almost all JCS components.

4.2 JUMP API

In order to carry out spatial data processing tasks a certain minimum level of general spatial functionality is required.   This includes things such as:
- spatial data input and output
- a model to represent spatial features with geometry and scalar attributes
- collections of features
- spatial access methods to improve performance

As part of the JCS project the JUMP Unified Mapping Platform has been developed to provide a common codebase for spatial functions and GUI tools.   The spatial functionality listed above is provided by the JUMP API.

4.3 CONFLATION API

The Conflation API is the core codebase for JCS. It provides algorithms for performing conflation tasks on spatial datasets, as well as supporting algorithms for other spatial functionality. In some cases these algorithms can be run in batch mode with predefined

input parameters; in other cases they are intended to support performing conflation tasks in an interactive scenario. The Conflation API includes many different kinds of spatial processing algorithms, including conflation between two datasets, spatial QA functions, and data cleaning functions for single datasets. Many of algorithms in the API are discussed in detail elsewhere in this document.

4.4 JUMP WORKBENCH

An important part of JCS is the exploration and development of tools for visualization of conflation tasks, and performing interactive conflation. The JUMP Workbench provides a highly functional and extensible GUI environment for spatial processing.   Many JCS functions and conflation tools are exposed as plugins for the JUMP Workbench.

5. CASE STUDIES OF CONFLATION PROBLEMS

This section lists various examples of conflation problems. These problems provide the motivation for the development of the conflation algorithms and processes in JCS.

5.1 COVERAGE CLEANING

A common kind of spatial data that is worked with is a polygonal coverage. This is a dataset which contains only polygons which do not overlap. In some cases the polygons are intended to completely cover a given area. In other cases, space or holes are allowed in the coverage to represent real-world entities (such as streets between blocks) (see Figure 4 below). In this case, however, the spaces are usually known to be larger than a certain amount.
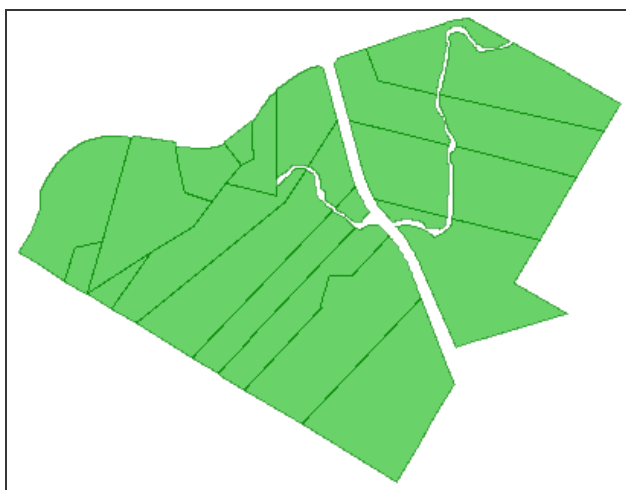

Figure 4 - A coverage containing spaces between areas covered by polygons

Two different kinds of errors can occur in datasets which are intended to be coverages. These are:

| Gaps	Places where two adjacent polygons are separated by a small amount along some or all of their boundary (see Figure 5a below). Since some coverages can contain legal spaces between polygons, gaps are always defined relative to a specified distance tolerance. Only spaces which have the adjacent line segments separated by less than the distance tolerance are considered to be gaps
Overlaps	Places where two polygons overlap (see Figure 5b below). No distance tolerance is needed, since **any** overlap is always considered to be an error in a coverage.
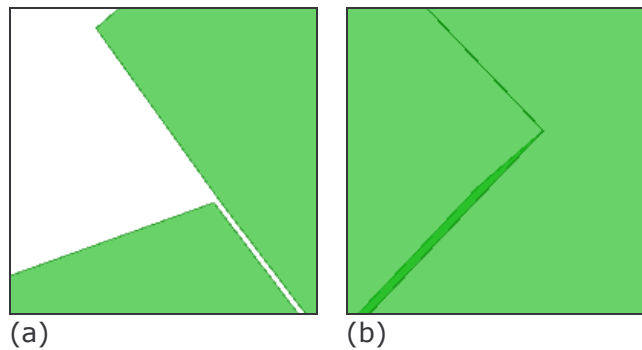
Figure 5 – Examples of (a) a gap (b) an overlap

Depending on how the coverage dataset was prepared, it can happen that gaps or overlaps are inadvertently introduced. Many GIS tools are not able to detect these problems. However, these gaps and overlaps are errors and should be removed from the dataset. Often these errors are very small and are undetectable by visual inspection. JCS provides tools to detect, display, and remove gaps and overlaps.

5.2 COVERAGE ALIGNMENT

Coverage Alignment is an example of Vertical conflation. It consists of aligning the edges of a Subject coverage to the edges of a Reference coverage that partially or completely overlaps the Subject.

