UNIVERSIDAD EMPRESARIAL SIGLO 21

Redistrictalización Automática, su complejidad computacional y solución algorítmica por Búsqueda Tabú

Trabajo Final de Graduación

Gastón Pezzuchi

2.011

Redistrictalización Automática, su complejidad computacional y solución algorítmica por Búsqueda Tabú

Trabajo Final de Graduación

En cumplimiento parcial a los requisitos de Licenciatura en Informática de la Universidad Empresarial Siglo 21

Gaston Pezzuchi
DNI 23.281.129 – Legajo VINF0283

Toutes les généralisations sont dangereuses, même celle-ci.
(Todas las generalizaciones son peligrosas, incluida esta).

Alexandre Dumas

Ficha Biblioteca

TRABAJO FINAL DE GRADUACION – PROYECTO DE APLICACIÓN PROFESIONAL
AUTOR: PEZZUCHI, Gastón APELLIDO(s) Nombre(s)
TITULO Y SUBTITULO: Redistrictalización Automática, su complejidad computacional y solución algorítmica por Búsqueda Tabú
LUGAR DE EDICION: Córdoba, Córdoba. República Argentina
A ÑO EN QUE SE OBTUVO: 2.011
NUM. DE PAGINAS: 159
NIVEL ACADEMICO OBTENIDO: Licenciatura
INSTITUCION Y DEPENDENCIA QUE OTORGA EL NIVEL ACADEMICO: Universidad Empresarial Siglo 21
NOMBRE DEL ASESSOR: MALDONADO, Calixto APELLIDO(s) Nombre(s)
LOCALIZACION DE LA OBRA:

Resumen

El presente trabajo explora brevemente la complejidad computacional inherente a los problemas de districtalización automática que surgen en diferentes áreas del conocimiento y de la realidad cuando es necesario definir regiones en base a información previamente agregada en unidades espaciales; generalmente estas regiones deben cumplir ciertas restricciones (balance poblacional, compacidad, integridad de comunidades, homogeneidad socioeconómica, etc.) que muchas veces compiten entre sí. Las características complejas del problema llevan necesariamente a la adopción de estrategias meta-heurísticas para la aproximación a soluciones óptimas o cuasi-óptimas. En este marco, se explora un algoritmo de Búsqueda Tabú y Memoria Adaptativa y su aplicabilidad a procesos de districtalización con radios censales del Área Metropolitana de Buenos Aires. En base a trabajos anteriores sobre este algoritmo, se procedió a ajustar los parámetros del mismo teniendo en cuenta las particularidades de las teselas elegidas y se obtuvo una solución óptima para el caso de ejemplo elegido del Partido de Berisso de la Provincia de Buenos Aires. Asimismo, este trabajo muestra que es factible embeber esta aproximación en Sistemas de Información Geográfica de uso masivo en nuestro medio.

Abstract

This paper briefly explores the computational complexity inherent in automatically redistricting problems (re-apportionment) that arise in different realms of science and in everyday reality when it is necessary to define regions based on information previously aggregated into smaller spatial units; usually these regions (districts) must meet certain restrictions (population balance, compactness, community integrity, socio-economic homogeneity, etc.) that often compete against each other. The complex features of the problem necessarily lead to the adoption of meta-heuristic strategies for approaching optimal or quasi-optimal solutions. Hence, we explore a Tabu Search and Adaptive Memory algorithm and its applicability to redistricting processes based on census block groups for the Metropolitan Area of Buenos Aires. Based on previous work on this algorithm, we proceeded to adjust its parameters taking into account the characteristics of the chosen tiles and we obtained an optimal solution for the Berisso county of Buenos Aires Province which was chosen as a use case scenario. Lastly this work shows that it is feasible to embed this approach in widespread Geographic Information Systems (GIS).

Índice de Contenidos

Índice de Tablas

Índice de Gráficos

Introducción y antecedentes generales

En el marco del *Trabajo Final de Graduación* (TFG) de la carrera de **Licenciatura en Informática** de la *Universidad Empresarial Siglo 21*, se ha elegido explorar la solución algorítmica de problemas de regionalización[1] utilizando una heurística[2] de *Búsqueda Tabú* y *memoria adaptativa*.

Este tipo de problemas ocurren con frecuencia en las ciencias regionales e inclusive en las ciencias políticas cuando es necesario dividir el espacio en un número predeterminado de zonas, partiendo de bloques constructivos individuales y sujeto a un conjunto de restricciones. Desafortunadamente no son muchas las posibilidades de acceso libre a herramientas informáticas de este tipo, debido esencialmente a cuestiones derivadas del costo de los softwares existentes[3], a las especificidades del mismo que lo hacen inviable para problemas similares (*e.g.* software diseñado para districtalización política que no puede ser aplicado en forma sencilla a zonificación de unidades de competencia policial, o zonas escolares) y a las complejidades inherentes al problema en sí. En este contexto, las posibilidades derivadas de un *TFG* de acceso público que permita una aproximación a la solución del problema no deben ser desechadas ligeramente.

En el contexto de nuestro desempeño laboral en el *Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires*, nos hemos encontrado en múltiples ocasiones con la necesidad de construir regionalizaciones para definir áreas de trabajo de la agencia policial, ya sea a nivel de micro-zonas para el despliegue táctico operativo policial como a nivel de macro-zonas para la gestión de la agencia policial. En todos esos casos la construcción fue realizada en forma manual "balanceando visualmente" las variables elegidas en un Sistema de Información Geográfica (SIG), en un laborioso proceso, costoso en tiempo, recursos humanos y sin una adecuada justificación de la calidad de la solución final presentada.

En particular, en el marco del *Departamento de Planeamiento y Mapeo Criminal de la Superintendencia de Investigaciones en Función Judicial del Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires*[4], se nos ha solicitado la provisión de una metodología de districtalización que pueda ser aplicada en el contexto de las necesidades institucionales, que admita una evaluación y justificación objetiva y libere al analista de una interacción permanente en el proceso de solución.

[1] También denominados problemas de districtalización o zonificación.
[2] O más técnicamente una *meta-heurística.*
[3] A la fecha la única solución gratuita y de código libre para zonificación (o redistrictalización) es el software BARD (*Better Automated Redistricting*) de Micah Altman y Michael P McDonald dentro del "*Project-R for Statistical computing*". Ver por ejemplo http://cran-r.c3sl.ufpr.br/ [Internet, fecha de último acceso enero 2011]
[4] Ver http://www.mseg.gba.gov.ar/mjysseg/Superintendencia/superintendencia.html [Internet, fecha de último acceso, enero 2011]

Como hemos mencionado, usualmente cuando se requiere definir zonas o distritos en forma computarizada se suele emplear *Sistemas de Información Geográfica* (*SIG*) y en forma manual intentar "balancear" los criterios pretendidos, o en el mejor de los casos, hacer uso de alguna función incorporada de districtalización que por lo general permite balancear un único criterio[5]. Es casi trivial notar que las probabilidades de que una solución obtenida manualmente intentando balancear unidades espaciales para construir distritos o zonas en aplicaciones de la vida real sea una solución óptima son esencialmente nulas, por lo que cobra particular importancia la necesidad de contar con un método automatizado para encontrar soluciones óptimas o cuasi-optimas en forma costo-efectiva.

La idea para este trabajo surgió de la lectura del documento ***A tabu search heuristic and adaptive memory procedure for political districting*** de Burcin Bozkaya, Erhan Erkut y Gilbert Laporte (2.003), publicado en el *European Journal of Operational Research*. En ese momento, hace ya unos años, estábamos investigando métodos automatizados que nos permitieran definir unidades territoriales para la gestión educativa, luego pudimos ver el mismo problema aplicado al despliegue territorial de agencias policiales, y en esencia a cualquier problema que pretenda la "repartición" del espacio optimizando un conjunto de criterios pre-establecidos.-

Es importante resaltar que es nuestra intención replicare implementar en la medida de lo posible la solución presentada por Bozkaya *et. al.* (2.003) *y* testearla utilizando datos reales provenientes de los radios censales del *Partido de La Plata de la Provincia de Buenos Aires*, *disponibilizando* un mecanismo útil para la resolución de este tipo de problemas a todos aquellos interesados. La elección del Partido de La Plata es consecuencia del mencionado pedido efectuado por el Departamento de Planeamiento y Mapeo Criminal[6] del Ministerio de Justicia y Seguridad Provincial, aunque a los fines de testeo de parámetros haremos uso de datos correspondientes al Partido de Berisso, dado su menor tamaño aunque de características similares al de La Plata[7].

Ahora bien, como mencionan Micah Altman y Michael P. McDonald (2.009) a pesar de una larga historia de experimentos en redistrictalización automática, no son muchas las herramientas accesibles para realizar esta función, entre ellas podemos mencionar rápidamente, los productos de:

[5] Ver por ejemplo en www.manifold.net[Internet, fecha de último acceso enero 2011]el manual de usuario del Software SIG *manifold* y consultar la categoría *districting*.
[6] La solicitud fue realizada por la Comisario Inspector (Prof.) Lic. Marisa A. Paviscov, Jefa del Departamento de Planeamiento y Mapeo Criminal.
[7] De hecho, los partidos de La Plata, Berisso y Ensenada forman un aglomerado urbano denominado "*Gran La Plata*".

1- ArcBridge[8].
2- Caliper Corporation[9] (Maptitude for districting).
3- City Gate GIS[10].
4- Corona Solutions[11].
5- Digital Engineering Corporation (DEC ©)[12].
6- Environmental Systems Research Institute, ESRI ©[13].
7- Manifold Systems[14].

En líneas generales, todas estas herramientas presentan severas limitaciones en distintos aspectos: a) Son capaces de producir solamente distritos utilizando un único criterio de igualdad de población[15] y no tienen forma de optimizar una función multicriterio general. b) Asimismo, usualmente todos los softwares que proveen mecanismos de automatización emplean alguna variante de heurísticas de ascenso de pendiente máxima (*steepest ascent*), con la notable excepción de *GeoBalance* de *Corona Solutions* que menciona emplea un algoritmo genético de criterio único.

Otros atributos compartidos por la mayoría de estas herramientas es la poca o nula documentación disponible sobre sus algoritmos (en todos los casos se trata de software propietario) y su costo generalmente de varios miles de dólares (a excepción de *Manifold,* cuyo costo promedio es de u$s 400, dependiendo del tipo de licencia, el sistema operativo (32 o 64 bits) y si se han adquirido o no otras extensiones), y la imposibilidad de utilizar diferentes medidas de compacidad.

[8] http://www.arcbridge.com/DISTRICTSolv.htm [Internet, fecha de último acceso enero 2.011]. Con su producto DistrictSolve.
[9] http://www.caliper.com/mtredist.htm Disponible como producto *stand-alone* y como extensión para el SIG ArcGIS de ESRI ©. [Internet, fecha de último acceso enero 2011]. Tanto la aplicación *stand-alone* como la extensión para ArcGIS tienen un costo de u$s 3.500 dólares por licencia monousuario, su énfasis es para redistrictalización en los EEUU y el costo es por condado, debiéndose agregar u$s 500 por estado que se agregue.
[10] http://www.citygategis.com/autobound9.htm [Internet, fecha de último acceso enero 2.011], tiene su extensión *Autobound Redistricting* para ArcGIS ©.
[11] http://www.coronasolutions.com/products/geobalance.shtml[Internet, fecha de último acceso enero 2.011]. Con su producto *GeoBalance* que tiene un costo de u$s 2.500 por licencia, y su énfasis de utilización en agencias de protección de la ley (policías).
[12] También provee una extensión para ArcGIS, al igual que programación sobre la plataforma ArcGIS (ver por ejemplo http://proceedings.esri.com/library/userconf/proc96/TO100/PAP088/P88.HTM [Internet, fecha de último acceso enero 2.011]
[13] http://www.esri.com/software/arcgis/extensions/districting/index.html, [Internet, fecha de último acceso enero 2011]se trata de una extensión gratuita para el SIG ArcGIS aunque no posee funciones automáticas de districtalización, sino que permite realizar "manualmente" la optimización de los criterios a utilizar (el costo de una licencia de ArcGIS es superior a los u$s 3.000 dependiendo del tipo de producto, (ArcView, ArcEditor o ArcInfo), llegando a u$s 10.000 para una licencia monousuario de ArcInfo). También está disponible una aplicación web paga (alrededor de u$s 4.500 al año) para el trabajo colaborativo, pero no posee funciones de districtalización automatizadas.
[14] www.manifold.net [Internet, fecha de último acceso enero 2.011].
[15] O en rigor son capaces de balancear una única variable, mientras respetan la contigüidad de las unidades espaciales básicas.

Es decir, las opciones generalmente disponibles son costosas, poco documentadas, y con un fuerte énfasis en aplicaciones locales de los EEUU, lo que seriamente dificulta su utilización en nuestro entorno local.

1- Descripción del problema en estudio y Justificación de este trabajo

En los problemas de districtalización, regionalización o zonificación, el objetivo es dividir el territorio en distritos (regiones, zonas) en base a ciertas restricciones. Los ejemplos clásicos de este tipo de problemas incluyen los distritos o secciones electorales, las zonas escolares, los límites de comisarías y sectores policiales, regiones de ventas, regiones sanitarias, etc.

Dependiendo el tipo particular de problema del que se trate, es posible encontrar diferentes juegos de restricciones aplicables para la definición de lo que configura un distrito aceptable, incluyendo igualdad de población; de problemática; respeto a fronteras naturales (como un cuerpo de agua o un río), a límites administrativos, a fronteras físicas (como una vía de tren o una autopista o carretera); homogeneidad socio-económica; similitud al actual esquema de distritos; respeto a la integridad de las comunidades (evitando partirlas en varios distritos); igualdad de representación (en el caso de los distritos o secciones electorales); compacidad; contigüidad; conectividad; etc.

Es importante notar que gran parte de la investigación sobre el tema se centra en la redistrictalización política en base a los requerimientos constitucionales de los Estados Unidos de América y a diversos fallos de la Suprema Corte de Justicia de ese país en relación a los criterios y metodologías a emplear, por lo que usualmente buena parte de los criterios a utilizar y de los ejemplos disponibles se centran en casos de los EEUU.

La districtalización se basa en un proceso de **optimización espacial** (Nelson, 2.000)[16] donde estamos buscando reducir C zonas originales en D zonas resultantes, con la restricción que cada una de las zonas originales pertenezca a una y solo una de las D zonas y que todos los miembros de las zonas resultantes (que llamaremos Distritos) estén conectados. Es por lo tanto un problema de optimización con restricciones, no-lineal y entero, que puede ser resuelto solamente mediante el empleo de métodos heurísticos[17].

Matemáticamente y en forma simplificada, este problema es una clase especial de los denominados problemas de *optimización combinatoria* [18] en los que el conjunto de

[16] Nelson, A., 2000. Activity 2 . 2 - Data reduction and low dimensional representation of complex spatial databases. *CIAT*, pp.1-26.

[17] Ver más adelante consideraciones y demostraciones.

[18] En estos problemas el objetivo es encontrar el máximo (o el mínimo) de una determinada función sobre un conjunto finito de soluciones que denotaremos por *S*. No se exige ninguna condición o propiedad sobre la función objetivo o la definición del conjunto *S*. Es importante notar que dada la finitud de *S*, las variables han de ser discretas, restringiendo su dominio a una serie finita de valores. Habitualmente, el número de elementos de *S* es muy elevado, haciendo impracticable la evaluación de todas sus soluciones para determinar el óptimo.

parámetros desconocidos es el sistema de districtalización. En esencia el problema se define como:

Optimizar f(z)

Donde **f(z)** es alguna función predefinida de los datos a nivel de zona creados por el operador de agregación espacial **z** al ser aplicado sobre un conjunto de **C** zonas más pequeñas. El problema es complicado dado que existen restricciones sobre **z**: por ejemplo, cada una de las **C** zonas más pequeñas puede ser asignada a una y sólo una de las **D** zonas más grandes en **z**, cada zona pequeña debe ser asignada a una zona grande, y todas las zonas pequeñas asignadas a la misma zona grande deben estar internamente conectadas.

Ahora bien, y adelantándonos a la discusión:

Es justamente la restricción de contigüidad la que agrega una complejidad única a este problema de optimización.

Siguiendo a Macmillan y Pierce (1.994), es posible primero describir la teoría de arreglar **C** unidades base(de aquí en adelante abreviadas como UB) en **D** distritos utilizando el vector:

(I.1) $$x \equiv (x_1^1, \dots, x_C^1; \dots, x_1^D, \dots, x_C^D)$$

Donde para i= 1,…,C y d = 1,…,D

(I.2) $$\left(x_i^d = 1\right) \rightarrow (la\ UB\ i\ esta\ en\ la\ region\ d);$$

(I.3) $$\left(x_i^d = 0\right) \rightarrow (la\ UB\ i\ NO\ esta\ en\ la\ region\ d);$$

Un plan de regionalización se dice factible si satisface las condiciones:

(I.4) $\sum_i x_i^d \geq 1 \qquad \forall d$

(I.5) $\sum_d x_i^d = 1 \qquad \forall i$

(I.6) $x_i^d = 0 \ \ o \ \ 1 \qquad \forall id$

La condición (I.4) implica que todo distrito contiene por lo menos una unidad base, mientras que las condiciones (I.5) e (I.6) aseguran en conjunto que cada unidad base esté en uno y solo un distrito.

Se trata en este caso de un modelo de optimización cuadrático y entero relativamente simple, donde una función posible podría ser la igualdad de población por zona. Como se dijo, las condiciones anteriores aseguran que cada uno de los **D** distritos contenga por lo menos una de las **C** unidades base, y que cada una de las **C** unidades base pertenezca a uno y solo uno de los **D** distritos.

La naturaleza entera del problema ayuda a explicar la variedad de aproximaciones que han sido adaptadas para la producción de planes de redistrictalización. Consideremos por ejemplo un problema en el que hay que asignar cuatro departamentos a dos distritos. La Figura 1, tomada de *Macmillan y Pierce* (1.994) muestra la representación del espacio de soluciones del problema como un árbol. La línea inferior de la figura contiene todas las posibles distribuciones. Obviamente no todas las distribuciones respetaran la contigüidad de las unidades base y dependerán de las relaciones de vecindad del mapa de UBs. La figura muestra dos mapas, las distribuciones contiguas asociadas con cada mapa se etiquetan como "**Co**" mientras que las no-contiguas se etiquetan como "**NC**". La figura 1 muestra asimismo dos algoritmos populares para resolver el problema:

a) Uno basado en un procedimiento de sembrado, en el que una UB semilla se selecciona para cada distrito y luego los distritos se "hacen crecer" al agregarles UBs a las semillas hasta que ya no queden UBs para asignar. Esta estrategia se representa en la figura por la selección de la semilla (1,2), es decir, por la asignación inicial de la Unidad Base 1 al Distrito 1 y de la Unidad Base 2 al Distrito 2. Todas las posibles asignaciones de las UBs remanentes se representan por las cajas de la fila inferior de la Figura 1 que están conectadas con (1,2) por debajo (ver las líneas gruesas). El proceso de crecimiento guía el procedimiento de solución a lo largo de la red y hacia abajo (en un camino hacia abajo a partir de la semilla), pero no en forma óptima. Es evidente que la estrategia empleada sufre por el hecho que busca en solamente una parte del espacio de soluciones por lo que no hay garantía de que se encuentre un mínimo global en la desviación de la población. Es más, las reglas para el proceso de crecimiento hacen bastante improbable que se llegue incluso a obtener un mínimo local… pero obviamente la virtud de esta estrategia es que por lo menos permite obtener una solución factible (si bien no necesariamente óptima como hemos dicho).

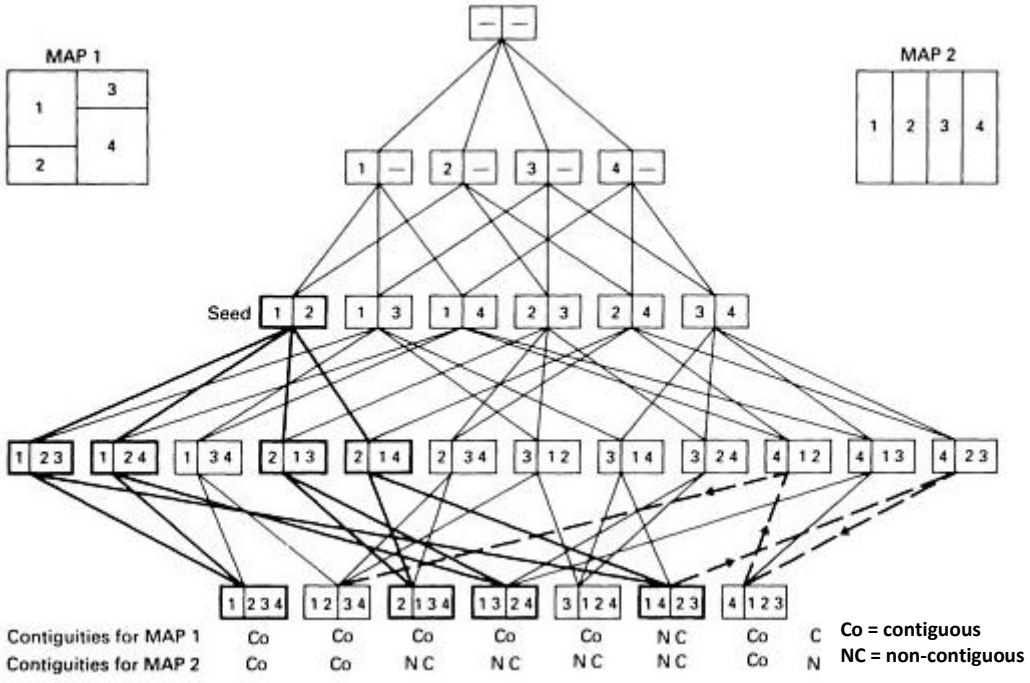


Figura 1 – Tomada de *Macmillan y Pierce* (1.994) Árbol de soluciones para el problema de districtalización

Tabla I.1 – Ejemplo de Poblaciones por departamento para la figura 1

Unidad Base	Población
1	6
2	5
3	4
4	5

Notemos que si consideramos el "Mapa 1" de la figura 1, y partimos de la semilla (1,2), entonces esa semilla conduce a cuatro (04) posibles configuraciones para los Distritos (señaladas con línea gruesa), de esas soluciones, tres (03) respetan el criterio de contigüidad:

Tabla I.2. – Configuraciones de Distritos y Población de los mismos para el proceso basado en sembrado de la Figura 1, Mapa 1, Semilla (1,2)

Distrito 1		**Distrito 2**	
UBs Componentes	**Población del Distrito**	**UBs Componentes**	**Población del Distrito**
1	6	2, 3, 4	14
1, 3, 4	15	2	5
1, 3	10	2, 4	10
1, 4	11	2, 3	9

(el *grisado* corresponde a la solución **no contigua**)

Por lo tanto, vemos que con este procedimiento y esa semilla, podemos obtener cuatro (04) soluciones, tres de las cuales son factibles (en cuanto a contigüidad), y se incluye la solución óptima (1,3 / 2, 4) entre las soluciones "alcanzables" a partir de esa semilla. Ahora bien, recordemos que en este tipo de problemas en general no podemos recorrer todo el espacio de soluciones y que en esta instancia particular estamos hablando de la solución a la que se arriba al recorrer hacia abajo la Figura 1. Por otro lado, si hubiéramos elegido la semilla (1,3) ninguna de las soluciones "alcanzables" incluiría la óptima. El punto es, justamente, que el proceso de agregación no asegura que al terminar el mismo (al llegar a la última fila), la solución hallada sea óptima, y de hecho es posible que ninguna de las soluciones a las que se pueda llegar a partir de una semilla dada sea óptima. Es decir, mientras que la Probabilidad de encontrar la solución óptima con este procedimiento para la Figura 1, Mapa 1 y Semilla (1,2) es de 0,25 (o un 25 %), para la Figura 1, Mapa 1 y Semilla (1,3) es directamente 0.Obviamente, en este sencillo caso, es posible calcular la probabilidad que eligiendo al azar una dada semilla lleguemos a la solución óptima, que es esencialmente pequeña (alrededor del 14 %).

b) La segunda estrategia involucra una optimización e intercambio explícito. Es decir, iniciando a partir de un plan completo (de una caja de la fila inferior de la figura 1), una UB es intercambiada de un distrito al otro de manera tal que la desviación cuadrática media se reduce a la mayor velocidad posible. El intercambio continúa hasta que se encuentra el *mejor* plan (ver las líneas punteadas que inician del plan (1 4, 2 3). Ahora bien, esta estrategia tiene también limitaciones. Por ejemplo, si las Unidades Base están conectadas como en el Mapa 1 de la Figura 1, y la población de las mismas es la de la Tabla I.1, entonces el plan (1 2, 3 4) será sub-óptimo pero inhibirá cualquier intercambio de UB posterior, dado que cualquier cambio (la liberación de una UB de un distrito y su anexión a otro) llevará a un deterioro en la desviación cuadrática media de la población.

Ahora bien, esta discusión no ha tenido en cuenta hasta ahora el problema de la **contigüidad**. El deseo de producir planes contiguos crea un problema verdaderamente formidable. *Informalmente, un plan contiguo es aquel en el que cada Unidad Base en un distrito está conectada con todas las otras Unidades Base vía Unidades Base que también están en el distrito. O dicho de otra forma, un plan es contiguo si ningún distrito tiene Unidades Base "islas" o grupos de Unidades Base separadas del cuerpo principal del distrito*.

