PROCEDIMIENTOS PROPUESTOS PARA INTEGRACIÓN DE CARTOGRAFÍAS

1. Armonización Geométrica

1.1.- Introducción y pertinencia

La integración de cartografías procedentes de diversas fuentes en una sola cobertura ha sido un requisito fundamental dentro del enfoque metodológico del proyecto.

La integración de la que hablamos no se basó en la simple superposición de capas, sino que debe tener en cuenta que las distintas capas proceden de diferentes interpretaciones del territorio, y en la mayoría de las ocasiones las líneas que el fotointérprete dibuja tienen diferentes interpretaciones geométricas, a pesar de simbolizar el mismo objeto físico.

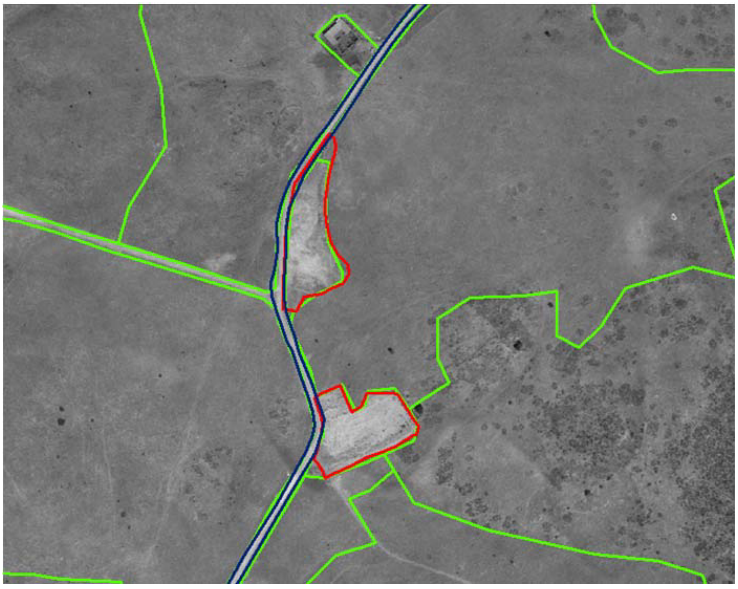


Figura 01.- El mismo objeto físico se fotointerpreta de diferente manera en las distintas bases de referencia

El proceso de **armonización geométrica** se basa en la siguiente premisa: dos líneas muy cercanas y geométricamente similares de distintas cartografías hacen referencia al mismo límite geográfico (fotointerpretado por distintas personas). La definición de lo “muy cercano” es variable dependiendo del nivel de detalle de la cartografía a armonizar y se concreta en una distancia lineal determinada; por otro lado “geométricamente similares” comprende distintas definiciones, siendo la más visual de todas, la evaluación del ángulo de incidencia de una línea sobre otra. El resultado del procedimiento es que en los casos en que se cumplen las condiciones impuestas, la línea a armonizar colapsa sobre la línea de referencia.

A través de operaciones espaciales que se efectúan sobre la capa a armonizar, se modifica sutilmente esta para que las líneas límite se acoplen a las propias de la capa hacia la que se armoniza. Finalmente, se aplican los valores alfanuméricos a los polígonos resultantes, eliminando los micropolígonos que puedan aparecer.

Figura 02.- Tras la armonización geométrica, las líneas límite coinciden.

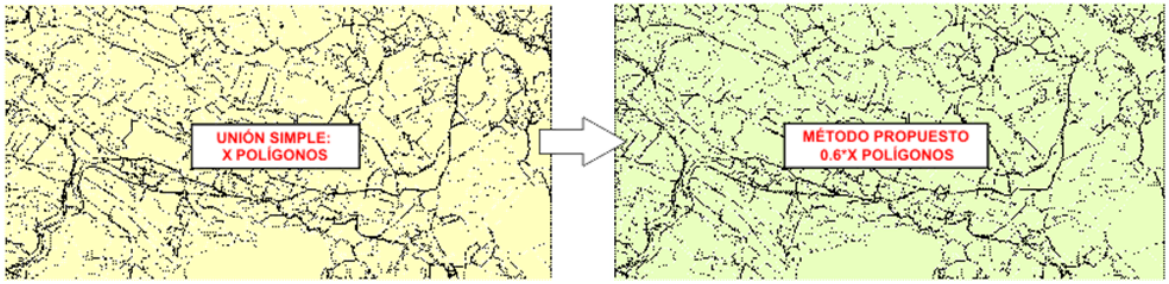
Como se ve en la imagen, y a diferencia de la imagen anterior, los polígonos producto del procedimiento se ajustan temáticamente, permitiendo un mejor entendimiento de la cartografía y una simplificación sustancial de la misma.

1.2..- La armonización geométrica vs la unión simple de polígonos

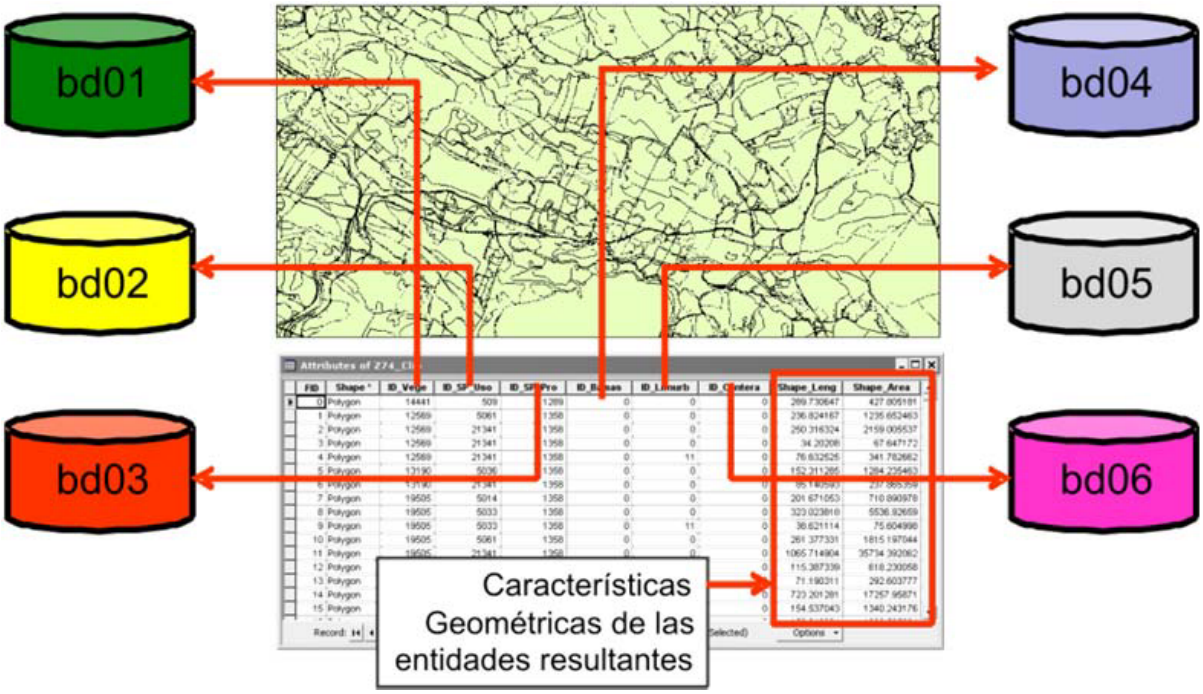
Para la mayoría de operaciones de gestión, o para la simple comparativa entre bases cartográficas, la operación de unión entre cartografías resulta necesaria, si bien normalmente se adecua la metodología al estudio de valores estadísticos. A través del procedimiento propuesto de armonización espacial, se consiguen los siguientes objetivos:

a.- **Eliminar definiciones de línea duplicadas** a favor de la perteneciente a la cartografía de mayor jerarquía, apoyando una mayor claridad en la interpretación de las bases cartográficas.

b.- **Evitar la superpoblación de micropolígonos**, y con ello las dificultades en la gestión de capas excesivamente pesadas, y permitiendo también una interpretación mas sencilla de la cartografía.



c.- **Realizar la operación espacial de unión de cartografías una sola vez**, para todo el territorio y entre todas las cartografías útiles para el objetivo final, con el consiguiente ahorro de recursos y facilidad para establecer consultas de explotación o coherencia entre las bases cartográficas que participan, lo que incidirá de forma evidente en la calidad de las mismas.



d.- **Al coste de una mínima modificación de las bases cartográficas** (por debajo del 0.1% en superficie), que además puede se modulada de forma específica según características temáticas de los polígonos.

Para ello en primer lugar, cuando a la información se le presupone interés cartográfico además de temático, se le debe realizar un proceso de armonización geométrica jerarquizada (si bien en el presente momento no está totalmente definida la jerarquía, salvo el hecho de que la capa que mejor defina la estructura de la propiedad debe quedar completamente inalterada). Para ello a cada una de estas bases de referencia se le

asigna, en función de sus características técnicas (escala, nivel de detalle, referencia temporal,…) un número de orden que indica a cuáles de las otras cartografías han de ajustarse espacialmente y cuáles han de ajustarse a ella, además de su tolerancia lineal y superficial en función de las cuales se realizará la armonización.

Estos procesos se realizarán sobre el total de la superficie regional organizada por bloques de trabajo, que constituyen unidades territoriales homogéneas.

La metodología para la realización de este proceso está basada en operaciones geométricas que modifican la definición de los polígonos de la base de referencia sobre la que se armoniza para que, dadas unas tolerancias métrica y superficial, se acoplen a la geometría de los polígonos de la capa que se integra. Todas las líneas que queden a menos distancia que la determinada por el parámetro de tolerancia métrica de alguna línea de la capa que se integra, se modifican para coincidir con las geometrías de la capa hacia las que se armoniza. Entidades sólo diferenciables por atributos de la capa a armonizar con superficie menor de la determinada en el parámetro de tolerancia superficial (que puede ser dependiente del contenido), se fusionan con las entidades anexas de igual contenido temático.

La capa hacia la que se armoniza no sufre ningún cambio, tan sólo sirve de molde hacia el que ajustar la nueva base de referencia. Éste sufre cambios sutiles controlados a través de los parámetros de tolerancia métrica y superficial.

Como quedará documentado más adelante en este apartado, RqueR tecnología y Sistemas cuenta con un avanzado procedimiento de armonización geométrica de desarrollo propio, que permite realizar esta compleja operación de forma prácticamente desatendida, ágil y segura.

La armonización geométrica de los polígonos se aplicará al conjunto del territorio entre las capas de información a integrar y la que deberá servir de base. También se incide en los procesos de armonización temática de los polígonos, de manera que la metodología debe utilizar herramientas que incorporen la información de las capas integradas dentro del modelo de datos implementado, con la interpretación y adaptación necesaria para cada uno de los posibles casos.

Los procedimientos garantizan la calidad de la información generada, incorporando toda información bajo los estándares de calidad definidos para la Red de Información Ambiental de Andalucía, tanto en lo que se refiera a la estructura de la información, como a los contenidos de la misma.

El conocimiento adquirido a través del tiempo sobre los distintos modelos de datos de SIOSE con el que cuenta nuestra empresa, nos ha permitido ofrecer con garantías métodos ágiles basados en programas ampliamente documentados para esta labor, que aseguran la comparabilidad de las bases de referencia entre ellas, y con el territorio que subyace.

De forma más concreta, la metodología que se llevado a cabo se ha basado en el estudio previo de las condiciones del proyecto SIOSE, en el análisis de las metodologías que habían sido implementadas para la versión de 2005 y en el estudio de la cartografía base a integrar para esta ocasión, para después desarrollar los procedimientos metodológicos necesarios para ejecutar los procedimientos de armonización mejorados sobre el conjunto de bases cartográficas propuestas a la Dirección Técnica, con el fin de obtener una base de referencia completa.

1.3. Procedimiento mejorado propuesto para la Armonización Geométrica

Las propuestas de mejora para el procedimiento de integración a aplicar están basadas en la eliminación o reducción de las debilidades detectadas en el procedimiento de integración seguido en 2005:

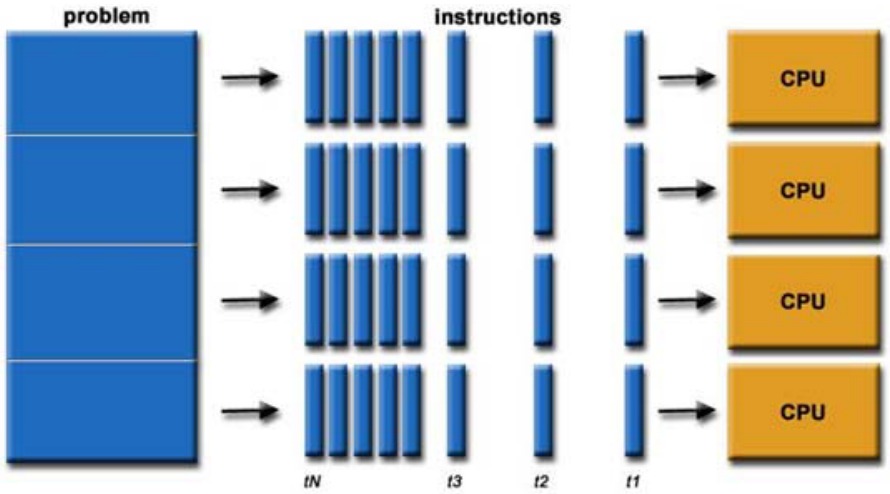
A) Reducir la dependencia de software propietario: Si bien dados los plazos de ejecución del servicio parece imposible eliminar esta dependencia, sí es posible suavizarla mediante dos acciones principalmente:

A.1.- Programación de herramientas que actúen sobre geometría y datos de forma directa. Evitar el uso indiscriminado de operaciones de geoprocesamiento incluidas en ArcGIS a favor de herramientas más especializadas que operen sobre geometría y datos alfanuméricos a nivel de matemática pura.