5.2.1 Suburb-Road/Lot Alignment

The following images shows a situation involving two coverages for the same are. The yellow/red polygons are Suburb areas. The green and red polygons are a coverage of road and lot polygons contained in the suburbs. The edges of the suburb polygons are fairly close to the road/lot polygons, but not precisely aligned.
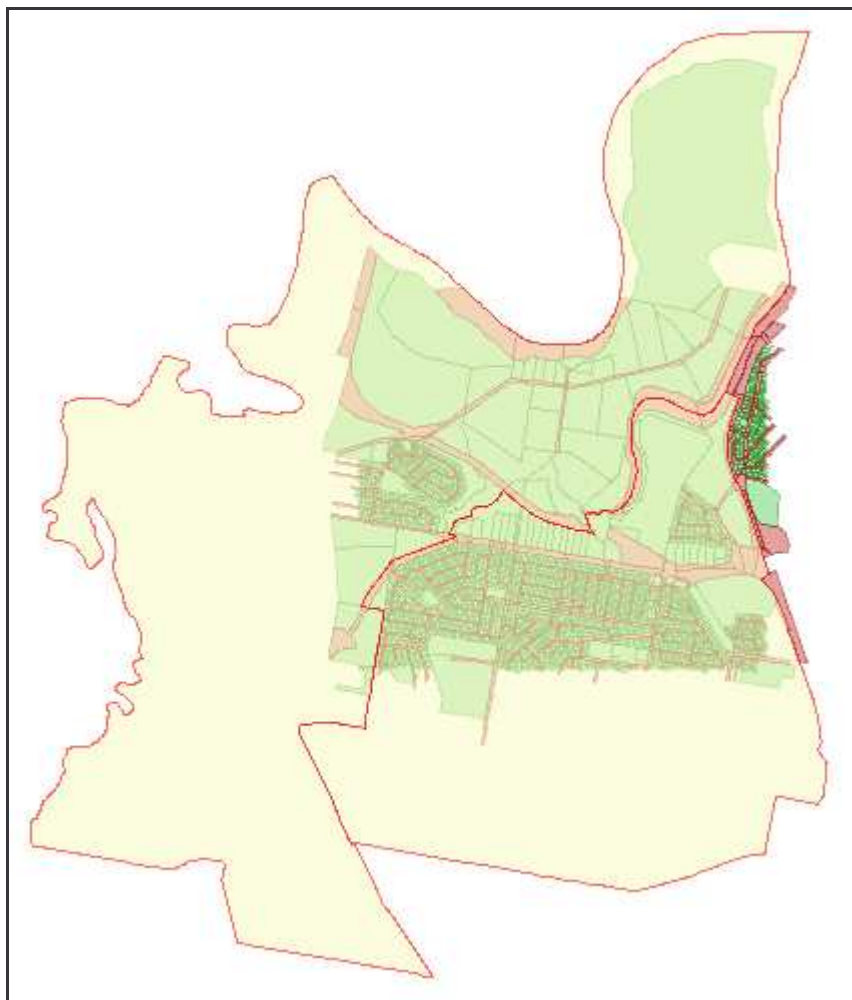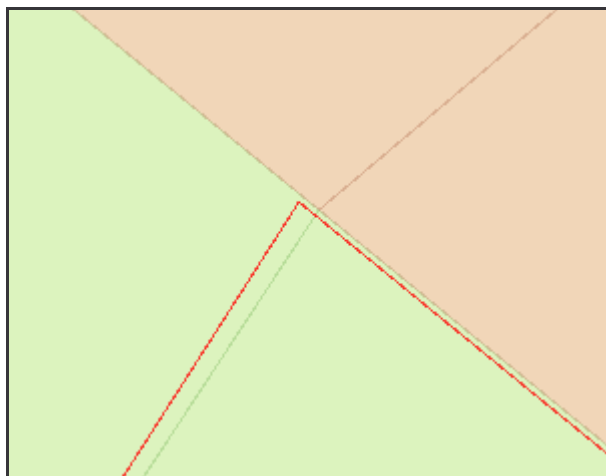
Figure 6 - Suburb and Road / Lot Coverages

Figure 7 - An example of a discrepancy between the Suburb boundary and the Road/Lot boundaries

Analysis: The conflation problem is to align the boundaries of the suburb coverage polygons to the road/lot coverage linework. The road/lot polygons are the Reference dataset, and the suburb polygons are the Subject dataset. The Subject dataset will be adjusted to align with the Reference dataset. The adjustments may include both moving and inserting vertices. A further conflation step is to insert vertices in the Reference dataset to ensure that it is noded exactly with the Subject dataset.

There are two options for noding when aligning Subject edges to matching Reference edges. If two contiguous Subject segments match a single Reference segment, so that a vertex exists in the Subject but not in the Reference, a choice can be made to either remove the vertex from the Subject (by merging the segments) or to insert the vertex into the Reference by splitting the segment. This choice can be made based on external requirements for how much change is allowable in each dataset.

Although not visible from the images, the polygons in the road and lot coverage in fact contain gaps and overlaps. Before conflation can be carried out, these problems must be corrected so that the road and lot polygons form a correct coverage.

5.2.2 Administrative Area-Watershed Boundary Alignment

In British Columbia the boundaries of many Administrative Areas follow watershed outlines (height-of-land) for portions of their length. Since the Admin boundaries were originally digitized the accuracy of the base watershed outlines has been improved. There is a need to align the Admin boundaries with the more accurate watersheds.

The image shows Admin boundaries in red and watershed outlines in blue. It clearly shows Admin boundary sections that follow the watersheds and sections, which follow surveyed lines.
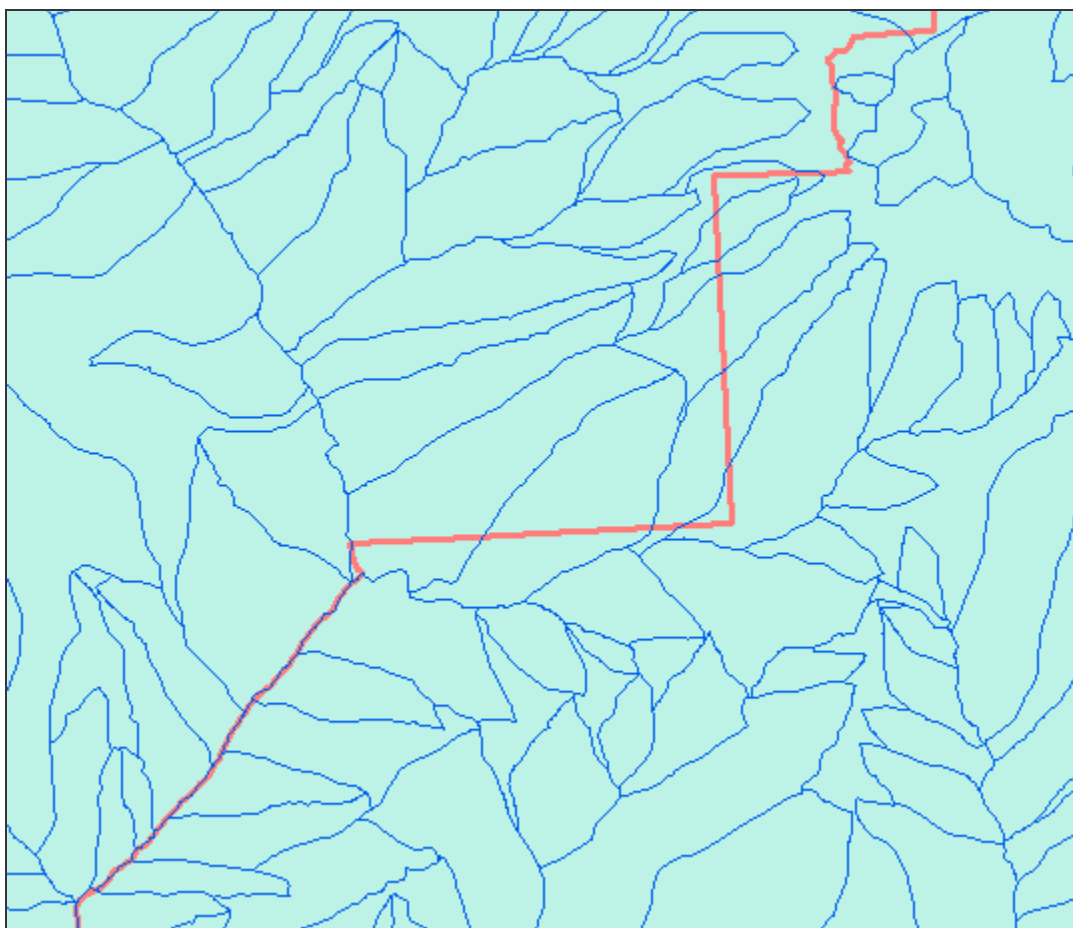
Figure 8 - Admin Boundaries and Watersheds

Analysis: The conflation problem is to align the boundaries of the Admin polygons with the boundaries of the watershed polygons. The watershed polygons are the Reference dataset, and the Admin boundaries are the Subject dataset.