La idea de contigüidad puede ser formalizada de diversas maneras, una es considerar una restricción de contigüidad que involucra potencias de la matriz de conectividad del problema. Una matriz de conectividad **k** tiene elementos $\boldsymbol{k}_{ij}$ que son **1** si i y j están simplemente conectadas (dos UB comparten una frontera) y **0** en cualquier otro caso. La matriz de conectividad para un distrito *d* consiste de elementos $k_{ij}x_i^d x_j^d$. Por lo tanto, si los departamentos i y j están ambos en *d* y se tocan (comparten una frontera), el elemento *ij* de la matriz de conectividad del distrito será 1. En cualquier otro caso, será 0.

Para asegurarse que las Unidades Base de un distrito de **N** Unidades Base son contiguas es suficiente comprobar que la **(N-1)-esima** potencia de esta matriz no tiene elementos nulos.

(I.7) $k_{ij}^{d1} \equiv k_{ij}x_i^d x_j^d$

(I.8) $k_{ij}^{d2} \equiv \sum_n k_{in}^{d1} k_{nj}^{d1}$

(I.9) $k_{ij}^{d3} \equiv \sum_n k_{in}^{d2} k_{nj}^{d2}$

⋮

(I.10) $k_{ij}^{d(N-1)} \equiv \sum_n k_{in}^{d(N-2)} k_{nj}^{d1}$

Es así que la condición de contigüidad toma la forma:

(I.11) $k_{ij}^{d(N-1)} - x_i^d x_j^d \geq 0 \qquad \forall ij$

Por lo tanto, con la restricción de contigüidad, el problema de optimización anterior se convierte en **Optimizar f**, sujeto a:

(I.12) $\sum_i x_i^d \geq 1 \qquad \forall d$

(I.13) $\sum_d x_i^d = 1 \qquad \forall i$

(I.14) $k_{ij}^{d1} \equiv k_{ij} x_i^d x_j^d \qquad \forall ijd$

(I.15) $k_{ij}^{d2} \equiv \sum_n k_{in}^{d1} k_{nj}^{d1} \qquad \forall ijd$

(I.16) $k_{ij}^{d3} \equiv \sum_n k_{in}^{d2} k_{nj}^{d1} \qquad \forall ijd$

⋮

(I.17) $k_{ij}^{d(N-M+1)} \equiv \sum_n k_{in}^{d(N-M)} k_{nj}^{d1} \qquad \forall ijd$

(I.18) $x_i^d = 0 \quad o \quad 1 \qquad \forall id$

Lo que pone al problema fuera del terreno de los métodos de optimización convencionales. La complejidad del problema surge en parte de su tamaño, tanto en término de su dimensionalidad como del número de restricciones, pero principalmente surge del hecho que el espacio definido por las restricciones es no-convexo. Es así que mientras que la región factible en el Problema original sería convexa sin la restricción entera (I.6), la remoción de esta restricción en el Problema con contigüidad (I.18) todavía deja una región no convexa en la que la función objetivo es probable que alcance numerosos valores óptimos no globales (locales). Es justamente esta característica de la región factible la que dificulta en extremo la resolución del problema de districtalización.

Antes de continuar con la exposición en relación a la solución del problema, es importante realizar algunos comentarios sobre las restricciones de contigüidad dado que cualquier algoritmo aceptable de redistrictalización *debe ser capaz de entregar planes contiguos*:

> Asumiendo que hay en total **C** unidades base, el número máximo de unidades base en cualquier distrito es C – D + 1, donde **D** es la cantidad de distritos. Los controles de contigüidad que hemos mencionado están sobre-especificados en el sentido que todos menos uno de los distritos en todas las iteraciones tendrá, necesariamente, menos del máximo de departamentos (C-D+1). Por lo tanto la mayor potencia a la que debe elevarse la matriz de conectividad del distrito es N-1, donde N es el número de unidades base departamentos en el distrito. Aunque de hecho, una potencia menor puede ser suficiente dado que la contigüidad del

distrito se establece tan pronto como los términos en la M-ésima potencia de la matriz es no nula, donde M es cualquier entero positivo menor o igual que N. Por lo tanto cierto grado de eficiencia computacional puede ser obtenido al relegar el chequeo de contigüidad a una restricción sobre la operación del algoritmo. Esto es, puede realizarse un control en cada iteración para asegurarse que el nuevo plan es contiguo. Por supuesto, esto no evita el problema de la existencia de óptimos locales no globales, pero si provee una forma de mejorar cualquier algoritmo de búsqueda de la solución óptima global. Ahora bien, es posible hacer mucho más que esto, una vez que se ha decidido relegar el control de contigüidad a una regla en el algoritmo, si se utiliza una solución factible y contigua inicial, es posible asegurar que el procedimiento de búsqueda estará restringido a soluciones contiguas y factibles mediante un simple control de contigüidad basado en observaciones topológicas. Veamos esto con un ejemplo[19]:

Dados C = 5 y D = 2 (*i.e.*5 UBs y 2 Distritos):

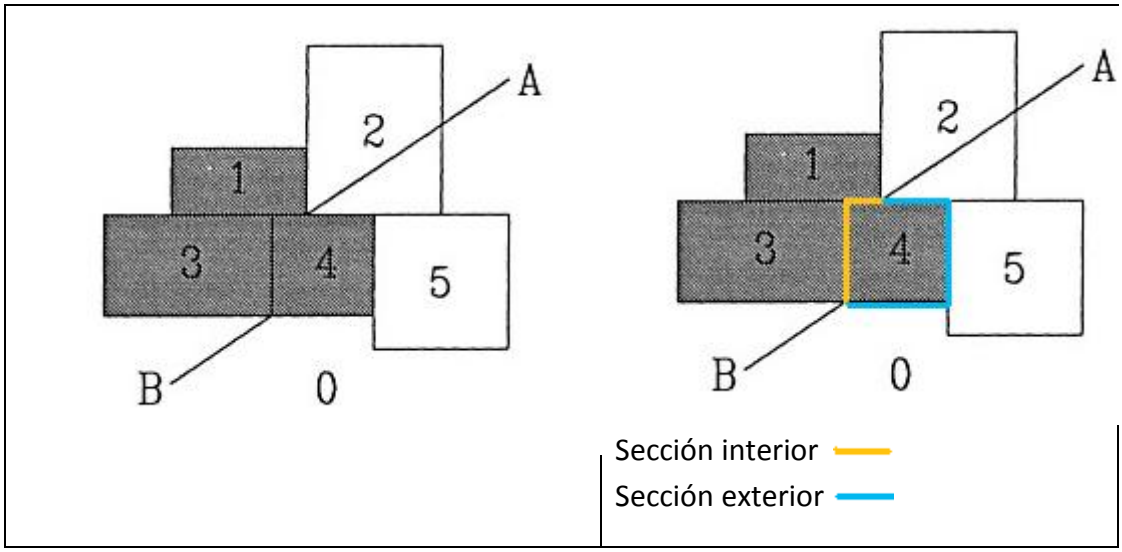


Figura 2 – Tomada y adaptada de Macmillan y Pierce 1.994

Supongamos que la UB 4 ha sido seleccionada para abandonar el distrito sombreado (distrito *d*). Puede apreciarse que la UB 4 puede abandonar el distrito sin pérdida de contigüidad dado que la frontera de 4 tiene solamente dos secciones: una sección interior a lo largo de la cual 4 está unida a otras UBs en el distrito; y una sección exterior a lo largo de la cual existen UBs que pertenecen a otros distritos (o al resto del mundo, representado por 0 en la figura). En la medida que la frontera tiene dos secciones, hay solamente dos puntos a lo largo de la misma en los que existe un cambio en el tipo de frontera (interior a exterior o viceversa). Comenzando por el punto A en la frontera y recorriendo en sentido

[19] Tomado de Macmillan y Pierce (1.994).

contrario a las agujas del reloj, es posible contar el número de puntos de inversión. Generalmente, una juntura en la frontera es un punto de inversión si la pertenencia al distrito en las dos UBs que se encuentran en la juntura cambia de "es la misma que la de la UB objeto" a "no es la misma de la UB objeto" o viceversa. Por lo tanto, en este ejemplo, el primer punto de inversión es el punto B dado que la UB 3 está en el mismo distrito que la UB 4, pero la "Unidad Base" 0 no está en el mismo distrito que la UB 4. El segundo punto de inversión es el punto A. Algebraicamente, para la figura 2, el número de puntos de inversión está dado por:

$$(x_1^d - x_3^d)^2 + (x_3^d - x_0^d)^2 + (x_0^d - x_5^d)^2 + (x_5^d - x_2^d)^2 + (x_2^d - x_1^d)^2$$

Por lo que el control de contigüidad es que la UB 4 puede ser separada del distrito sombreado siempre y cuando:

$$(x_1^d - x_3^d)^2 + (x_3^d - x_0^d)^2 + (x_0^d - x_5^d)^2 + (x_5^d - x_2^d)^2 + (x_2^d - x_1^d)^2 = 2$$

Ahora bien, si estuviéramos en la situación de la figura 3:

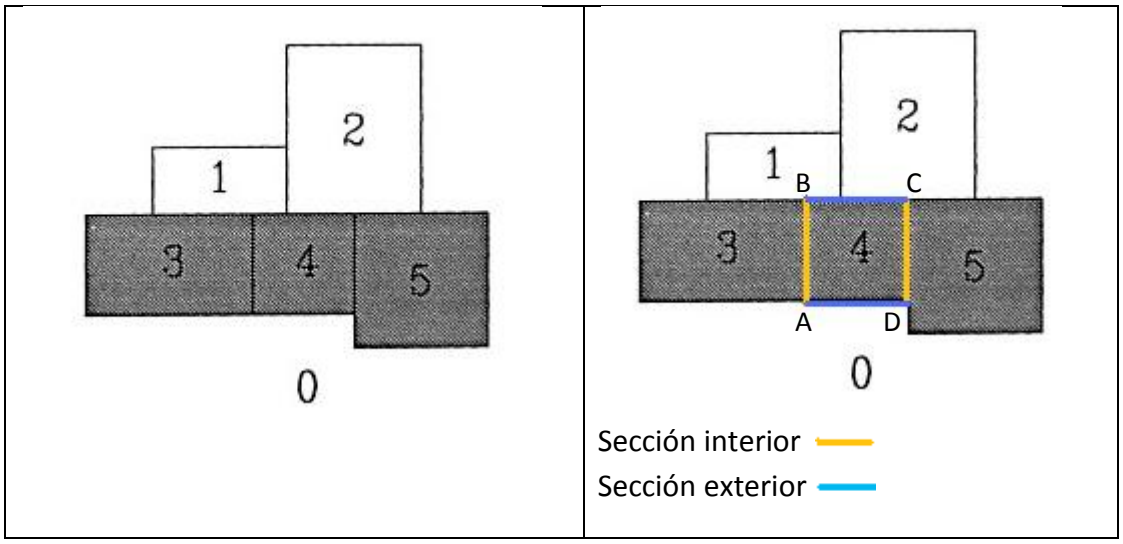


Figura 3 – Tomada y adaptada de Macmillan y Pierce 1.994

es fácil comprobar que el número de puntos de inversión es 4, por lo que la UB 4 **no** puede ser separada del distrito sombreado sin que exista perdida de contigüidad. Para una UB completamente rodeada por UBs del mismo distrito, el número de puntos de inversión será siempre 0, por lo que el control de contigüidad prevendrá nuevamente su remoción. Ahora bien, el tema no es tan simple, dado que existen situaciones en las que la estructura topológica de los distritos permite el cambio aun cuando los puntos de inversión sean 4. El punto no es tanto la restricción topológica en sí, sino entender que la forma de la restricción

va a variar conforme la topología del sistema de Unidades Base, pero que si se trata la restricción de contigüidad como una regla del algoritmo es posible entonces volver al problema original sin perder de vista que la no-linealidad sigue estando presente.

Se han sugerido diversos métodos para encontrar distritos óptimos (es decir Planes de Districtalización Óptimos). Básicamente podemos englobarlos en dos categorías amplias (Altman, 1997): *métodos exactos* y *métodos heurísticos*.

- *Limitaciones de los Métodos Exactos*: sistemáticamente examinan todos los distritos posibles, ya sea en forma explícita o implícita. Los métodos explícitos o de búsqueda por "fuerza bruta" literalmente evalúan cada distrito. Otros métodos más sofisticados como los de enumeración implícita, o las técnicas "*branch and bound*"[20] o "*branch and cut*"[21] excluyen clases de soluciones que pueden ser inferidas como sub-óptimas sin necesitar un examen explícito. Para encontrar los distritos óptimos en este caso solamente se requiere ordenar la lista de puntajes de los mismos y obviamente estos métodos funcionan con problemas muy pequeños de districtalización.
  Ahora bien no se ha encontrado aún un método exacto que resuelva el problema para un plan de tamaño razonable[22].

- *Limitaciones a los métodos heurísticos:* los procedimientos heurísticos utilizan una variedad de métodos para estructurar la búsqueda de planes de districtalización con puntajes elevados. Ninguno garantiza convergencia al plan óptimo en tiempo finito, y dentro de sus posibilidades se trata de "buenas conjeturas". Todas las heurísticas citadas en la bibliografía conocida se centran en realizar *mejoras iterativas* a un plan de districtalización propuesto.
  La principal queja con cualquier heurística es que en su corazón no son otra cosa que buenas conjeturas.

Ahora bien, existen diferentes estrategias heurísticas de solución propuestas para estos problemas, las tres más comunes incluyen el método de Monte Carlo, la **Búsqueda Tabú** y el "*simulated annealing*" (recocido o templado simulado)[23]. Tal como menciona Rafael Martí (2.003), hay otros motivos posibles para emplear métodos heurísticos además de que se trate de un problema difícil, por ejemplo: (1) El problema es de una naturaleza tal que no se conoce ningún método exacto para su resolución. (2) Aunque existe un método exacto para resolver

[20] Ramificación y acotación.
[21] Ramificación y corte.
[22] Ver el Anexo I para la demostración de Altman en relación a la imposibilidad de existencia de soluciones exactas que funcionen para planes reales de tamaño razonable.
[23] De hecho Openshaw y Rao(1994), muestran justamente estos tres métodos partiendo de un trabajo original de la década de 1970 de Openshaw con su algoritmo AZP (Automatic Zoning Procedure (AZP) of Openshaw (1977, 1978a, 1978b))

el problema, su uso es computacionalmente muy costoso. (3) El método heurístico es más flexible que un método exacto, permitiendo, por ejemplo, la incorporación de condiciones de difícil modelización. (4) El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema. En este caso, existen dos posibilidades: - El método heurístico proporciona una buena solución inicial de partida. - El método heurístico participa en un paso intermedio del procedimiento, como por ejemplo las reglas de selección de la variable a entrar en la base en el método Simplex.

En este punto hay que enfatizar el tema no menor que **este tipo de problemas no puede ser atacado por algoritmos de "fuerza bruta"**, dado el tamaño del espacio de soluciones. Sin incluir restricciones de contigüidad, el número total de planes distintos *S(C,D)*[24]que pueden ser creados utilizando **C** Unidades Base para construir **D** distritos está caracterizado por[25]:

(I.19) $$S(C,D) = \frac{1}{D!}\sum_{i=0}^{D}(-1)^i \left(\frac{D!}{(D-i)!i!}\right)(D-i)^C$$

y aún bajo la presunción que cada distrito está compuesto de exactamente **k** bloques (y por lo tanto D = C/k), el número de planes posibles es una función rápidamente creciente:

(I.20) $$S'(C,D,k) = \frac{C!}{k!(D!)^k}$$

La magnitud del problema no suele ser generalmente reconocida, dado que aun para una pequeña cantidad de unidades base y distritos, el número posible de arreglos de distritos es enorme. Por ejemplo, el número de planes posibles que se pueden utilizar para dividir una región *R* en dos distritos es:

Tabla I.3 – Cantidad de planes posibles si D = 2

Número total de Unidades Base (C)	Ejemplo del tipo de Unidades Base	Cantidad de UBs por Distrito (k)	Cantidad de Planes posibles *S'(C,D,k)*
10	Departamentos	5	945
50	Fracciones Censales	25	5.8×10^{31}
250	Radios Censales	125	4×10^{245}

Como esta tabla ejemplifica, el tamaño del problema de redistrictalización crece rápidamente en la medida que lo hacen las unidades constructivas (las UBs), y notemos que por ejemplo, para el caso de C = 12, D = 2, k = 6, tenemos 10.399 planes posibles, y para C = 14, D = 2, k = 7, se tienen 135.135 planes posibles, como se ve, el crecimiento es *extremadamente rápido*.

[24] El número de Stirling de segunda especie, *S(C,D)* o $\left\{ {C \atop D} \right\}$se define como la cantidad de maneras que existen de hacer una partición de un conjunto de C elementos en D grupos.
[25] Ver Altman (1997).

Ahora bien, la tabla anterior minimiza el problema dado que asume que todos los distritos tienen la misma cantidad de bloques (en todos los casos k = C/D). Por otro lado, la cantidad de distritos a construir (**D**) es también un factor importante al determinar la cantidad de planes posibles. La cantidad de planes posibles crecerá con **D** hasta un punto a partir del cual comenzara a decrecer. Por ejemplo, si consideramos 24 bloques de construcción para construir **D** distritos, tenemos:

Tabla I.4. Cantidad de Planes y Distritos cuando C = 24

Cantidad de Distritos	UBs por Distrito	Cantidad de Planes
1	24	1
2	12	3.2×10^{11}
3	8	9.2×10^{12}
4	6	4.5×10^{12}
6	4	$9{,}6 \times 10^{10}$
8	3	1.6×10^{9}
12	2	1.3×10^{6}
24	1	1

Puede verse que la naturaleza combinatoria del problema sugiere que aún con una única y sencilla función objetivo, el problema puede ser muy difícil (sino imposible[26]) de resolver en forma óptima, si el tamaño es lo suficientemente grande.

Asimismo, dado la naturaleza combinatoria del problema, la mayoría de los algoritmos de solución propuestos son como ya hemos dicho aproximaciones heurísticas que buscan soluciones cuasi-optimas.

[26]Nuevamente ver Altman (1.997).

Objetivos Generales:

En base a lo expuesto precedentemente, hemos decidido centrar este trabajo en esencialmente los siguientes objetivos:

1- **Estudiar la posibilidad de proveer una herramienta informática al Departamento de Planeamiento y Mapeo Criminal del Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires, para calcular una solución de "*buena calidad*" al problema de la districtalización. Definiendo la calidad de la solución en base al valor de la función objetivo que se minimiza y las propiedades del método de solución utilizado;**
2- **Determinar los criterios aplicables a una función objetivo a ser minimizada en un proceso de districtalización (es decir determinar los criterios específicos para la districtalización), y su implementación algorítmica en el contexto del objetivo anterior;**
3- **Realizar un análisis general de la complejidad computacional[27] del problema**;
4- **Definir y ajustar el algoritmo de solución por búsqueda tabú, su implementación informática (software) y el testeo de los parámetros fundamentales utilizando datos reales siguiendo los lineamientos del trabajo de Bozkaya et. al (2003);**
5- **Determinar la aplicabilidad de la solución a problemas equivalentes en el ámbito de la Provincia de Buenos Aires[28]**.

En esencia el objetivo es seguir en la medida de lo posible el trabajo de Bozkaya et. al (2003), ajustando el algoritmo y la implementación a datos de la Provincia de Buenos Aires, lo que permitirá su eventual utilización rápida en problemas de districtalización en nuestro entorno local (provincial y nacional).

En este punto enfatizamos que la elección de de la solución por búsqueda tabú en lugar de estrategias de "*simulated annealing"*o algoritmos genéticos surge fundamentalmente del hecho que mientras que estos dos últimos se basan exclusivamente en principios de procesos biológicos o físicos, la filosofía de la *búsqueda tabú es derivar y explotar una colección de principios de resolución inteligente de problemas[29]. En este sentido, puede decirse que la búsqueda tabú se basa en conceptos seleccionados que unifican los campos de la inteligencia artificial y la optimización*. Una característica distintiva de la búsqueda tabú, representada por su explotación de formas adaptativas de memoria, la equipa para penetrar

[27] Especialmente como justificación al empleo de *meta-heurísticas.*-
[28]Los datos a utilizar para el testeo provienen del censo nacional de población, hogares y vivienda 2.001, y la metodología empleada para el ajuste del algoritmo se ha descripto lo suficiente como para permitir su replicabilidad casi inmediata.-
[29]Glover y Laguna (1997), pág. 1.

complejidades que muchas veces confunden a otras aproximaciones alternativas (como el ya mencionado *simulated annealing*[30]).

La búsqueda tabú hace uso de técnicas que combinan la inteligencia artificial con la optimización, y su metodología se basa principalmente en los principios de la búsqueda local, pero con una utilización inteligente de la memoria y con la habilidad de cruzar las fronteras de la factibilidad. Esto permite que un algoritmo de búsqueda tabú recorra el espacio de soluciones de un problema particular de forma más intensa, alcanzando soluciones que los métodos tradicionales usualmente no alcanzan. Mediante **diversificación** el algoritmo visita partes radicalmente diferentes del espacio de soluciones factibles, mientras que mediante **intensificación** se centra en las soluciones de elite y sus vecindades. Estas características junto con el uso inteligente de la memoria suelen denominarse **memoria adaptativa**, en la terminología de esta clase de algoritmos.

Es importante resaltar que siendo este un trabajo final de graduación y no una tesis doctoral no estamos proponiendo innovaciones radicales, sino que en rigor estamos intentando seguir los pasos propuestos originalmente por Bozkaya et al. (2003) y evaluar su aplicabilidad en nuestro mandante (Departamento de Planeamiento y Mapeo Criminal del Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires).

Finalmente consideramos útil remarcar que coincidiendo con Glover y Laguna (1997) nonos sentimos conceptualmente cercanos con la ola de *Nuevo Romanticismo* que impregna ciertos métodos heurísticos que se asocian con procesos encontrados en la naturaleza. Las metáforas de la naturaleza ciertamente tienen un lugar en la investigación científica, aparecen especialmente como disparadoras durante las primeras fases de una investigación. En la medida que se tenga cuidado de que esas metáforas no prevengan ciertas líneas de exploración, nos proveen de una forma de "vestir" las descripciones de varias meta-heurísticas de manera tal de satisfacer nuestros instintos de trazar paralelos entre fenómenos simples y diseños abstractos. Es fundamental estar atentos de manera tal de determinar cuando el simbolismo del *Nuevo Romanticismo* obscurece en lugar de iluminar el camino hacia un mejor entendimiento… y es probable que no siempre lo que sirve para abejas, hormigas o raíces de abetos sea igualmente funcional y óptimo para la resolución de problemas complejos.

[30] Notemos incidentalmente que las últimas implementaciones de SA apuntan a olvidarse de la analogía física y manejar la cola de enfriamiento como una estructura de memoria. Es decir, la probabilidad de aceptar o rechazar un movimiento de no mejora depende no de la iteración (tiempo transcurrido) sino de lo sucedido en la búsqueda. En este sentido, la probabilidad será función de algunas variables de estado del proceso. En la actualidad se están diseñando numerosos algoritmos híbridos en donde la búsqueda local se realiza con un procedimiento basado en SA, en ocasiones combinado con Búsqueda Tabú.

2- Fundamentación Teórica

A) Consideraciones sobre el problema de districtalización[31]

1- El Problema de la districtalización automática puede ser intratable

Para poder encontrar los distritos *óptimos* primero debemos obviamente formular el problema en términos matemáticos y resolver ese problema en forma matemática. Altman (1997) demuestra que sin importar esa formulación, el problema de districtalización es formalmente intratable desde el punto de vista computacional, es decir es prácticamente imposible resolver el problema en forma exacta.

1.1 La districtalización es un Problema Matemático de tamaño considerable:

Es posible caracterizar el problema de districtalización de diferentes formas. Una forma sencilla de "matematizar" el problema es pensarlo como un problema de *repartimiento de un conjunto*, que es la forma utilizada por Altman (1.997) en su trabajo[32]. Su demostración en relación a la *inmanejabilidad* del problema no es dependiente de la caracterización utilizada para definirlo, de hecho todas las caracterizaciones son computacionalmente equivalentes.

Si consideramos al problema como un caso de **optimización combinatoria** (que es la caracterización que estamos usando a lo largo de este trabajo), consideramos a las Unidades Base (UB) como indivisibles, de las que contamos con información completa en relación a las necesidades para el problema puntual a resolver (cantidad de votantes y demografía en el caso de districtalización política, cantidad de delitos, niveles de violencia, etc. en el caso de districtalización policial, datos de matrícula educativa en el caso de regiones educativas, etc.) El problema de districtalización es entonces **dividir** el conjunto completo de UB de manera tal que una dada **función objetivo** sea optimizada (por ejemplo, minimizada). El problema de *repartimiento* puede complicarse aún más al agregar un conjunto de restricciones sobre los distritos a formar. Estas restricciones, tales como la contigüidad, pueden limitar considerablemente el conjunto de planes viables.

El problema presenta dificultades especiales por el tamaño del conjunto de soluciones. En general resulta imposible atacar el mismo mediante una búsqueda de fuerza

[31] Altman (1997)

[32] Existen, por supuesto, otras formas que se pueden utilizar, tales como *división de grafos, disección poligonal y programación entera*.

bruta sobre todos los posibles distritos a ser formados, esto ya ha sido adecuadamente ilustrado en porciones precedentes de este trabajo.