A.2.- Sustitución de scripts y programas de terceros por propios o documentación exhaustiva de los primeros. Esta acción redunda en la transparencia de las operaciones que se realizan.

B) Programación de todas las operaciones en el mínimo número de paquetes ejecutables posible, eliminando la dependencia del operador y minimizando la posibilidad de error. Esta operación puede incluir, de forma ordenada, tanto operaciones de las antes mencionadas (en las que no se depende de software propietario), como otras difícilmente replicables o en las que este último ofrece un fuerte impacto positivo sobre el rendimiento (en tiempo o calidad). Asimismo, se maximiza el rendimiento de la ejecución sobre los paquetes de información que cubren la geografía Andaluza, pudiendo ser programados en forma de proceso por lotes. En cualquier caso, conviene mantener cierta separación entre aquellas operaciones dependientes de software propietario y las que no lo son, puesto que las primeras podrían ser paulatinamente sustituidas por desarrollos libres con idénticas funcionalidades.

C) Eliminación o reducción drástica de las operaciones de edición manual sobre los resultados. Esta mejora es quizá la más importante, puesto que se evitarían los errores implícitos a los procedimientos de edición manual, y se mejoraría el rendimiento sobremanera.

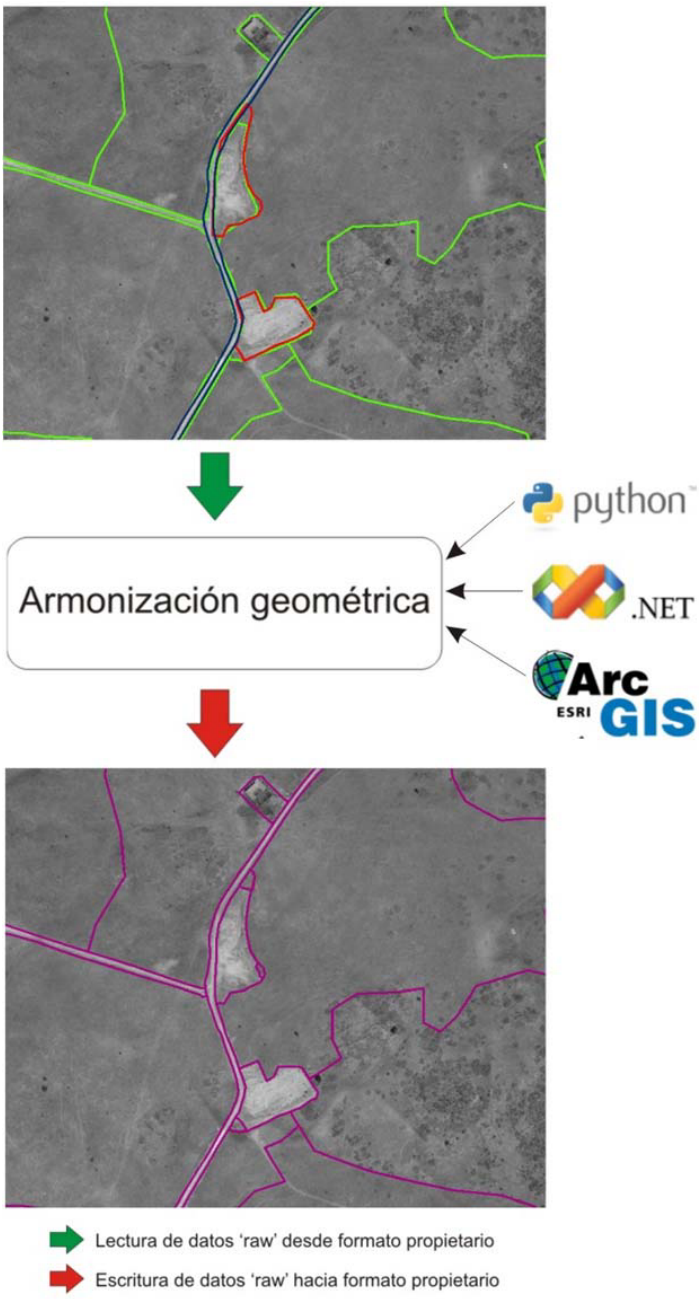


Además, se pretende la introducción de las siguientes mejoras instrumentales:

D) Introducción de procesos multitarea. La reducción de las operaciones de geoprocesamiento basadas en software propietario, que generalmente admiten mal la multitarea, y la programación del mínimo de paquetes ejecutables posible, permitirán la introducción de tecnologías de paralelización de procesos que aumentan el rendimiento de las operaciones automáticas y reducen los tiempos de ejecución hasta un 600%.

E) Documentación ‘al vuelo’ de los procesos seguidos: se implementarán mecanismos de ‘logging’ en los que se guardará un histórico de los procesos ejecutados y los errores recibidos en cada uno de ellos. Esto asegura la trazabilidad de los datos desde su forma de origen hasta su forma final.

F) Generación de metadatos para los productos obtenidos, según la normativa y modelos que la dirección técnica precise.



Estas propuestas de mejora suponen los siguientes avances, de forma sintética:

1.- Mejorar la transparencia y usabilidad del método o de cualquiera de sus partes, dentro del proyecto o de cualquier otro entorno proyectual, al quedar correctamente documentado y parametrizado para su uso.

2.- Mejorar de forma ostensible el rendimiento de los programas implementados, lo que supondría un impacto crítico sobre los tiempos de implementación, que mejoraría la usabilidad del método.

De forma gráfica y muy resumida, el esquema de ejecución propuesto para la armonización se resume en la imagen superior.

La obtención de los parámetros descritos ha de realizarse mediante la aplicación de algoritmos sobre los datos de origen (en particular, sobre el conjunto de cartografías implicadas para su armonización). Estos algoritmos han de estar implementados en un programa informático, para su fácil ejecución a demanda del usuario. Este programa informático ha de cumplir, bajo nuestro entender determinadas condiciones:

1.- El programa en el que se implementen los algoritmos ha de estar realizado íntegramente en software libre, y si bien el apoyo en herramientas propietarias está permitido, éste estará justificado y de alguna manera 'aislado', para poder ser sustituido por herramientas libres en sucesivas actualizaciones. Esta característica favorece la independencia, transparencia y sentido de la economía de medios de los resultados obtenidos, puesto que, si bien para la aplicación de los algoritmos a un conjunto de datos dado será al principio necesario contar con alguna licencia de uso para otros programas, posteriormente se podrá evitar.

2.- El programa en el que se implementen los algoritmos ha de realizar los cálculos necesarios de forma ágil, a través de la optimización tanto del código como del aprovechamiento de los recursos de la máquina sobre la que sean ejecutados, reforzándose aún más el sentido de la economía de medios. Se espera que, por la naturaleza de los objetivos, la aplicación de los algoritmos sobre un conjunto de datos útil suponga un elevado consumo de recursos informáticos. Tanto la depuración del código como el aprovechamiento máximo de la capacidad de cálculo de la máquina (o conjunto de máquinas) sobre la que corran los procesos pueden suponer un ahorro muy importante en tiempo de ejecución del programa.

3.- El programa en el que se implementen los algoritmos ha de estar suficientemente documentado, para la correcta comprensión, por parte de personas no expertas en programación, de los procesos que se ejecutan. Si bien el presente informe sirve como documentación precisa del programa implementado, dentro del propio código podemos encontrar comentarios, precedidos por uno o varios símbolos "#", que ayudan a la correcta comprensión del mismo.

4.- Deberá minimizarse la acción de operadores en 'revisiones manuales'. En caso de ser insoslayables, las operaciones de revisión deberán estar agrupadas de la forma más compacta posible, permitiéndose una programación oportuna que redundará en una fuerte reducción de las oportunidades de error y en el aumento de la calidad final del producto.

1.3.1. Lenguaje de programación

El programa en el que los algoritmos están implementados está realizado íntegramente en el lenguaje de programación Python, concretamente en su versión 2.4.

Python es un lenguaje de programación interpretado creado por Guido van Rossum en el año 1991. Se compara habitualmente con Tcl, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a GUI (interfaz gráfica con el usuario) como Tk, GTK, Qt entre otros.

Python se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.

El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño. Por tanto, consideramos que es un lenguaje muy apropiado para la implementación de algoritmos que han de ser validados por personal no experto, dada su facilidad de lectura. El uso de lenguajes no interpretados (como C en sus distintas variantes) estaría más indicado para reforzar su rendimiento, una vez validado el código. En cualquier caso, para esta primera versión de mejora del método de armonización geométrica de bases de referencia, en la que aún se mantienen ciertas operaciones de geoprocesamiento basadas en software propietario, en concreto ArcGIS, se ha elegido python, además de por las razones antes descritas, por ser el lenguaje de scripting de este paquete GIS, que se puede utilizar para 'empaquetar' o ejecutar por lotes operaciones sencillas de geoproceso.

1.3.2. Funciones y módulos.

De forma general, una función es una relación entre dos variables, de forma que a cada valor de la variable independiente x, le asocia un único valor de la variable dependiente y, que llamaremos imagen de x.

En Python, la estructura básica de una función es la siguiente:

Valor = algunaFunción(Arg01, Arg02, etc...)

Esto significa que la variable Valor obtiene su valor al llamar a la función. Una función puede aceptar varios, uno o ningún argumento. La función utiliza estos argumentos como variables internas e incluso puede llamar a otras funciones.
Un módulo es una colección de funciones (también de clases), de forma que al importar ese módulo, se agregan nuevas funcionalidades. En este sentido, módulos y funciones sirven para formar el 'contenido' del programa, permitiendo las dependencias a variables necesarias, mientras que la 'estructura' irá llamando a estas funciones según vaya siendo necesario.

1.3.3. Módulos estándar utilizados.

En Python con cada paquete de instalación se ofrece una serie de módulos estándar llamados 'librerías'. Estos módulos temáticos pueden ser importados con la sentencia 'import', añadiendo nuevas funcionalidades al programa.

En los desarrollos a los que se refiere este documento, los módulos que se han utilizado son:

Time.- Módulo de control de tiempo transcurrido, así como de intercambio entre distintos formatos de tiempo.

Os.- Módulo que da acceso a diversas funciones del Sistema Operativo, como pueden ser obtención de rutas, control de procesos…

Math.- Módulo que proporciona una gran variedad de funciones matemáticas avanzadas, más allá de las que están incluidas en el intérprete.

Operator.- Exporta un conjunto de funciones implementadas en C que se corresponden con los operadores intrínsecos de Python. Por ejemplo, operator.add(x, y) equivale a la expresión x+y. Los nombres de las funciones son los utilizados como métodos de clase especiales. En especial, la función itemgetter devuelve un objeto relacionado con su operando a través del método __getitem__ del operando (). Si hay varios elementos, devuelve una tupla de valores de búsqueda. En general este función se usa para ordenar listas

de listas según uno de los valores.

Sys.- Este módulo proporciona acceso a variables utilizadas o mantenidas por el intérprete y a funciones que interactúan estrechamente con el intérprete, como puede ser el tráfico de argumentos, o las rutas definidas para la lectura de módulos.

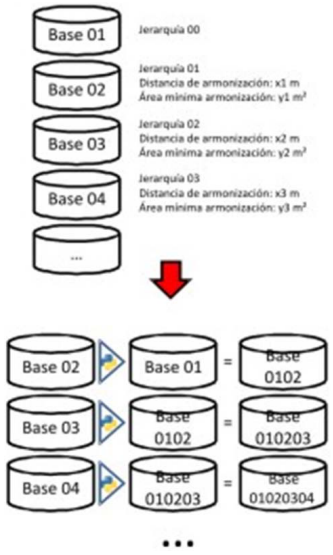
Uuid.- Esta módulo provee de objetos inmutables tipo uuid (Universally unique identifier) según la norma RFC4122, a través de algoritmos generalmente dependientes del tiempo de máquina, y otros parámetros accesorios (dirección MAC de la maquina, etc). Es prácticamente imposible la generación de uuids duplicados. Un uuid tiene la forma:

xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Donde 'x' significa cualquier número del sistema hexadecimal.

Arcgisscripting.- Introducido por ESRI desde la versión ArcGis 9.2, Arcgisscripting es un módulo que, de forma nativa, soporta las operaciones de geoprocesamiento con el motor (generalmente programado en Vb.net) de ArcGIS. En este acercamiento se ha utilizado puesto que entendemos que no era posible prescindir completamente del software propietario, y este proceder fue autorizado por la dirección técnica del proyecto. Esta dependencia se irá disolviendo con la introducción de nuevos programas que, con la forma de módulos python, vayan implementándose en el proceso, sustituyendo a los procedimientos propietarios.

1.3.4. Esquema general del método



El esquema general de armonización de un conjunto de capas se basa en realizar la operación de armonización geométrica de forma iterativa sobre ese conjunto de capas, tomando la capa de mayor invariancia como la primera a la que ajustar y la segunda en el orden de invariancia como la primera que ajustar. Una vez armonizadas ambas, el producto de esta operación puede ser usado como capa a la que ajustar la tercera de las capas en el orden jerárquico.

Esta metodología general no ha sufrido cambios en esencia, pero al mejorarse la operatividad de la operación atómica (armonización de una capa a otra), resulta bastante más sencilla de implementar.

1.3.5.. Armonización de una capa a otra

Para la operación atómica de armonización se ha escogido un esquema que se divide en cinco pasos. Cada uno de estos pasos tiene como entrada una serie de archivos o capas y determinados parámetros y da como salida otros, siendo la entrada al primer paso las dos capas a armonizar y la salida del último la capa armonizada. Cada uno de estos pasos toma la forma de una función python, y todas las funciones que se han generado para el Servicio se han agrupado en un módulo llamado magbar.py.