One of the challenges in this problem is that only some portions of the Admin boundaries are intended to follow the watersheds. The non-watershed sections need to be distinguished from sections that are intended to be aligned with watershed boundaries. The non-watershed sections should not be altered by the conflation process.

5.3 BOUNDARY ALIGNMENT

An example of Boundary Alignment is aligning municipal lot coverages along common borders. The image shows datasets of lots in two municipalities that share a common border. The datasets are not perfectly aligned along the boundary, likely because they were digitized at different times without reference to the other dataset. It is desired to ensure that the two datasets can be merged to form a continuous, correct coverage. This requires conflating the datasets along their common boundary.

The image also shows lots that have a high percentage of overlap. Handling these overlaps is outside the scope of the alignment process, since they constitute a violation of business rules.
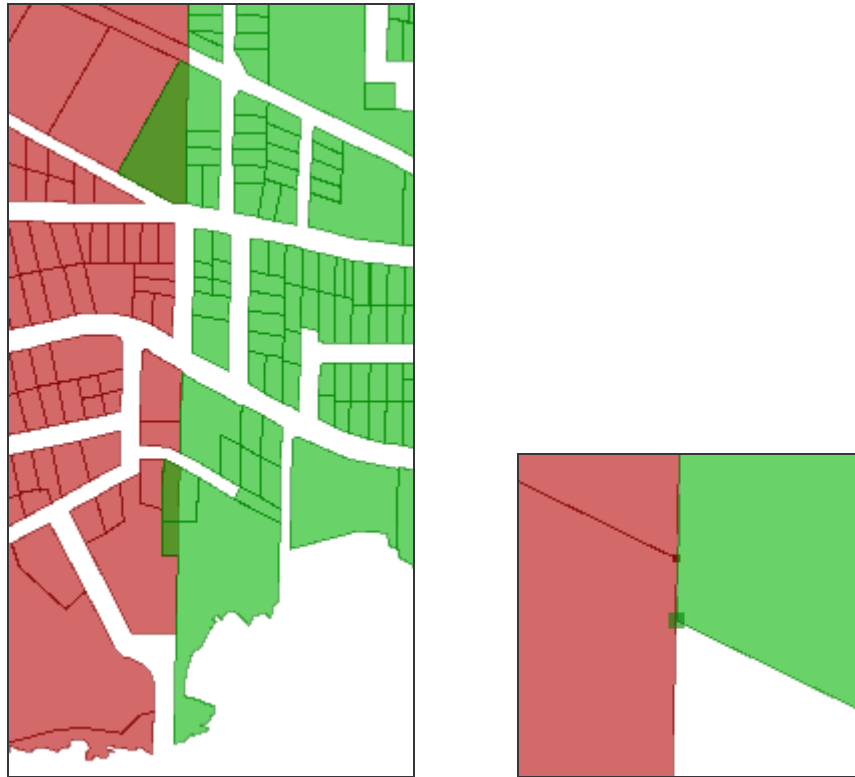

Figure 9 – (a) An example of boundaries to be aligned; (b) A close-up of misaligned edges

Analysis: Boundary alignment is a special case of Coverage Alignment. In Boundary Alignment, the Reference and Subject coverages are intended to touch, but not overlap, along a common boundary.

The conflation problem is to align the boundary edges of the Subject dataset with the matching boundary edges of the Reference dataset (so that the datasets would form a continuous, clean coverage if merged). One dataset is chosen to be the Reference dataset, and the other is chosen to be the Subject dataset (this choice may be arbitrary, or it may be made based on the known accuracies of the data). The Subject dataset will be adjusted to align with the Reference dataset. The adjustments may include both moving and inserting vertices. A further conflation step is to insert vertices in the Reference dataset to ensure that it is noded exactly with the Subject dataset.

As mentioned in the problem statement, polygons with large overlaps are outside the scope of the conflation process. However, they may need to be detected and eliminated from the alignment process, since they do not fit the underlying model of boundary alignment.

For further comments, see the Analysis of Coverage Alignment.

Boundary Alignment is an example of horizontal conflation.

5.4 ROAD NETWORK MATCHING

The following image is an example of two different Road Networks in the same area. The red roads are the Reference dataset, and the green roads are the Subject dataset. The Reference dataset is more accurate than the Subject dataset, but the Subject dataset contains attributes and some road features which do not appear in the Reference dataset.

Figure 10 - An example of two road networks to be conflated

Analysis: The conflation problem is to match the same roads between the two datasets and transfer attributes from the Reference to the Subject matched roads. Also, Reference roads, which do not appear in the Subject dataset, need to be warped to fit the Subject network and inserted into the Subject dataset. This may require adding nodes to Subject edges.

This problem is an example of Vertical Conflation.

6. DISTANCE METRICS FOR MATCHING

6.1 EUCLIDEAN DISTANCE

The **Euclidean distance** between two points a and b is defined as:

d(a, b) = sqrt(del x ^ 2 + del y ^2)

The Euclidean distance from a point to a geometry (set of points) can be defined as

d(a B) = min [b e B] d(a, b)

The Euclidean distance between two geometries is defined as:

d(A, B) = min [a e A] (a, B)

In general the Euclidean distance is not very useful for computing a match distance. This is because it provides a measure of how close two geometries are, not a measure of *how far apart* they are. Two geometries can be very close according to their Euclidean distance, but still be a very poor match.

6.2 HAUSDORFF DISTANCE

The **directed Hausdorff distance** is defined as:

dH(A, B) = max min d(a,b)

(d may be any metric, but for JCS purposes the Euclidean distance is used)

The general **Hausdorff distance** is defined as:

H(A, B) = max { dH(A, B), dH(B, A) }

The Hausdorff distance can be thought of as computing how far A is from B, measured at two single points. This is much more useful that the Euclidean distance for the purpose of estimating the closeness of a match between two geometries.
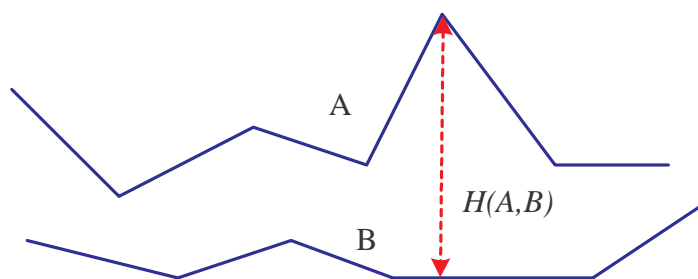

Figure 11 – An example of the Hausdorff Distance

One deficiency of the Hausdorff distance for matching purposes is that it is highly sensitive to outliers (such as may occur in noisy data). However, in cases where the geometries can be assumed to be free from noise (as is the case in hand-digitized datasets) the Hausdorff distance is very useful.