1.1.1 Funciones objetivo candidatas

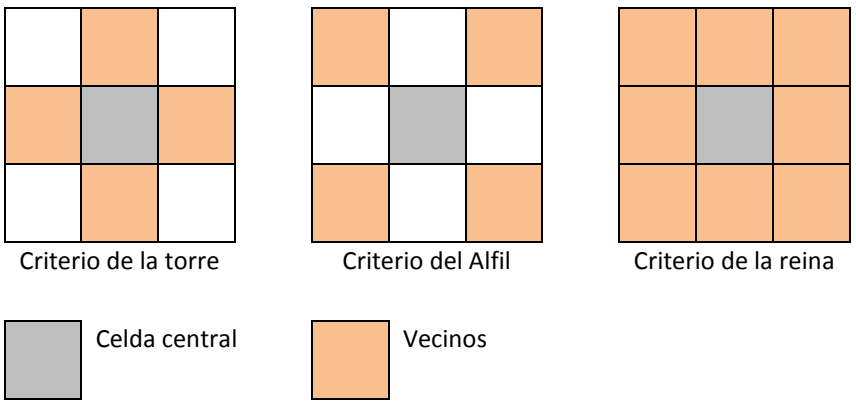
La dificultad para resolver un dado problema de districtalización dependerá en esencia de la función objetivo particular y de las restricciones que se utilicen. Algunos de los criterios comúnmente utilizados en los diferentes tipos de problemas son:

i. *Igualdad de Población*: fundamental en la districtalización electoral dado que está asociado al concepto de equidad política, pero es también fundamental en la districtalización policial, o en la planificación educativa[33].

ii. *Contigüidad:* donde entran en juego las diferentes conceptualizaciones de vecindad en el espacio, y en particular si se considerará por ejemplo el criterio "*de la reina*" o el criterio de la "*torre*"[34].

[33] En esencia en todos los casos se tiene algún tipo de población base a repartir.

[34] La forma más usual para describir la relación espacial de adyacencia o vecindad entre un conjunto de objetos espaciales es mediante el empleo de matrices de ponderación y ordenamiento espacial, generalmente denotadas con la letra **W**. En estas matrices, cada elemento w_{ij} registra la relación espacial entre los objetos. Los elementos de la matriz pueden ser de diversa naturaleza, por ejemplo si **W** se expresa en términos de distancias, los w_{ij} pueden representar la inversa de las distancias entre los objetos *i* y *j*, si operamos bajo la presunción que la interacción entre vecinos depende negativamente de la distancia (es decir decae a medida que nos alejamos). En otros casos, **W** representa una matriz de contigüidad, donde w_{ij} toma valores binarios en función de si hay contigüidad o no. Generalmente en el caso de la contigüidad hay tres variantes para identificarla, denominadas criterios de la torre, alfil o reina (cuyos nombres provienen de los movimientos de las piezas de ajedrez). La torre puede moverse entre los escaques solamente hacia los escaques que comparten una frontera de longitud positiva con el escaque actual. Por otro lado, el alfil solamente puede moverse en diagonal, es decir, utiliza un criterio de punto adyacencia, y la reina puede desplazarse hacia cualquier polígono que comparta frontera de al menos un punto. Obviamente la contigüidad de la reina puede definirse como la suma de la de la torre más el alfil, y el criterio de la torre es claramente más restrictivo en cuanto a vecindad que el de la reina:



iii. *Compacidad*: es decir cuan compacto es el distrito que se forma, y que a pesar de una aparente simplicidad encierra una muy interesante cantidad de criterios disímiles y en muchos casos incompatibles entre sí.

iv. *Integridad de Comunidades*: es decir, adherencia a ciertas unidades espaciales que se trata de no separar en la medida de lo posible.

v. *Etc.*

1.2 ¿Qué es un Problema difícil computacionalmente?

La teoría de la *Complejidad computacional* y el campo relacionado de la teoría de la *computabilidad* son dos ramas de la ciencia informática. Estas disciplinas se dedican a analizar la dificultad para la resolución de problemas discretos[35] específicos utilizando computadoras.

Investigadores de la ciencia informática y de la investigación operativa utilizan la teoría de la complejidad computacional cuando analizan problemas.

Central en la teoría de la Complejidad Computacional es la definición de *problema*. Un **problema** es una pregunta general a ser respondida. En el caso de la districtalización se trata de encontrar el plan de districtalización que optimiza nuestra función objetivo (o más formalmente, debemos encontrar la partición óptima). Altman (1997) utiliza el término *sub-problema de redistrictalización* para distinguir el caso en el que tiene una pre-especificada función objetiva particular, tal como la compacidad, en lugar de considerar el valor de la función en sí mismo como un parámetro. Por lo tanto la districtalización para optimizar la (una particular y formalmente definida medida de) igualdad poblacional, es un sub-problema.

Un dado problema posee diversos *parámetros*, o variables libres, cuyos valores no están especificados. Un problema se describe indicando (Garey & Johnson, 1979): (1) una descripción general de todos sus parámetros, y (2) una declaración de que propiedades se requiere que satisfaga la respuesta o *solución*. Para cualquier sub-problema de districtalización, los parámetros consisten de las unidades de población de las que se va a determinar el plan y el vector de valores asignados a aquellas unidades de población. Una *instancia* del problema se crea al asignar valores a todos los parámetros del mismo.

Por otro lado, un *algoritmo* es un conjunto general de instrucciones *paso a paso* en un lenguaje formal de computadora que, cuando se ejecuta, resuelve un problema

[35] Para problemas que son continuos en lugar de discretos como el que nos ocupa tiene también relevancia el campo de la *complejidad basada en información*.

especificado. Se dice que un algoritmo *resuelve* un problema si y sólo si puede ser aplicado a *cualquier* instancia de un problema y está garantizado que produzca una solución exacta para esa instancia. Por lo tanto, un algoritmo se dice que resuelve un problema de districtalización de igualdad de población solamente si está garantizado que encuentre una solución óptima para cada conjunto de unidades base que se le den.

En general, se está interesado en encontrar el algoritmo *más eficiente* para resolver un problema. Y en su sentido más amplio la noción de eficiencia involucra todos los diferentes recursos computacionales necesarios para ejecutar un algoritmo. Aun así, por algoritmo *más eficiente* generalmente se entiende el más rápido.

Finalmente, se dice que un problema es *computable* si y sólo si existe un algoritmo que resuelve el problema. Para un dado problema computable, definimos la *complejidad temporal* (o *complejidad* a secas) de un algoritmo como una función que representa el número de instrucciones que el algoritmo debe ejecutar para alcanzar la solución. La complejidad de un algoritmo se expresa en términos del *tamaño* del problema, groseramente equivalente al número de parámetros de entrada[36]. El tamaño de la districtalización es simplemente el número de UBs que se emplean como entrada. Se dice que un algoritmo tarda un *tiempo polinomial* si su *función de complejidad temporal*[37] es polinomial, y se dice que toma un *tiempo exponencial* en cualquier otro caso. Algo más formalmente, un *algoritmo de tiempo polinomial* se define como uno en el que su función de complejidad temporal es de *O(p(n))* para alguna función polinomial *p*, donde *n* se utiliza para indicar el tamaño (longitud) del problema. Asimismo, notemos que la definición dada anteriormente para algoritmos de tiempo exponencial incluye ciertas funciones no-polinomiales de complejidad temporal que usualmente no son consideradas exponenciales, como por ejemplo $n^{\log n}$.

Notemos que la distinción entre esos dos tipos de algoritmos tiene una importancia particular para instancias de gran tamaño, por ejemplo, la tabla M.1 muestra las diferencias de tasa de crecimiento de diferentes funciones de complejidad, donde las mismas representan el tiempo de ejecución en microsegundos (tomada de Garey (1979)) donde se observa claramente las tasas de crecimiento de las funciones de complejidad exponencial:

[36]La complejidad temporal de un algoritmo se denota generalmente como *O(g(n))* donde *n* es el tamaño del problema. Se omiten constantes multiplicativas y aditivas. Por lo tanto el número de pasos para resolver un algoritmo de complejidad *O(n)* es una función lineal del número de entradas. Decimos que una función *f(n)* es *O(g(n))* siempre que exista una constante *c* tal que $|f(n)| \leq c \cdot |g(n)|$ para todos los valores de $n \geq 0$. Ver Anexo II para un breve resumen de *Órdenes de Magnitud*.-

[37]La *función de complejidad temporal* de un algoritmo, expresa sus requerimientos de tiempo al dar, para cada posible longitud (tamaño) del mismo, la mayor cantidad de tiempo que necesita el algoritmo para resolver una instancia del problema de ese tamaño.

Tabla M.1. – Comparación de diferentes funciones de complejidad temporal polinomiales y exponenciales

Función de Complejidad Temporal	**Tamaño n**					
	10	20	30	40	50	60
n	0.00001 segundos	0.00002 segundos	0.00003 segundos	0.00004 segundos	0.00005 segundos	0.00006 segundos
n^2	0.0001 segundos	0.0004 segundos	0.0009 segundos	0.0016 segundos	0.0025 segundos	0.0036 segundos
n^3	0.001 segundos	0.008 segundos	0.027 segundos	0.064 segundos	0.125 segundos	0.216 segundos
n^5	0.1 segundos	3.2 segundos	24.3 segundos	1.7 minutos	5.2 minutos	13.0 minutos
2^n	0.001 segundos	1.0 segundos	17.9 minutos	12.7 días	35.7 años	366 siglos
3^n	0.059 segundos	58 minutos	6.5 años	3855 siglos	2×10^8 siglos	1.3×10^{13} siglos

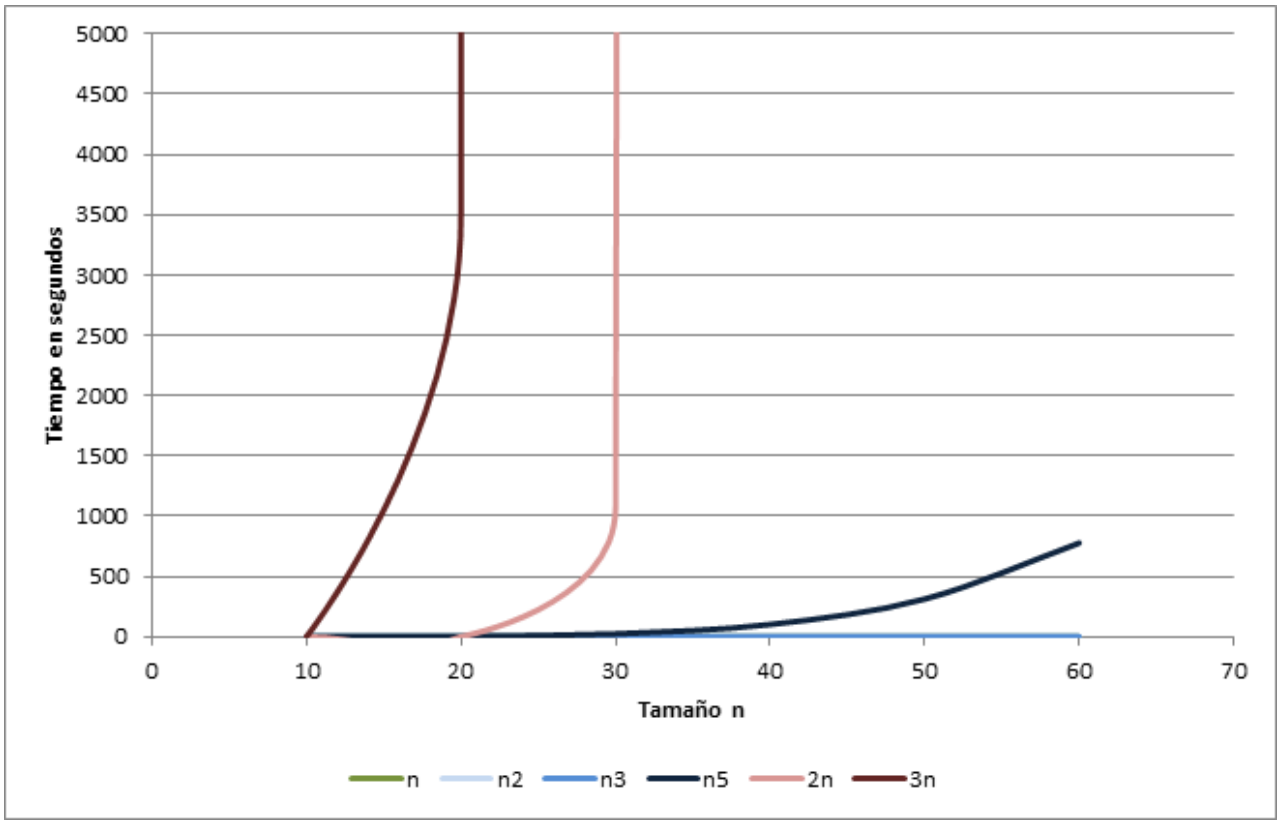


Figura 4 – Funciones de complejidad temporal de la Tabla M.1.

Asimismo, se dice que es *computacionalmente tratable* si existe un algoritmo que es de complejidad polinomial para todas las instancias y que resuelve el problema. Mientras que se dice que es *computacionalmente intratable* (o *complejo*, o *duro*, o *difícil*) si el algoritmo (probablemente) óptimo para resolver el problema no puede resolver todas las instancias en tiempo polinomial.

A pesar que se ha definido la complejidad en función del tiempo, podemos pensarla también en función del *costo*. Si el tiempo es costoso, y si no existen economías exponenciales de escala asociadas con el tiempo, los problemas computacionalmente intratables serán prohibitivamente costosos, dado que el costo de resolver tales problemas también crecerá en forma exponencial.

Esta caracterización de la dificultad del problema tiene dos principales fortalezas. Es independiente de cualquier diseño de hardware de computación y clasifica la dificultad de los problemas en sí mismos sin recurrir a los métodos utilizados para resolverlos.

Asimismo, los resultados bajo esta caracterización son *independientes de la implementación*. Diferentes lenguajes pueden alterar la complejidad temporal de un algoritmo, pero ningún lenguaje "razonable" convertirá un algoritmo polinomial en exponencial. Por otro lado, computadoras más potentes pueden realizar cada operación atómica más rápidamente, pero no pueden alterar la función de complejidad temporal del problema. Es decir, problemas intratables no pueden hacerse tratables mediante mejoras en la tecnología de hardware.

En esta línea de argumentación, esta caracterización aplica al problema en sí mismo y no a un método particular utilizado para resolverlo. Problemas que se demuestra son difíciles bajo esta caracterización son difíciles para *cualquier* método computacional. Dado que es el problema en sí mismo el que requiere tiempo exponencial, no puede convertirse en "tratable" mediante avances en software o en diseño algorítmico.

Aun así es igualmente cierto que esta caracterización presenta importantes limitaciones. Primero, la distinción entre problemas tratables e intratables es más importante para instancias de gran tamaño – donde el factor exponencial en los requerimientos de tiempo se convierte en dominante. Por ejemplo, consideremos dos problemas, el "A" es computacionalmente intratable y toma $O(1.1^n)$ pasos para ser resuelto; mientras que el "B" es computacionalmente tratable y toma $O(n^{14})$ pasos. A pesar que el tiempo necesario para resolver el problema A será eventualmente mucho más grande que el requerido para el problema B, para tamaños menores a 1000, podemos de hecho resolver A en forma mucho más rápida:

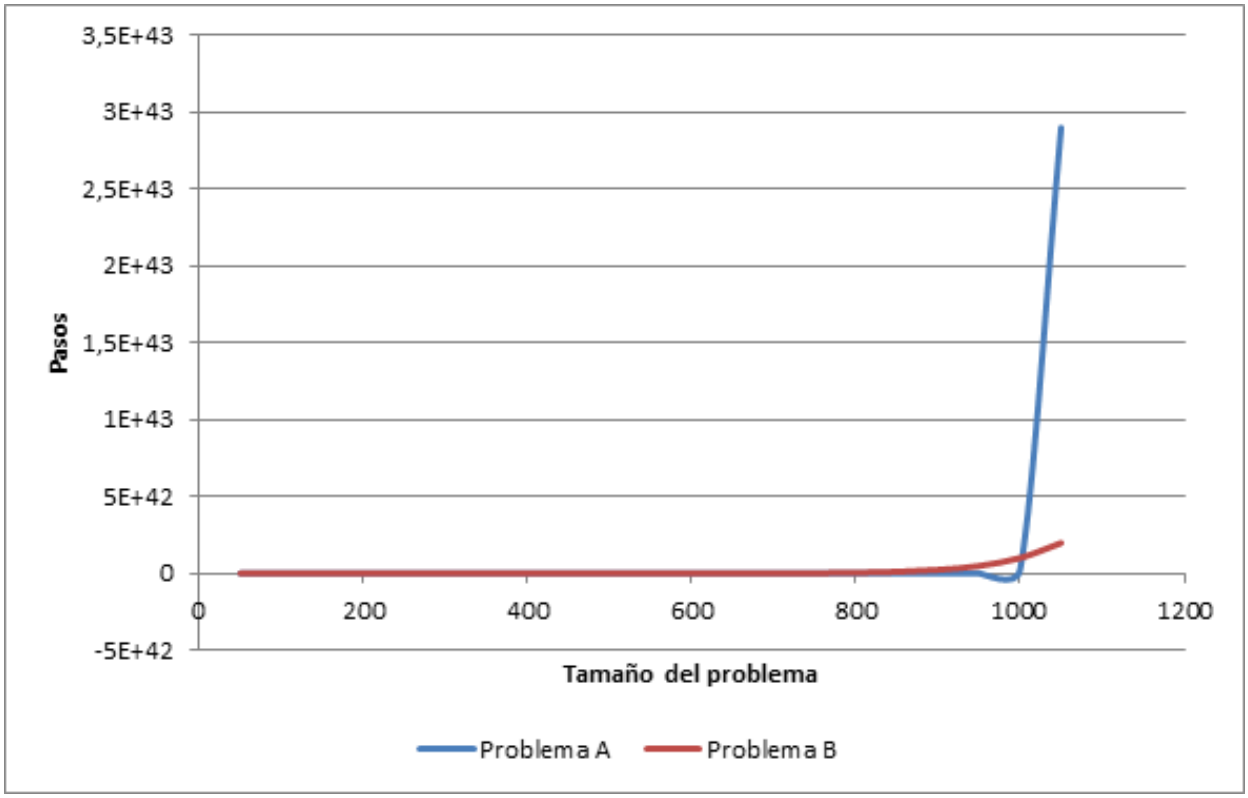


Figura 5 – Pasos para el Problema A y B

Asimismo, aún en la tabla M.1 y la Figura 4, se ve que el algoritmo de función de complejidad temporal 2^n es más rápido que el de función de complejidad temporal n^5 para todo $n \leq 20$. Insistimos, la complejidad temporal tal como ha sido definida es una medida del *peor escenario*, y por ejemplo, el hecho que un algoritmo posea una función de complejidad temporal 2^n significa solamente que por existe por lo menos una instancia del problema de tamaño *n* que requiere como mucho ese tiempo. Y de hecho muchas de las instancias del problema pueden requerir mucho menos tiempo que eso[38].

Segundo, cuando usamos esta caracterización, requerimos que el problema sea resuelto en forma exacta. Algunos problemas que son computacionalmente difíciles de resolver pueden ser *aproximados* en forma mucho más rápida, y si esta aproximación es (probablemente o empíricamente) suficientemente cercana a la solución del problema, para todos los fines prácticos podemos no necesitar la mejor solución exacta.

Tercero, cuando utilizamos esta caracterización requerimos que nuestro algoritmo *siempre* alcance una solución *correcta* para cada instancia del problema, requerimientos que hace de la complejidad computacional función de la peor instancia del problema. Dado que basamos el análisis en el peor caso, existe la posibilidad de que estemos sobre-enfatizando la complejidad del problema en promedio. Asimismo, dado que requerimos

[38]Por ejemplo el algoritmo *simplex* para programación lineal tiene complejidad de tiempo exponencial, pero tiene un registro impresionante de corridas rápidas en la práctica.

que nuestro algoritmo de solución nunca se equivoque o se dé por vencido, estamos removiendo del análisis algunos algoritmos que son probabilísticos. Si bien es cierto que esta clase de algoritmos no resuelven formalmente un problema computacionalmente complejo, pueden ser bastante útiles si sus tasas de error y de falla son lo suficientemente bajas.

Estas tres consideraciones anteriores nos ofrecen rutas de escape de problemas computacionalmente intratables, pero se trata de rutas de escape *posibles*, no necesariamente seguras.

2- El Problema de la districtalización es Computacionalmente Complejo

Demostrar que un problema es intratable es difícil, pero existe una gran clase de problemas que los informáticos consideran intratables. La más antigua clase de estos problemas es la denominadas problemas *NP-completos*.

Además de buscar métodos poderosos para probar que un problema es intratable, se ha invertido mucho tiempo y esfuerzo en encontrar relaciones entre los problemas. La principal técnica empleada para demostrar que dos problemas están relacionados es la de "*reducir*" uno en el otro mediante la presentación de una transformación constructiva que mapea cualquier instancia del primer problema en una instancia equivalente del segundo. Esta transformación provee también la forma de convertir cualquier algoritmo que resuelva el segundo problema en un algoritmo correspondiente que resuelva el primer problema.

Una de las características más importantes de los problemas NP-completos es su posibilidad de *reducción en tiempo polinomial ("polynomial-time reducibility")*. Cualquier problema NP-completo puede ser transformado en cualquier otro problema NP-completo en tiempo polinomial, de esta forma si se puede probar que un problema cualquiera NP-completo es formalmente intratable, se habrá probado que todos los problemas NP-completos son intratables y vice-versa. Las bases teóricas para la teoría de la completitud-NP fueron presentadas por Stephen Cook en 1.971 en su *paper* titulado *The complexity of Theorem Proving Procedures*, además de enfatizar el significado de la *reducción en tiempo polinomial*, Cook se centra en la clase de los problemas de decisión NP, que pueden ser resueltos en tiempo polinomial **por una computadora no determinística** (un problema de decisión es un problema cuya solución es **SI** o **NO**). La mayoría de los problemas aparentemente intratables encontrados en la práctica, cuando se los presenta como problemas de decisión, pertenecen a esta clase. Por otro lado Cook muestra que un problema particular en NP, denominado el problema de la *satisfactoriedad* (*the satisfiability problem*) tiene la propiedad que cualquier otro problema en NP puede ser polinomialmente reducido a él. Por lo tanto, si el problema de la *satisfactoriedad* puede ser resuelto con un algoritmo de

tiempo polinomial, **todo** problema en NP puede ser resuelto de esta forma, y si cualquier problema en NP es intratable, entonces, el problema de la *satisfactoriedad* también debe ser intratable. Es decir, el problema de la *satisfactoriedad* es el problema más difícil en NP[39].

Más tarde, Richard Karp (1.972) presenta una lista de 21 problemas que prueba son NP-completos, y provee versiones de decisión de varios de los problemas combinatorios conocidos, incluyendo el problema del vendedor viajero (*the traveling salesman problem*), demostrando que estos problemas son por lo menos **tan difíciles** como el problema de la *satisfactoriedad*, es así que nace la clase de equivalencia compuesta por los problemas más difíciles en NP, los problemas NP-completos.

La pregunta esencial derivada de las ideas de Cook no es otra que: **¿Son los problemas NP-completos intratables?** A la fecha no ha sido posible demostrar la intratabilidad de los problemas NP-completos, aunque tampoco se ha logrado encontrar un algoritmo polinomial que resuelva alguno de estos problemas, y el consenso en los expertos es que no existe un algoritmo de este tipo. Ahora bien, es importante recordar que aun sin una prueba de que la NP-completitud implica intratabilidad, el conocimiento de que un problema es NP-completo, sugiere, como mínimo, que se necesitará una avance muy importante para poder resolverlo utilizando un algoritmo de tiempo polinomial.

Además, la clase de problemas NP-completos no es la única clase que se considera que es intratable. En nuestro caso necesitamos considerar también la clase NP-compleja o NP-difícil (*NP-hard*): La clase NP-difícil es un súper conjunto que contiene la clase NP-completa; esta clase es potencialmente más difícil de resolver que los problemas NP-completos dado que a pesar que si cualquier problema NP-completo es intratable, entonces todos los problemas NP-difíciles son intratables, la reversa no es cierta[40].

[39] De hecho Cook introdujo en ese *paper* el mayor problema no resuelto de la ciencia computacional, el Problema de **P vs. NP**, que esencialmente (e informalmente) se pregunta si todo problema cuya solución puede ser eficientemente comprobada por una computadora también puede ser eficientemente resuelto por una computadora, la pregunta sobre si P = NP? Implica preguntar si la solución a un problema puede ser verificada "rápidamente", entonces las soluciones en sí mismas, pueden ser calculadas rápidamente?, donde "rápidamente" equivale a una resolución con función de complejidad temporal polinomial. Ver por ejemplo Goldreich Oded (2006). **Computational Complexity: A conceptual Perspective.** Department of Computer Science and Applied Mathematics. Weizmann Institute of Science, Rehovot.Israel, para un tratamiento del problema P vs. NP ligeramente diferente al tratamiento clásico (que implica el uso de una máquina no determinística de tiempo polinomial).

[40] Cualquier problema NP-difícil se puede demostrar que es NP-completo para por lo menos algunas instancias, pero no necesariamente para todas las instancias.