La división en pasos responde efectivamente a tres razones: la primera de estas razones es la exigencia de mantener aisladas las operaciones que dependen de software propietario, para poder sustituirlas por herramientas de software libre cuando sea posible. Por otro lado, el procedimiento se segmenta debido a la necesidad de ejecutar una pequeña sesión de revisión y edición manual a la finalización de los pasos 04 y 05. La última de las razones para dividir el proceso en pasos es la diferencia de precisión entre unos procedimientos y otros, que promueve la separación de ambos.

Aun así, podemos ejecutar el conjunto de estos cinco pasos de forma automática, a través del pequeño script siguiente:

```
##importamos los modulos necesarios
import time, uuid, os, magbar
##ruta hacia la capa que ha de ajustarse
fc01='C:\\_dehesas\\integracion.gdb\\integ\\fc19cotos02_SP'
##ruta hacia la capa a la que hay que ajustar
fc02='C:\\_dehesas\\integracion.gdb\\integ\\i01_13_1415161718_1'
##float de distancia de ajuste
dmax=20.0
##float de angulo de ajuste
angulomax=30.0
##float de area de ajuste
arealimite=500.0
##preparacion de ruta de directorio de trabajo
uu=str(uuid.uuid4())
dirtrab='C:\\_dehesas\\int18_19'
tempdir=dirtrab+'\\'+uu
os.makedirs (tempdir)
##preparacion de rutas de directorios temporales del sistema
Temp_Dir = tempdir+'\\'+"temp01"
Temp_Dir2 = tempdir+'\\'+"temp02"
os.makedirs (Temp_Dir)
os.makedirs (Temp_Dir2)
##comienzo del paso 01
siose2.paso01(fc01, fc02, dirtrab, tempdir, dmax)
##comienzo del paso02
siose2.paso02(dirtrab, dmax, angulomax)
##comienzo del paso03
siose2.paso03(dirtrab, tempdir, dmax, angulomax)
##comienzo del paso04
siose2.paso04(dirtrab, tempdir, dmax, angulomax, arealimite)
pausa=raw_input('Pulsa intro cuando la sesion de edicion 01 este terminada.')
##comienzo del paso05
siose2.paso05(tempdir, dirtrab, arealimite)
print 'Proceso terminado. Efectuar la sesion de edicion 02'
```

En este pequeño script, en primer lugar se cargan los módulos python que serán necesarios; posteriormente se declaran las variables necesarias:

fc01: Esta es la feature class que debe modificarse ajustándose de forma métrica a la de mayor jerarquía (fc02)

fc02: feature class a la que se ajusta la fc01.

dmax: distancia métrica de ajuste.

angulomax: ángulo máximo que deben formar las líneas de fc01 respecto a las de fc02 para que se produzca el ajuste.

arealimite: área mínima del polígono diferenciado por su código proveniente de fc01. Cualquier polígono menor que esta área que no esté definido en fc02 será absorbido.

Posteriormente, se preparan los directorios de trabajo en los que se generarán los archivos intermedios, que en algunos casos serán necesarios para la siguiente función, mientras que en otros casos sólo se mantienen para asegurar la trazabilidad de las capas producidas.

Por último, se ejecutan de forma sucesiva cada una de las funciones que producirán finalmente la capa armonizada. Nótese que entre los pasos 4 y 5 se efectúa una pausa para realizar la primera de las sesiones de edición, y que al final del script se advierte de la necesidad de efectuar la segunda.

1.3.6. Código del módulo magbar.py.

Como ya se dijo más arriba, magbar.py es el módulo que contiene todas las funciones necesarias para la ejecución correcta de una armonización geométrica atómica, esto es, del ajuste de una capa a otra de mayor jerarquía dada una tolerancia métrica de ajuste, un ángulo de similitud y una tolerancia superficial para el producto (polígono mínimo).

El procedimiento es similar al ya usado en el proyecto SIOSE1: se basa en la premisa de que dos líneas muy cercanas y geométricamente similares pertenecientes a dos bases de referencia distintas responden en realidad al mismo límite geográfico, sólo que dibujado por distintos fotointérpretes. Lo que se persigue con esta metodología es realizar un análisis de estos límites para la capa que puede modificarse, modificándola efectivamente en aquellas situaciones en las que se detecta el conflicto, tomando la forma para esa situación de la capa a la que se ajusta (la de jerarquía superior). Con este procedimiento se eliminan los micropolígonos que resultarían de una unión simple, manteniendo la integridad topológica y la funcionalidad de un procedimiento de unión, con lo que la capa resultante puede ser utilizada para ejecutar sobre ella consultas cruzadas de integridad temática.

Importación de librerías estándar necesarias:

Al cargar el módulo se importan las librerías estándar python que resultan necesarias para la ejecución de todos los pasos. En "Módulos estándar utilizados" se describe sucintamente cada una de ellas.

```
import os, arcgisscripting, time, math, sys
from operator import itemgetter
ini=time.time()
```

También se dainicio a un contador de tiempo.

Función paso01:

En esta función se ejecutan las operaciones de geoproceso necesarias para la lectura de datos en formato propietario y su conversión a un formato operable. La primera parte del módulo define las variables de entorno del módulo arcgisscripting.

```
def paso01(fc01, fc02, dirtrab, tempdir, dmax):

    dirfin=dirtrab+'\\paso01'
    ini=time.time()

    Temp_Dir = tempdir+'\\'+"temp01"
    Temp_Dir2 = tempdir+'\\'+"temp02"

    os.environ['TEMP'] = Temp_Dir
    os.environ['TMP'] = Temp_Dir2

    gp=arcgisscripting.create()
    gp.workspace=tempdir
    gp.overwriteoutput=1
```

A continuación se añaden los campos identificadores de cada entidad en las feature classes originales, para poder recuperar el modelo de datos original tras la armonización.

```
##anadimos campos id, y calculamos
gp.addfield(fc01, 'id_01', 'LONG')
gp.addfield(fc02, 'id_02', 'LONG')
rows=gp.updatecursor(fc01)
row=rows.next()
while row:
    row.id_01=row.objectid
    rows.updaterow(row)
    row=rows.next()
rows=gp.updatecursor(fc02)
row=rows.next()
while row:
    row.id_02=row.objectid
    rows.updaterow(row)
    row=rows.next()
del rows, row
```

Seguidamente se realiza una copia temporal singlepart de cada una de las feature classes para su modificación.

```
    gp.multiparttosinglepart(fc01, tempdir+'\\0
1.shp')
    gp.multiparttosinglepart(fc02, tempdir+'\\0
2.shp')
```

En las feature classes de copia, se borran todos los campos que no son los identificadores, puesto que no los necesitaremos.

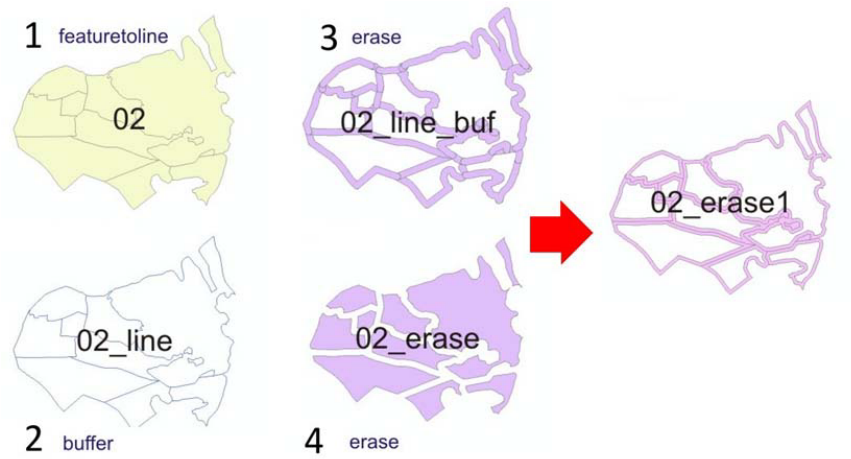
```
##borramos el resto de los campos
fields=gp.listfields(tempdir+'\\01.shp')
field=fields.next()
Lf=[]
while field:
    Lf.append(field.name)
    field=fields.next()
del fields, field
t=''
for f in Lf:
    if f<>'FID' and f<>'id_01' and f<>'Shap
e':
        t=t+f+';'
t=t[:-1]
gp.deletefield(tempdir+'\\01.shp', t)
fields=gp.listfields(tempdir+'\\02.shp')
field=fields.next()
Lf=[]
while field:
    Lf.append(field.name)
    field=fields.next()
del fields, field
t=''
for f in Lf:
    if f<>'FID' and f<>'id_02' and f<>'Shap
e':
        t=t+f+';'
t=t[:-1]
gp.deletefield(tempdir+'\\02.shp', t)
```

Convertimos ambas capas temporales a líneas, sin mantener los atributos (no habrá líneas superpuestas)

```
    ##Coversion de ambas capas a lineas
    gp.featuretoline(tempdir+'\\01.shp', tempdi
r+'\\01_line.shp', '#', 'No_Attributes')
    gp.featuretoline(tempdir+'\\02.shp', tempdi
r+'\\02_l.shp', '#', 'No_Attributes')
    gp.dissolve(tempdir+'\\02_l.shp', tempdir+'
\\02_li.shp')
    gp.multiparttosinglepart(tempdir+'\\02_li.s
hp', tempdir+'\\02_line.shp')
```

En cada capa de líneas añadimos un campo identificador de estas y lo calculamos.

```
    ##Inclusion y calculo id_01_line e id_02_li
ne
    gp.addfield(tempdir+'\\01_line.shp', 'id_01
_line', 'LONG')
    gp.deletefield(tempdir+'\\01_line.shp', 'id
')
    gp.addfield(tempdir+'\\02_line.shp', 'id_02
_line', 'LONG')
    gp.deletefield(tempdir+'\\02_line.shp', 'id
')
    rows=gp.updatecursor(tempdir+'\\01_line.shp
')
    row=rows.next()
    while row:
        row.id_01_line=str(int(row.fid)+1)
        rows.updaterow(row)
        row=rows.next()
    rows=gp.updatecursor(tempdir+'\\02_line.shp
')
    row=rows.next()
    while row:
        row.id_02_line=str(int(row.fid)+1)
        rows.updaterow(row)
        row=rows.next()
```



Ejecutamos un buffer de tamaño dmax sobre la capa de líneas a la que debemos ajustarnos, y realizamos una serie de comprobaciones, puesto que este geoproceso puede dar como resultado errores. Una vez depurados estos errores, el buffer se borra a la capa a la que ajustar, y el resultado se borra de nuevo a la capa a la que ajustar. Con esta operación conseguimos los ‘aros’ de buffer identificados por el identificador de la entidad de la que salen.

```
    ##Ejecucion buffer dmax sobre O2
    gp.buffer(tempdir+'\\O2_line.shp', tempdir+
'\\O2_line_buf.shp', str(dmax))
    ##Comprobacion buffer
    rows=gp.searchcursor(tempdir+'\\O2_line_buf
.shp')
    row=rows.next()
    Lerrores=[]
    while row:
        if row.shape.area==0.0:
            Lerrores.append(row.id_O2_line)
        row=rows.next()
    if len(Lerrores)>0:
        rows=gp.updatecursor(tempdir+'\\O2_line
_buf.shp')
        row=rows.next()
        while row:
            if row.id_O2_line in Lerrores:
                rows.deleterow(row)
            row=rows.next()
        clause=''
        for item in Lerrores:
            clause=clause+'"id_O2_line"='+str(i
tem)+' AND '


         clause=clause[:-4]
         gp.select(tempdir+'\\O2_line.shp', temp
 dir+'\\lineerr.shp', clause)
         gp.buffer(tempdir+'\\lineerr.shp', temp
 dir+'\\lineerr_buf.shp', str(dmax/2))
         gp.buffer(tempdir+'\\lineerr_buf.shp',
 tempdir+'\\lineerr_buf2.shp', str(dmax/2))
         gp.append(tempdir+'\\lineerr_buf2.shp',
 tempdir+'\\O2_line_buf.shp', 'TEST')
    rows=gp.searchcursor(tempdir+'\\O2_line_buf
 .shp')
    row=rows.next()
    Lerrores=[]
    while row:
        if row.shape.area==0.0:
            Lerrores.append(row.id_O2_line)
        row=rows.next()
    if len(Lerrores)>0:corta
    gp.erase(tempdir+'\\O2.shp', tempdir+'\\O2_
 line_buf.shp', tempdir+'\\O2_erase.shp')
    gp.erase(tempdir+'\\O2.shp', tempdir+'\\O2_
 erase.shp', tempdir+'\\O2_erase1.shp')
```

Como no se imponen condiciones de superposición perfecta de ambas capas, es necesario conseguir el aro exterior.

```
    ##aro exterior
    gp.dissolve(tempdir+'\\02.shp', tempdir+'\\
02_diss.shp')
    gp.multiparttosinglepart(tempdir+'\\02_diss
.shp', tempdir+'\\02_diss_SP.shp')
    nfeatdiss=gp.getcount(tempdir+'\\02_diss_SP
.shp')
    gp.featuretoline(tempdir+'\\02_diss_SP.shp'
, tempdir+'\\02_diss_SP_line.shp', '#', 'No_Att
ributes')
    nfeatdissline=gp.getcount(tempdir+'\\02_dis
s_SP_line.shp')
    gp.buffer(tempdir+'\\02_diss_SP_line.shp',
tempdir+'\\02_diss_SP_line_buf.shp', str(dmax))
    gp.erase(tempdir+'\\02_diss_SP_line_buf.shp
', tempdir+'\\02_diss.shp', tempdir+'\\02_diss_
SP_line_buf_erase.shp')
    gp.addfield(tempdir+'\\02_diss_SP_line_buf_
erase.shp', 'id_02', 'LONG')
    gp.append(tempdir+'\\02_diss_SP_line_buf_er
ase.shp', tempdir+'\\02_erase1.shp', 'NO_TEST')
```

Todas las porciones de la capa que ajustamos (en línea – 01_line) que queden dentro de los aros que acabamos de calcular pueden ser susceptibles de modificación, pero las porciones que quedan fuera deben ser guardadas para utilizarlas en la reconstrucción de polígonos armonizados. De entre las que puedan ser modificadas, hay que eliminar las que sean coincidentes con las líneas a las que ajustar.