The Hausdorff distance between two line segments is straightforward to compute . It is simply the maximum of the distances from each vertex to the other line segment.

   H(L1, L2) = max (d (L1[0], L2) …

Unfortunately, computing the Hausdorff distance between geometries containing multiple line segments is complex and time-consuming. It can be shown that the endpoints of the line segment forming the Hausdorff distance lie on an intersection point of one geometry with the line Voronoi edges of the other geometry. Thus computing the Hausdorff distance in general requires computing the line Voronoi diagram of the geometries. Implementing a robust line Voronoi algorithm is non-trivial.
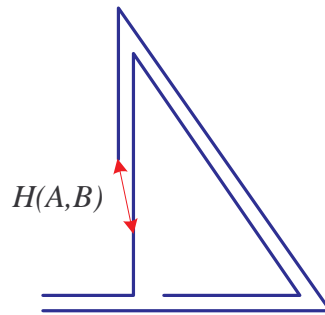

H(A,B)

Figure 12 - A Hausdorff Distance which has a non-vertex endpoint

6.3 VERTEX HAUSDORFF DISTANCE

The **Vertex Hausdorff Distance** is a simplification of the Hausdorff distance which is easier to compute in terms of both algorithmic complexity and performance. It is equal to the Hausdorff distance in many cases. Where it is not equal still provides a useful measure of how far apart two geometries are. The Hausdorff Vertex Distance is simply the Hausdorff distance restricted to the vertices of one of the geometries. As with the Hausdorff distance, the metric has both directed and non-directed versions.

   dVH(A, B) = max [a e V(A)] d(a, B)

    VH(A, B) = max { dVH(A, B), dVH(B, A) }

For linear geometries that are "mutually monotonic" the Vertex Hausdorff distance is a very close approximation of the Hausdorff distance.

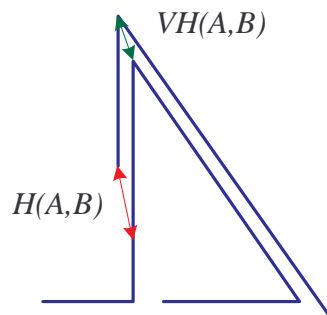**Figure 13 – A case where the Hausdorff Distance and the Vertex Hausdorff Distance are not equal**

7. FINDING GEOMETRY DIFFERENCES

A common problem in spatial data processing is to find differences in the geometries in two datasets. Geometry differences can be caused by:

- Ø Features being added or deleted
- Ø Geometry linework being altered
- Ø Data format conversion or other processing altering the values of coordinates, e.g. by explicitly reducing the precision or by numerical roundoff errors

A common situation is that one dataset is an updated version of the other, and it is required to determine the location and number of changes made to the original dataset.

These changes can be detected by means of a spatial difference algorithm. Difference algorithms work by matching geometries between the two datasets, and reporting the geometries in each dataset which do not have a match in the other dataset. Matching can be carried out in two different ways:

- Ø **Exact Matching** tests if geometries are the same on a point-by-point basis. This is useful if the datasets are known to be very similar and only small changes have occurred.
- Ø **Matching within a Distance Tolerance** tests if each point in the geometries lies within a given distance of some other point on the other geometry. This has the effect of ignoring changes smaller than the distance tolerance. This is useful in situations where many points have been shifted by small amounts (e.g. by a change in precision caused by format conversion) and only large-scale differences are of interest.

7.1 DIFF GEOMETRY

JCS provides a function called Diff Geometry to detect the differences between the geometry of features in two datasets. (The term "Diff" is a reference to a well-known Unix application for detecting the differences between two files. The JCS Diff Geometry function performs a similar function for spatial datasets.)

The basic algorithm is:

1. For each geometry a in dataset A
   a. attempt to find a matching geometry b in dataset B. (Since matches can only occur if two geometries are within the distance tolerance on one another, a spatial index is used to improve performance of this query)
   b. If a matches b, mark both as being matched
2. Report all a in A and b in B which are not matched

A slight variation on this algorithm involves matching components of geometries, rather than the geometries themselves. This allows ignoring cases where (for instance) a MultiPolygon in one dataset has be split into two separate Polygons in another dataset.

7.2 GEOMETRY MATCHING

The key algorithm in the Diff Geometry function is that of matching Geometrys. Two algorithms have been developed: **Exact Match** and **Match Within Distance Tolerance**.

7.2.1 Exact Match

Exact Matching tests whether the type and the contents of the Geometrys are equal. Testing type equality is straightforward. To test equality of the contents, Exact Match compares the components and pointlists of the Geometrys in sequence. In order to eliminate trivial mismatches caused by the ordering of the points within rings and of components within collections, the components and pointlists can be normalized before comparison. Normalizing sorts components and rings according to an order function based on the common "dictionary" ordering for coordinates.

7.2.2 Match Within Distance Tolerance

The fundamental idea behind matching with a distance tolerance is to be able to match Geometrys in spite of small differences between them.   "Small difference" is an imprecise notion which is difficult to formalize. In practice we have found that a useful definition involves using the Hausdorff distance.   We say that two Geometrys match if the Hausdorff distance between both the Geometrys and their boundaries is less than the distance tolerance. More formally,

> Geometry A matches Geometry B iff
> > $H(A, B) <= d$
> > and $H(boundary(A), boundary(B) <= d$

We do not need to compute the Hausdorff distance explicitly. Instead, we need to solve the decision problem: is $H(A, B) < d$? This can be solved in a straightforward way by using buffering to test for an upper bound on the Hausdorff distance. We have the following relationship:

> $H(A, B) <= d$ if
> > buffer(A, d) contains B
> > and buffer(B, d) contains A

This gives the following algorithm:

> Geometry A matches Geometry B if
>
> > A is contained in buffer(B, d)
> > **and** B is contained in buffer(A, d)
> > **and** boundary(A) is contained in buffer(boundary(B), d)
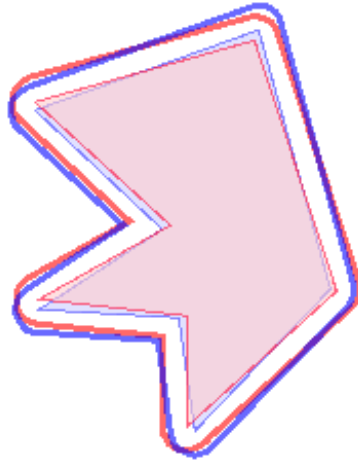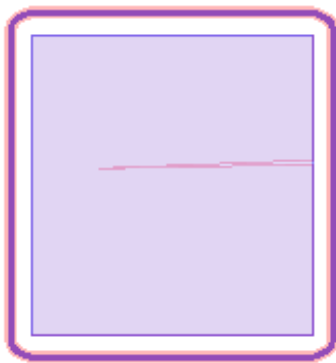> > **and** boundary(B) is contained in buffer(boundary(A), d)

Figure 14 - An example of using buffers to match within a distance tolerance

The two examples in Figure 15 illustrate that both the geometry buffer test and the boundary buffer test are necessary; neither one alone is sufficient.