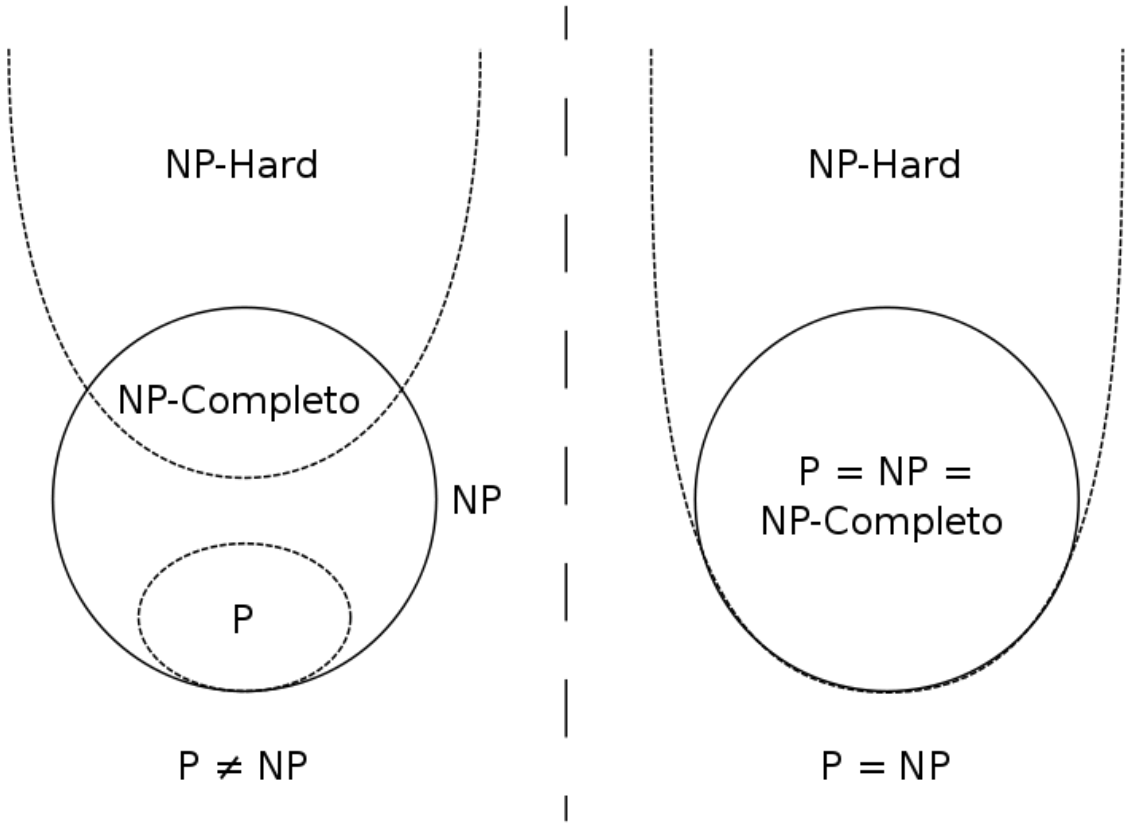


Figura 6 – Diagrama de Venn de las familias de problemas P, NP, NP-completo y NP-difícil[41]

Entender claramente la relación entre las clases de equivalencia P y NP es fundamental para la teoría de la NP completitud[42]. Obviamente que en el contexto de la discusión precedente e independientemente de la resolución del problema de P vs. NP, implícitamente hemos mostrado que $P \subseteq NP$. Es decir, todo problema resoluble por un algoritmo determinístico de tiempo polinomial es también resoluble por un algoritmo no determinístico de tiempo polinomial. Ahora bien, si P difiere de NP ($P \neq NP$), entonces la distinción entre P y NP – P es importante, dado que todos los problemas en P pueden ser resueltos mediante algoritmos de tiempo polinomial y que todos los problemas en NP – P son intratables. Por lo tanto, dado un problema de decisión $\Pi \in NP$, si $P \neq NP$ queremos saber a cuál conjunto pertenece (es decir si Π pertenece a P y por lo tanto es resoluble en tiempo polinomial o si pertenece a NP – P y es por lo tanto intratable). Desafortunadamente, hasta

[41] Tomada de www.wikipedia.org [Internet, fecha de último acceso Octubre 2.010].
Un problema en NP es NP-completo si todo problema en NP es reducible a él. Por lo tanto, el destino de la clase de equivalencia NP con respecto a su inclusión en P está en las manos de cada problema NP-completo. Es más, probar que un problema es NP-completo implica que ese problema no está en P, a menos que NP sea igual a P. Es importante tener siempre presente que los problemas NP-completos NO son los únicos problemas difíciles en NP (es decir, si P es diferente a NP, entonces NP contiene problemas que no son NP-completos y que no están en P).

[42] Notemos que consistentemente estamos evitando entrar en las intrincaciones de definir máquinas de Turing determinísticas de una sola cinta, máquinas no determinísticas, etc. Pero reteniendo las nociones mínimas necesarias aún con el riesgo de cierta pérdida de formalidad pero ganando en simpleza.

tanto no podamos probar que P ≠ NP, no tenemos esperanzas de mostrar que un dado problema particular pertenece a NP – P. Por esta razón, la teoría de la NP-completitud se centra en proveer resultados de la forma más débil "Si P ≠ NP, entonces $\Pi \in$ NP – P, es decir, es intratable".

Altman (1.997) muestra que el problema de districtalización, *en su forma más general*, es por lo menos tan difícil como cualquier problema NP-completo. Asimismo, otra implicación de sus resultados es que encontrar un distrito óptimo bajo cualquier *combinación* de objetivos *difíciles* es también *difícil*. Es decir, no podemos esperar que la districtalización automática óptima sea tratable para cualquier elección arbitraria de Unidades Base, número de distritos y función objetivo arbitraria. Es más, aun la tarea de construir distritos que minimizan solamente la desviación de población es una tarea computacionalmente difícil[43].

Como forma de ilustrar esto, primero notemos que los problemas de particionamiento, cobertura y empaquetado están generalmente muy relacionados. En un problema de particionamiento se nos da un objeto y se nos pide una colección de sub-objetos disjuntos cuya unión equivalga al objeto dado; en un problema de cobertura, se nos pide una colección (no necesariamente disjunta) cuyas uniones contengan el objeto dado; y en el problema de empaquetado se nos pide una colección de objetos disjuntos cuya unión contenga el objeto dado. Si consideramos rápidamente el problema de encontrar un plan contiguo con distritos equi-poblados, ese problema es equivalente al problema conocido como "Cortar un grafo en componentes conectados de tamaño acotado" (*Cut into connected components of Bounded Size)*[44]:

INSTANCIA: Grafo $G = (V,E)$ con cota entera $K \geq |V| / 2$

PREGUNTA: Existe una partición de los vértices, V, en subconjuntos disjuntos V_1 y V_2 tal que máx($|V_1|, |V_2|$) $\leq K$ y tanto V_1 como V_2 inducen subgrafos conectados de G.

Este problema es NP-completo (ver Johnson, 1.984).Y ya hemos visto que el número de distritos posibles para un dado número de Unidades Base de igual población está dado por un número de Stirling del segundo tipo (ver ecuación I.19) y aún en el caso de relativamente baja cantidad de UBs la cantidad de distritos posibles es extremadamente grande. En la práctica, las restricciones de contigüidad reducen el número posible de distritos, que aun así sigue siendo extremadamente grande, por lo que seleccionar el plan de districtalización globalmente óptimo a partir de una enumeración de todos los planes posibles no es factible ni siquiera utilizando la más rápida de las computadoras disponibles. Por razones similares, la generación aleatoria de planes de districtalización (mediante caminatas al azar por ejemplo) tampoco es probable que genere planes factibles.

[43] Ver Anexo I con las demostraciones.
[44] Johnson (1.984)

Al fallar la enumeración las opciones restantes son las heurísticas o los algoritmos, aunque el hecho de haber probado que se trata de un problema NP-completo es fuerte evidencia de que no existen algoritmos eficientes capaces de resolver el problema con una aproximación garantizada de optimalidad, es por eso que se tiende a pensar en soluciones heurísticas que si bien no necesariamente garantizan la producción de soluciones de calidad, si las calculan en un tiempo razonablemente corto.

3- **Intentos de escapar de la intratabilidad**

Hemos dicho que un problema es tratable solamente si existe un algoritmo de tiempo polinomial que garantiza resolverlo en forma exacta para todas las instancias del problema. ¿Estaremos pidiendo demasiado?, ¿Si se pide un "poco menos" a nuestros algoritmos de solución, podemos encontrar soluciones prácticas a problemas de districtalización automatizados?

La NP-completitud es una forma limitada de intratabilidad[45], hay un número posible de opciones para tratar con este tipo de problemas, y si restringimos los valores lo suficientemente podemos encontrar formas tratables de encontrar distritos óptimos bajo esas condiciones. Asimismo, si restringimos el conjunto de entradas lo suficiente, también es posible encontrar instancias tratables.

Por otro lado, si el problema no puede hacerse tratable mediante restricciones sobre los valores o sobre los datos, aún es posible encontrar un método probabilístico que resuelva el problema la mayoría de las veces, o un algoritmo determinístico que trabaje bien

[45]Los problemas NP-completos no incluyen **todos** los problemas intratables, y tampoco son la "peor" clase de problemas intratables. La intratabilidad de los problemas NP-completos toma una forma limitada. Un problema puede ser intratable por diferentes razones. Como menciona Altman (1.997):

1- Puede ser que la solución en si misma sea inmanejablemente grande o inmanejable de alguna otra forma.
2- Puede ser que la solución sea manejable, pero es difícil de resolver y verificar
3- En el caso menos difícil, puede ser que sea difícil solamente encontrar la solución, pero una vez encontrada es fácilmente verificada y puesta en uso.

Un ejemplo de problemas del primer tipo es: *enumerar el conjunto de todos los planes de districtalización posibles*. Simplemente imprimir esta solución para este tipo de problemas es inmanejable, salvo para las instancias más minúsculas. Este problema es probablemente inmanejable, aunque afortunadamente de alguna forma se trata de problemas de poco interés, e inclusive aun en el caso de obtener una solución es poco probable usarla. Problemas del segundo tipo, son al decir de Altman (1.997) "esotéricos y demasiados complicados". Ahora bien, los problemas NP-completos caen en la tercera categoría. Un ejemplo de problema del tercer tipo es: *¿Existe un plan de districtalización donde la máxima diferencia de población entre distritos es menos que B, y de ser así, especifique el plan?* Para este problema, en la medida que el cálculo de la función objetivo *f* sea tratable, encontrar el plan puede ser difícil, aunque dado un plan satisfactorio, es sencillo verificar si el valor del plan excede *B*.

en la mayoría de los casos. Asimismo, también podemos encontrar un método que rápidamente encuentre una solución aproximadamente óptima o cuasi-optima.

3.1. Ejemplo de resolución de problemas NP-completos:

Si bien es cierto que todos los problemas NP-completos son reducibles unos a otros, en el sentido práctico, no son todos igualmente difíciles de resolver: algunos problemas pueden ser fácilmente restringidos, respondidos en forma probabilística o aproximados adecuadamente.

Altman (1.997) da dos ejemplos sobre las diferencias prácticas entre dos problemas formalmente intratables:

a- Imaginemos que fallece un pariente y nos deja como albacea de su herencia. Se trataba de un coleccionista de arte y antigüedades, y su herencia se compone exclusivamente de elementos únicos y valiosos. Se nos dice que debemos dividir la herencia de manera tal que (1) todos los elementos son entregados (no se puede venderlos y dar el dinero), (2) cada elemento debe ir a un único heredero, y (3) se debe maximizar la *igualdad subjetiva* de la división (el valor monetario que cada persona asigna a su parte debe ser tan próximo como sea posible al valor que otro heredero asigna a la suya).
Formalmente se trata de un problema NP-completo[46], pero si todos los herederos compartes valores comunes para cada objeto (si fueran todos ellos coleccionistas de arte que conocen los precios de mercado de los objetos), el problema no es muy "difícil" de resolver, debido a varios casos prácticos: Si hay solamente dos herederos y no hay elementos "invaluables" podemos obtener la solución óptima en tiempo polinomial utilizando programación dinámica. Por otro lado, si hubiera más herederos, pero todos los objetos fueran del mismo valor aproximado, podemos fácilmente minimizar la desigualdad entre las partes.

b- Ahora supongamos que cada heredero posea diferentes valores privados para conjuntos de objetos, esto es especialmente difícil cuando los conjuntos relevantes difieren también entre cada heredero (por ejemplo, la tía Marta le da un gran valor sentimental a la combinación de sofá y mesita de té, mientras que el tío Enrique odia la mesita pero siempre quiso el sofá y el tapiz que le hace juego). Si tenemos una gran cantidad de antigüedades a distribuir, a pesar

[46]Conocido como el problema de partición de un conjunto ponderado.

que el problema es básicamente el mismo que el anterior en estructura, en este caso se ha puesto mucho más difícil de resolver en la práctica.

3.2. Tamaño del Problema y Complejidad Computacional:

Aun problemas que son computacionalmente difíciles pueden ser resueltos fácilmente para casos lo suficientemente pequeños. ¿Son entonces los tamaños de los problemas de districtalización típicos, lo suficientemente pequeños para que la intratabilidad no es un tema práctico?

Desafortunadamente, la respuesta a esta pregunta es **no**. Como ya hemos mostrado, el número de soluciones posibles crece tanto en función de la cantidad de distritos a ser diagramados (hasta un cierto punto) como en función de la cantidad de Unidades Base que se utilizan. Más adelante mostraremos que el tiempo necesario para resolver sub-problemas de districtalización crece exponencialmente como factor tanto de los distritos como de las UB.

Mientras que el número de distritos es típicamente pequeño (tanto en districtalizaciones electorales, como policiales, o escolares), yendo de uno a cincuenta en líneas generales[47], el número de UB es cualquier caso práctico es grande, tanto como para hacer que los requerimientos de tiempo exponencial del algoritmo sean dominante aun cuando el crecimiento exponencial sea relativamente pequeño[48].

3.3. Restringiendo el Problema de Districtalización:

Si bien es cierto que cualquier problema adecuadamente entendido, se convierte en computacionalmente difícil cuando se lo generaliza lo suficiente, y se convierte en trivial cuando se lo restringe lo suficiente, debemos primero considerar si existen restricciones naturales sobre el problema que lo convierta en tratable.

Hay tres tipos básicos de restricciones que podemos colocar sobre nuestro problema:

[47] Por ejemplo, la mayor cantidad de comisarias por partido en la Provincia de Buenos Aires es de 20, mientras que la mayor cantidad de cuadriculas o sectores por comisaria no supera los 8. Asimismo, para el partido de La Plata, las fracciones censales son 47 (censo 2001), mientras que en promedio hay 15 radios censales por fracción (con un máximo de 28 y un mínimo de 2), y el 77 % de las fracciones censales está compuesta por entre 10 y 20 radios censales.

[48] Un factor de crecimiento exponencial tan pequeño como 1.001^n es probable que haga intratables a estos problemas.

i- Podemos restringir los objetivos de districtalización que vamos a considerar.

ii- Podemos descartar algunos planes de districtalización, tales como los que contienen distritos no contiguos.

iii- Podemos restringir los datos o las UB con las que alimentamos el proceso de districtalización automático.

3.3.1 Restricciones en la función objetivo:

Podemos facilitar la tarea computacional limitando el número y tipo de objetivos que la función debe optimizar. De esta forma podemos hacer uso de ventajas específicas de ciertos algoritmos de optimización, o podemos eliminar requerimientos de tiempo exponencial.

Ahora bien, aunque es cierto que existen funciones que son "fáciles" de optimizar, hay dos problemas con esta posibilidad. La primera es puramente práctica, cualquier objetivo razonable para districtalización parece ser computacionalmente difícil de optimizar. Aun las más simples y populares funciones objetivo son *todas* computacionalmente difíciles de optimizar. Y lo que es peor, cualquier (simplemente ponderada) combinación de estas funciones entre sí, o con cualquier otra función, también es computacionalmente difícil. Es decir, cualquier función objetivo de importancia práctica no permite una fácil *computabilidad*.

La segunda dificultad es normativa y política. Los proponentes de la districtalización automática señalan que una de las fortalezas de la automatización es permitir que se definan los objetivos a optimizar y no su implementación. Este argumento también se sostiene en la mayoría de los casos de districtalización analizados (policial, escolar, de marketing, etc.). Esta separación entre elección de objetivos e implementación se viola si se colocan restricciones sobre los objetivos permitidos. Permitir que las limitaciones de un proceso de computación restrinjan *ex ante* el tipo de objetivos de representación que una sociedad o una organización pueda pretender puede ser normativamente y/o políticamente inaceptable.

3.3.2 Restricciones en las entradas:

Restricciones en las entradas nos permiten también, teóricamente, escapar de la intratabilidad. Generalmente en la mayoría de la investigación sobre districtalización

automática, las entradas básicas se definen en términos de unidades base tales como los radios censales. Es posible aplicar restricciones sobre las UB en forma directa o indirecta.

Podemos utilizar limitaciones en la entrada *directamente* para reducir la complejidad computacional de un problema mediante la eliminación de sus peores casos (si es que podemos predecir cuales son los casos que crean problemas para nuestros algoritmos de districtalización). Por ejemplo, si queremos crear planes de districtalización solamente para minimizar desigualdades de población, podemos simplificar el problema al restringir todos los bloques de población a ser iguales en tamaño[49]. En este caso restringido, cualquiera de las heurísticas usuales rápidamente encontrará una solución óptima al problema. Si bien esto no trivializa completamente el problema, si elimina la posibilidad de casos que puedan causar que el algoritmo necesite tiempo exponencial para completar.

Para problemas más complicados, podemos utilizar restricciones indirectas en las entradas, y combinar esto con restricciones en los objetivos del proceso, de manera tal de reducir el número de soluciones que un algoritmo de optimización necesita considerar. Métodos tales como los de "ramificación y acotamiento (*"branch and bound"*), o métodos de planos de corte (*"cutting planes methods"*) utilizan en esencia diferentes aplicaciones de este principio general. Balas (1985) da una descripción sucinta de estos métodos:

> *"Métodos enumerativos (ramificar y acotar, enumeración implícita) resuelven un problema discreto de optimización al partir su conjunto factible en subconjuntos sucesivamente más pequeños, calculando limites en los valores de la función objetiva sobre cada subconjunto, y usándolos para descartar ciertos subconjuntos de consideraciones posteriores. Los límites se obtienen al reemplazar el problema sobre un dado subconjunto, con un problema más sencillo (relajado), de manera tal que el valor de la solución para este último acote el del primero. El procedimiento termina cuando cada subconjunto ha sido reducido a una solución posible o se ha probado que no contiene una mejor solución que la que ya se tiene. La mejor solución encontrada durante este procedimiento es un óptimo global."*

Estos métodos son por lo general muy sensibles a la elección de la función objetivo (se necesita conocer muy bien esta función objetivo para poder derivar problemas "relajados" que puedan utilizarse para definir los límites). Luego tenemos que restringir nuestro conjunto de entrada de manera tal de garantizar que el procedimiento de "ramificar y acotar" no tarde tiempo exponencial.

[49] Una estrategia alternativa y similar para obtener distritos con igual cantidad de población es arbitrariamente partir las UB de manera tal que sean aproximadamente iguales. Si bien esto es popular en los métodos de districtalización política, debe tenerse presente que en toda nuestra argumentación se considera a la UB como las unidades de granularidad mínima para las que se cuenta con información censal disponible, por lo tanto partirlas no es otra cosa que un método aproximado…

A pesar que podemos hacer más fácil el proceso de districtalización mediante restricciones en las entradas, muchas restricciones útiles pueden no ser prácticas o posibles. Para eliminar el peor caso necesitamos forzar a la regularidad sobre las UB que utilizamos, pero muchas de estas características son exógenas. Por ejemplo, regularidades geográficas nos pueden ayudar a definir distritos compactos. De hecho hay problemas que pueden ser muy sencillos gracias al azar, pero no es posible pretender tener siempre buena suerte.

Por otro lado, solamente podemos confiar en restricciones que eliminen los peores casos cuando restringimos con cuidado la función objetivo. Restricciones que hacen sencillo el cumplimiento de un criterio pueden exacerbar la dificultad para satisfacer otro, y se ha encontrado que son pocas las restricciones que son útiles para una variedad de funciones objetivo.

3.3.3 Restricciones en los planes:

En lugar de tratar de restringir los objetivos o los datos, podemos en forma más razonable restringir los tipos de planes de districtalización a ser considerados. Por ejemplo, podemos considerar distritos que son rectángulos contiguos, si nuestra área de interés no tiene "huecos", y si estamos interesados solamente en maximizar una medida de compacidad (*e.g.* minimización del perímetro), podemos encontrar con mayor facilidad "distritos óptimos".

Ahora bien, a pesar que puede ser matemáticamente conveniente expresar estas restricciones de esta forma, políticamente y/o normativamente tales restricciones implican un conjunto de restricciones sobre las funciones objetivo o sobre las entradas. Por ejemplo, si requerimos que nuestros planes sean contiguos, excluimos cualquier función objetivo que permita a la contigüidad ser ponderada contra otros criterios. Debido a esta equivalencia, no podemos escapar a las limitaciones de las restricciones a la función objetivo o a las entradas al reformularlas como restricciones a nivel del plan. A pesar que una restricción a nivel del plan puede ser útil para la formulación clara del problema, debemos tener sumo cuidado que no se las manipule para oscurecer los otros tipos de restricciones que ellas implican.

4- ¿Y si consideramos Disctrictalizaciones sub-óptimas?

Cuando se definió la complejidad computacional teórica, el concepto de solución fue demasiado demandante. Las soluciones políticas o policiales, rara vez requieren el grado

de precisión que demanda la teoría. Por lo tanto, ¿si relajamos nuestros requerimientos de precisión, podemos encontrar en general, fáciles y prácticos métodos de districtalización? ¿Podemos encontrar planes óptimos la mayoría de las veces, o encontrar aproximaciones lo "suficientemente buenas"?

4.1 Districtalización Óptima la mayoría de las veces:

Hay dos formas posibles en las que podemos rápidamente realizar districtalizaciones óptimas la mayoría de las veces: Podemos desarrollar un método para obtener soluciones cuasi-óptimas para todos los casos, o encontrar la solución óptima para la mayoría de los casos.

Cuando definimos la complejidad computacional, pedimos que nuestros algoritmos garanticen generar la solución **correcta** para un problema. Pero podemos relajar esta garantía de corrección y permitir a nuestros métodos que cometan ocasionales errores[50]. La posibilidad de "resolver" algunos problemas NP-completos no ha sido teóricamente excluida, pero desafortunadamente se conjetura que los problemas NP-completos no son susceptibles de este tipo de solución[51].

En lugar de confiar en métodos probabilísticos, podemos esperar tener un problema lo suficientemente simple, podría ser que un algoritmo resuelva el problema con certeza ya sea en promedio o en la práctica[52]. En la medida en que exista algún caso para el que se requiera tiempo exponencial el problema es aun, por definición intratable, pero puede ser más sencillo la mayor parte del tiempo.

Se trata de una pregunta eminentemente empírica: ¿Cuán bien responden los métodos reales sobre datos reales?... desafortunadamente el registro histórico de la districtalización automática no es muy bueno. La mayoría de los investigadores que han intentado generar planes de districtalización han enfatizado que se trata de un problema

[50]Expandimos nuestras soluciones a algoritmos que producen los resultados con una probabilidad ε. Obviamente no estamos acotando la magnitud del error sino su **probabilidad de ocurrencia.**

[51]Se dice que un algoritmo es "acotado probabilísticamente polinomial" (*boundedly probabilistic polynomial BPP*) si computa una "solución" a todas las instancias de un problema en tiempo polinomial, y la "solución" computada tiene una probabilidad estrictamente mayor que el 50 % de ser correcta, cada vez que se ejecuta el algoritmo. Dado que la probabilidad de error en *cada* ejecución del algoritmo es independiente, los algoritmos en esta clase se pueden ejecutar repetidamente para obtener cualquier probabilidad de corrección deseada. Desafortunadamente la actual conjetura es que ningún problema NP-completo es miembro de la clase de problemas BPP.

[52]Podemos decir que un algoritmo trabaja bien *en promedio* cuando describimos su comportamiento en base a problemas obtenidos de una distribución de probabilidad teórica y matemáticamente definida; mientras que podemos decir que un algoritmo funciona bien *en la práctica* cuando lo hemos evaluado en base a datos empíricos reales.

difícil en la práctica, y la literatura especializada revela que no existe un procedimiento que sea demostrablemente óptimo y que haya sido generalmente efectivo en conjuntos de datos lo suficientemente grandes como para ser útiles en la práctica.

Ahora bien, se ha podido probar que para la clase de distribuciones computables simples, la complejidad promedio de casos de todos los problemas NP-completos es exponencial[53]. Para algunas distribuciones, la complejidad promedio no puede escapar la intratabilidad.

Por lo tanto, la posibilidad de descubrir un procedimiento tratable para una districtalización óptima es muy tenue. En la práctica, los procedimientos automáticos resultaran casi con seguridad en planes sub-óptimos.