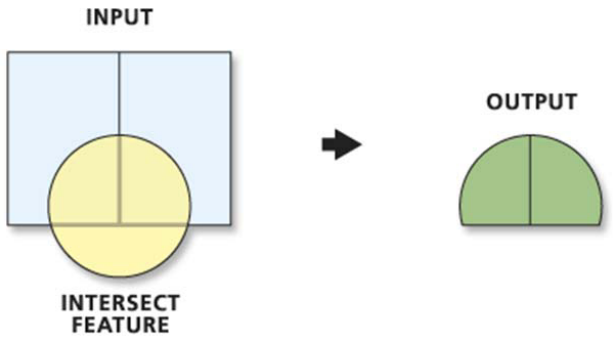
```
    gp.erase(tempdir+'\\01_line.shp', tempdir+'
\\02_erase1.shp', tempdir+'\\01_lineval.shp')
    gp.intersect(tempdir+'\\01_line.shp;'+tempd
ir+'\\02_erase1.shp', tempdir+'\\01_linemod.shp
', 'NO_FID')
    gp.erase(tempdir+'\\01_linemod.shp', tempdi
r+'\\02_line.shp', tempdir+'\\01_linemod_er.shp
')
```

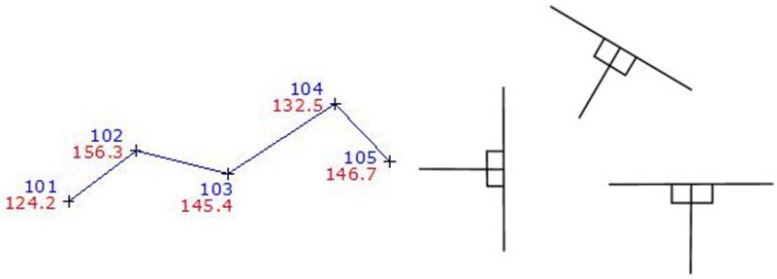
En este punto del procedimiento se necesitan montar grupos según la conectividad de las líneas que han de modificarse, para después ejecutar determinados procesos iterando a través de estos grupos. Los conjuntos se logran convirtiendo las líneas en polígonos a través de un buffer de muy pequeño tamaño, disolviendo esos polígonos y convirtiéndolos en singlepart, identificándolos e intersecándolos con las propias líneas a modificar.

```
    gp.multiparttosinglepart(tempdir+'\\01_line
mod_er.shp', tempdir+'\\01_linemod_SP.shp')
    gp.addfield(tempdir+'\\01_linemod_SP.shp',
'id_vlp', 'LONG')
    rows=gp.updatecursor(tempdir+'\\01_linemod_
SP.shp')
    row=rows.next()
    while row:
        row.id_vlp=str(int(row.fid)+1)
        rows.updaterow(row)
        row=rows.next()
    del rows, row
    gp.addfield(tempdir+'\\01_linemod_SP.shp',
'id_punto', 'LONG')
    gp.dissolve(tempdir+'\\01_linemod_SP.shp',
tempdir+'\\01_linemod_SP_diss.shp')
    gp.buffer(tempdir+'\\01_linemod_SP_diss.shp
', tempdir+'\\01_linemod_SP_diss_buf.shp', '0.0
05')
    gp.multiparttosinglepart(tempdir+'\\01_line
mod_SP_diss_buf.shp', tempdir+'\\01_linemod_SP_
diss_buf_SP.shp')
    gp.addfield(tempdir+'\\01_linemod_SP_diss_b
uf_SP.shp', 'id_grupo', 'LONG')
    rows=gp.updatecursor(tempdir+'\\01_linemod_
SP_diss_buf_SP.shp')
    row=rows.next()
    while row:
        row.id_grupo=str(int(row.fid)+1)
        rows.updaterow(row)
        row=rows.next()
    del rows, row
    gp.deletefield(tempdir+'\\01_linemod_SP_dis
s_buf_SP.shp', 'id')
    gp.deletefield(tempdir+'\\01_linemod_SP_dis
s_buf_SP.shp', 'buff_dist')
    gp.intersect(tempdir+'\\01_linemod_SP.shp;'
+tempdir+'\\01_linemod_SP_diss_buf_SP.shp', tem
pdir+'\\01_linemod_int.shp', 'NO_FID')
```

Para cada línea de entre las implicadas en los grupos de modificación, se obtienen los puntos de inicio y final; se les da un identificador único y posteriormente se catalogan según su situación, a través del valor de un nuevo campo llamado ‘situ’: valor 0 para los puntos que se encuentran en la línea de buffer (entrada a zona de modificación); valor 1 para aquellos puntos que se encuentran sobre una línea perteneciente a la capa a la que ajustar (fc02), y valor 2 para los puntos flotantes de conexión entre líneas de la capa que ajustar (fc01). A la vez se van acumulando en diversas listas los identificadores de puntos en cada situación.

```
    gp.FeatureVerticesToPoints_management (temp
dir+'\\01_linemod_int.shp', tempdir+'\\01_punto
s.shp', 'BOTH_ENDS')
    gp.addfield(tempdir+'\\01_puntos.shp', 'sit
u', 'LONG')
    gp.addfield(tempdir+'\\01_puntos.shp', 'id_
pto', 'LONG')
    rows=gp.updatecursor(tempdir+'\\01_puntos.s
hp')
    row=rows.next()
    while row:
        row.situ='2'
        row.id_pto=str(row.fid+1)
        rows.updaterow(row)
        row=rows.next()
    del rows, row
    gp.intersect(tempdir+'\\01_puntos.shp;'+tem
pdir+'\\02_line.shp', tempdir+'\\01_puntos_t1.s
hp', 'NO_FID')
    L1=[]
    rows=gp.searchcursor(tempdir+'\\01_puntos_t
1.shp')
    row=rows.next()
    while row:
        L1.append(row.id_pto)
        row=rows.next()
    del rows, row
    S1=set(L1)
```





```
    S1=set(L1)
    gp.featuretoline(tempdir+'\\02_erase1.shp',
tempdir+'\\02_erase1_line.shp', '#', 'No_Attrib
utes')
    gp.erase(tempdir+'\\02_erase1_line.shp', te
mpdir+'\\02_line.shp', tempdir+'\\02_erase1_lin
e_erase.shp')
    gp.intersect(tempdir+'\\01_puntos.shp;'+tem
pdir+'\\02_erase1_line_erase.shp', tempdir+'\\0
1_puntos_t0.shp', 'NO_FID')
    L0=[]
    rows=gp.searchcursor(tempdir+'\\01_puntos_t
0.shp')
    row=rows.next()
    while row:
        L0.append(row.id_pto)
        row=rows.next()
    del rows, row
    S0=set(L0)
    rows=gp.updatecursor(tempdir+'\\01_puntos.s
hp')
    row=rows.next()
    while row:
        if row.id_pto in S0:
            row.situ='0'
        if row.id_pto in S1:
            row.situ='1'
        rows.updaterow(row)
        row=rows.next()
    del rows, row
```

En este momento se calculan, en unos campos x e y, las coordenadas de los puntos, se recopila la información necesaria para el siguiente paso y se escribe esta información en un archivo de texto plano que podrá ser leído por python sin necesidad de cargar ningún conector a software propietario.

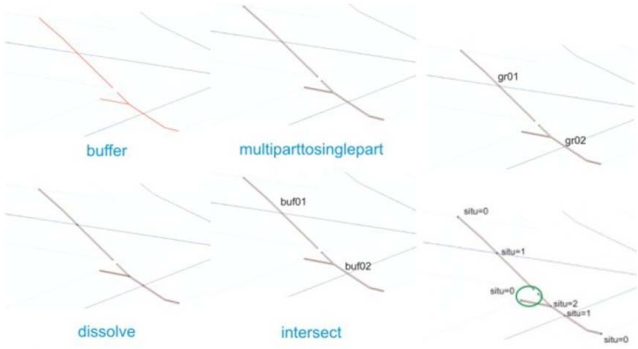
En archivo de texto (puntos.txt) constará de una línea por cada punto, con los datos propios de punto según el siguiente formato:

id_grupo, id_vlp, situ, id_01_line, x, y, id_02

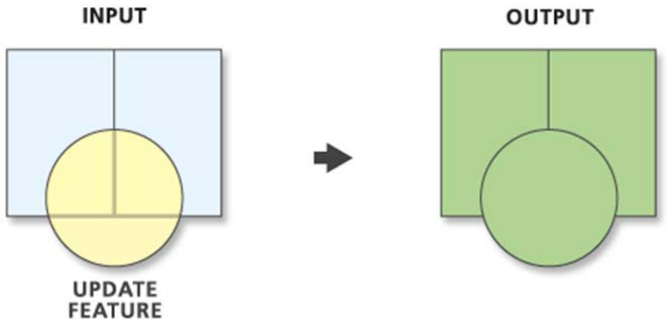
Es decir, el identificador de grupo seguido del identificador de línea (dentro de cada polígono de fc02), seguido de la situación del punto según se explicaba en el párrafo anterior, seguido del identificador de la línea de la que la modificable es segmento, seguido de las coordenadas x e y del punto y por último del polígono de fc02 en el que se encuentra.

```
    gp.deletefield(tempdir+'\\01_puntos.shp', '
id_pto')
    gp.addfield(tempdir+'\\01_puntos.shp', 'x',
'DOUBLE')
    gp.addfield(tempdir+'\\01_puntos.shp', 'y',
'DOUBLE')
    rows=gp.updatecursor(tempdir+'\\01_puntos.s
hp')
    row=rows.next()
    while row:
        row.x=row.shape.getpart().x
        row.y=row.shape.getpart().y
        rows.updaterow(row)
        row=rows.next()
    del rows, row
    rows=gp.searchcursor(tempdir+'\\01_puntos.s
hp')
    row=rows.next()
    w=open(dirfin+'\\puntos.txt', 'w')
    while row:
        w.write(str(row.id_grupo)+','+str(row.i
d_vlp)+','+str(row.situ)+','+str(row.id_01_line
)+','+str(row.x)+','+str(row.y)+','+str(row.id_
02)+'\n')
        row=rows.next()
    w.close()
```

Finalmente, se ejecutan una serie de operaciones de geoproceso destinadas a conseguir una feature class de líneas que permite la resolución de determinadas indefiniciones del método general, que suponían alrededor del 0.1% de error final. Estas situaciones son debidas a la coincidencia espacial de varias incidencias, como agujeros en polígonos, ajuste geométrico necesario en varios lugares cercanos entre sí.




```
    gp.intersect(tempdir+'\\02_line_buf.shp', t
empdir+'\\02_line_buf_int.shp', 'ONLY_FID')
    gp.intersect(tempdir+'\\02_line_buf_int.shp
;'+tempdir+'\\01_line.shp', tempdir+'\\01_linei
nt.shp', 'ONLY_FID')
    gp.dissolve(tempdir+'\\01_lineint.shp', tem
pdir+'\\01_lineint_diss.shp')
    gp.multiparttosinglepart(tempdir+'\\01_line
int_diss.shp', tempdir+'\\01_lineint_diss_SP.sh
p')
    gp.buffer(tempdir+'\\01_lineint_diss_SP.shp
', tempdir+'\\01_lineint_diss_SP_buf.shp', '0.0
1')
    gp.featuretopoint(tempdir+'\\01_lineint_dis
s_SP_buf.shp', tempdir+'\\01_lineint_points.shp
')
    gp.addfield(tempdir+'\\01_lineint_points.sh
p', 'id_comp', 'LONG')
    gp.deletefield(tempdir+'\\01_lineint_points
.shp', 'Id')
    rows=gp.updatecursor(tempdir+'\\01_lineint_
points.shp')
    row=rows.next()
    idc=0
    while row:
        row.id_comp=str(idc+1)
        rows.updaterow(row)
        idc=idc+1
        row=rows.next()
    gp.buffer(tempdir+'\\01_lineint_points.shp'
, tempdir+'\\01_points_buf.shp', str(dmax+0.1))
    gp.intersect(tempdir+'\\01_points_buf.shp;'
+tempdir+'\\02_line.shp', tempdir+'\\01_points_
buf_int.shp', 'NO_FID')
    gp.multiparttosinglepart(tempdir+'\\01_poin
ts_buf_int.shp', tempdir+'\\01_points_buf_int_S
P.shp')
    gp.FeatureVerticesToPoints_management (temp
dir+'\\01_points_buf_int_SP.shp', tempdir+'\\02
_midpoints.shp', 'MID')
    rows=gp.searchcursor(tempdir+'\\02_midpoint
s.shp')
```




```
    Lcomp=[]
    row=rows.next()
    while row:
        Lcomp.append([row.id_comp, row.shape.ge
tpart().x, row.shape.getpart().y])
        row=rows.next()
    if len(Lcomp)>0:
        Lcomp.sort()
        inig=Lcomp[0][0]
        LLcomp=[]
        Lco=[]
        for item in Lcomp:
            if item[0]==inig:
                Lco.append(item)
            else:
                LLcomp.append(Lco)
                Lco=[]
                Lco.append(item)
                inig=item[0]
        LLcomp.append(Lco)
        gp.CreateFeatureClass_management (tempd
ir, 'linecomp.shp', 'POLYLINE')
        rows=gp.insertcursor(tempdir+'\\linecom
p.shp')
        idl=1
```

```
        for grupo in LLcomp:
            Lhechos=[]
            for pto in grupo:
                for pto2 in grupo:
                    if pto2<>pto and [pto2,pto]
not in Lhechos:
                        lineArray = gp.CreateOb
ject("Array")
                        pnt = gp.CreateObject("
Point")
                        pnt.id, pnt.x, pnt.y =
str(idl), str(pto[1]), str(pto[2])
                        lineArray.add(pnt)
                        pnt.id, pnt.x, pnt.y =
str(idl), str(pto2[1]), str(pto2[2])
                        lineArray.add(pnt)
                        feat = rows.NewRow()
                        feat.shape = lineArray
                        rows.InsertRow(feat)
                        Lhechos.append([pto,pto
2])
        del pnt, lineArray, feat, rows
    else:
        gp.CreateFeatureClass_management (tempd
ir, 'linecomp.shp', 'POLYLINE')
    print 'Paso 01:', (time.time()-ini)/60
```

Finalmente, la función informa de que el proceso de preparación de las capas ha terminado de forma satisfactoria y del tiempo (en minutos) que ha sido necesario para ello.