In this case the geometry buffers are almost identical, but the linework is different. The boundary buffer containment test will detect this difference.
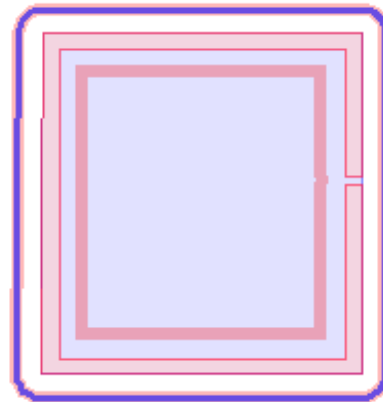
In this case the boundary buffers are almost identical, but the geometrys are different. The geometry buffer containment test will detect this difference.

Figure 15 - Examples showing that both boundary buffers and geometry buffers must be tested to properly detect geometry differences

7.3 FURTHER WORK

In practice, the above algorithm works well in finding differences between geometry datasets. However, there are pathological examples which are reported as matches by the algorithm but which would probably not be considered as matching by a human. An example is shown below. It would be interesting to experiment with some other matching algorithms to reduce the likelihood of invalid matches. As a precursor to this, it may be

necessary to categorize the kinds of differences which are considered as matches or mismatches in particular application areas.
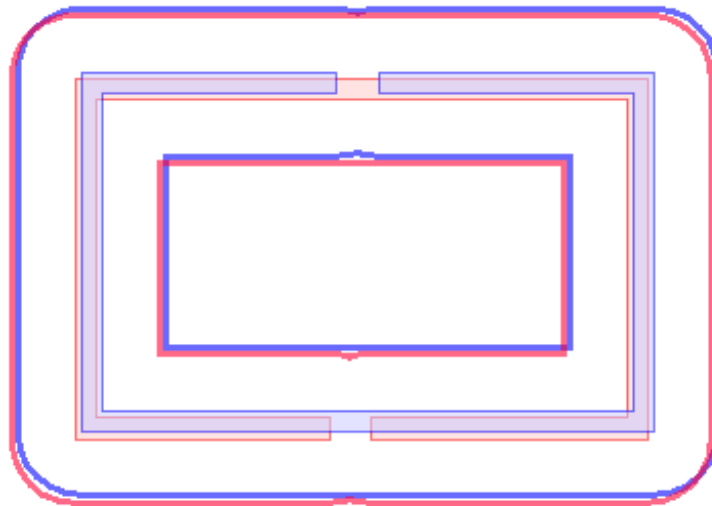

Figure 16 - A example of different Geometrys reported as matching by the buffer-based algorithm

8. COVERAGE CLEANING

JCS provides tools to do the following tasks involved in coverage cleaning:

- QA coverages to detect errors (gaps and overlaps)
- Automatically fix most or all errors in a coverage
- Allow manual fixing of errors not fixed automatically
-

8.1 GAP DETECTION

An automated tool has been developed to find gaps in coverages. Gap detection is based on detecting misaligned line segments within a certain distance tolerance.

8.1.1 Output Data

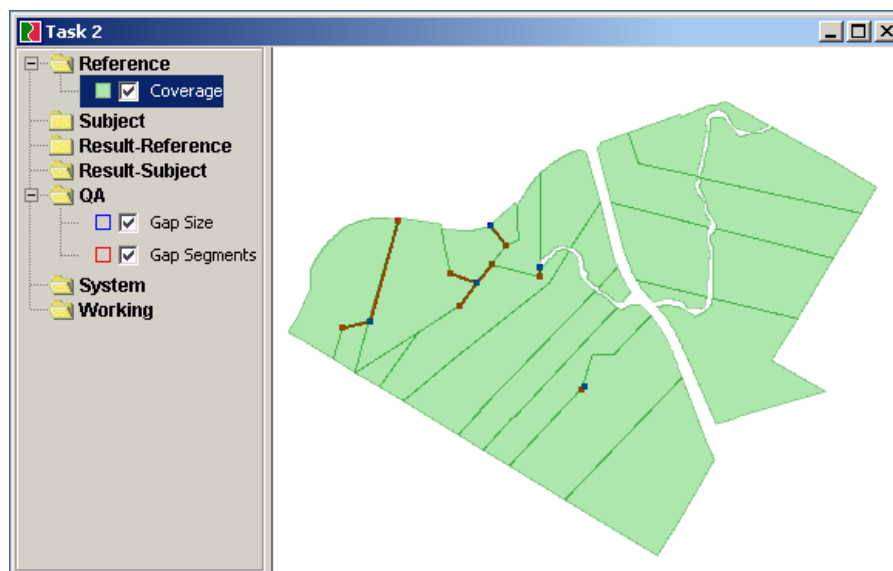| Layer	Description
Gap Size	Shows the segments on either side of detected gaps
Gap Segments	Contains indicators showing the size of gaps

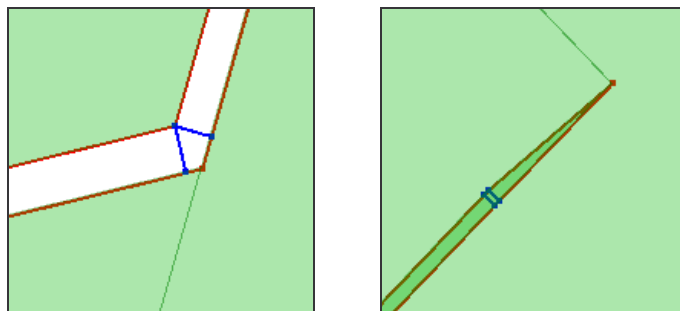**Figure 17 – QA layers created by the Find Coverage Gap function**

Figure 18 - Closeups of gaps showing Gap Segment indicators (red) and Gap Size indicators (blue)

8.2 OVERLAP DETECTION

An automated tool has been developed to find overlaps in coverages. Overlap detection is based on the JTS capability to compute exactly whether the interiors of two polygons overlap