4.2 Aproximaciones Garantizadas

La complejidad computacional es una medida de la dificultad de obtener soluciones *óptimas*, pero no dice nada sobre la dificultad de las aproximaciones. De hecho, los problemas NP-completos, si bien son equivalentes para soluciones óptimas, no lo son para aproximaciones. Algunos problemas son mucho más fáciles de aproximar que otros. Para algunos problemas de optimización NP-completos, existen métodos que, en tiempo polinomial, generan una solución que se **garantiza** este dentro de un porcentaje predefinido del valor óptimo[54]. Los métodos que obtienen "soluciones" arbitrariamente cercanas en tiempo polinomial se denominan *aproximaciones polinomiales completas.* Desafortunadamente, puede demostrarse que no existen aproximaciones polinomiales completas para muchos de los sub-problemas de districtalización discutidos precedentemente[55].

A pesar que el problema de districtalización no permite aproximaciones arbitrariamente cercanas, no hemos excluido la posibilidad de aproximar a la solución dentro de un dado porcentaje fijo del óptimo. Si bien no se ha encontrado a la fecha un

[53] Li y Vitanyi (1992).

[54] Hay que tener extremo cuidado de distinguir entre medidas de aproximación basadas en el valor de las soluciones producidas versus medidas basadas en el porcentaje de soluciones que son excluidas. Por ejemplo, distritos dibujados a mano es probable que estén en el 99 % superior de todos los distritos posibles, simplemente debido a que hay muchos distritos posibles con bajo valor. Aun así, estos distritos dibujados a mano pueden tener un valor mucho menor que el del distrito óptimo.

[55] Las aproximaciones épsilon son métodos que garantizan que la razón del valor de la solución aproximada en relación al valor de la solución óptima sea no menor que 1-ε. Las aproximaciones completamente polinomiales son aproximaciones épsilon que tienen requerimientos de tiempo acotados por una función polinomial de ε para todo ε.

procedimiento de aproximación de este tipo para un sub-problema de districtalización, la pregunta permanece abierta.

4.3 Conjeturas informadas

La forma más común de aproximarse a la districtalización automática es a través del uso de *heurísticas* (o *meta-heurísticas*). Una heurística es un método que se considera útil para resolver un problema, pero que no garantiza optimalidad u optimalidad aproximada. Mientras que los métodos heurísticos no garantizan optimalidad, si pueden reducir el conjunto de planes de districtalización a ser considerados.

La literatura de la ciencia informática y de investigación operativa presenta diversas heurísticas que han sido aplicadas para resolver matemáticamente problemas de partición similares:

- *Simulated annealing*, que es un proceso análogo al enfriamiento lento del metal.
- *Algoritmos genéticos* que utilizan procedimientos análogos al cruce, mutación y selección evolutiva para generar soluciones en problemas de optimización.
- *Búsqueda Tabú,*
- *"Weighted Voronoi region procedures",*
- *"Divide and conquer",*
- *"Lagrangean relaxation",*
- *Métodos de Monte Carlo*
- Etc.[56]

Ahora bien, no siempre estas heurísticas funcionan bien con tamaños razonables para la cantidad de UBs y distritos a formar en un caso práctico, y es por lo tanto aún necesaria mucha investigación al respecto.

[56] Para algunos ejemplos de modelos y algoritmos recientes aplicados a districtalización, ver por ejemplo: Li, Wang, R. S., y Wang, Y. (2007), o Ricca y Simeone (2008), o bien Ricca, Scozzari y Simeone (2008) o Yamada (2009).

B) Búsqueda Tabú[57]

Las raíces de la *Búsqueda Tabú* surgen en los 1970s, pero fue presentada por primera vez por Glover (1986), y formalizada con mayor detalle nuevamente por Glover (1989 y 1990) y de Werra y Hertz (1989). Si bien no existe una clara demostración de su convergencia[58,59], la técnica ha demostrado una considerable eficiencia en muchos problemas de optimización.

Entre los procedimientos de optimización, las técnicas iterativas juegan un papel importante dado que para la mayoría de los problemas de optimización no se conoce un procedimiento general que permita obtener la solución "óptima". El paso general de un procedimiento iterativo consiste en construir a partir de una dada solución *i* la siguiente solución *j* y controlar si se debe parar en ese punto o realizar otro paso. Los métodos de búsqueda local son procedimientos iterativos en los que una vecindad *N(i)* se define para cada solución factible *i*, y la siguiente solución *j* se busca entre las soluciones que se encuentran en *N(i)*.

El método de búsqueda de vecindad más famoso que ha sido utilizado para encontrar una aproximación al valor mínimo de una función real *f* en un conjunto *S* es el método de descenso:

Método de descenso

Paso 1. Elegir una solución inicial *i* en *S*

Paso 2. Encontrar el mejor *j* en *N(i)* (de manera tal que $f(j) \leq f(k)$ para cualquier *k* en *N(i)*).

Paso 3. Si $f(j) \geq f(i)$ parar. De otro modo, hacer $i = j$ e ir al Paso 2.

Claramente este método es capaz de frenar en un mínimo local, pero no necesariamente en un mínimo global de *f*. Por lo general *N(i)* no se define explícitamente: podemos buscar *j* explorando en alguna dirección a partir de *i.*

[57] Hertz, Taillard y Werra (1.990)

[58] Faigle y Kern(1992), empleando consideraciones teóricas basadas en un modelo probabilístico de la Búsqueda Tabú muestran que cuando movimientos a partir de una solución *i* hacia una solución *j* se realizan de acuerdo con una distribución de probabilidad p_{ij} , entonces bajo ciertas suposiciones débiles es posible probar que se obtiene la convergencia a una solución óptima para probabilidades p_{ij} que se permite que sean elegidas de un gran subintervalo dentro de [0,1]

[59] Glover y Hanafi (2002) muestran convergencia finita para ciertos algoritmos de búsqueda tabú basados en memoria de inmediatez o memoria de frecuencia, distinguiendo entre estructuras de vecindad simétricas y asimétricas

En este contexto, tanto la Búsqueda Tabú como el *"Simulated Annealing"* pueden ser considerados métodos de búsqueda de vecindad bastante más elaborados que el método de descenso.

1. Ideas Principales de la Búsqueda Tabú

Para mejorar la eficiencia del proceso exploratorio, se necesita no solo guardar la información local (como por ejemplo el valor actual de la función objetivo), sino también alguna información relacionada con el proceso de exploración. Este uso sistemático de la ***memoria*** es una característica esencial de la Búsqueda Tabú (BT).

La Búsqueda Tabú se basa en la premisa de que la resolución de problemas, para poder ser calificada como inteligente, debe incorporar ***memoria adaptativa*** y una ***exploración responsiva***. Una buena analogía es la del montañismo, donde un montañista debe selectivamente recordar elementos clave del camino recorrido (utilizando memoria adaptativa) a la vez que debe ser capaz de realizar elecciones estratégicas durante el camino (empleando la exploración responsiva). La característica de memoria adaptativa de la BT (cuya importancia se aprecia en la analogía del montañista que debe analizar las alternativas actuales en relación con ascensos previos en terrenos similares) permite la implementación de procedimientos que son capaces de buscar el espacio de solución en forma económica y eficiente. Dado que las elecciones locales son guiadas por la información recolectada durante la búsqueda, la BT contrasta fuertemente con procedimientos que carecen de memoria y que por lo tanto dependen fuertemente de procesos semi-aleatorios que implementan alguna forma de muestreo (heurísticas semi-voraces y aproximaciones tradicionales de *annealing* y evolutivas). Asimismo, la memoria adaptativa también contrasta con los esquemas de memoria rígidos típicos de estrategias de ramificación y acotamiento (*branch and bound*). Por otro lado, el énfasis en la exploración responsiva (y por lo tanto en el propósito) en la BT, deriva de la suposición que una mala elección estratégica puede producir más información que una buena elección aleatoria. La exploración responsiva integra los principios básicos de la búsqueda inteligente (es decir, explotar características de las buenas soluciones mientras que se exploran nuevas regiones promisorias). En este sentido el principio de la BT puede sintetizarse como[60]:

> ***Es mejor una mala decisión basada en información que una buena decisión al azar, ya que, en un sistema que emplea*** **memoria**, ***una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones.***

[60] Martí (2003)

Mientras que la mayoría de los métodos de exploración retienen en memoria el valor *f(i*)* de la mejor solución *i** visitada al momento, la BT también mantendrá información sobre el itinerario recorrido hasta las últimas soluciones visitadas. Esta información será empleada para guiar el movimiento de *i* a la siguiente solución *j* que se elegirá dentro de *N(i)*. El rol de la memoria es el de restringir la elección de algún subconjunto de *N(i)* al prohibir movimientos a ciertas soluciones vecinas.

La estructura de la vecindad *N(i)* de una solución *i* será variable de iteración a iteración, por lo que podemos incluir a la BT en el conjunto de procedimientos denominados *Técnicas dinámicas de búsqueda por vecindad*.

Más formalmente, consideremos un problema de optimización: dado un conjunto *S* de soluciones factibles y una función $f: S \rightarrow \mathbb{R}$, encontrar alguna solución *i** en *S*, tal que *f(i*)* es aceptable con respecto a algún criterio o criterios. Generalmente un criterio de aceptabilidad para una solución *i** será tener $f(i^*) \leq f(i)$ para todo *i* en S. En esta situación, la BT será un algoritmo de minimización exacto, provisto que el proceso de exploración garantice que luego de un número finito de pasos, es posible alcanzar uno de esos *i**.

Desafortunadamente, en muchos contextos, no se tiene garantía de que pueda obtenerse un *i** de esas características, por lo que la BT puede ser considerado entonces como un procedimiento heurístico general. Ahora bien, dado que la BT de hecho incluye en sus reglas operativas algunas técnicas heurísticas, sería más apropiado caracterizar a la BT como una *metaheurística*, su rol será usualmente el de guiar y orientar la búsqueda de otro procedimiento (más local) de búsqueda.

Incidentalmente, la palabra *meta-heurística* o *metaheurística*[61] fue acuñada en el mismo *paper* que introdujo el término *Búsqueda Tabú* (Glover, 1986). Una *metaheurística* se

[61] Glover y Laguna (1998) muestran una clasificación de las *meta-heurísticas* en término de sus características con respecto a tres elecciones de diseño básicas (págs. 17-18): (1) el uso de memoria adaptativa, (2) el tipo de exploración de vecindad empleado y (3) el número de soluciones actuales que se mantiene de una iteración a la siguiente. Emplean un esquema de clasificación de la forma *x/y/z*, donde las opciones para x son A (si la *meta-heurística* emplea memoria adaptativa) y SM (si el método no posee memoria). Las elecciones para *y* son N (para un método que emplea alguna forma sistemática de búsqueda de vecindad, ya sea para seleccionar el siguiente movimiento o para mejorar una dada solución) y S (para aquellos métodos que se basan en muestreos aleatorios). Finalmente, *z*, puede ser 1 (si el método se mueve de una solución actual a la siguiente en cada iteración) o P (para métodos basados en aproximaciones de población con una población de tamaño P). Ahora bien, a pesar que los criterios de clasificación son generalmente aceptados por todos, no todos los investigadores asignan los mismos valores a las mismas técnicas, mientras que la Clasificación 1 hace referencia a la "concepción popular", la Clasificación 2 hace referencia a la opinión de un grupo significativo aunque minoritario de investigadores:

Tabla BT-8 Clasificación de *meta-heurísticas* (Glover y Laguna 1998)

Meta-heurística	*Clasificación 1*	*Clasificación 2*
Algoritmos Genéticos	SM/S/P	SM/N/P
Búsqueda dispersa	SM/N/P	A/N/P
Simulated Annealing	SM/S/1	SM/N/1
Búsqueda Tabú	A/N/1	A/N/P

refiere a una estrategia maestra que guía y modifica otras heurísticas de manera tal de producir soluciones más allá de aquellas que son normalmente generadas in una búsqueda de un mínimo local. Las heurísticas guiadas por esta *meta-heurística* pueden ser procedimientos de alto nivel o pueden no ser más que una descripción de los movimientos disponibles para transformar una solución en otra, junto con una regla de evaluación asociada.

Siguiendo a Hertz (1.990) es posible reformular el método de descenso de la siguiente manera:

Paso 1. Elegir una solución inicial *i* en *S*

Paso 2. Generar un subconjunto *V** de solución en *N(i)*

Paso 3. Encontrar el mejor *j* en *V** (de manera tal que $f(j) \leq f(k)$ para cualquier *k* en *V**) y hacer $i = j$.

Paso 4. Si $f(j) \geq f(i)$ parar. De otro modo, ir al Paso 2.

El procedimiento estándar de descenso, haríamos $V^* = N(i)$. Pero esto es por lo general muy costoso en tiempo de cálculo, y por lo tanto una adecuada elección de *V** es vista usualmente como una mejora sustancial. El caso opuesto sería hacer $|V^*| = 1$; lo que eliminaría la fase de búsqueda de un mejor j. El "*Simulated Annealing*" puede claramente ser integrado en este marco. Una solución *j* será aceptada si $f(j) \leq f(i)$, de otro modo será aceptada con cierta probabilidad dependiendo de los valores de *f* en *i* y *j* al igual que de un parámetro denominado *temperatura*.

Si bien no hay "temperatura" en la BT, la elección de *V** es vital; para definirlo en cada paso uno debe utilizar sistemáticamente la memoria para explotar el conocimiento que surge de la función *f* y de la vecindad *N(i)*.

Excepto en algunos casos especiales de convexidad, el uso de procedimientos de descenso es frustrante dado que es muy posible quedar atrapado en un mínimo local que de hecho está muy lejos del mínimo global. Por lo tanto, cualquier proceso de exploración

---

Por ejemplo, en el caso de *Simulated Annealing* en su forma original (Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P., 1983. **Optimization by simulated annealing**. *Science*, 220(4598), p.671-680.) fue modificada por un grupo de investigadores que considera que deben ser incorporados elementos más fuertes de búsqueda de vecindad (por eso la N en la clasificación 2). Por otro lado, los documentos iniciales de Glover sobre la Búsqueda Tabú incluían elementos basados en la población en la forma de estrategias para la explotación de colecciones de soluciones de elite guardadas durante la búsqueda, pero buena parte de la literatura sobre el tema no incorporó estos elementos de población hasta hace relativamente poco tiempo.

iterativa debe en algunos casos aceptar también movimientos de *i* a *j en* V*que no mejoran la solución (es decir, con *f(j) > f(i)*) si es que uno quiere escapar de un mínimo local[62].

Tan pronto como son posibles movimientos que no mejoran la solución, aparece el riesgo de visitar nuevamente una solución, y en líneas generales aparece el riesgo de entrar en ciclos. Este es juntamente el punto donde el empleo de la memoria es útil para prohibir movimientos que pueden llevar a soluciones recientemente visitadas. Si introducimos esa memoria, podemos considerar que la estructura de *N(i)* dependerá del itinerario y por lo tanto de la iteración *k*; por lo que nos referimos a *N(i,k)* en lugar de a *N(i)*. Con estas modificaciones, volvemos a reformular el método de descenso (*i**es la mejor solución encontrada al momento, y *k* es el contador de iteraciones):

Paso 1. Elegir una solución inicial *i* en *S*. Hacer *i** = *i* y *k = 0*.

Paso 2. Hacer k = k + 1 y generar un subconjunto *V** de solución en *N(i,k)*

Paso 3. Encontrar el mejor *j* en *V** (con respecto a f o a alguna función modificada $\tilde{f}$) y hacer i = j.

Paso 4. Si *f(i)* < *f(i*)* hacer *i** = *i*

Paso 5. Si alguna condición de parada se cumple, parar. De otro modo, ir al Paso 2.

Obviamente el procedimiento de descenso clásico está incluido en esta formulación (la regla de parada es simplemente $f(i) \geq f(i^*)$ e *i** siempre será la última solución).

En la BT, algunas condiciones de parada inmediata son:

- $N(i, k+1) = \varnothing$,
- *K* es mayor que el número máximo de iteraciones permitidas,
- El número de iteraciones desde la última mejora de *i** es mayor que un número dado,
- Se cuenta con evidencia de que se ha obtenido una solución óptima.

Ahora bien, mientras que las reglas de parada tienen alguna influencia en el procedimiento de búsqueda y en sus resultados, es importante realizar que la definición de *N(i,k)* en cada iteración *k* al igual que la elección de *V** son cruciales.

La definición de *N(i,k)* implica que algunas soluciones recientemente visitadas son removidas de *N(i)*; se las considera soluciones tabú, que deben ser evitadas en la siguiente

[62] Si bien el *"Simulated Annealing"* también hace esto, no guía en la elección de *j*. Mientras que la BT elige el mejor *j* en *V**.

iteración. Este tipo de memoria basada en la inmediatez (memoria reciente) prevendrá en forma parcial las posibilidades de entrar en ciclos. Por ejemplo, mantener en la iteración *k* una lista *T* (lista tabú) de las $|T|$ soluciones visitadas prevendrá ciclos de tamaño $|T|$. En esos casos, podemos hacer $N(i,k) = N(i) - T$. Ahora bien, esta lista *T* puede ser extremadamente impráctica para usar; por lo tanto, podemos describir el proceso de exploración en *S* en base a movimientos de una solución a la siguiente. Para cada solución *i* en *S*, se define *M(i)* como el conjunto de movimientos que pueden ser aplicados a *i* de manera tal de obtener una nueva solución *j* ($j = i \oplus m$). Por lo tanto, $N(i) = \{j \ / \exists m \in M(i)$ con $j = i \oplus m \}$. Por lo general empleamos movimientos que son reversibles: para cada *m* existe un movimiento m^{-1} tal que $(i \oplus m) \oplus m^{-1} = i$. Por lo tanto en lugar de mantener una lista *T* de las últimas $|T|$ soluciones visitadas, podemos mantener los $|T|$ movimientos o los $|T|$ movimientos reversos asociados con los movimientos realmente efectuados[63].

Puede ser conveniente por cuestiones de eficiencia utilizar varias listas T_r en un dado momento. En este caso, algunos constituyentes t_r (de *i* o de *m*) tendrán un estado tabú para indicar que esos constituyentes no pueden en ese momento estar involucrados en un movimiento. Por lo general el estado tabú de un movimiento es una función del estado tabú de sus componentes, que puede cambiar en cada iteración. Es así que se puede formular una colección de condiciones tabú como $t_r(i, m) \in T_r$ $(r = 1,\ldots, t)$. Un movimiento *m* (aplicado a una solución *i*) será tabú si ***todas*** las condiciones se satisfacen.

Por otro lado, otra consecuencia de la simplificación de la lista tabú (del reemplazo de soluciones por movimientos) es el hecho que es posible asignar el estado tabú a soluciones que no han sido visitadas al momento. Por lo tanto, es necesaria cierta relajación del estado tabú; se anulará el estado tabú cuando algunas condiciones tabú sean atractivas. Esto se realizará mediante *condiciones de niveles de aspiración.*

Un movimiento tabú *m* aplicado a una solución actual *i* puede ser atractivo dado que provee, por ejemplo, de una mejor solución que la mejor encontrada hasta el momento. En esas condiciones queremos aceptar *m* a pesar de su estado "prohibido"; haremos justamente esto si posee un *nivel de aspiración a(i,m)* que es mejor que cierto valor umbral *A(i,m)*.

Por lo general, *A(i,m)* puede ser considerado como un conjunto de valores preferidos para una función *a(i,m)*. Con esto, las condiciones de aspiración pueden ser escritas en la forma $a_r(i,m) \in A_r(i,m)$ $(r = 1,\ldots,a)$.

Si por lo menos una de estas condiciones es satisfecha por un movimiento tabú *m* aplicado a *i*, entonces *m* será aceptado (a pesar de su estado).

[63] Claramente esta restricción es una pérdida de información y no se garantiza que no se entre en un ciclo de longitud como mucho de $|T|$.

Por otro lado, es posible que en el proceso *f* sea en algunas instancias reemplazada por otra función $\tilde{f}$, lo que permite introducir la **intensificación** y la **diversificación** en la búsqueda.

En la BT, la memoria se utiliza de diferentes maneras para guiar el procedimiento de búsqueda, la memoria de "corto plazo", reciente o de inmediatez tiene el rol de prohibir ciertos movimientos con alta probabilidad de llevarnos nuevamente a soluciones recientemente visitadas, pero hay también otras formas de memoria que pueden ser consideradas.

En el proceso de búsqueda a veces es útil intensificar la búsqueda en alguna región de *S* dado que puede contener algunas soluciones aceptables. Este tipo de *intensificación* puede ser llevado a cabo al dar una mayor prioridad a soluciones que tienen atributos comunes con la actual solución; esto puede hacerse mediante la introducción de un término adicional en la función objetivo; este término penalizará soluciones que se alejen de la actual. Esto debe hacerse durante unas pocas iteraciones y luego puede ser útil explorar otra región de *S*; la *diversificación* tenderá a ampliar los esfuerzos de exploración a regiones diferentes de *S*. Nuevamente, la diversificación puede ser forzada mediante la introducción de un término adicional en la función objetivo; este término penalizará en cierta etapa soluciones que están cerca de la solución actual.

Tanto los términos de intensificación como de diversificación tienen ponderaciones que son modificadas durante el proceso de búsqueda de manera tal que las fases de intensificación y de diversificación se alternan. La función objetivo modificada es la función $\tilde{f}$ mencionada precedentemente, por lo que $\tilde{f}$ = *f* + Intensificación + Diversificación.

Con lo anterior las características esenciales del procedimiento de BT pueden definirse como:

Búsqueda Tabú

Paso 1. Elegir una solución inicial *i* en *S*. Hacer *i** = *i* y *k = 0*.

Paso 2. Hacer k = k + 1 y generar un subconjunto *V** de solución en *N(i,k)* tal que

alguna de las condiciones tabú $t_r(i,m) \in T_r$ se viola (r = 1,…,t)

o por lo menos una de las condiciones de aspiración $a_r(i,m) \in A_r(i,m)$ se cumple (r = 1,…,a)

Paso 3. Encontrar el mejor $j = i \oplus m$ en *V** (con respecto a *f* o a la función modificada $\tilde{f}$) y hacer i = j.

Paso 4. Si *f(i)* < *f(i*)* hacer *i** = *i*

Paso 5. Actualizar las condiciones tabú y de aspiración.

Paso 6. Si alguna condición de parada se cumple, parar. De otro modo, ir al Paso 2.

2. Eficiencia de los métodos iterativos de Solución

Es importante resaltar que la eficiencia de los métodos iterativos de solución depende fundamentalmente del modelado. Un ajuste cuidadoso de los parámetros nunca podrá balancear una mala elección de la estructura de vecindad o de la función objetivo. Por el contrario, un modelado efectivo conduce a técnicas robustas que no son demasiado sensibles a configuraciones diferentes de los parámetros.

Los métodos iterativos de solución pueden ser vistos como caminatas en un grafo del espacio de estados *G(S,A)* donde el conjunto de vértices *S* es el conjunto de soluciones factibles y donde existe un arco $(i,j) \in A$ que une *i* con *j* si $j \in N(i)$. La elección de una solución inicial es, por tanto, equivalente a generar un vértice en *G* y un paso del procedimiento iterativo consiste en moverse del vértice actual *i* a un vértice adyacente *i*. Ahora bien, para un dado problema de optimización, existen usualmente diversas formas de definir el grafo del espacio de estados *G*, y es por lo tanto esencial elegir uno que satisfaga la siguiente condición:

O1- Dado cualquier *i* en *S,* debe existir un camino que una *i* con una solución óptima *i**.

Si una solución que no satisface esta condición es visitada durante el proceso iterativo, es obvio que nunca se alcanzará una solución óptima.

Para ilustrar esto, consideremos el problema de colorear un grafo: dado un grafo *H* = (*V, E*), se debe encontrar un esquema de asignación de colores de cada uno de sus vértices con tan pocos colores como sea posible de manera tal que no haya dos vértices unidos por una arista tengan el mismo color (estos esquemas de asignación de colores se denominan factibles). Si uno define el conjunto de soluciones factibles *S* como el conjunto de esquemas de asignación de colores que no usan más que un dado número de colores (por ejemplo esa cota superior al número de colores podría ser el número de colores utilizado en la solución inicial), y si se define la vecindad *N(i)* de una solución *i* como el conjunto de esquemas de asignación de colores factible que puede ser obtenido a partir de *i* intercambiando el color de exactamente un vértice, se induce un grafo de espacio de estados que no satisface la condición O1.

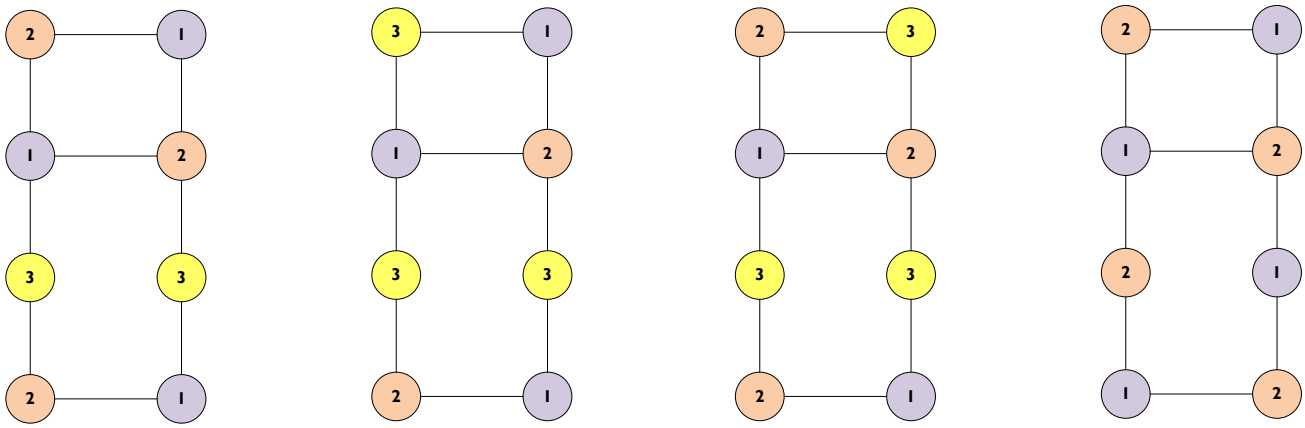


Figura 7 – Grafo y esquemas de colores

En la figura 7, si un método iterativo de solución visita el esquema factible de asignación de colores c_1, y si el límite superior a la cantidad de colores a usar es 3, entonces la solución óptima de asignación de colores usando dos colores nunca será alcanzada[64].