Función paso02:

Esta función no es dependiente del módulo arcgisscripting, por lo que no contiene ninguna operación de geoproceso, al menos desde el punto de vista de la gestión de licencias de software propietario. La posición de los mismos para decidir qué de debe hacer con cada uno de ellos. Dentro de cada grupo de puntos conectado, se analizan todas los posibles caminos que podrían usarse para, partiendo de un punto de situación 0 (en la línea de buffer), lleguemos a otro punto de la misma situación, y según la casuística de las relaciones entre puntos de los mismos segmentos, se preparen líneas a trazar que sustituirán a estos segmentos en la reconstrucción de la capa a armonizar.

```
def paso02(dirtrab, dmax, angulomax):
    timeini=time.time()
    txtent=dirtrab+'\\paso01\\puntos.txt'
    dirsal=dirtrab+'\\paso02'

    r=open(txtent)
    t=r.read()
    r.close()
    L=t.split('\n')[:-1]
    del t
    Lp=[]
    Lg=[]
    for item in L:
        Lt=item.split(',')
        Lg.append(str(float(Lt[4]))+'_'+str(floa
t(Lt[5])))
        Lp.append([int(Lt[0]), int(Lt[1]), int(L
t[2]), int(Lt[3]), float(Lt[4]), float(Lt[5]), i
nt(Lt[6])])
    del L
    Sg=set(Lg)
    Lg=list(Sg)
    del Sg
    Lg.sort()
    Lp2=[]
    for item in Lp:
        g=str(item[4])+'_'+str(item[5])
        Lp2.append([item[0], item[1], item[2], i
tem[3], Lg.index(g), item[6]])
    Lp=Lp2
    del Lp2
    Lp.sort(key=itemgetter(0))
    Lgrupos=[]
    grupo=Lp[0][0]
    Lgr=[]
    for item in Lp:
        if item[0]==grupo:
            Lgr.append(item[1:])
        else:
```

```
            else:
                Lgrupos.append(Lgr)
                grupo=item[0]
                Lgr=[]
                Lgr.append(item[1:])
        Lgrupos.append(Lgr)
        for item in Lgrupos:
            item.sort(key=itemgetter(0))
        Lvpl=[]
        for item in Lgrupos:
            L=[]
            i=0
            while i<len(item):
                L.append([item[i][1:], item[i+1][1:]
])
                i=i+2
            Lvpl.append(L)
        for grupo in Lvpl:
            for trazo in grupo:
                trazo.sort(key=itemgetter(0))
```

La función formatea los datos en primer lugar para poder usarlos según convenga, para después iterar a través de ellos e ir catalogando los trazos según las siguientes categorías: de la posición 0 a la posición 0, de 0 a 1, de 0 a 2, de 1 a 1, de 1 a 2 y de 2 a 2. Seguidamente, se analizan todas las opciones en las que la situación 2 está implicada, buscando la conectividad geográfica para convertir el 2 en una situación 0 o 1. Este funcionamiento es similar al de los programas de routing.

En la imagen lateral, los puntos tienen las siguientes situaciones:

a-0; b-0; c-2; d-1; e-2; f-0; g-0.

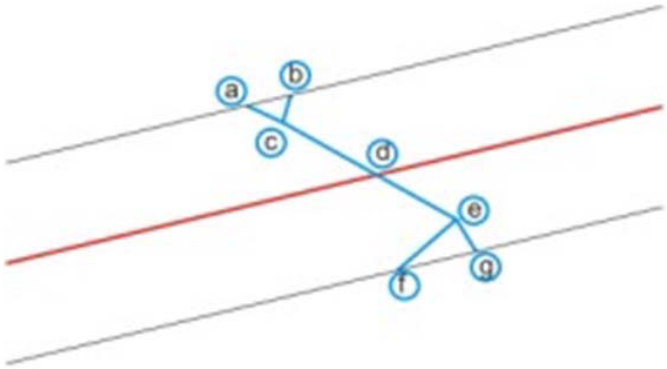
Y los trazos quedan:

a_c:0_2 ; b_c:0_2; c_d:2_1; d_e:1_2; e_g:2_0; e_f:2_0

Eliminando las situaciones de tipo 2 mediante la conectividad geográfica, los trazos quedan:

a_d:0_1; b_d:0_1;d_f:1_0;d_g:1_0

Que ya pueden ser analizados para ser sustituidos por líneas simples.



```
##deteccion de los situ=2
Lreduc=[]
LLLfines=[]
for grupo in Lvp1:
    LLfines=[]
    for trazo in grupo:
        if trazo[0][0]<>2:##si el trazo no es 2_2
            if trazo[1][0]==2:##si el trazo es 0_2 o 1_2
                disp=0
                Lfines=[]
                Linis=[trazo[1]]
                Lusados=[trazo]
            else:##si el trazo es 0_1 o 0_0 o 1_1
                Lfines=[trazo]
                disp=1
            while disp==0:
                disp=1
                for ini in Linis:
                    if ini[0]==2:
                        for item in grupo:
                            if item not in Lusados:
                                if item[0][2]==ini[2]:
                                    disp=0
                                    Linis.append(item[1])
                                    Lusados.append(item)
                                if item[1][2]==ini[2]:
                                    disp=0
                                    Linis.append(item[0])
                                    Lusados.append(item)
                    else:
                        Lfines.append([trazo[0], ini])
            LLfines=LLfines+Lfines
    LLLfines.append(LLfines)
LLLfines2=[]
```

Ahora los casos se han reducido a 0_0, 0_1 y 1_1. Para cada situación se desarrolla una casuística:

0_0: Esta situación tiene dos variantes:

```
    ##reduccion de casos repetidos por geometria
    for grupo in LLLfines:
        ngrupo=[]
        ngeoms=[]
        for trazo in grupo:
            if [trazo[0][2], trazo[1][2]] not in
ngeoms and [trazo[1][2], trazo[0][2]] not in ngeo
ms:
                ngeoms.append([trazo[0][2], trazo
[1][2]])
                ngrupo.append(trazo)
        LLLfines2.append(ngrupo)

    Lp=[]
    for item in Lg:
        Lp.append([float(item.split('_')[0]), flo
at(item.split('_')[1])])
```

a.- ambos puntos están a una distancia menor de dmax entre ellos: en esta situación hay que trazar la línea entre ambos puntos.

b.- ambos puntos están a una distancia mayor de dmax entre ellos: se trazarán las perpendiculares

a las líneas de fc02 que pasen por cada uno de los puntos de situ=0.

0_1: También se observan dos variantes:

a.- el ángulo que forma la unión de ambos puntos con la línea de fc02 es menor que angulomax: en esta situación se debe trazar la perpendicular a la línea de fc02 que pasa por el punto con situ=0

b.- el ángulo que forma la unión de ambos puntos con la línea de fc02 es mayor que angulomax: se traza la unión de ambos puntos.

1_1: Esta situación tiene una sola solución, que pasa por trazar para cada punto la perpendicular a la línea de fc02 en la que no se encuentra el punto.

Por tanto, tendremos como resultado de la función 3 archivos que definen diferentes situaciones: en el primero de ellos se definirán los puntos que deben de unirse (0_0.a y 0_1.b); en el segundo se especificarán aquellos puntos desde los que deben trazarse perpendiculares a fc02 (0_0.b y 0_1.a); por último, se especifican aquellos puntos englobados en la situación 1_1, que reciben un tratamiento particular, puesto que debe eliminarse la línea en la que se encuentran antes de trazarse líneas perpendiculares a fc02.

```
    ##casos
    puntos_near=[]
    puntos_unir=[]
    puntos_nearcomp=[]
    for grupo in LLLfines2:
        for trazo in grupo:
            caso=0
            if trazo[0][0]==0 and trazo[1][0]==0:
caso=1 ##caso de buf a buf
            if trazo[0][0]==0 and trazo[1][0]==1:
caso=2 ##caso de buf a prop
            if trazo[0][0]==1 and trazo[1][0]==0:
caso=3 ##caso de prop a buf
            if trazo[0][0]==1 and trazo[1][0]==1:
caso=4 ##caso de prop a prop
            if caso==1:
                d=((Lp[trazo[0][2]][0]-Lp[trazo[1
][2]][0])**2+(Lp[trazo[0][2]][1]-Lp[trazo[1][2]][
1])**2)**.5
                if d<=dmax:
                    puntos_unir.append([Lp[trazo[
0][2]], Lp[trazo[1][2]]])##almacenamos pares de g
eometrias
                else:
                    puntos_near.append(Lp[trazo[0
][2]])##almacenamos geometrias
                    puntos_near.append(Lp[trazo[1
][2]])
```

```
            if caso==2:
                d=((Lp[trazo[0][2]][0]-Lp[trazo[1
][2]][0])**2+(Lp[trazo[0][2]][1]-Lp[trazo[1][2]][
1])**2)**.5
                if d<=dmax/math.sin(math.radians(
angulomax)):
                    puntos_unir.append([Lp[trazo[
0][2]], Lp[trazo[1][2]]])##almacenamos pares de g
eometrias
                else:
                    puntos_near.append(Lp[trazo[0
][2]])##almacenamos geometrias
            if caso==3:
                d=((Lp[trazo[0][2]][0]-Lp[trazo[1
][2]][0])**2+(Lp[trazo[0][2]][1]-Lp[trazo[1][2]][
1])**2)**.5
                if d<=dmax/math.sin(math.radians(
angulomax)):
                    puntos_unir.append([Lp[trazo[
0][2]], Lp[trazo[1][2]]])##almacenamos pares de g
eometrias
                else:
                    puntos_near.append(Lp[trazo[1
][2]])##almacenamos geometrias
            if caso==4:
                puntos_nearcomp.append([Lp[trazo[
0][2]], trazo[0][3]])##almacenamos geometrias e i
d_02
                puntos_nearcomp.append([Lp[trazo[
1][2]], trazo[1][3]])##almacenamos geometrias e i
d_02
            if caso==0:corta



    w=open(dirsal+'\\near.txt', 'w')
    for item in puntos_near:
        w.write(str(item)[1:-1]+'\n')
    w.close()
    w=open(dirsal+'\\unir.txt', 'w')
    for item in puntos_unir:
        w.write(str(item[0])[1:-1]+'_'+str(item[1
])[1:-1]+'\n')
    w.close()
    w=open(dirsal+'\\comp.txt', 'w')
    for item in puntos_nearcomp:
        if item[1]<>0:
            w.write(str(item[0])[1:-1]+', '+str(i
tem[1])+'\n')
    w.close()

    print time.time()-timeini
```

Función paso03:

Esta función traza, apoyándose en herramientas de geoprocesamiento propietarias, las líneas que deben sustituir a las de la feature class que queremos armonizar (fc01) que quedan dentro del buffer de tamaño dmax contado desde las líneas de la feature class invariante (fc02)

En primer lugar se definen las variables derivadas necesarias y los directorios de temporales adecuados. Posteriormente se lee el archivo near.txt procedente de la función anterior y se procede a almacenar las definiciones de líneas perpendiculares a las de la fc02 que pasan por los puntos que se han definido en el archivo de texto, a través de la función de arcgisscripting near.

```
def paso03(dirtrab, tempdir, dmax, angulomax):
    Temp_Dir = tempdir+'\\'+"temp01"
    Temp_Dir2 = tempdir+'\\'+"temp02"
    os.environ['TEMP'] = Temp_Dir
    os.environ['TMP'] = Temp_Dir2
    dirent=dirtrab+'\\paso02'
    fcprop=tempdir+'\\02.shp'
    fclinesprop=tempdir+'\\02_line.shp'
    dirfin=dirtrab+'\\paso03'
    linfin='linfin.shp'

    gp=arcgisscripting.create()
    gp.workspace=tempdir
    gp.overwriteoutput=1

    ##Trabajo sobre Lnear
    r=open(dirent+'\\near.txt', 'r')
    Lnear_coords=r.read().split('\n')[:-1]
    r.close()
    gp.CreateFeatureClass_management (tempdir, 'pa
ranear.shp', 'POINT')
    rows=gp.insertcursor(tempdir+'\\paranear.shp')
    for item in Lnear_coords:
        pnt = gp.CreateObject("Point")
        pnt.x, pnt.y = item.split(', ')[0], item.s
plit(', ')[1]
        feat = rows.NewRow()
        feat.shape = pnt
        rows.InsertRow(feat)
    del rows
    gp.featureclasstofeatureclass(tempdir+'\\paran
ear.shp', tempdir, 'paranear2.shp')
    gp.Near_analysis (tempdir+'\\paranear2.shp', f
clinesprop, str(dmax*1.5), 'LOCATION')
    Ltrazar=[]
    rows=gp.searchcursor(tempdir+'\\paranear2.shp'
)
    row=rows.next()
    while row:
        Ltrazar.append(str(row.shape.getpart().x)+
', '+str(row.shape.getpart().y)+'_'+str(row.near_x
)+', '+str(row.near_y))
        row=rows.next()
    del rows, row
    print 'Lineas near anadidas'
```

A continuación se procede a leer el archivo unir.txt (procedente de la función anterior) y a almacenar las definiciones de líneas que unen parejas de puntos definidas en este archivo de texto.

```
    ##Trabajo sobre Lunir

    r=open(dirent+'\\unir.txt', 'r')
    Ltrazar=Ltrazar+r.read().split('\n')[:-1]
    r.close()
    print 'Lineas unir anadidas'
```

Seguidamente se procede a añadir a la lista de trazado las parejas de puntos de las que un participante de cada pareja se extrae del archivo nearcomp.txt (procedente de la función anterior) y el otro participante de la pareja ha de calcularse como el punto más cercano de entre los contenidos en las líneas de fc02 que no pertenece al segmento en el que está el primero de los puntos. Esta parte de la función consume una gran cantidad de tiempo, por lo que se va informando al operador del transcurso de la misma (a través de una línea impresa cada 100 operaciones ejecutadas).