8.2.1 Output Data

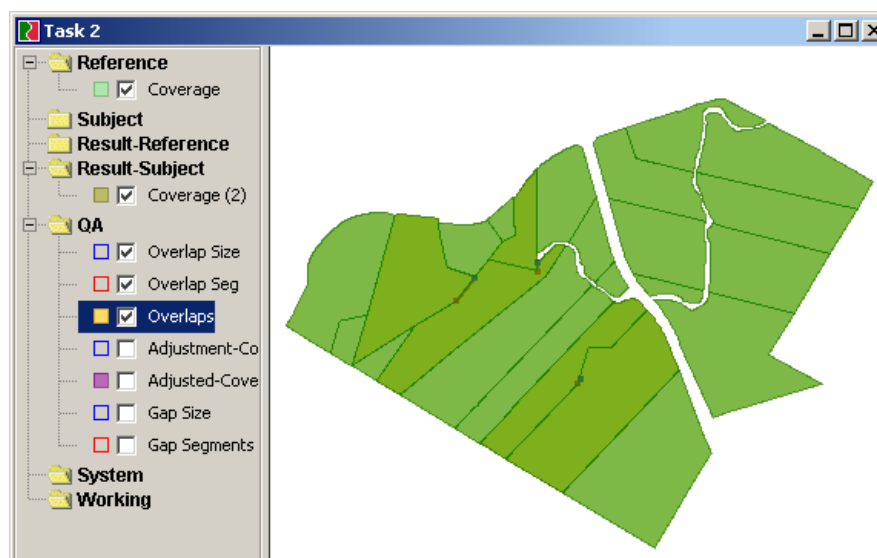| Layer	Description
Overlap Size	Contains indicators showing the size of overlaps
Overlap Seg	Shows the segments forming overlaps
Overlaps	Contains copies of the features which overlap

Figure 19 - Layers created by the Find Coverage Overlaps function

8.3 COVERAGE GAP AND OVERLAP REMOVAL

8.3.1 Automated Gap Correction

JCS contains an algorithm to correct many gaps automatically. Since it is not always well-defined how a gap should be corrected without negatively impacting the topology of the surrounding geometry, the algorithm contains heuristics that help to prevent causing further topology errors. This means that not all gaps can be automatically fixed. However, tests run on many real-world datasets show that usually 95% or better of gaps can be fixed automatically.

8.3.2 Manual Gap Correction

To correct gaps that cannot be fixed automatically JCS provides manual editing tools. The tools are **Snap Vertices** (⊞) and **Delete Vertex** (✖).
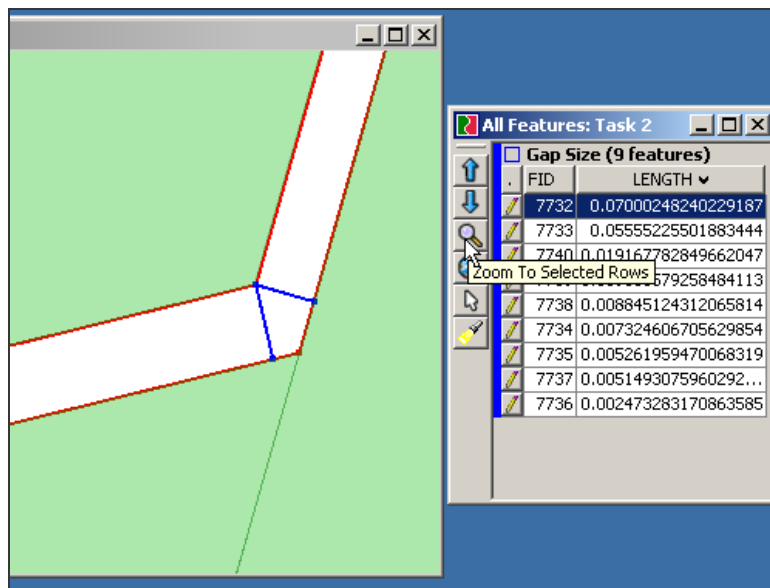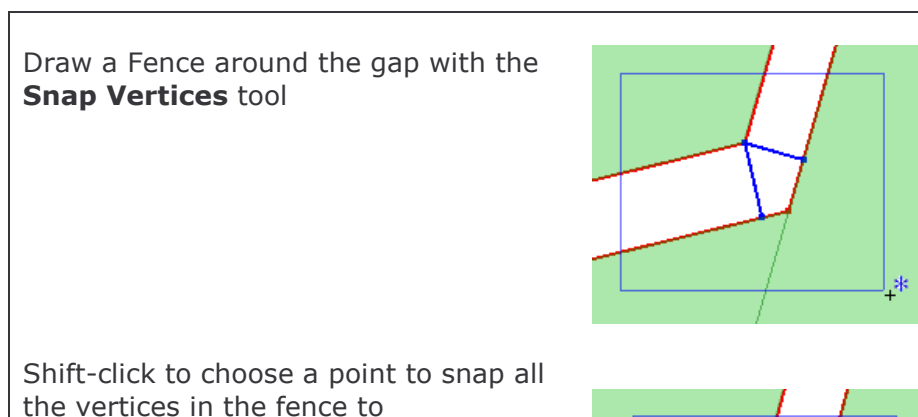

Figure 20 - Zooming to a gap using the Attribute View Zoom tool

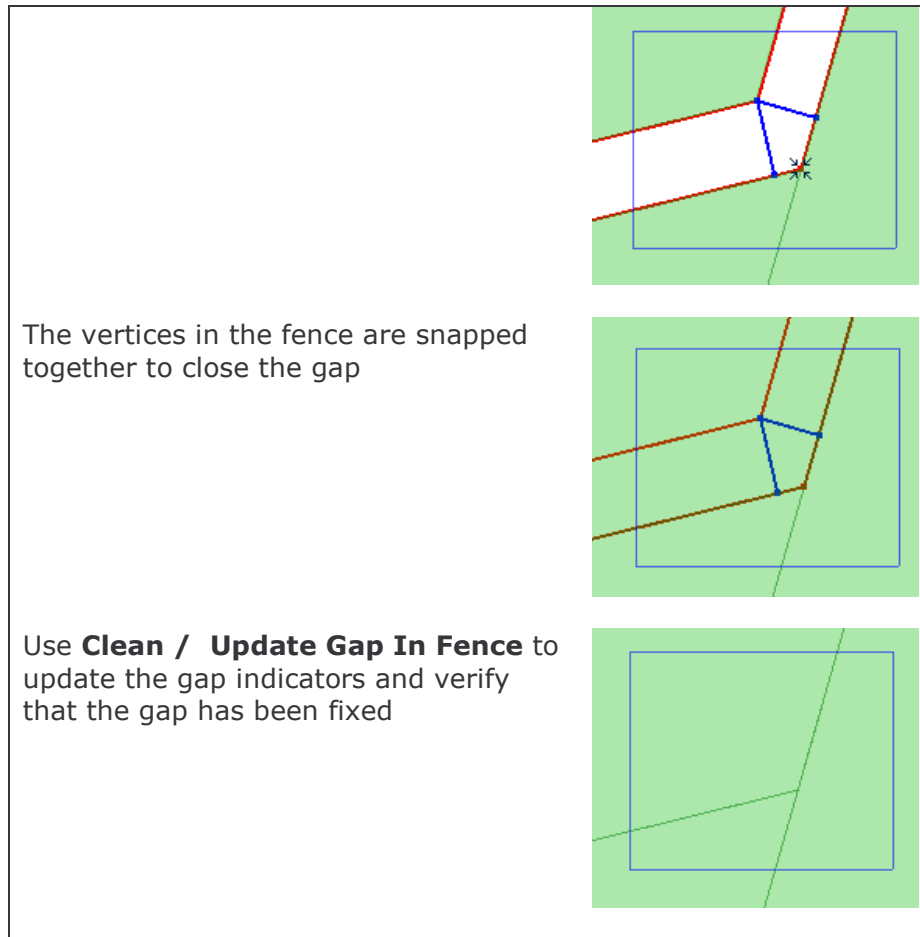The following images illustrate how a gap is manually fixed using the **Snap Vertices** tool.