Es importante resaltar que el conjunto *S* de soluciones factibles no es necesariamente el conjunto de soluciones del problema original, y de hecho en varias situaciones puede ser difícil definir una estructura de vecindad que satisfaga O1. Si las soluciones del problema original deben satisfacer un conjunto de restricciones *C*, es útil en ciertos casos definir *S* como el conjunto de soluciones que satisfacen un adecuado subconjunto $C' \subset C$ de restricciones.

[64] Notemos que los esquemas C_2 y C_3 son alcanzables y vecinos de C_1, dado que en ambos casos es posible ir desde C_1 simplemente intercambiando el color de un vértice.

Cada vértice *i* del grafo del espacio de estados *G* tiene un valor *f(i)*. Si consideramos a estos valores como si fueran altimetrías, se induce una topología con valles que contienen óptimos locales. Es importante definir *f* de una manera tal que no hayan ni demasiados valles ni demasiadas mesetas. Si esto no fuera así, es muy difícil guiar la búsqueda hacia la solución óptima, es decir hacia el fondo del valle más profundo. Para el problema de coloreo del grafo, definir el valor *f(i)* de un esquema de colores *i* como el número de colores empleado en *i*, induce grandes mesetas, dado que todas las soluciones tienen un valor entero contenido en el pequeño intervalo [*f(i*),u*], donde *f(i*)* es el valor de una solución óptima y *u* es el límite superior. Como ya se ha dicho, la función objetivo *f* puede ser modificada durante la búsqueda. Estos cambios corresponden a una modificación de la topología. Durante la fase de diversificación, por ejemplo, se trata de reemplazar los valles visitados frecuentemente por cadenas montañosas y de esa forma dirigir la búsqueda hacia regiones inexploradas.

3. Computo Efectivo

En cada paso de un método iterativo de solución, hay que evaluar muchas soluciones y es importante realizar estos cómputos de manera eficiente.

Para una solución *i* y un movimiento *m*, es muchas veces más sencillo calcular $\Delta(i,m)$ definido como $f(i \oplus m) - f(i)$ en lugar de $f(i \oplus m)$. En algunos casos, es posible almacenar los valores de $\Delta(i,m)$ para todos los $m \in M(i)$. En esos casos, si un movimiento *m* puede ser aplicado a dos soluciones consecutivas *i* y *j*, ($m \in M(i) \cap M(j)$), es usualmente común que $\Delta(i,m)$ pueda ser calculado aún más fácilmente. Para algunos problemas, dada una solución *i* y un movimiento $m \in M(i)$, es difícil calcular $f(i \oplus m)$.[65]

En los casos donde todos los valores $\Delta(i,m)\ \forall\ m \in M(i)$ puedan ser almacenados, es muchas veces posible mantener una lista ordenada de estos valores con un bajo costo computacional.

Más adelante se describe una adaptación del problema de BT de encontrar un conjunto independiente de un tamaño dado *k* en un grafo con *n* vértices. Los *O(nk)* vecinos pueden ser eficientemente evaluados mediante el almacenamiento de dos listas ordenadas que contienen la información pertinente. Un paso de la BT que es encontrar la mejor vecindad y actualizar las listas ordenadas, puede ser realizado en ese caso en tiempo *O(n)*.

[65] En el problema de Ruteo de Vehículo, si uno define una solución *i* como la asignación de clientes a los vehículos, y un movimiento *m* se define como la transferencia de exactamente un cliente de un vehículo a otro, y si el objetivo es minimizar la distancia total de viaje, entonces el cálculo de $f(i \oplus m)$ es equivalente a resolver varios problemas del vendedor viajero (*travelling salesman problems)*, que como se sabe es un problema NP-difícil.

4. Uso eficiente de la memoria[66]

El uso de la memoria es una característica esencial de la BT. Las condiciones tabú pueden ser consideradas generalmente como una memoria a corto plazo que previene en cierta medida los ciclos, ahora bien, es posible utilizar la memoria de manera tal de intensificar la búsqueda en regiones "buenas" o diversificarla hacia regiones inexploradas.

En ciertas configuraciones puede ser conveniente utilizar en algún punto diferentes listas tabú, obviamente cualquier implementación que se discuta para el caso de una única lista puede ser extendido al caso de varias.

Como se ha dicho, el rol básico y fundamental de la lista tabú es evitar los ciclos. Si la longitud de la lista es demasiado corta, este rol puede no ser obtenido; e inversamente, si la lista es demasiado larga se crean demasiadas restricciones y se ha observado en la práctica que el valor promedio de la cantidad de soluciones visitadas crece con el incremento del tamaño de la lista tabú. Ahora bien, dado un problema de optimización, es muchas veces difícil o incluso imposible encontrar un valor que prevenga los ciclos y a la vez no restrinja excesivamente la búsqueda para todas las instancias del problema de un dado tamaño. Por lo tanto, una forma de evitar esto es utilizar una lista tabú de tamaño variable. Cada elemento de la lista pertenece a la misma por un dado número de iteraciones que ha sido acotado en forma máxima y mínima.

Para **intensificar la búsqueda en regiones promisorias**, debemos volver a una de las mejores soluciones encontradas hasta ese punto. Entonces podemos disminuir el tamaño de la lista tabú durante un número "pequeño" de iteraciones. En ciertos casos se pueden utilizar técnicas más elaboradas, por ejemplo algunos problemas pueden ser particionados en subproblemas, donde las soluciones óptimas de los subproblemas se combinan en una solución óptima, obviamente la dificultad principal en estos casos estriba en el hecho de encontrar esa partición "adecuada". Ahora bien, por razones de tiempo computacional, en cada paso de la BT se utilizan vecindades de tamaño adecuado y heurísticas rápidas, por lo tanto, otras formas de intensificar la búsqueda son el empleo de heurísticas más elaboradas o un incremento en la vecindad.

[66] Las estructuras de memoria en la búsqueda tabú operan mediante referencia a cuatro dimensiones principales: ***inmediatez, frecuencia, calidad*** e ***influencia***. Las memorias basadas en la *inmediatez* (lo reciente) y en la *frecuencia* se complementan entre sí. La dimensión de *calidad* se refiere a la habilidad de diferenciar el merito de las soluciones visitadas durante la búsqueda. En este contexto, la memoria se puede emplear para identificar elementos que son comunes a las buenas soluciones o a los caminos que llevan a esas soluciones. Operacionalmente, la calidad se convierte en la base para el aprendizaje en base a incentivos, donde se proveen incentivos que refuerzan las acciones que llevan a buenas soluciones, y penalidades se proveen para desalentar las acciones que llevan a malas soluciones. La cuarta dimensión, la *influencia* considera el impacto de las elecciones realizadas durante la búsqueda, no solo en calidad sino también en estructura.

Ahora bien, es también posible realizar una intensificación basada en una memoria a *largo plazo*. Cada solución o movimiento puede ser caracterizado por un conjunto de componentes. Los componentes de "buenos" movimientos o de "buenas" soluciones son memorizados. Durante la fase de intensificación, los movimientos o las soluciones son evaluados teniendo en cuenta su cantidad de "buenos" componentes, en este contexto se puede considerar a la memoria a largo plazo como un tipo de proceso de aprendizaje.

Por otro lado, para evitar que una gran porción del espacio de estados permanezca inexplorada, es importante diversificar la búsqueda. La forma más sencilla de hacer esto es realizar diferentes inicios aleatorios. Otra forma de hacer lo mismo es penalizar movimientos realizados frecuentemente o soluciones visitadas con alta frecuencia. Esta penalidad se ajusta lo suficientemente grande como para asegurar el escape de la región actual, usando una función objetivo modificada durante una dada cantidad de iteraciones, y es también posible emplear una penalidad sobre movimientos frecuentemente realizados durante todo el proceso de búsqueda.

En el caso en el que las soluciones factibles tienen que satisfacer un dado conjunto de restricciones, estas aparecen como montañas infranqueables (de altura infinita). Relajar estas restricciones y penalizar su violación se corresponde con reducir la altura de las montañas. Es entonces posible pasar esta barrera y alcanzar otro valle en unos pocos pasos. Durante esta fase de diversificación, las soluciones visitadas no son necesariamente factibles (dado que las restricciones han sido relajadas). Para obtener una solución factible nuevamente, la penalización para la violación de las restricciones relajadas se incrementa gradualmente.

Un algoritmo de BT está fuertemente basado en la interacción entre la memoria a corto plazo y la memoria a largo plazo. Ambos tipos de memoria llevan asociadas sus propias estrategias y atributos, y actúan en ámbitos diferentes. Un buen resumen de estos elementos está dado por la siguiente tabla tomada de Martí (2.003):

Tabla M.2 – Resumen de la memoria y sus características

Memoria	Atributos	Estrategias	Ámbito
Corto Plazo	Reciente	Tabú – Aspiración Lista de Candidatos	Local
Largo Plazo	Frecuente	Intensificación y Diversificación	Global

Existen otros elementos más sofisticados en la BT, sugerimos al lector interesado la consulta a la profusa bibliografía sobre el tema.

Tabla M.3 – Adaptada de Glover, Laguna y Martí[67]

Características Principales de la Búsqueda Tabú
Memoria Adaptativa Selectividad (incluyendo olvidos estratégicos) Abstracción y Descomposición (a través de memoria explícita y atributiva) *Timing*: Inmediatez de los eventos (eventos recientes) Frecuencia de los eventos Diferenciación entre memoria a corto y a largo plazo Calidad e impacto: Atractividad relativa de elecciones alternativas Magnitud de los cambios en la estructura o en las relaciones de restricción Contexto: Interdependencia regional Interdependencia estructural Interdependencia secuencial **Exploración responsiva** Incentivos y restricciones impuestas estratégicamente (*condiciones tabú* y *niveles de aspiración*) Foco concentrado en buenas regiones y buenas características de solución (*procesos de intensificación*) Caracterización y exploración de nuevas regiones promisorias (*procesos de diversificación*) Patrones de búsqueda no monotónicos (*oscilación estratégica*) Integración y extensión de soluciones (*path relinking*)

[67] González (2007)

5. Dos ejemplos de aplicación de Búsqueda Tabú

a) El problema de Colorear un grafo[68]

Dado un grafo *G = (V,E)* tenemos que encontrar un esquema de colores de sus vértices con tan pocos colores como sea posible. La única restricción es que dos vértices unidos por una arista no deben recibir el mismo color.

En otras palabras, un esquema de colores de los vértices de *G* en *k* colores es una partición del conjunto de vértices *V* en *k* conjuntos independientes V_1, …, V_k. Para definir el conjunto *S* de soluciones factibles, la única restricción ha sido relajada y la BT se emplea para encontrar en la medida de lo posible un esquema de colores en un número *k* de colores. Por lo tanto, una solución factible es cualquier partición (V_1, …, V_k) de *V* en *k* conjuntos. Si definimos $E(V_r)$como el conjunto de aristas que tienen ambos puntos terminales en V_r, el objetivo es minimizar $\sum_{r=1}^{k}|E(V_r)|$. Este es uno de los primeros problemas que se probó que era intratable (NP-difícil), por lo que usualmente se emplean heurísticas para su resolución

Una solución de valor 0 corresponde a un esquema de asignación de colores *G* en *k* colores. En un modelado eficiente de la BT[69], la vecindad *N(i)* de una solución *i*, consiste de todas las soluciones generadas a partir de *i*, mediante la siguiente modificación local: mover un vértice que es un punto final de un borde monocromático (ambos puntos terminales están en el mismo V_r) a un conjunto V_q (donde $q \neq r$). Cuando un vértice *v* se mueve de V_r a V_q, el par *(v, r)* se introduce en la lista tabú *T*. Esto implica que está prohibido introducir *v* en V_r durante $|T|$ iteraciones.

b) El problema del Conjunto Independiente Máximo[70, 71]

Dado un grafo *G = (V,E)* se debe determinar un subconjunto *X* de *V* tan grande como sea posible tal que *E(X)* = ∅ (donde *E(X)* es el conjunto de aristas que tienen sus dos extremos terminales en *X*)[72].

La BT ha sido adaptada para tratar de determinar *G*en un conjunto independiente de un dado tamaño *k*. Nuevamente la única restricción del problema ha sido relajada. Una solución es cualquier conjunto *X* de *V* de tamaño *k* y la función objetivo es |*E(X)*|. La vecindad consiste

[68] Hertz y De Werra (1.987).
[69] Comparar este modelado con el dado precedentemente para el mismo problema.
[70] Friden, Hertz y De Werra (1.989).
[71] Friden, Hertz y De Werra (1.990).
[72] Un conjunto independiente es un conjunto de vértices en un grafo tal que ninguno es adyacente a otro (no existen aristas que los conecten *E(X)* = ∅). La determinación del conjunto independiente máximo es un problema NP-completo, ver por ejemplo: Garey y Johnson (1.979)

en el intercambio de un vértice $v \in V$ con un vértice $w \in V - X$. El tamaño de la vecindad de una solución es igual a $k(n-k) \in O(nk)$ donde $n = |V|$.

Para un vértice $v \in V$, llamamos $\Gamma_X(v)$ al conjunto de vértices w en X que están unidos a v por una arista. Los vértices v en X se ordenan de acuerdo a valores no crecientes de $\Gamma_X(v)$ y los vértices $w \in V - X$ se ordenan de acuerdo a valores no crecientes de $\Gamma_X(w)$. Mantener estas dos listas ordenadas requiere un tiempo $O(n)$.

En este caso se emplean tres listas tabú:

i) La primera T_1 almacena las últimas soluciones visitadas.
ii) La segunda T_2 contiene los últimos vértices introducidos en X.
iii) La tercera T_3 contiene los últimos vértices removidos de X.

En este caso, Friden, Hertz y de Werra (1989) muestran que mediante el empleo de técnicas *hash* para implementar T_1 y la elección de un valor pequeño y constante para $|T_2|$ y $|T_3|$, es posible encontrar la mejor vecindad en la práctica en tiempo casi constante, y por lo tanto un paso de BT toma un tiempo $O(n)$ en lugar de $O(nk)$.

Por otro lado, una adaptación completamente diferente de la BT a este problema fue descripto por Friden, Hertz y De Werra (1990), en este caso se utilizó la BT para obtener cotas en un algoritmo de ramificación y acotamiento (*branch and* bound). En cada nodo del algoritmo enumerativo, se definen dos conjuntos I y O de vértices que se fuerzan a estar respectivamente dentro y fuera del conjunto máximo independiente. Sea X el mayor conjunto independiente obtenido hasta el momento, la BT se emplea para cubrir con $|I| - |O|$ *cliques*[73] tanta cantidad de vértices como sea posible en el grafo H inducido por $U = V - \{I \cup O\}$. Sea W el conjunto de vértices cubierto por los cliques. Si $|W| = |U|$ entonces ocurre una marcha atrás dado que H no puede contener un conjunto estable[74] de tamaño más grande que $|X| - |I|$. De cualquier otro modo, hemos generado un nuevo sub-problema para cada vértice $v \in U - W$ mediante la introducción de v en I y mediante la anexión en O de todos los vértices adyacentes a v.

El empleo de una técnica menos elaborada para cubrir los vértices de H por *cliques*, requiere menos tiempo, pero el número de ramas generadas en cada nodo del algoritmo de ramificación y acotamiento es generalmente mucho mayor. Se ha demostrado (Friden, Hertz y

[73] Un **clique** en un grafo no dirigido $G = (V,E)$ es un conjunto C del conjunto de vértices $C \subseteq V$, tal que para cada par de vértices en C existe una arista que los conecta. Esto es equivalente a decir que el subgrafo inducido por C es completo. El *Problema del clique* que consiste en dado un grafo $G = (V,E)$ encontrar el clique de mayor tamaño también es NP-completo, ver por ejemplo: Karp (1.972), en particular a partir de la pág. 94: Teorema Principal: todos los problemas de la siguiente lista son completos (3. CLIQUE…)

[74] o independiente.

de Werra, 1.990) que es una buena idea emplear la BT en cada nodo para la reducción de la cantidad de ramas.

3- Desarrollo: Solución al Problema de Districtalización por Búsqueda Tabú y Memoria adaptativa (Bozkaya, 1.999)

Una solución al problema de la districtalización es una colección de distritos que son mutuamente exclusivos y que colectivamente "cubren" el territorio que se "districtaliza". Esta colección de distritos será denominada un ***Plan de districtalización.***

El territorio queda dividido entonces en una determinada cantidad **D** de distritos (regiones, zonas). Los distritos están constituidos por unidades geográficas más pequeñas, denominadas ***unidades base (UB)*** que son los elementos constructivos de los distritos. Dependiendo el nivel de la districtalización, estos UB pueden ser por ejemplo, radios censales, fracciones censales, departamentos, cuadriculas, comisarías, provincias, etc.

Como ya hemos mencionado, una solución al problema de districtalización es una partición del conjunto de todas las UB en **D** subconjuntos. Si x_{ij} es una variable binaria que indica si una UB *i* ha sido asignada al distrito *j*, y si **C** es la cantidad total de todas las UB en el área a regionalizar, entonces una solución X al problema se define como:

(S.1) $$X = \{x_{ij} : i \in I, j \in J\}$$

donde *I* = {1,…,C} es el conjunto de índices para las UB y *J* = {1,…,D} es el conjunto de índices de los distritos. En otras palabras la solución completa al problema determina cuáles son las UB que están contenidas en cada distrito. Dado que cada UB puede ser asignada a uno y solo un distrito, se tiene que $\sum_{j \in J} x_{ij} = 1$.

Utilizaremos dos criterios estrictos para definir *si una solución es o no factible*, si alguna de estas condiciones se viola, el plan de districtalización es *inviable (**no factible**)*:

- Balance poblacional: La población de todos los distritos en un plan de districtalización dado, debe hallarse dentro de un rango predeterminado [P_{min}. P_{max}]. Donde $P_{min} = (1-\beta)\bar{P}$ y $P_{max} = (1+\beta)\bar{P}$, donde $\bar{P} = {}^{P}/_{D}$ es el cociente del distrito y β es la desviación porcentual permitida ± a partir de $\bar{P}$. La notación $P = \sum_{i \in I} p_i$ representa la población total que reside en el territorio conde p_i es la población de la UB i-esima.

- Contigüidad: Un plan de districtalización debe ser contiguo, es decir, cada UB dentro de un distrito debe poder ser "alcanzable" desde todas las UB en el distrito mediante el recorrido de una secuencia de UBs en ese distrito.

Desde la lógica del modelado, estas dos condiciones no son otra cosa que restricciones que una solución factible debe satisfacer. Ahora bien el algoritmo de Búsqueda Tabú (BT) que

estamos utilizando va a tratar la primera condición como una condición débil, permitiendo violaciones temporarias de la misma; mientras que tratará a la segunda condición como una restricción fuerte, lo que implica que el algoritmo de BT mantendrá **siempre** la contigüidad del plan de districtalización considerado. Una ventaja de asegurar siempre la contigüidad es que se restringe el número de soluciones posibles, al igual que se facilita el seguimiento de los componentes de cada distrito una vez que el algoritmo de BT realiza sus movimientos.

Función Objetivo:

Dado que estamos en presencia de un problema multi-criterio, usaremos una aproximación ponderada, es decir, la función objetivo será construida mediante un conjunto de múltiples términos, cada uno es uno de los criterios relevantes para el problema, ponderados mediante pesos no negativos:

(S.2) $$f(X) = w'h(X) + w_1 f_1(X) + w_2 f_2(X) + \cdots$$

en todos los términos de la función objetivo, se prefieren los menores a los mayores valores, es decir, *f*(X) debe ser minimizada.

El primer término en (S.2) es distinto del resto[75], se trata obviamente de la primera condición de factibilidad (el requerimiento de que todos los distritos estén balanceados con respecto a la población). Por lo tanto, este término mide la magnitud de la violación de la restricción poblacional.

a) Balance Poblacional

El balance poblacional es una de las dos condiciones que definen la factibilidad de la solución. Como ya hemos dicho, será tratada como una restricción débil por el algoritmo, lo que implica que se permitirá visitar soluciones temporarias que poseen distritos con poblaciones por fuera del rango dado por [P_{min}, P_{max}]. Esto se permite en la intención de que soluciones factibles de mejor calidad sean encontradas en posteriores iteraciones del algoritmo. Otro objetivo es hacer más fácil para el algoritmo dirigir su búsqueda a partes inexploradas del espacio de soluciones.

Ahora bien, dado que soluciones que vulneran este rango no son deseables, se introduce un término de penalización en la función objetivo. Se lo calcula como la cantidad

[75] En Bozkaya *et al.* (2003)se utiliza $F(x) = \sum_r \alpha_r f_r(x)$, donde $\alpha_r$es la ponderación y $f_r(x)$ es el valor de una función que asigna un valor para el criterio *r* a cualquier solución dada x, y se aclara que de todos los diferentesα, solamente α_{pop} (el α en S.3), se ajusta a 1 inicialmente y luego se permite que varíe durante la ejecución del algoritmo, mientras que todos los otros los define el usuario.

total de violaciones (si es que hay alguna) relativa al cociente del distrito $\bar{P} = {}^{P}/_{D}$, multiplicada por un parámetro ajustable α:

(S.3) $$h(X) = \alpha \frac{\sum_{j \in J} \max\{P_j - P_{max}\ ,\ P_{min} - P_j,\ 0\}}{\bar{P}}$$

donde $P_j = \sum_{i \in D_j} p_i$ es la población del distrito j. Esta última cantidad se calcula como la suma total de población de las UBs en el distrito *j* con $D_j = \{i \in I: x_{ij} = 1\}$ es el conjunto de índices de las UB para el distrito *j*.

El numerador en (S.3) es la violación total absoluta por encima y por debajo de los límites del rango permitido de población. La violación *relativa* se calcula simplemente al dividir este valor por $\bar{P}$. Si $P_j \in [P_{min}, P_{max}]$, $\forall j \in J$, entonces no hay violación y por lo tanto el termino de penalización *h(X)* es igual a cero.

El coeficiente α en (S.3) es utilizado por el algoritmo para ajustar la frecuencia de los cambios entre las soluciones no factibles y las factibles. Inicialmente α = 1, y se revisa continuamente para su ajuste durante el transcurso del algoritmo. A este parámetro se lo multiplica o divide por 2, dependiendo de la historia de las soluciones visitadas en las pasadas iteraciones, según el siguiente criterio:

- Si por lo menos $\bar{\mu}$ de las últimas μ soluciones visitadas eran factibles, entonces α←α/2,
- De otro modo, si por lo menos $\bar{\mu}$ de las últimas μ soluciones visitadas eran no factibles, entonces α←2α,
- De otro modo, dejar α inalterado.

Estos controles se realizan cada μ iteraciones; $\bar{\mu}$ especifica exactamente cuántas soluciones deben ser factibles o no para que α deba ser ajustada. En esencia, lo que α hace es incrementar la penalidad por las violaciones si todas o casi todas las soluciones visitadas son no factibles (asumiendo $\bar{\mu}$ lo suficientemente largo). En forma análoga, disminuye la penalidad si todas o casi todas las últimas soluciones son factibles. Es importante notar que $\bar{\mu}$ debe ser más grande que μ/2 de manera tal que las dos primeras condiciones no se satisfagan simultáneamente. Con este esquema de ajustar α, se permite que el algoritmo seleccione entre soluciones factibles y no factibles en forma continua. Si el algoritmo visita demasiadas soluciones no factibles sucesivamente, se favorecen las soluciones factibles mediante la duplicación de la penalidad.