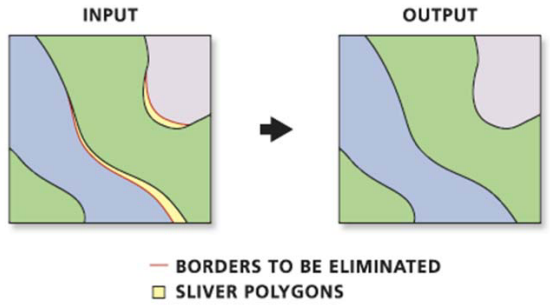
```
    ##Trabajo sobre Lnearcomp
    r=open(dirent+'\\comp.txt', 'r')
    Lpt=r.read().split('\n')[:-1]
    Lptnc=[]
    for it in Lpt:
        Lptnc.append(it.split(', '))

    Lptnc.sort(key=itemgetter(2))
    Lnum=[]
    for item in Lptnc:
        Lnum.append(item[2])
    Snum=set(Lnum)

    Lencargos=[]
    propini=int(Lptnc[0][2])
    print 'Enviando', len(Snum), 'grupos.'
    pos=0
    for item in Lptnc:
        if int(item[2])==propini:
            Lencargos.append(item)
        else:
            if pos/100==float(pos)/100:print 'Grup
o nro', pos, 'tiempo', (time.time()-ini)/60, 'minu
tos.'
            pos=pos+1
            Ltrazar=Ltrazar+nearcomp(fcprop, Lenca
rgos, tempdir, dmax).split('\n')[:-1]
            Lencargos=[]
            Lencargos.append(item)
            propini=int(item[2])
    if len(Lencargos)>0:
        Ltrazar=Ltrazar+nearcomp(fcprop, Lencargos
, tempdir, dmax).split('\n')[:-1]
    print 'Lineas comp anadidas'
```

Para la definición del segundo de los puntos de cada pareja, se establece una función accesoria (nearcomp), que, iterando a través de cada polígono de propiedad, con los puntos implicados en cada polígono, calcula el punto más cercano no contenido en el segmento en el que se encuentra el primero de los puntos.

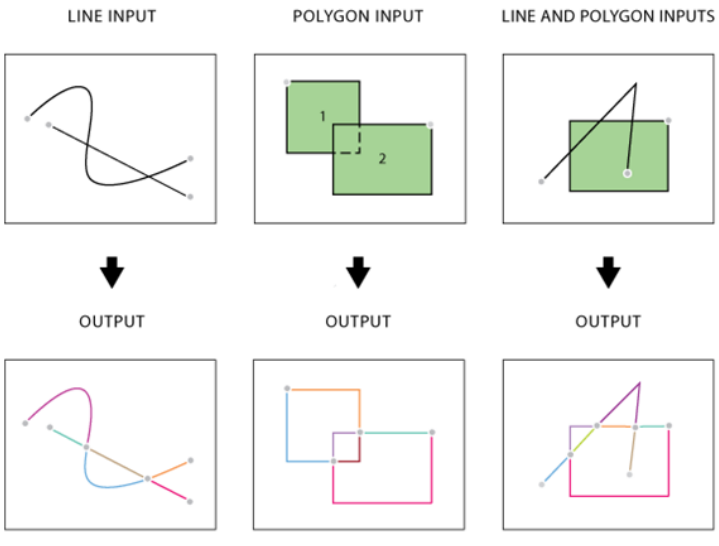
```
def nearcomp(fcprop, Lpuntos, tempdir, dmax):
    gp.select_analysis (fcprop, tempdir+'\\polprop
.shp', '"id_02"='+Lpuntos[0][2])
    gp.FeatureToLine_management (tempdir+'\\polpro
p.shp', tempdir+'\\polprop_line.shp', '#', 'No_Att
ributes')
    gp.SplitLine_management (tempdir+'\\polprop.sh
p', tempdir+'\\polprop_spline.shp')
    rows=gp.searchcursor(tempdir+'\\polprop.shp')
    row=rows.next()
    geom=row.shape
    part=geom.getpart(0)
    pnt=part.next()
    Lxx=[]
    Lyy=[]
    while pnt:
        Lxx.append(pnt.x)
        Lyy.append(pnt.y)
        pnt=part.next()
```

```
        #bound[l,r,b,t]
        bounds=[min(Lxx), max(Lxx), min(Lyy), max(Lyy)
    ]
        gp.MakeFeatureLayer(tempdir+'\\polprop_spline.
    shp', "lines_lyr")
        tres=''
        for punto in Lpuntos:
            gp.createfeatureclass(tempdir, 'puntoorig.
    shp', 'POINT')
            rows=gp.insertcursor(tempdir+'\\puntoorig.
    shp')
            pnt = gp.CreateObject("Point")
            pnt.x, pnt.y = punto[0], punto[1]
            feat = rows.NewRow()
            feat.shape = pnt
            rows.InsertRow(feat)
            del pnt, feat, rows
            gp.buffer(tempdir+'\\puntoorig.shp', tempd
    ir+'\\pto_buf.shp', str(dmax*3))
            gp.intersect(tempdir+'\\pto_buf.shp;'+temp
    dir+'\\polprop_line.shp', tempdir+'\\bufline.shp')
            gp.multiparttosinglepart(tempdir+'\\buflin
    e.shp', tempdir+'\\bufline_SP.shp')
            if gp.getcount(tempdir+'\\bufline_SP.shp')
    >1:
                gp.MakeFeatureLayer(tempdir+'\\puntoor
    ig.shp', "punto_lyr")
                gp.SelectLayerByLocation("lines_lyr",
    "WITHIN_A_DISTANCE", "punto_lyr", '.01')
```

```
            rows=gp.searchcursor("lines_lyr")
            row=rows.next()
            geom=row.shape
            part=geom.getpart(0)
            pnt=part.next()
            origx=pnt.x
            origy=pnt.y
            pnt=part.next()
            finx=pnt.x
            finy=pnt.y
            del pnt, part, geom, row, rows
            if origy-finy<>0.0:
                tg_ang_perp=(finx-origx)/(origy-fi
ny)
                if tg_ang_perp==0.0:
                    yl1=float(punto[1])
                    xl1=bounds[0]
                    yl2=float(punto[1])
                    xl2=bounds[1]
                else:
                    yl1=tg_ang_perp*(bounds[0]-flo
at(punto[0]))+float(punto[1])
                    xl1=(yl1-float(punto[1]))/tg_a
ng_perp+float(punto[0])
                    yl2=tg_ang_perp*(bounds[1]-flo
at(punto[0]))+float(punto[1])
                    xl2=(yl2-float(punto[1]))/tg_a
ng_perp+float(punto[0])
            if origy-finy==0.0:
                yl1=bounds[2]
                xl1=float(punto[0])
                yl2=bounds[3]
                xl2=float(punto[0])
            gp.CreateFeatureClass_management (temp
dir, 'lineperp.shp', 'POLYLINE')
            rows=gp.insertcursor(tempdir+'\\linepe
rp.shp')
            lineArray = gp.CreateObject("Array")
            pnt = gp.CreateObject("Point")
            id=0
            pnt.id, pnt.x, pnt.y = str(id), str(xl
1), str(yl1)
            lineArray.add(pnt)
```

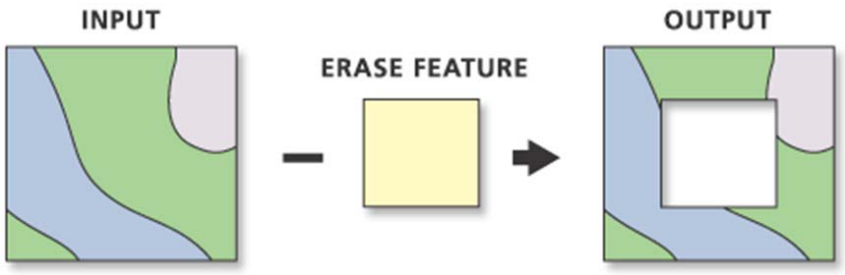


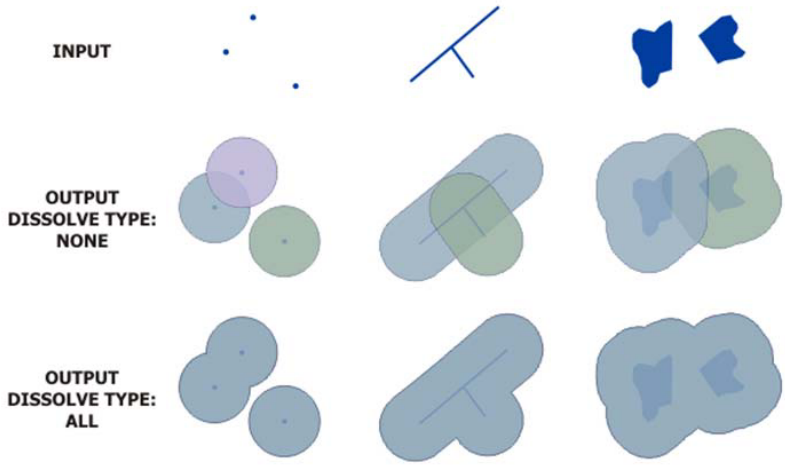

```
            feat = rows.NewRow()
            feat.shape = lineArray
            rows.InsertRow(feat)
            del pnt, lineArray, feat, rows
            gp.intersect(tempdir+'\\polprop.shp;'+
tempdir+'\\lineperp.shp', tempdir+'\\lineint.shp')
            gp.multiparttosinglepart(tempdir+'\\li
neint.shp', tempdir+'\\lineint_SP.shp')
            gp.MakeFeatureLayer(tempdir+'\\lineint
_SP.shp', "lineint_lyr")
            try:
                gp.SelectLayerByLocation("lineint_
lyr", "WITHIN_A_DISTANCE", tempdir+'\\puntoorig.sh
p', '.01')
                rows=gp.searchcursor("lineint_lyr"
)
                row=rows.next()
                geom=row.shape
                part=geom.getpart(0)
                pnt=part.next()
                origx=pnt.x
                origy=pnt.y
                pnt=part.next()
                finx=pnt.x
                finy=pnt.y
                longline=((finy-origy)**2+(finx-or
igx)**2)**.5
            except:
                print 'Problema en punto', str(pun
to)[1:-1]
                longline=dmax*4##para que no pase
a tres
            if longline<dmax*3:
                if abs(origx-float(punto[0]))<abs(
finx-float(punto[0])):
                    tres=tres+punto[0]+', '+punto[
1]+'_'+str(finx)+', '+str(finy)+'\n'
                else:
                    tres=tres+punto[0]+', '+punto[
1]+'_'+str(origx)+', '+str(origy)+'\n'
    return tres
```



Por último, sólo queda escribir los archivos de trazos.

```
    gp.CreateFeatureClass_management (dirfin, linf
in, 'POLYLINE')
    rows=gp.insertcursor(dirfin+'\\'+linfin)
    id=0
    for item in Ltrazar:
        lineArray = gp.CreateObject("Array")
        pnt = gp.CreateObject("Point")
        pnt.id, pnt.x, pnt.y = str(id), item.split
('_')[0].split(', ')[0], item.split('_')[0].split(
', ')[1]
        lineArray.add(pnt)
        pnt.id, pnt.x, pnt.y = str(id), item.split
('_')[1].split(', ')[0], item.split('_')[1].split(
', ')[1]
        lineArray.add(pnt)
        row = rows.NewRow()
        row.shape = lineArray
        rows.InsertRow(row)
        id=id+1
    del rows, row

    print 'Tiempo empleado en paso03:',(time.time(
)-ini)/60, 'minutos.'
```





Función paso04:

La siguiente función genera los polígonos a través de las líneas no modificables de fc01 (aquellas que quedan a más de la distancia dmax de las líneas de la fc02), las propias líneas de fc02 y las líneas trazadas en el paso anterior, que sustituyen a los tramos que quedan dentro del buffer de distancia dmax, atribuyéndoles la información original a través de porcentajes de acierto de un polígono en otro.

Finalmente, se establece un sistema de comprobación y se marcan los polígonos que no superan esta comprobación para su revisión manual. Esta función es dependiente del módulo arcgisscripting, y por tanto está sujeta a licencia.