Draw a Fence around the gap with the **Snap Vertices** tool

Shift-click to choose a point to snap all the vertices in the fence to

The vertices in the fence are snapped together to close the gap

Use **Clean / Update Gap In Fence** to update the gap indicators and verify that the gap has been fixed

Figure 21 - Fixing a gap manually

8.3.3 Manual Overlap Correction

Most small overlaps should have been detected and fixed using the Gap correction tools. Overlaps that remain may be due to two things:

- The **Distance Tolerance** used in gap cleaning may have been too small to correct the gap due to the overlap. If possible, increase the Distance Tolerance value to allow the gap to be corrected. However, do not increase the Distance Tolerance to the point at which valid spaces begin to be affected.

- Overlaps may be due to duplicate features in the datasets.   In this case, it may be possible to simply remove one of the features. (The best way to do this is to use the **Feature Info** tool to view all features at a specified point. One of the features can be selected in the **Feature Info** view and deleted using **Cut Selected Features**)

- Overlaps may be present in the input dataset for a business reason. In this case it is necessary to manually decide how to deal with the overlap.

9. BOUNDARY ALIGNMENT

9.1 PROBLEM DESCRIPTION

Along common edges polygons may not meet exactly. There may be gaps or overlaps between adjacent polygons. Vertices may be present on one edge that are not present on its neighbour.

The following example illustrates gaps, overlaps, missing vertices, and large gaps
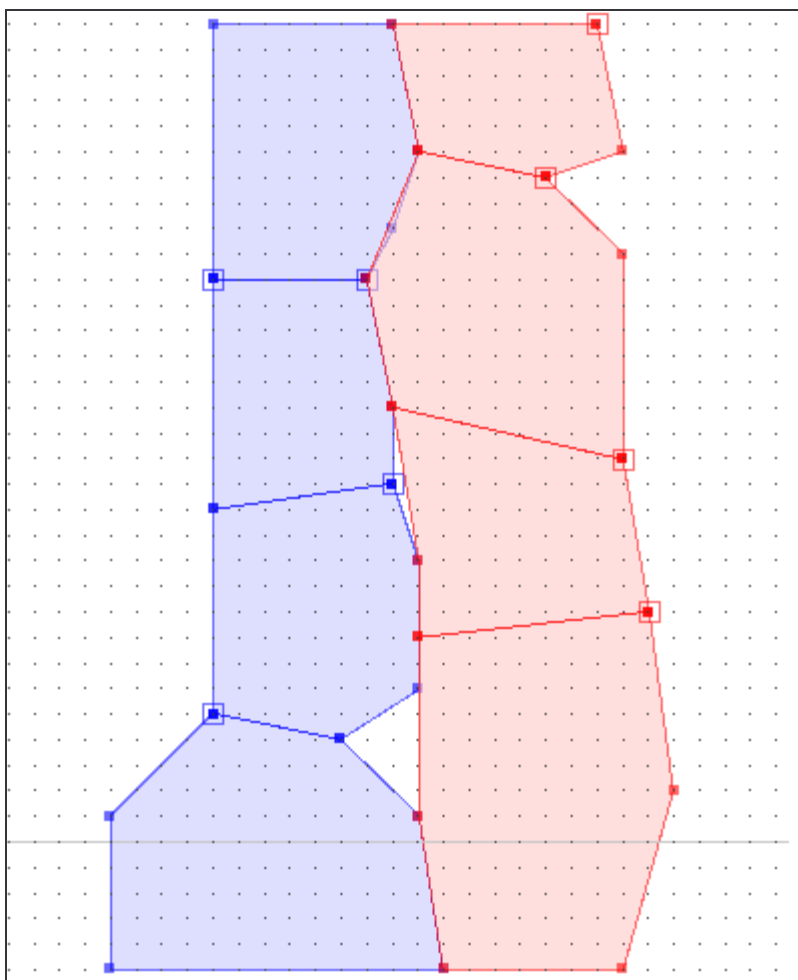

Figure 22 - Example of datasets requiring edge-matching

9.2 DATASET DESCRIPTIONS

9.2.1 Input Data

A set of two (or more?) coverages (collections of non-overlapping polygons). Each input coverage is assumed to be clean. The coverages are intended to form a single larger coverage, with no gaps or overlaps at their common edges, and correctly noded along their common edges.

Other terms: Gap Removal, Sliver Removal

9.2.2 Output Data

Updated versions of the input coverages, edge-matched along their common edges.

9.3 ASSUMPTIONS & RESTRICTIONS ON INPUT DATA

1. The process assumes that the discrepancies that exist between the datasets are small relative to the minimum segment size in the datasets, and to the minimum gap size between the two datasets. (If this is not the case, it becomes difficult to automatically decide whether differences are legal or are due to conflation problems).

9.4 CONDITIONS ON OUTPUT DATA

1. No existing vertices in either input dataset will be deleted.
2. No existing vertices other than boundary vertices will be moved in either dataset.
3. No existing vertices in the Fixed dataset will be moved
4. New vertices may be created along the common boundary in either dataset.

9.5 PARAMETERS

Parameter	Description
Tolerance	A Subject segment or vertex, which is within this distance of a Reference segment or vertex, will be adjusted to align to the Reference.

9.6 ADJUSTMENTS PERFORMED

Match	Adjustment
A Subject vertex and Reference vertex are within the Tolerance	The Subject vertex is adjusted to the Reference vertex location.
A Subject segment has a Reference vertex within the Tolerance	A new vertex is created on the Subject segment at the location of the Reference vertex.
A Reference segment has a Subject vertex within the Tolerance	A new vertex is created in the Reference segment at the closest point to the Subject vertex (the projection of the Subject vertex on the reference segment). The Subject vertex is adjusted to the location of the new vertex.

Note: matches are processed in order of priority. If two or more matches occur, the higher priority one determines the adjustment carried out.