A modo de ejemplo, si D = 10, P = 800000, $\bar{P}$ = 80000, P_{max} = 100000, P_{min} = 60000 (es decir, β = 0.25), y α = 1

Tabla S.1 – Ejemplo de cálculo del término de penalización para la violación del balance poblacional

Distrito	**P_j**	**Violación**	**Penalidad**[76]
1	85000	-	
2	73000	-	
3	58000	2000	
4	70000	-	
5	87000	-	
6	105000	5000	
7	87000	-	
8	104000	4000	
9	56000	4000	
10	75000	-	
TOTAL	**800000**	**15000**	***h(X)*= 1 •15000/80000 = 0.1875**

Por lo tanto, el termino de penalización *h(X)* es parte de la función objetivo *f(X)* y es no nulo dondequiera que haya una violación al rango permitido [P_{min}, P_{max}]. El término *h(X)* se multiplica además por una ponderación *w*' para ajustar su balance con relación a los demás términos en *f*.

b) **Compacidad**

Es el segundo criterio que forma parte de la función objetivo y que está asociado con el término $f_1(X)$ y ponderación w_1. Siguiendo la tesis doctoral de Bozkaya (1.999) son dos las medidas que se consideran al evaluar la compacidad de una solución:

[76] Notemos primero que el "0" en el cálculo de la determinación de la violación absoluta máxima es necesario para asegurar que los valores no sean negativos (lo que podría ocurrir si el valor del distrito estuviera completamente contenido en el intervalo [P_{min}, P_{max}], el cálculo detallado de valores que definen la tabla S.1 es:

Distrito	**P_j**	**P_j-P_{max}**	**P_{min}-P_j**	**Constante**	**Violación** (max{ **P_j - P_{max} , P_{min}-P_j** , 0 }
1	85000	-15000	-25000	0	0
2	73000	-27000	-13000	0	0
3	58000	-42000	2000	0	2000
4	70000	-30000	-10000	0	0
5	87000	-13000	-27000	0	0
6	105000	5000	-45000	0	5000
7	87000	-13000	-27000	0	0
8	104000	4000	-44000	0	4000
9	56000	-44000	4000	0	4000
10	75000	-25000	-15000	0	0

i- La longitud total de los perímetros "interiores" de los distritos,

ii- La comparación entre el perímetro del distrito con el perímetro de un círculo con la misma superficie que el distrito.

Medida i: se trata de una función de la longitud total de los perímetros de los distritos. El objetivo es minimizar esta función de manera tal que el algoritmo sea forzado a definir distritos con fronteras que sigan bordes rectos. Específicamente, se calcula tomando la longitud total de los perímetros *interiores* de los distritos y escalando el valor por *R* (el perímetro del área en estudio):

(S.4) $$f_1(X) = C_1(X) = \frac{\frac{1}{2}(\sum_j R_j - R)}{R}$$

donde R_j es el perímetro del distrito *j* y está dado por:

(S.5) $$R_j = \sum_{i \in D_j} r_i - \sum_{i.k \in D_j} \sum_{i \neq k} t_{ik}$$

donde r_i es el perímetro de la UB *i*, y t_{ik} es la longitud de la frontera común entre la UB *i* y la *k*. Por lo tanto, R_j se calcula primero sumando el perímetro de todas las UB en el distrito *j*, y luego se restan todos los perímetros de las UB que se cuentan dos veces en $\sum_{i \in D_j} r_i$. El numerador del cociente en (S.4) es la longitud total de los perímetros internos, que es la mitad de la diferencia entre la suma de todos los perímetros de los distritos y *R* (1/2 debido al doble conteo). Al calcular C_1(X), el valor de compacidad de *X*, se utiliza *R* como un factor de escala constante.

Por ejemplo, tomando el partido de La Plata en la Provincia de Buenos Aires, con un perímetro de aproximadamente 165.4 km, 704 radios censales (UBs) y 47 fracciones censales (distritos, representados con su frontera en línea puntuada), tenemos:

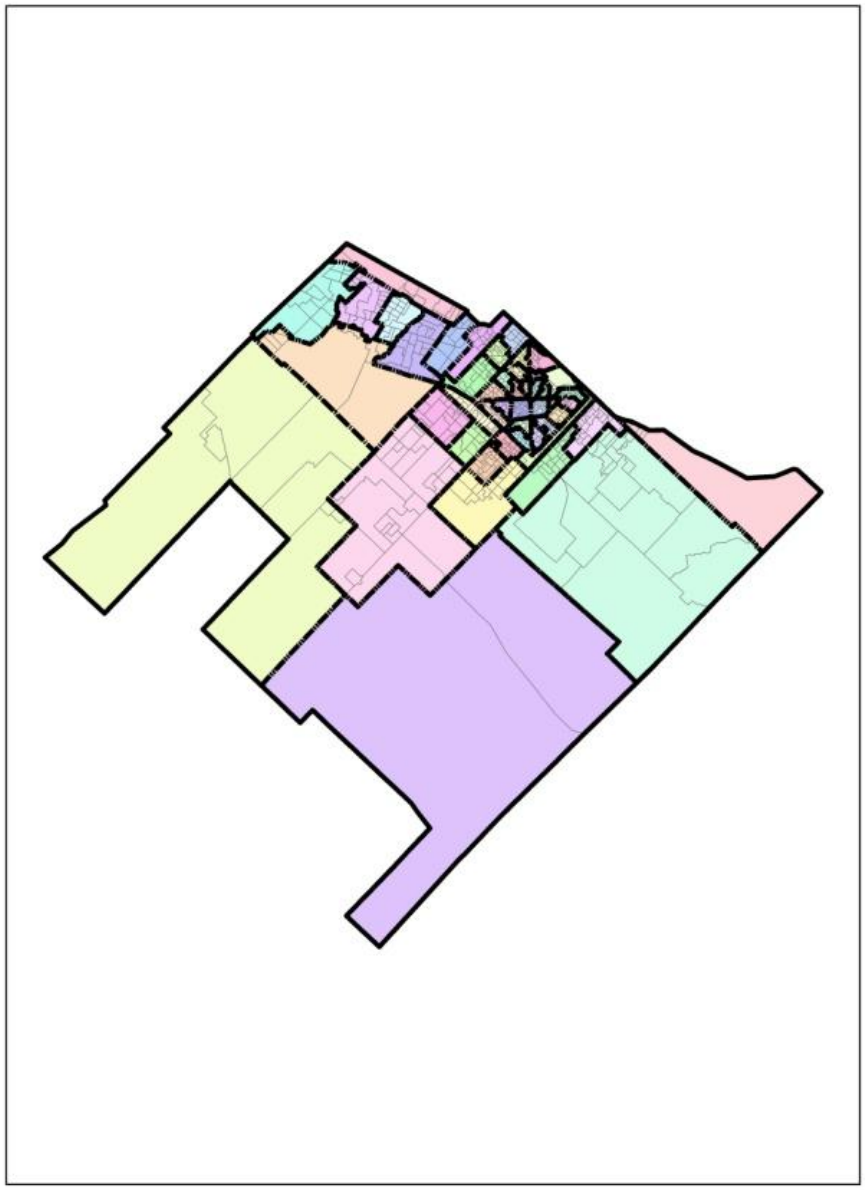


Figura 8 - Medida de Compacidad i, radios censales del partido de La Plata

En el plan de districtalización de la figura 8, las líneas puntuadas corresponden justamente a los perímetros interiores de los distritos. Las líneas sólidas se corresponden con el perímetro total de la zona en estudio y su valor es *R* (165.4 km). Por lo tanto, la compacidad de este plan se calcula como la razón de la longitud total de las líneas punteadas en relación a *R*. En este caso puntual se tiene:

$$C_1(X) = \frac{\frac{1}{2}\left(\sum_j R_j - R\right)}{R} = \frac{\frac{1}{2}(709.4\ km\ - 165.4\ km)}{165.4\ km} = 1.644$$

Medida ii: Se trata ahora de considerar el perímetro del distrito vs. El perímetro del círculo con la misma superficie. En este caso, la compacidad del distrito *j* se define como:

(S.6) $$C_{2j}(X) = 1 - \frac{2\pi\sqrt{A_j/\pi}}{R_j} = 1 - \frac{2\sqrt{\pi A_j}}{R_j}$$

donde $A_j = \sum_{i \in D_j} a_i$ es el área del distrito *j*, y a_i es el área de la unidad base *i*. Asimismo, el numerador de esta razón es el perímetro del círculo que tiene superficie A_j y por lo tanto tiene radio $\sqrt{A_j/\pi}$.

Ahora bien, (S.6) indica la compacidad de **cada** distrito. La Pregunta obvia es como combinar estas medidas individuales de compacidad en un valor agregado para el plan completo. Dos aproximaciones inmediatas involucran el promedio simple de los valores individuales de compacidad o el valor máximo de los valores individuales. Con el promedio simple, un distrito puede ser altamente no-compacto, pero esto puede quedar enmascarado por una determinada cantidad de distritos altamente compactos. Por otro lado, con la aproximación minimax, se tiene la garantía de que no existe un distrito altamente no compacto, pero al mismo tiempo se pierde potencia en el valor global.

Hay quienes han planteado como tercer alternativa minimizar el promedio de los *cuadrados* de (1 + valores individuales de compacidad), que a su vez penaliza distritos no compactos en mayor medida que distritos compactos (los valores compactos se incrementan en 1, por lo que al elevarlos al cuadrado se magnifica la distancia relativa del valor máximo). Entre estas alternativas y nuevamente siguiendo a Bozkaya (1.999), se elije por simplicidad el promedio simple. Por lo tanto la compacidad de un plan de districtalización es:

(S.7) $$f_1(X) = C_2(X) = \frac{1}{D}\sum_{j \in J} C_{2j}(X) = \frac{1}{D}\sum_{j \in J}\left(1 - \frac{2\pi\sqrt{A_j/\pi}}{R_j}\right)$$

Nuevamente, a modo de ejemplo, para el plan de districtalización de la figura 8, se tiene:

Tabla S.2 – Ejemplo de determinación de la compacidad para el plan de districtalización de la figura 8, usando la medida ii

Distrito	Cantidad de UBs	A_j[km^2]	R_j[km]	C_{2j}
01	13	2,098	7,154	**0,282**
02	15	2,535	8,218	**0,313**
03	17	1,203	6,668	**0,417**
04	14	1,207	6,875	**0,434**
05	16	0,552	3,552	**0,259**
06	17	0,849	4,919	**0,336**
07	13	0,464	3,277	**0,263**
08	13	1,254	6,595	**0,398**
09	14	1,096	5,078	**0,269**
10	12	1,206	5,301	**0,265**
11	12	0,844	4,540	**0,283**
12	12	0,548	4,001	**0,344**
13	21	1,205	4,245	**0,083**
14	13	1,428	5,202	**0,186**
15	13	1,257	6,810	**0,416**
16	9	1,267	6,235	**0,360**
17	12	1,045	4,945	**0,267**
18	19	1,636	5,501	**0,176**
19	10	0,988	4,524	**0,221**
20	16	1,872	7,177	**0,324**
21	11	1,483	6,473	**0,333**
22	13	1,365	6,890	**0,399**
23	26	6,067	13,022	**0,329**
24	21	3,578	17,643	**0,620**
25	15	2,244	7,329	**0,275**
26	18	3,670	8,452	**0,197**
27	12	4,252	8,412	**0,131**
28	18	8,145	11,522	**0,122**
29	17	3,443	10,851	**0,394**
30	18	4,899	10,969	**0,285**
31	20	2,197	6,552	**0,198**
32	15	2,791	9,805	**0,396**
33	16	5,398	13,399	**0,385**
34	12	6,212	11,981	**0,263**
35	17	10,285	16,850	**0,325**
36	16	5,676	10,610	**0,204**
37	18	9,030	18,584	**0,427**
38	11	10,195	21,200	**0,466**
39	17	16,270	24,055	**0,406**
40	9	40,863	34,796	**0,349**

41	8	187,603	86,261	**0,437**
42	28	66,061	39,971	**0,279**
43	2	272,076	84,540	**0,308**
44	17	17,323	19,752	**0,253**
45	22	8,690	15,770	**0,337**
46	21	134,278	49,960	**0,178**
47	5	32,557	33,008	**0,387**

Por lo que $C_2(X) = 0.310$ para este plan de districtalización y esta métrica de compacidad.

c) **Cambios con respecto a un plan de districtalización existente**

El objetivo en este caso es minimizar los cambios en relación a planes ya existentes. Para esto se utiliza una medida que mide la *similitud* entre dos planes de districtalización que se comparan. Se ha elegido una medida que varía entre 0 y 1, donde 0 se obtiene cuando los dos planes son idénticos (perfectamente similares), y valores más altos indican aumento en la distancia entre la perfecta similitud y la absoluta disimilaridad. El término de la función objetivo asociado con esta medida es $w_2 f_2(X)$.

Una forma obvia de reconocer la similitud perfecta es observar las UBs componentes de cada distrito, si todas las UB en cada distrito del primer plan de districtalización se encuentran también en el segundo plan, ambos planes son idénticos.

En nuestro caso, para medir la similitud entre dos planes de districtalización, se calcula para cada distrito en el plan base, la superficie más grande dentro de ese distrito que se forma como resultado de la superposición. El índice de similitud se calcula como el cociente de la suma de todas las superposiciones dividida por la superficie total del área en estudio. Para hacer el índice *minimizable*, se sustrae este valor de 1 (por lo que en esencia se trata de un índice de ***dis-similitud*** más que de similitud):

(S.8) $$f_2(X) = 1 - \frac{\bar{A}_1 + \cdots + \bar{A}_n}{A}$$

donde $\bar{A}_j$ es el área de la mayor porción del distrito j que se forma como resultado de la superposición. Para este término de la función objetivo, el plan base es el plan de districtalización actualmente en funciones. Una ventaja clara de esta medida de similitud es que permite que la cantidad de distritos en los dos planes que se comparan sean diferentes.

Por ejemplo, si tomamos la porción Sur Oeste del Partido de La Plata de la Provincia de Buenos Aires (ver Figura 8) y consideramos las 23 UBs (radios censales) que configuran los 3 distritos (fracciones censales) originales con A = 509,2 km^2, tenemos:

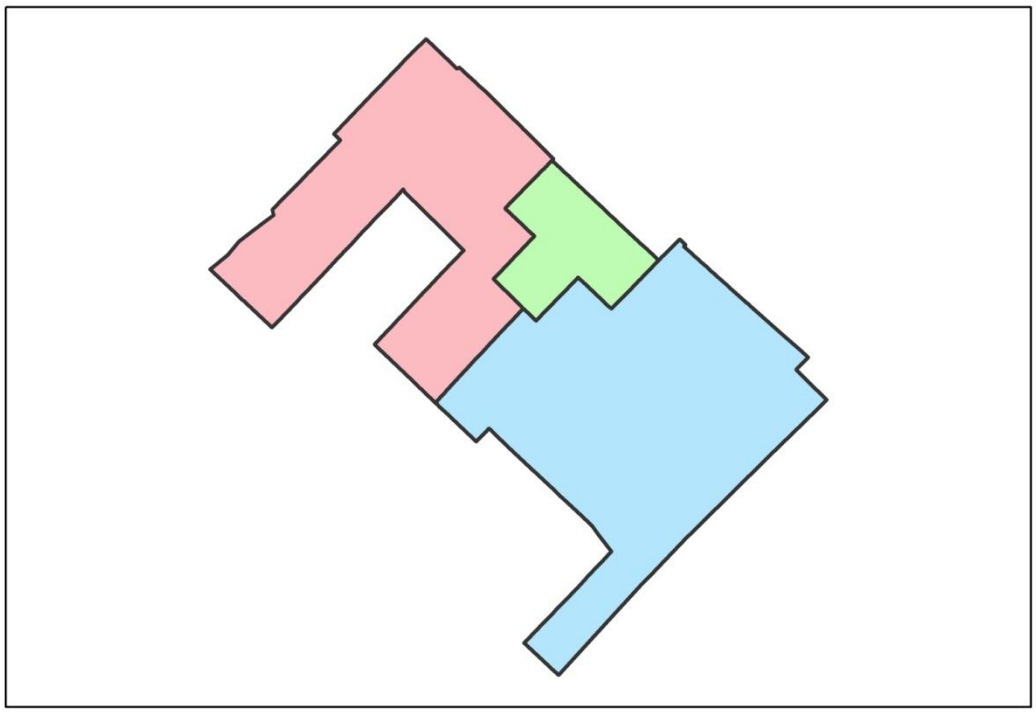


Figura 9 - Plan de districtalización original (3 distritos)

Si consideramos ahora una districtalización alternativa, dada por:

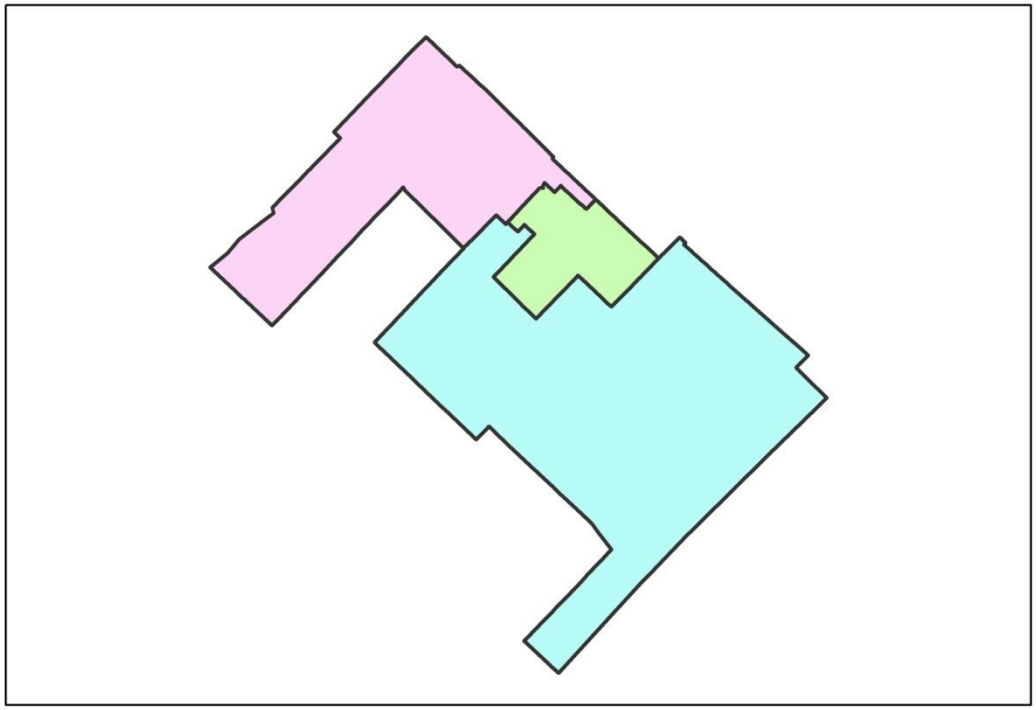


Figura 10 - Plan de districtalización alternativa (3 distritos)

Y si ahora intersecamos este plan con el original, y marcamos las porciones con mayor superficie, tenemos:

Figura 11 - Plan de districtalización alternativo y porciones $\bar{A}_j$

En este caso, tenemos:

Tabla S.3 – Superficies de las porciones $\bar{A}_j$ de mayor tamaño

Distrito	$\bar{A}_j$[km^2]
01	43,6
02	270,0
03	139,4

Por lo tanto $f_2(X) = 0{,}1$ (notemos que si ambos planes fueran idénticos $f_2(X) = 0$), por lo que el valor obtenido confirma la impresión visual de que ambos planes son bastante similares.

d) **Integridad de Comunidades**

En este caso, la idea es análoga a la anterior, pero el "plan base" contra el que se compara es ahora el "plan de comunidades", donde el área en estudio se divide en regiones que representan las diferentes "comunidades". Esta medida fuerza a las UBs que son parte de una comunidad a ser asignadas al mismo distrito en la medida de lo posible, y consecuentemente reduce el número de comunidades que son divididas en dos o más distritos.

Por lo tanto, la aproximación computacional es idéntica al caso anterior. La única diferencia es que los números de distritos en el plan base y en el plan que se está comparando pueden (y de hecho generalmente lo son) ser distintos[77]. Y consecuentemente en esos casos, no será posible obtener un puntaje óptimo. Si el número de distritos excede el de comunidades, es inevitable que algunas comunidades sean divididas entre distritos, pero en la medida que las porciones de comunidad que cada distrito tiene sean grandes, mejor será el puntaje del plan de districtalización (más pequeño). En resumen, si el criterio se minimiza, es razonable esperar que el plan resultante tenga grandes bloques de la misma comunidad juntos y en líneas generales las comunidades serán divididas en menor cantidad de distritos.

e) **Proporcionalidad**

Este es un punto importante en la districtalización política aunque no tanto en otras aplicaciones (salud, educación, justicia, marketing, etc.). La *proporcionalidad* implica que la cantidad de bancas que un partido gana en una elección debe ser en líneas generales igual a la proporción de votos que ha obtenido ese partido. Si bien en nuestra aplicación no haremos uso de este requisito, si fuera necesaria su implementación es sencilla, ver por ejemplo Bozkaya (1.999) o Bozkaya *et al.* (2.003) .

Notemos por otro lado, que tanto en la formulación de Bozkaya *et al*. (2.003) o en la de Bozkaya *et. al* (2010), se propone el modelado de los criterios para la districtalización utilizando una función objetivo ponderada de la forma:

$$F(X) = \alpha_{pop} f_{pop}(x) + \alpha_{comp} f_{comp}(x) + \alpha_{soc} f_{soc}(x) + \alpha_{sim} f_{sim}(x) + \alpha_{int} f_{int}(x)$$

dondeα_i representa la ponderación elegida para el criterio de districtalización i-ésimo, y $f_i(x)$ representa el valor cuantificable de dicho criterio. Estos criterios son, los antedichos de balance poblacional, compacidad, homogeneidad socio-económica[78], similitud con planes existentes, e integridad de comunidades. En este caso, cada criterio se mide utilizando una función $f_i(x)$ que se normaliza de manera tal de tomar valores en el rango [0,1], donde se prefieren los valores más pequeños y donde:

$$f_{pop}(x) = \left(\sum_{j \in J} max\left\{P_j(x) - (1+\beta)\bar{P}, (1-\beta)\bar{P} - P_j(x), 0\right\}\right) \Big/ \bar{P}$$

[77] Es decir la cantidad de distritos y de comunidades no es necesariamente la misma.
[78] No utilizado en la formulación de Bozkaya (1.999), y que obviamente no es necesariamente aplicable a todo ejercicio de districtalización.

$$f_{comp1}(x) = \left(\sum_{j\in J} R_j(x) - R\right)/(2R)$$

$$f_{comp2}(x) = \sum_{j\in J}\left(1 - \frac{2\pi\sqrt{A_j(x)/\pi}}{R_j(x)}\right)/m$$

$$f_{soc}(x) = \left(\sum_{j\in J} S_j(x)\right)/\bar{S}$$

$$f_{sim}(x) = 1 - \left(\sum_{j\in J} O_j(x)\right)/A$$

$$f_{int}(x) = 1 - \frac{\sum_{j\in J} G_j(x)}{\sum_{j\in J} P_j(x)}$$

en todos los casos *J* representa el índice de distritos con $|J| = m$. $P_j(x)$ indica la población del distrito j, y *ϐ* es la desviación máxima permitida a partir del promedio de población por distrito $\bar{P}$. Asimismo, $A_j(x)$ y $R_j(x)$ denotan respectivamente el área y perímetro del distrito *j*, y *A* y *R* corresponden al área y perímetro de toda el área en estudio. $S_j(x)$representa la desviación estándar interna (dentro de cada distrito) de cualquier cantidad que se emplee como *proxy* para la homogeneidad socioeconómica, mientras que $\bar{S}$ representa el promedio de dicha cantidad. Por último, $O_j(x)$ representa la superficie de la mayor porción que se superpone del distrito *j* en comparación con el plan existente, y $G_j(x)$ representa la población de la mayor comunidad representada en el distrito *j*.

Características del Algoritmo

1. *Write down the problem.*
2. *Think real hard.*
3. *Write down the solution.*

"***The Feynman Algorithm***"

a) **Construcción de la solución inicial**

Siguiendo a Bozkaya (1.999) el algoritmo primero construye una solución inicial y luego realiza iteraciones de movimientos de Búsqueda Tabú (BT) para mejorar el plan de districtalización. El método de construcción de la solución inicial es en este caso la heurística dada por Vickery (1.961), primero se selecciona una UB semilla para inicializar un distrito, luego se extiende gradualmente el distrito mediante la anexión de una de sus unidades adyacentes. El distrito está completo cuando su población alcanza $\bar{P}$ por primera vez o cuando no hay unidades adyacentes disponibles. Si el número de distritos creados es más grande que *D* (la cantidad total de distritos necesarios), se los reduce combinando iterativamente el distrito menos poblado con su vecino de menor población. Por otro lado, si el número de distritos creados es más quico que *D*, se incrementa gradualmente la cantidad de distritos dividiendo en dos el distrito más poblado, preservando la contigüidad. Al final de este proceso se tiene una solución inicial X_0 de *D* distritos contiguos, algunos de los cuales pueden ser inviables en relación al requisito de población:

INITSOLN[79]

```
1. j := 0
2. Repetir hasta que cada UB sea asignada a un distrito
   2.1j := j + 1
   2.2 Elegir una UB al azar como semilla de acuerdo con:
       2.2.1 Construir la lista de todas las UBs que AUN NO han sido
       asignadas a un distrito, Y que están también sobre la periferia
       de la región que se districtaliza. Si esta lista es Vacía, ir
       al Paso 2.2.3
       2.2.2 Elegir una UB al azar de la lista construida en el paso
       2.2.1 y luego ir al Paso 2.3
       2.2.3 Construir la lista de todas las UBs NO asignadas
       2.2.4 Seleccionar al azar una UB de la lista construida en el
       paso 2.2.3
   2.3 Asignar la semilla encontrada en el Paso 2.2 al distrito j.
   2.4 Crear el distrito j agrupando alrededor de la semilla de
       acuerdo con:
       2.4.1 Identificar una UB "pivote" entre las UBs que ya se
       encuentran asignadas al distrito j y que NO han sido utilizadas
       aun como pivote. Marcar el pivote como "utilizado". Si no
       existe pivote disponible, la construcción del distrito j está
       completa; ir al paso 2.1
```

[79] En todos los casos mantendremos los nombres dados por Bozkaya (1.999)

2.4.2 Asignar los vecinos de la UB pivote al distrito j hasta que la población del distrito P_j alcance o supere $\bar{P}$ por primera vez.
2.4.3 Si todos los vecinos del pivote están asignados y aun así Pj $<\bar{P}$, repetir los pasos 2.4.1 y 2.4.2. En cualquier otro caso, la construcción del distrito j está terminada; ir al paso 2.1

3. Si j = m, PARAR. Else si j > m, repetir los pasos 3.1-3.4 hasta que j se reduzca a m. Si j < m dividir el distrito más poblado en dos distritos de aproximadamente la misma población y repetir el proceso hasta que j se incremente hasta m.
3.1 Encontrar el distrito q con la menor población (rompiendo los empates en forma arbitraria).
3.2 Encontrar el distrito l ≠ q, como el distrito menos poblado vecino de q.
3.3 Combinar los distritos q y l en un único distrito
3.4j := (j-1)

En esencia, este módulo primero toma semillas a partir de las UBs y luego agrupa otras UBs alrededor de estas semillas hasta formar un distrito por semilla. La asignación de UBs a un distrito (semilla) se frena cuanto el total de población del distrito alcanza o excede $\bar{P}$por primera vez. Este proceso se repite hasta que no existen UBs sin asignar. Es posible que este algoritmo genere *"enclaves"* (un conjunto de UBs o un distrito completamente incluido por una única UB o distrito) y áreas residuales, y por lo tanto puede no ser posible formar exactamente *m* distritos en el Paso 2. Si este es el caso, el número de distritos será usualmente mayor que *m.* Lo que se trata en el Paso 3, donde el número de distritos se reduce a *m*, mediante la combinación del distrito de menor población con su vecino de menor población. Notemos especialmente que se mantiene la contigüidad en todos los pasos del algoritmo INITSOLN.