En primer lugar, se declaran las variables de entorno y derivadas.

```
def paso04(dirtrab, temp, dmax, angulomax, arepp):
    ini=time.time()
    Temp_Dir = temp+'\\'+"temp01"
    Temp_Dir2 = temp+'\\'+"temp02"
    os.environ['TEMP'] = Temp_Dir
    os.environ['TMP'] = Temp_Dir2

    dirent=dirtrab+'\\paso03'
    linfin=dirent+'\\linfin.shp'
    fc02=temp+'\\02.shp'
    fclines02=temp+'\\02_line.shp'
    fc01=temp+'\\01.shp'
    fclines01=temp+'\\01_line.shp'
    fclinesval01=temp+'\\01_lineval.shp'
    fclinecomp=temp+'\\linecomp.shp'
    dirfin=dirtrab+'\\paso04'

    gp=arcgisscripting.create()
    gp.workspace=temp
    gp.overwriteoutput=1
```

En el siguiente paso se generan los polígonos a través de las feature classes antes mencionadas y se establece una comparación estadística con la unión simple de capas, de la que nuestra capa armonizada, hasta ahora sin datos asociados, adquiere los datos alfanuméricos.

```
    ##Conversion a poligonos
    gp.FeatureToPolygon(linfin+';'+fclines02+';'+f
clinesval01+';'+fclinecomp, temp+'\\poly.shp', '#'
, 'No_Attributes')
    gp.addfield(temp+'\\poly.shp', 'id_poly', 'LON
G')
```

```
    rows=gp.updatecursor(temp+'\\poly.shp')
    row=rows.next()
    while row:
        row.id_poly=str(row.FID+1)
        rows.updaterow(row)
        row=rows.next()
    gp.union(fc02+';'+fc01, temp+'\\fc01_02.shp',
'NO_FID')
    gp.multiparttosinglepart(temp+'\\fc01_02.shp',
temp+'\\fc01_02_SP.shp')
    gp.addfield(temp+'\\fc01_02_SP.shp', 'id_union
', 'LONG')
    rows=gp.updatecursor(temp+'\\fc01_02_SP.shp')
    row=rows.next()
    while row:
        row.id_union=str(row.FID+1)
        rows.updaterow(row)
        row=rows.next()
    gp.union(temp+'\\poly.shp'+';'+temp+'\\fc01_02
_SP.shp', temp+'\\fcpoly_01_02.shp', 'NO_FID')
    gp.multiparttosinglepart(temp+'\\fcpoly_01_02.
shp', temp+'\\fcpoly_01_02_SP.shp')
    gp.addfield(temp+'\\fcpoly_01_02_SP.shp', 'ar_
uni', 'DOUBLE')
    rows=gp.updatecursor(temp+'\\fcpoly_01_02_SP.s
hp')
    row=rows.next()
    while row:
        row.ar_uni=str(row.shape.area)
        rows.updaterow(row)
        row=rows.next()
    gp.Statistics(temp+'\\fcpoly_01_02_SP.shp', te
mp+'\\summary.dbf', 'ar_uni max', 'id_poly')
    gp.MakeFeatureLayer(temp+'\\fcpoly_01_02_SP.sh
p', 'lyr')
    gp.addjoin('lyr', 'id_poly', temp+'\\summary.d
bf', 'id_poly')
    gp.selectlayerbyattribute('lyr', 'NEW_SELECTIO
N', '"ar_uni"="max_ar_uni"')
    gp.featureclasstofeatureclass('lyr', temp, 'at
trs.shp')
    gp.MakeFeatureLayer(temp+'\\poly.shp', 'lyr')
    gp.addjoin('lyr', 'id_poly', temp+'\\attrs.shp
', 'id_poly')
    gp.featureclasstofeatureclass('lyr', temp, 'po
ly_attrs.shp')
    flds=gp.listfields(temp+'\\poly_attrs.shp')
    Lf=[]
    fld=flds.next()
    while fld:
        Lf.append(fld.name)
        fld=flds.next()
    tb=''
    for item in Lf:
        if item<>'FID' and item<>'Shape' and item<
>'id_01' and item<>'id_02' and item<>'ar_uni':
            tb=tb+item+';'
    tb=tb[:-1]
    gp.deletefield(temp+'\\poly_attrs.shp', tb)
```

Por último, se añaden una serie de campos y se ejecutan varias consultas para detectar errores en la generación de los polígonos o en la asignación de atributos. Concretamente se buscan porciones de polígono que hayan sido modificadas durante la armonización, que tengan una de sus dimensiones lineales mayor que dmax y que tengan un área mayor que la del polígono mínimo autorizado.

```
        gp.addfield(temp+'\\poly_attrs.shp', 'area', 'D
    OUBLE')
        rows=gp.updatecursor(temp+'\\poly_attrs.shp')
        row=rows.next()
        while row:
            row.area=str(row.shape.area)
            rows.updaterow(row)
            row=rows.next()
        gp.addfield(temp+'\\poly_attrs.shp', 'dif', 'DO
    UBLE')
        gp.addfield(temp+'\\poly_attrs.shp', 'porcac',
    'DOUBLE')
        rows=gp.updatecursor(temp+'\\poly_attrs.shp')
        row=rows.next()
        while row:
            row.dif=str(abs(row.area-row.ar_uni))
            row.porcac=str((row.area-row.dif)/row.area*
    100)
            rows.updaterow(row)
            row=rows.next()
        print 'Comenzando comprobaciones...'
        gp.dissolve(temp+'\\poly_attrs.shp', temp+'\\01
    _mod.shp', 'id_01')
        gp.union(temp+'\\01.shp'+';'+temp+'\\01_mod.shp
    ', temp+'\\01_comp_u.shp')
        gp.multiparttosinglepart(temp+'\\01_comp_u.shp'
    , temp+'\\01_comp_u_SP.shp')
        gp.addfield(temp+'\\01_comp_u_SP.shp', 'revisar
    ', 'SHORT')
        rows=gp.updatecursor(temp+'\\01_comp_u_SP.shp')
        row=rows.next()
        while row:
            ar=row.shape.area
            per=row.shape.length
            h1=(per/2+(abs(per**2/4-4*ar))**.5)/2
            h2=(per/2-(abs(per**2/4-4*ar))**.5)/2
            h=min([h1,h2])
            if h<0:corta#####
            if row.id_01<>row.id_012 and h>dmax and ar>
    arepp:
                row.revisar=str(1)
                rows.updaterow(row)
            row=rows.next()
        del rows, row
        gp.copy(temp+'\\01_comp_u_SP.shp', dirfin+'\\co
    mparacion.shp')
        gp.copy(temp+'\\01_mod.shp', dirfin+'\\01_mod.s
    hp')

    gp.copy(temp+'\\poly_attrs.shp', dirfin+'\\poly
_attrs.shp')
    gp.copy(temp+'\\01.shp', dirfin+'\\01.shp')
    gp.copy(temp+'\\02.shp', dirfin+'\\02.shp')

    print (time.time()-ini)/60
```

Los polígonos de la capa 01_comp_u_SP (generada en el directorio temp\paso04) que en el campo 'revisar' tengan valor 1 serán marcadores de revisión manual sobre la capa 01_mod, a través de la comparación con la capa original a armonizar (01). Tras la realización de varios procesos de armonización, se estima que menos de 1 de cada 5000 polígonos han de ser revisados manualmente en este paso.

Función paso05:

Esta función realiza la eliminación de pequeños fragmentos de polígono que quedan como subproducto del proceso de armonización geométrica. Se eliminarán aquellos polígonos que:

- Tengan un área menor que el parámetro 'arealimite' (polígono mínimo autorizado)

- No supongan en sí mismos un polígono completo de la fc02 (a la que ajustamos)

En otras palabras, todos los polígonos menores del área mínima permitida y para los que exista algún otro polígono dentro de la misma definición en fc02, serán absorbidos por el polígono con el que comparta mayor límite común de entre los que tengan el mismo atributo de fc02.
La función, como las demás, comienza con la definición de variables de entorno y derivadas necesarias.

```
def paso05(temp, dirtrab, arealimite):
    Temp_Dir = temp+'\\'+"temp01"
    Temp_Dir2 = temp+'\\'+"temp02"
    os.environ['TEMP'] = Temp_Dir
    os.environ['TMP'] = Temp_Dir2
    ini=time.time()
    gp=arcgisscripting.create()
    gp.workspace = temp
    gp.overwriteoutput = 1
    dirent=dirtrab+'\\paso04'
    dirsal=dirtrab+'\\paso05'
```

Posteriormente se procede a la detección de la situación antes descrita. En general la mecánica el similar a la del script EPPDAP ya ensayado para la versión 2005 de SIOSE, pero se ha mejorado enormemente su rendimiento a través de la optimización del número de operaciones de eliminate ejecutadas. Para este fin, la función agrupa los polígonos que han de ser eliminados de tal manera que se incluya en cada grupo el mayor número de estos que no estén geográficamente conectados, para que la eliminación se produzca de forma controlada, y no se unan polígonos separados por líneas pertenecientes a la fc02.

```
        gp.union(dirent+'\\01_mod.shp;'+dirent+'\\02.sh
    p', temp+'\\u01mod02.shp', 'NO_FID')
        gp.multiparttosinglepart(temp+'\\u01mod02.shp',
    temp+'\\u01mod02_SP.shp')
        gp.addfield(temp+'\\u01mod02_SP.shp', 'idu_unio
    n', 'LONG')
        gp.addfield(temp+'\\u01mod02_SP.shp', 'area', '
    DOUBLE')
        rows=gp.updatecursor(temp+'\\u01mod02_SP.shp')
        row=rows.next()
        while row:
            row.idu_union=str(row.FID+1)
            row.area=str(row.shape.area)
            rows.updaterow(row)
            row=rows.next()
        del rows, row
        rows=gp.searchcursor(temp+'\\u01mod02_SP.shp')
        row=rows.next()
        Lprops=[]
        while row:
            Lprops.append(row.id_02)
            row=rows.next()
        del rows, row
        Lprops.sort()
        Lpr=[]
        Lfr=[]
        frec=0
        pos=0
        while pos<len(Lprops)-1:
```

```
            frec=frec+1
            if Lprops[pos+1]<>Lprops[pos]:
                    Lpr.append(Lprops[pos])
                    Lfr.append(frec)
                    frec=0
            pos=pos+1
        Lpropsunicas=[]
        pos=0
        for item in Lfr:
            if item==1:
                Lpropsunicas.append(Lpr[pos])
            pos=pos+1
        gp.copy(temp+'\\u01mod02_SP.shp', temp+'\\u01mo
    d02_SP_nounicas.shp')
        rows=gp.updatecursor(temp+'\\u01mod02_SP_nounic
    as.shp')
        row=rows.next()
        while row:
            if row.id_02 in Lpropsunicas:
                rows.deleterow(row)
            row=rows.next()
        del rows, row
        gp.MakeFeatureLayer_management (temp+'\\u01mod0
    2_SP_nounicas.shp', 'lyr')
        gp.MakeFeatureLayer_management (temp+'\\u01mod0
    2_SP_nounicas.shp', 'lyr_sel', '"area"<'+str(areali
    mite))
        rows=gp.searchcursor('lyr_sel')
        Lpr_peques=[]
        row=rows.next()
```

```
    Lpr_peques=[]
    row=rows.next()
    while row:
        Lpr_peques.append(row.id_02)
        row=rows.next()
    del rows, row
    gp.SelectLayerByLocation_management ('lyr', 'IN
TERSECT', 'lyr_sel')
    gp.FeatureclassToFeatureclass_conversion ('lyr'
, temp, 'sujetosyadys.shp')
```

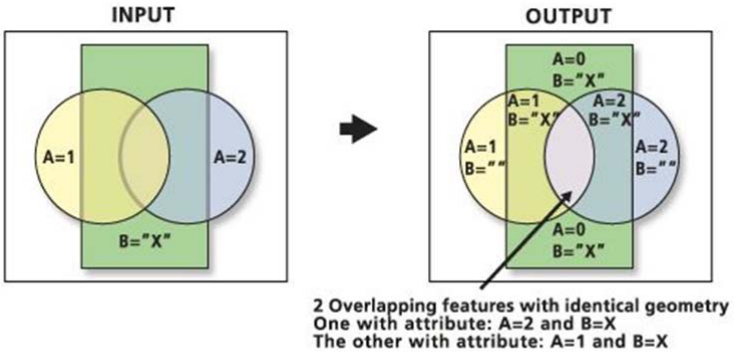
```
    rows=gp.updatecursor(temp+'\\sujetosyadys.shp')
    row=rows.next()
    while row:
        if row.id_02 not in Lpr_peques:
            rows.deleterow(row)
        row=rows.next()
    del rows, row
    Lsujyadys=[]
    rows=gp.searchcursor(temp+'\\sujetosyadys.shp')
    row=rows.next()
    while row:
        Lsujyadys.append(row.idu_union)
        row=rows.next()
    del rows, row
    gp.copy(temp+'\\u01mod02_SP.shp', temp+'\\resto
.shp')
    rows=gp.updatecursor(temp+'\\resto.shp')
    row=rows.next()
    while row:
        if row.idu_union in Lsujyadys:
            rows.deleterow(row)
        row=rows.next()
    del rows, row
    gp.dissolve(temp+'\\sujetosyadys.shp', temp+'\\
grupos.shp', 'id_02')
    gp.dissolve(temp+'\\sujetosyadys.shp', temp+'\\
grupos_diss.shp')
    gp.multiparttosinglepart(temp+'\\grupos_diss.sh
p', temp+'\\grupos_diss_SP.shp')
    gp.MakeFeatureLayer_management (temp+'\\grupos.
shp', 'lyr_grupos')
    conjunto=1
    LLgrupos=[]
    while conjunto<=gp.getcount(temp+'\\grupos_diss
_SP.shp'):
        gp.MakeFeatureLayer_management (temp+'\\gru
pos_diss_SP.shp', 'lyr_grupos_diss', '"FID"='+str(c
onjunto-1))
        gp.SelectLayerByLocation_management ('lyr_g
rupos', 'INTERSECT', 'lyr_grupos_diss')
        L=[]
        rows=gp.searchcursor('lyr_grupos')
        row=rows.next()


    while row:
        L.append(row.id_02)
        row=rows.next()
    del rows, row
    LLgrupos.append(L)
    gp.delete('lyr_grupos_diss')
    conjunto=conjunto+1

LLgrupos_orden=[]
a=0
pos=0
while a==0:
    a=1
    L=[]
    for item in LLgrupos:
        try:
            L.append(item[pos])
            a=0
        except:
            pass
    LLgrupos_orden.append(L)
    pos=pos+1
LLgrupos_orden=LLgrupos_orden[:-1]
```

```
        LLgrupos_orden2=[]
        for item in LLgrupos_orden:
            L=[]
            for item2 in set(item):
                L.append(item2)
            LLgrupos_orden2.append(L)
        LLgrupos_orden2.reverse()

        LLgrupos_orden3=[]
        Ltot=[]
        for item in LLgrupos_orden2:
            L=[]
            for item2 in item:
                if item2 not in Ltot:
                    L.append(item2)
                    Ltot.append(item2)
            LLgrupos_orden3.append(L)

        vuelta=1
        print 'Comenzando proceso con', len(LLgrupos_or
    den3), 'iteraciones'
        for item in LLgrupos_orden3:
            if vuelta/100==float(vuelta)/100:print "Vue
    lta nro "+str(vuelta)+" de "+str(len(LLgrupos_orden
    3))
            gp.copy(temp+'\\sujetosyadys.shp', temp+'\\
    paraeliminate.shp')
            rows=gp.updatecursor(temp+'\\paraeliminate.
    shp')
            row=rows.next()
            while row:
```