9.7 ISSUES

1. If a Subject boundary edge reverses direction at a vertex and both vertices are close to a Reference segment, the vertices will both be adjusted to the same segment. This will cause adjusted segments to self-intersect and the adjusted polygon to have invalid topology. This can be defined as a requirement for the boundary to be "smooth" relative to the tolerance value. This issue may be able to be solved by constraining matched segments to be in opposite directions (where opposite is defined as having slopes that are at least 135 degrees apart).
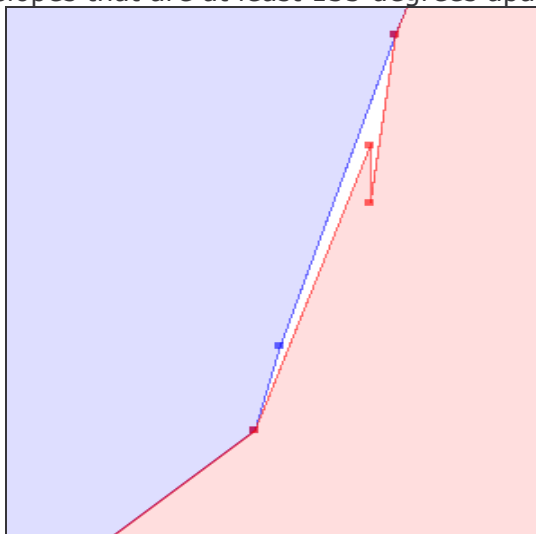

Figure 23 - A case where adjusting vertices would cause self-intersection

2. If a boundary polygon completely overlaps a neighbour polygon, the segments of the boundary polygon may incorrectly match segments of the neighbour polygon that are not on its boundary. *[diagram?]*

9.8 ALGORITHM

### 9.8.1 High-Level Description

3. Identify boundary edgelists that are close together.
4. In the Subject dataset, adjust vertices to the closest Reference vertex within the Tolerance.
5. In the Subject dataset, insert vertices along line segments where there is a Base vertex closer than the Tolerance to the segment. The inserted vertices will be inserted at the location of the Reference vertex.
6. In the Subject dataset, existing vertices that are within the Tolerance of a reference line segment will be moved to lie as close as possible to the Reference line segment.
7. In the Reference dataset, insert vertices along line segments where source vertices are within the Tolerance of the segment.

10.   ROAD NETWORK CONFLATION

The Road Network Conflation algorithms and tools are currently being developed. Issues being tackled include:
- How to identify matching nodes and edges
- How best to visualize matches and mismatches
- What manual tools are required to allow specifying matches and mismatches

The screenshot below illustrates the current state of development for matching and visualization.
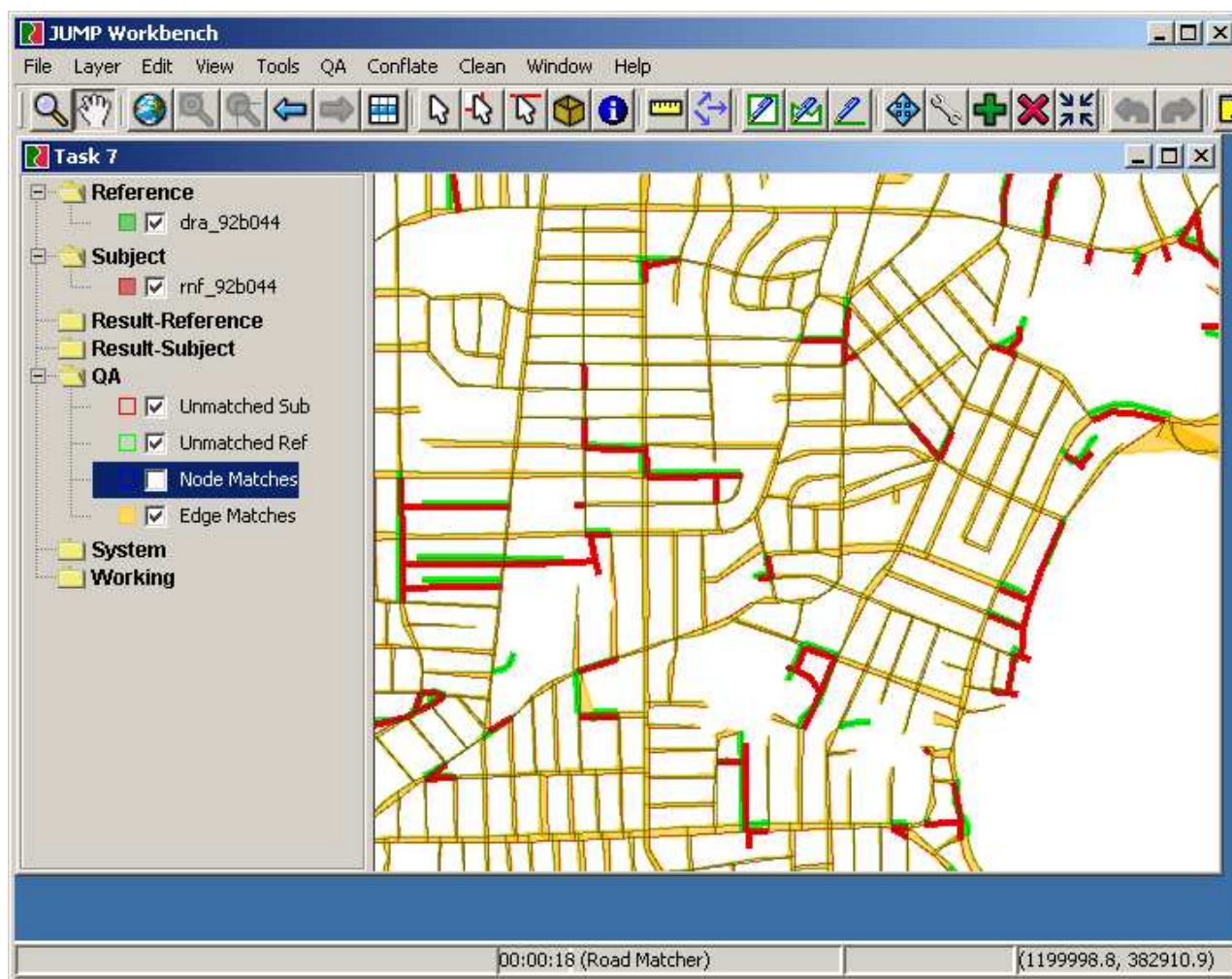
10.1 PROBLEM DESCRIPTION

10.2 DATASET DESCRIPTIONS

10.2.1 Input Data

10.2.2 Output Data

10.3 ASSUMPTIONS & RESTRICTIONS ON INPUT DATA

10.4 CONDITIONS ON OUTPUT DATA

10.5 PARAMETERS

Parameter	Description

10.6 ADJUSTMENTS PERFORMED

10.7 ISSUES

10.8 ALGORITHM

11.   REFERENCES

[Yua99]   Shuxin Yuan, Chuang Tao. "Development of Conflation Components". 1999.
               http://www.umich.edu/~iinet/chinadata/geoim99/Proceedings/yuan_shuxin.p
               df