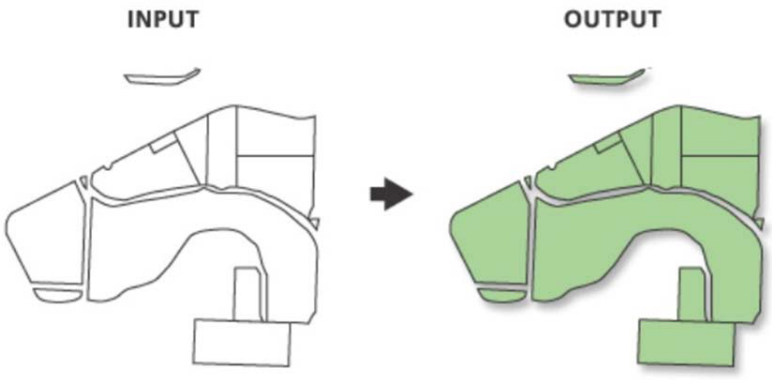
```
            while row:
                if row.id_02 not in item:
                    rows.deleterow(row)
                else:
                    row.area=str(row.shape.area)
                    rows.updaterow(row)
                row=rows.next()
            del rows, row
            a=0
            while a<10:
                gp.MakeFeatureLayer_management (temp+'\
\paraeliminate.shp', 'lyr_eli')
                gp.selectlayerbyattribute('lyr_eli', 'N
EW_SELECTION', '"area"<'+str(arealimite))
                if gp.getcount('lyr_eli')>0:

                    rows=gp.searchcursor(temp+'\\parael
iminate.shp')
                    row=rows.next()
                    Lits=[]
                    while row:
                        Lits.append([row.id_02, row.idu
_union, row.shape.area])
                        row=rows.next()
                    Lits.sort()
                    LLits=[]
                    prop=Lits[0][0]
                    LOO=[]
                    for its in Lits:
                        if its[0]==prop:
                            LOO.append(its)
                        else:
                            LLits.append(LOO)
                            LOO=[]
                            LOO.append(its)
                            prop=its[0]
                    LLits.append(LOO)
                    Lquitar=[]
```

```
                for Lits in LLits:
                    Lar=[]
                    for its in Lits:
                        Lar.append(its[2])
                    if max(Lar)<arealimite:
                        for its in Lits:
                            if its[2]==max(Lar):
                                Lquitar.append(its[
1])

                for quitar in Lquitar:
                    gp.selectlayerbyattribute('lyr_
eli', 'REMOVE_FROM_SELECTION', '"idu_union"='+str(q
uitar))

                gp.eliminate('lyr_eli', temp+'\\eli
minate.shp')
                if gp.getcount(temp+'\\paraeliminat
e.shp')-gp.getcount('lyr_eli')==gp.getcount(temp+'\
\eliminate.shp'):




                    a=10
                gp.delete(temp+'\\paraeliminate.shp
')
                gp.delete('lyr_eli')
                gp.rename(temp+'\\eliminate.shp', t
emp+'\\paraeliminate.shp')
            else:
                gp.delete('lyr_eli')
                a=10
            a=a+1
        rows=gp.updatecursor(temp+'\\sujetosyadys.s
hp')
        row=rows.next()
        while row:
            if row.id_02 in item:
                rows.deleterow(row)
            row=rows.next()
        del rows, row
        gp.append(temp+'\\paraeliminate.shp', temp+
'\\sujetosyadys.shp', 'NO_TEST')
        gp.delete(temp+'\\paraeliminate.shp')
        vuelta=vuelta+1

    gp.append(temp+'\\sujetosyadys.shp', temp+'\\re
sto.shp', 'NO_TEST')
    gp.copy(temp+'\\resto.shp', dirsal+'\\01_02_EPP
.shp')
```

Tras el proceso iterativo antes comentado, se realiza un conteo de polígonos menores que el área mínima tanto en la capa producida como en la fc02. Si en número no es coincidente, se genera un archivo txt con aquellos polígonos que no cumplen las condiciones finales exigidas. Este archivo txt puede ser cargado como una tabla (tipo csv) en un editor, con lo que la segunda sesión de edición también es muy dirigida.




```
    npequesprop=0
    rows=gp.searchcursor(dirent+'\\02.shp')
    row=rows.next()
    while row:
        if row.shape.area<arealimite:
            npequesprop=npequesprop+1
        row=rows.next()
    del rows, row

    npequesu=0
    Lerrores=[]
    rows=gp.searchcursor(dirsal+'\\01_02_EPP.shp')
    row=rows.next()
    while row:
        if row.shape.area<arealimite:
            npequesu=npequesu+1
            if row.id_02 in Ltot:
                Lerrores.append(row.idu_union)
        row=rows.next()
    del rows, row
    print "Proceso terminado; "+dirsal+'\\01_02_EPP
.shp'+" generada, con "+str(npequesu)+ " poligonos
menores de "+str(arealimite)+" m2, de los cuales "+
str(npequesprop)+" corresponden a la capa de propie
dad."
    if len(Lerrores)>0 and npequesu>npequesprop:
        w=open(dirsal+'\\errores.txt', 'w')
        w.write('idu_union, revisar\n')
        for item in Lerrores:
            w.write(str(item)+', 1\n')
        w.close()
        print "Errores guardados en tabla", dirsal,
"errores.txt."
    print (time.time()-ini)/60
```



2. Armonización Temática

Una vez ejecutada la armonización geométrica, se diseña e implementa una batería de consultas con el fin hacer patentes las posibles incoherencias entre las distintas bases de referencia, jerarquizándolas en una escala que va desde las simples imprecisiones a las incompatibilidades más manifiestas.

Este listado jerarquizado sirve como base a la labor de fotointerpretación y corrección de errores, tras la que quedará asegurado el máximo nivel de precisión y coherencia posible. La base de referencia resultante se somete posteriormente a un estricto control de calidad quedando a disposición de gestores, investigadores y público para su uso.

El resultado del procedimiento anterior permite obtener una base cartográfica en la que cada entidad (polígono) contiene identificadores que apuntan a cada una de las características territoriales que se determinan en las cartografías que se armonizan, a la manera de un procedimiento de unión. La única diferencia estriba en que las operaciones realizadas permiten una geometría mucho más depurada, lo que supone una enorme mejora en cuanto a la claridad y precisión de las cartografías.

Pero este procedimiento puede poner de manifiesto multitud de incoherencias entre las cartografías participantes. Por ejemplo, podemos armonizar una cartografía de ocupación/uso como SIOSE con otra de uso como MUCVA, cuidando de que las versiones sean contemporáneas, obteniendo para algunos polígonos resultados confusos: por ejemplo, SIOSE puede decirnos que esa porción del territorio es dehesa con determinados valores de ocupación, mientras que MUCVA nos dice que ese mismo territorio es de tipo urbano. Evidentemente, una de ambas bases de referencia estará equivocada, o bien ambas no son exactamente contemporáneas, y en el intervalo temporal han acaecido cambios territoriales.

A partir de la detección de estas incoherencias, fue necesario establecer un procedimiento para la jerarquización de las mismas, y para su cualificación, puesto que en algunos casos estas incoherencias pudieron ser corregidas automáticamente, mientras que en otros casos se hace necesaria la revisión por parte de un fotointérprete, de manera que la incoherencia sea corregida de forma satisfactoria.

Esta jerarquización permite una mejora tanto de la cartografía producida como de las bases de referencia a partir de las cuales se obtiene (SIOSE, SIGPAC…) a las que pueden volcarse las correcciones oportunas de forma automática, mejora que además puede ser gestionada de forma óptima, puesto que se pueden tratar preferentemente aquellas que más urgentes resulten, sea por su tamaño, o por la importancia de las temáticas a las que afectan.

También puede darse el caso de que una base de referencia complemente la información ofrecida por otra: un polígono puede ser interpretado como ‘quercínea con matorral denso’ en MUCVA, mientras que SIOSE nos da ocupaciones de 20% de arbolado forestal (sin especificar), 60% de matorral y 20% de pastos, y la base de referencia de Vegetación (VEGE10), que se apoya en muestreos de campo, nos especifica que la especie arbórea es ‘Quercus ilex’. Por tanto, podemos sumar toda la información y determinar ese polígono en la nueva base siendo 20% de quercus illex, 60% de matorral y 20% de pastos, y pudiendo ahora actualizar SIOSE, especificando que el 20% de arbolado forestal se corresponde con el tipo ‘quercínea perennifolia’.

Por último, es necesario en muchos casos realizar el traslado de datos de los modelos desde las bases de referencia a armonizar al modelo de datos propio de la base resultante. Estas operaciones se realizan a través del desarrollo de programas llamados ‘pasarelas de datos’, desarrollos en los que RqueR tys tiene una dilatada experiencia.

La resolución, a través del procedimiento de armonización espacial, de las incoherencias geométricas que subyacen en las diferentes bases de referencia, y que son producto de la diferencia de trazo entre dos fotointérpretes a la hora de delimitar un fenómeno sobre el territorio, de alguna manera evidencia la necesidad de una coherencia en el ámbito temático mucho más fuerte.

Esto es debido a que las distintas interpretaciones del territorio que quedan representadas en las distintas cartografías no sólo son parte ahora de un mismo objeto cartográfico (un polígono) sino que comparten de alguna manera un mismo modelo de datos. De forma más precisa, se confronta el contenido de un modelo con el de otro, para la misma situación geográfica.

En orden a establecer una metodología de actuación ante estas imprecisiones o incoherencias que previsiblemente afloran una vez realizada la armonización espacial, es importante destacar previamente el potencial previsible: no sólo permitirá conseguir una base de referencia mucho más correcta, sino que a su vez permite mejorar el resto de bases de referencia participantes, en particular en aquellos casos en los que se detecte un error, que podrá ser indicado a los responsables de esta base de referencia.
Pero es también vital tener prevista una jerarquización de estas incoherencias, de manera que puedan ser objeto de actualización o revisión de forma ordenada. Todo ello debe formar parte de una estrategia de detección, jerarquización y corrección de incoherencias temáticas. Resultarando imprescindible para esta labor el apoyo de los técnicos de la Dirección Facultativa del proyecto, es importante definir las líneas guía:

- En primer lugar es necesario conocer de forma detallada la filosofía de caracterización territorial que subyace a cada una de las bases de referencia implicadas: para ello es necesario entrevistarse con los responsables temáticos de cada una de ellas, con el fin de conocer los pormenores sobre la interpretación de la realidad que ofrecen; un estudio pormenorizado del modelo de datos de cada una de ellas es asimismo vital para la compleción de esta fase.

- Posteriormente, es necesario realizar una 'matriz de confrontación' que presupone evaluar las diferentes posibilidades de combinación de temáticas en un polígono armonizado; este proceso puede ser de alguna manera facilitado a través del estudio de casos más relevantes, tanto en el plano probabilístico (lo que sucede en gran parte de las ocasiones) como desde el punto de vista de la importancia (lo que debería suceder siempre o por el contrario, qué usos simultáneos nunca deberían coincidir en un polígono armonizado).

- En tercer lugar, la caracterización de cada una de estas combinaciones (o en el peor de los casos las combinaciones más frecuentes o más extremas) con un cierto índice de compatibilidad: este índice, que a veces puede ser cualitativo (del tipo 'compatible, neutro, no compatible') o cuantitativo (porcentual o como factor) revela cuando se aplica a la base armonizada, las incoherencias entre las distintas bases.

- En cuarto lugar, se debe establecer una jerarquía de actuación, de forma que se de una posición más alta en esa jerarquía a imprecisiones o incoherencias muy llamativas, que afectaran a una superficie muy grande, o que sean de una gran importancia temática (por ejemplo, cuando se afecte a temáticas que estén relacionadas con la evaluación de ayudas, o especies protegidas…)

- Como quinto paso, y sabiendo que el procedimiento de armonización afecta a más de dos bases de referencia, es necesario detectar un buen número de situaciones de concomitancia o de alineación: esto significa que si de cuatro bases de referencia cuya armonización espacial ya ha sido efectuada, tres de ellas dicen que un determinado espacio está poblado por coníferas, una dice que son quercíneas y otra que es pasto, es posible, conociendo las fiabilidades de cada una de ellas, determinar de forma automática que el polígono es de coníferas, informar a los responsables de las bases con menores posibilidades de acierto que existe un probable error en esa situación.

- El sexto paso es la corrección efectiva, que es un proceso en sí misma, puesto que a través de procedimientos de medida de la fiabilidad (como el que hemos visto antes) se puede de forma automática acometer una primera corrección, mientras que ciertas incoherencias para las que los datos son muy dispersos o bien no hay suficiente fiabilidad, deben ser revisados por un técnico fotointérprete especializado, que es quien en última instancia debe tener voz. Por ello las correcciones realizadas de forma automática deben tener la consideración de 'estadísticas', puesto que siempre cabe la posibilidad de que el polígono sea, en el caso antes expuesto, efectivamente de quercíneas, y que las otras bases de referencia estuvieran equivocadas.