Obviamente existen diferentes formas de crear planes de districtalización iniciales, y mientras que el resultado de INITSOLN es un plan que respeta la contigüidad, no necesariamente se obtienen valores adecuados de compacidad. Asimismo, es probable que algunos distritos tengan poblaciones fuera del rango [P_{min}, P_{max}] y que no se cumplan los otros criterios. Pero es importante recordar que el objetivo de INITSOLN no es producir un plan de districtalización óptimo sino una solución inicial lo más rápidamente posible, para que luego pueda actuar el algoritmo de BT. Es justamente la tarea de la BT arreglar cualquier deficiencia de la solución inicial.

b) Movimientos de la Búsqueda Tabú y Vecindades

Una vez que se ha construido la solución inicial, el siguiente paso es realizar una búsqueda iterativa del espacio de soluciones, basándose en los principios de la búsqueda tabú.

En cuanto a los movimientos de la BT, el algoritmo se mueve de una solución a la siguiente implementando dos tipos de movimientos:

- Movimientos Tipo 1: Transferir una única UB *i* del distrito *j* a un distrito vecino *l*. Usaremos la notación (índice de la UB, desde, hacia) = (*i, j, l*)

- Movimientos Tipo 2: Intercambiar dos UBs *i* y *k* entre los distritos *j* y *l*. Usaremos la notación (índice_UB_1, índice_UB-2, desde_1, hacia_1) = (*i, k, j, l*)

Asimismo $M_t^1(X) = \{(i,j,l)\}$ es el conjunto de todos los movimientos Tipo 1 elegibles en la iteración *t*, mientras que X_{ijl} denota la solución que resulta de realizar el movimiento (*i, j, l*) sobre la solución actual *X*. Por otro lado, $M_t^2(X) = \{(i,k,j,l)\}$ y X_{ikjl} se definen en forma análoga para los movimientos Tipo 2.

Los movimientos Tipo 1 y 2 se utilizan para construir la vecindad de la solución actual en cada iteración, específicamente $N_1(X)$ es la vecindad que contiene todas las soluciones resultantes de todos los movimientos Tipo 1 posibles realizados sobre *X*, mientras que $N_2(X)$ es la vecindad análoga para el caso de los movimientos Tipo 2. Por lo tanto:

(S.14) $$N_1(X) = \left\{X_{ijl}: (i,j,l) \in M_t^1(X)\right\}$$

(S.15) $$N_2(X) = \left\{X_{ikjl}: (i,k,j,l) \in M_t^2(X)\right\}$$

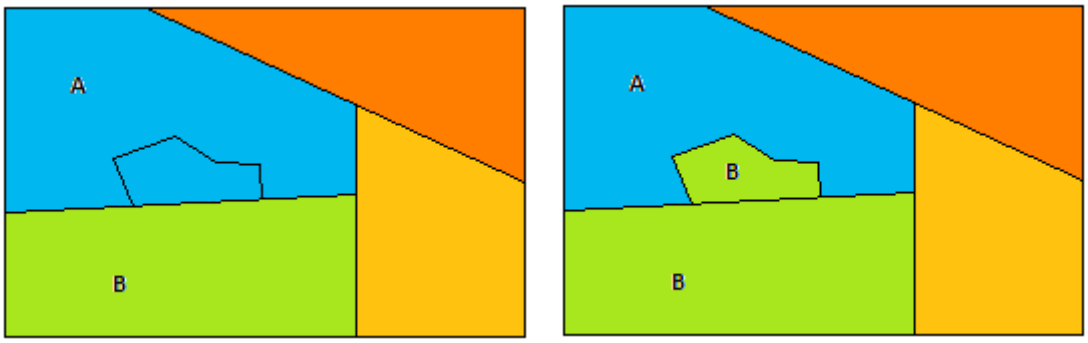


Figura 12 – Ejemplo de ***Movimiento Tipo 1***, transfiriendo una única UB del distrito A al distrito B adyacente (cada distrito se ha representado con un color diferente).

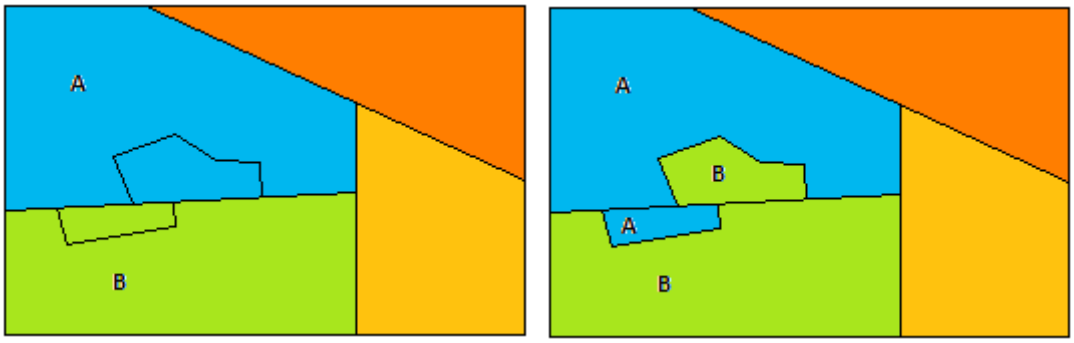


Figura 13 – Ejemplo de ***Movimiento Tipo 2***, intercambio de dos UBs entre dos distritos adyacentes A y B.

Ahora bien, si la UB *i* que se transfiere o intercambia posee enclaves, la realización del movimiento genera una discontigüidad (ver figura 14). El algoritmo puede manejar esta situación transfiriendo todos los enclaves de la UB junto con la UB *i* (si esto no fuera posible, la UB *i* debe permanecer asignada al mismo distrito durante toda la duración de una corrida BT, lo que obviamente afectará las posibilidades de alcanzar ciertas soluciones). Obviamente este no es un problema serio, dado que en lo que respecta a la composición de los distritos, la propia unidad *i* y todos sus enclaves deben pertenecer necesariamente al mismo distrito. La única excepción a esta regla es cuando las UB completamente contenidas tienen la suficiente población como para formar un distrito por sí mismas diferente al de la UB que las contiene. En este caso, solamente la UB *i* se transfiere o intercambia.

Ahora bien, notemos que con un adecuado pre-procesamiento de datos, los enclaves desaparecen del problema. Si por su tamaño la UB completamente contenida no posee la cantidad de población necesaria para transformarse en un distrito, tanto ella como su UB contenedora siempre deberán formar parte del mismo distrito, por lo tanto, cualesquiera dos unidades que *deban* ser parte del mismo distrito en la solución final pueden ser fusionadas al inicio, y cualquier UB enclave puede ser fusionada con su contenedora (a menos que tenga una población lo suficientemente grande como para constituirse en un distrito ella misma).

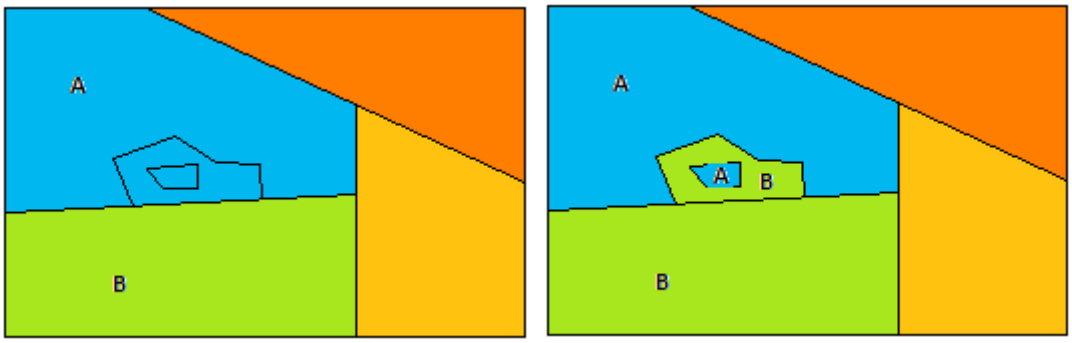


Figura 14 – Discontigüidad generada como resultado de la transferencia de UBs con enclaves

El algoritmo de BT propuesto por Bozkaya (1.999) y Bozkaya *et. al*(2.003). primero utiliza la vecindad $N(X) = N_1(X)$ hasta que no es posible mejorar la mejor solución por las siguientes *T* iteraciones. En ese momento se cambia a $N'(X) = N_1(X) \cup N_2(X)$ para el resto de la corrida. Se trata de no utilizar *N*'(X) en la medida de lo posible dado que el conjunto extendido *N*'(X) es computacionalmente mucho más costoso de procesar.

La razón para usar dos vecindades $N_1(X)$ y $N_2(X)$ es que los movimientos Tipo 1 no son, como regla general, lo suficientemente potentes para identificar soluciones de buena calidad. Los movimientos Tipo 2 funcionan mucho mejor, pero son computacionalmente más costosos, es por eso que se realizan dos "pasadas" en el algoritmo, la primera usando solamente movimientos del Tipo 1, mientras que en la segunda pasada se emplean ambos tipos de movimientos.

Por otro lado, mantener la contigüidad es una tarea sumamente importante mientras se implementan los movimientos.

Si definimos a una *unidad base de frontera (UBF)* como la Unidad Base que es adyacente a por lo menos una UB de otro distrito, entonces una condición necesaria para la contigüidad es que una UB que se transfiere o intercambia ***debe ser una UBF***. Esto es, ***UBs del "interior" de un distrito no pueden ser transferidas o intercambiadas sin violar la contigüidad***.

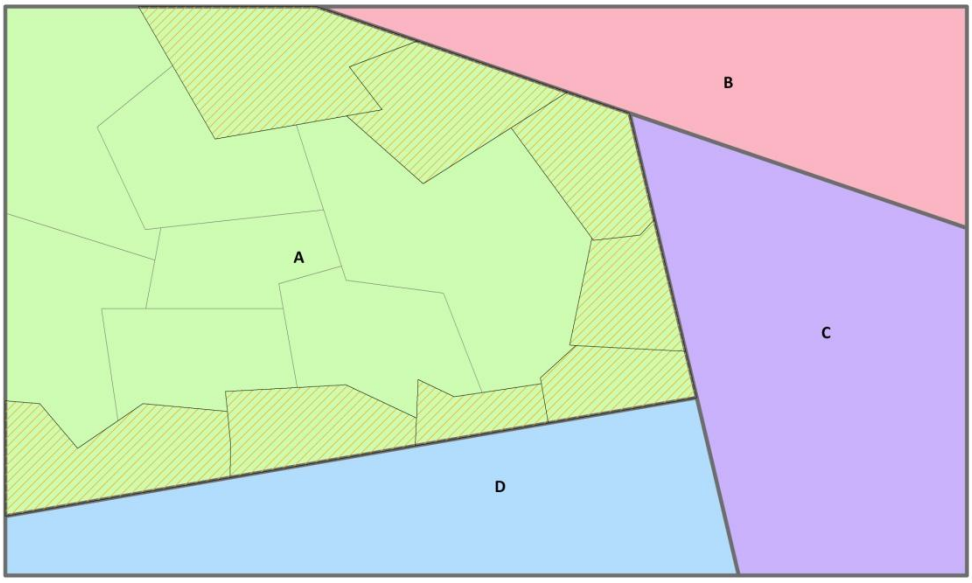


Figura 15 – Unidades base de frontera (UBFs) para el distrito A

Desafortunadamente, como muestra la figura 15 esta no es una condición suficiente:

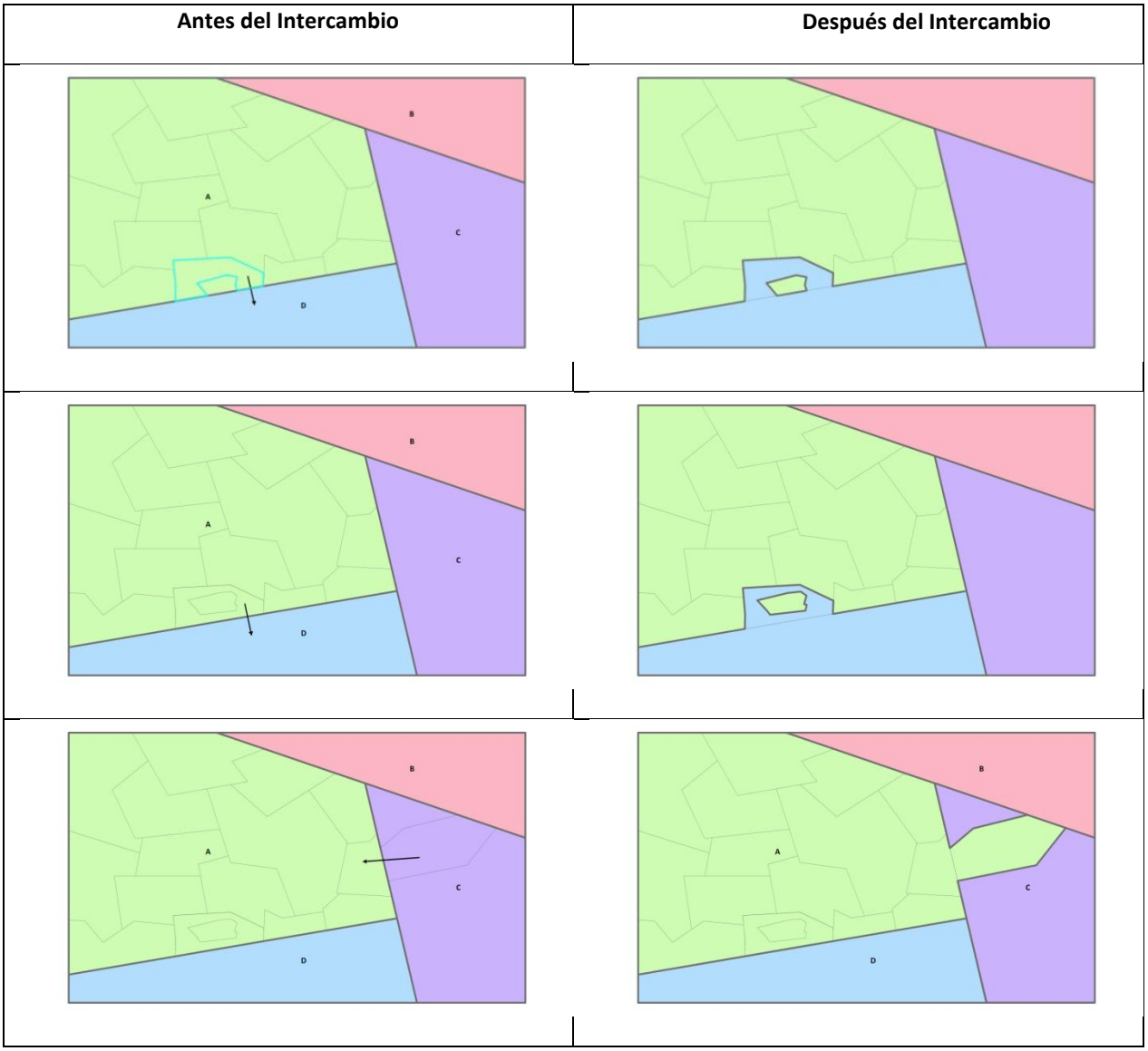


Figura 16 – Soluciones no contiguas que pueden ocurrir como resultado de estas movidas

Notemos que aun cuando los enclaves fueran pre-procesados y eliminados, aún es posible encontrar casos como el último de la figura 16. Como resultado, la condición necesaria se utiliza solamente para limitar la lista de UBs elegibles para ser transferidas o intercambiadas. Cada UB en esta lista debe aún ser sujeta a un control de contigüidad. Si una UB o un par de UBs fallan en el test de contigüidad, la solución resultante se excluye de la vecindad de la solución. Bozkaya (1.999) implementa este control para las movidas Tipo I (*i, j, l*) mediante el algoritmo ISCONTIGUOS que se detalla más adelante, y que nosotros hemos modificado en nuestra implementación.

Por otro lado, si dos UBs *i* y *k* se definen como *adyacentes por puntos* si su intersección es un conjunto finito de puntos (en oposición a un conjunto de segmentos de línea), las UBs denominadas “1” y “2” en la siguiente figura son adyacentes por puntos conforme a este

criterio. Cuando esto ocurre, t_{ik} que es la longitud de la frontera común entre las UBs *i* y *k*, es igual a cero, aun cuando las dos UBs se "tocan". Esta definición es necesaria dado que el algoritmo ISCONTIGUOS de Bozkaya (1.999) opera sobre un subconjunto *L* de las UBs de los distritos fuente: trabaja sobre el conjunto de UBs en el distrito fuente *j* que son adyacente **O** adyacentes por puntos a la UB *i*, que es la UB que está siendo transferida o intercambiada. La idea es controlar si *L* es contiguo después de que la UB *i* se remueve del distrito *j*. Dado que la UB *i* es de hecho también una UBF, el control de contigüidad necesita realizarse solamente para el distrito fuente *j*. Esto es así dado que el distrito objetivo *l* nunca puede hacerse no contiguo al ganar la UBF *i* del distrito fuente.

ISCONTIGUOUS(i,j,l)

1. Sea L = {k: k ∈ D_j, UBs k e i son adyacentes o adyacentes por puntos}.
2. Particionar L en subconjuntos contiguos $L_1,\ldots,L_q$.
3. Si q = 1, devolver VERDADERO. Esto es la UB i puede ser transferida del distrito *j* al distrito *l* sin destruir la contigüidad del distrito *j*.
4. ELSE, comprobar si los L_js pueden ser combinados en un único conjunto contiguo, utilizando las adyacencias entre los UBs L y $D_j \setminus (L \cup \{i\})$. Si es posible, devolver VERDADERO.
5. ELSE, devolver FALSO. Es decir, la transferencia de la UB i del distrito j al distrito l destruye la contigüidad del distrito j.

Notemos que en nuestra propia implementación haremos uso de otro algoritmo que testea la contigüidad de un conjunto de UBs y hace uso de la matriz de Adyacencias de las Unidades Base, esencialmente[80]:

```
ISCONTIGUOUS'(i,j,l)
#L = listado de índices a controlar, misma definición que en el
algoritmo de Bozkaya.
#W = matriz de adyacencia (wij = -1 si i y j no son adyacentes, 0 si
#son adyacentes por puntos y tij (longitud de la frontera común) si
son # adyacentes)
Definir L'= [L[0]]
# Se crea una lista con el primer elemento de L, esta lista va a
crecer
#con cada anexión contigua
for i = 0 to len(L) - 2
   L'' = L - L[i]
# Arma el conjunto de índices a comparar por vecindad contra i
   for j = 0 to len(L') - 1
   t1 = L[i]
   t2 = L[j]
   if W[t1, t2] >= 0 and L''[j] not in L':
# Es decir t1 y t2 son contiguos y t2 no está en L'
      L' = L' + L''[j]
```

[80] Notemos que estamos presentando el código en una suerte de pseudocódigo con fuerte influencia de **Python.**

```
   j = j + 1
i = i + 1
if len(L') == len(L):
   return VERDADERO          # El conjunto es contiguo
elif return FALSO            # El conjunto NO es contiguo
```

Ahora bien, para una medida Tipo II, es suficiente aplicar el algoritmo dos veces, una para el distrito fuente y una para el distrito objetivo, dado que ambos distritos ganan y pierden una UB, a menos que haya enclaves. El punto crítico es ejecutar ISCONTIGUOUS como si el intercambio ya hubiera tenido lugar. Para una medida Tipo II (*i*, *k*, *j*, *l*), esto significa que ISCONTIGUOUS debe correrse para el distrito *j* asumiendo que el distrito ya ha ganado la UB *k*, es decir $D_j \leftarrow D_j \cup \{k\}$. Similarmente para el distrito *l*, ISCONTIGUOUS debe ser corrido como si el distrito *l* está compuesto de las UBs en $D_l \cup \{i\}$. Con esta implementación, el movimiento (*i*, *k*, *j*, *l*) (solución X_{ikjl}) no será permitido en el conjunto de movimientos elegibles $M_t^2(X)$ (entorno $N_2(X)$) si uno (o ambos) de los dos controles de contigüidad fallan. En el caso de una transferencia / intercambio que involucre una UB con enclaves, todas las UBs que son enclavadas por la UB *i* se excluyen del conjunto *L*. Asimismo, en el paso 4, $L \cup \{i\}$ se reemplaza con $L \cup \{i\} \cup E$, donde *E* es el conjunto de las UB enclavadas por la UB *i*[81].

Una vez que se construye la vecindad de la solución, el siguiente paso es elegir una solución $X' \in N(X)$, que el algoritmo visitará en la siguiente iteración. El algoritmo de Búsqueda Tabú selecciona la solución con el mejor valor de la función objetivo, aun cuando esta solución puede no ser una solución mejor. El estado de tabú de los movimientos y los criterios de aspiración aplicables son también fuertemente relevantes para identificar *X*'. En líneas generales el algoritmo elegirá la mejor solución no-tabú. Una solución tabú será elegida solamente si tiene el menor valor de la función objetivo que es también menor que el de la mejor solución conocida.

Es importante resaltar que tanto la vecindad $N_1(X)$ como $N_2(X)$ pueden contener soluciones inviables, es decir, **soluciones que violan las restricciones de balance de población**. Como resultado, es posible elegir una solución inviable de la vecindad si dicha solución es de una "suficientemente buena" calidad. De hecho, todos los siguientes conjuntos de movimientos están disponibles para ser seleccionados:

- Todos los movimientos no-tabú (sea que resulten o no en soluciones inviables).
- Todos los movimientos no-tabú que resulten en soluciones **inviables**, pero que también mejoren el mejor valor conocido para la función objetivo (viable o inviable).

[81] Recordemos, que en nuestra implementación no existen los enclaves dado que en los pasos de pre-procesamiento, si existieran enclaves estos se fusionan con la Unidad Base que los contiene, salvo que tengan la población suficiente como para ser un Distrito por derecho propio.

- Todos los movimientos tabú que resulten en soluciones **viables**, pero que también mejoren el mejor valor conocido de una solución **viable**.

Memoria de Búsqueda Tabú Básica

Bozkaya (1.999) implementa dos tipos básicos de memoria en su algoritmo. El primero es el denominado *recency-based* (memoria basada en la inmediatez o memoria reciente) y se emplea para almacenar y administrar el estatus tabú de los movimientos. El segundo, denominado *frequency-based* (memoria basada en la frecuencia o memoria frecuente) provee una diversificación sobre el espacio de soluciones.

Memoria basada en la inmediatez (memoria reciente)

La principal función de esta memoria es almacenar el estatus de los movimientos tabú, es decir, si un dado movimiento es considerado en ese momento o no como tabú, y de ser tabú cuando expira dicho estado. En el algoritmo de BT un movimiento se declara tabú tan pronto como es ejecutado, y permanece tabú por la duración de su permanencia tabú θ. La parte interesante del problema es como declarar una movida como tabú y que valor de θ debe emplearse.

Un movimiento se declara tabú basándose en uno o más de sus atributos. En el algoritmo de Bozkaya, esto se implementa sobre la base del índice de la UB que está siendo transferida y el índice del distrito desde donde dicha UB se transfiere. Para un movimiento Tipo I, los atributos correspondientes son *UB_indice* y *desde*. Al movimiento le es inmediatamente asignado una permanencia tabú dada por θ. Esto implica que una vez que la UB deja el distrito, no podrá volver a regresar al mismo distrito por θ iteraciones.

Toda la información necesaria para administrar el estado tabú y las permanencias tabú se almacena en un arreglo bidimensional de tamaño *C x D*. Siguiendo la notación de Bozkaya, denominaremos a ese arreglo $\Gamma(i,j)$ donde el índice de fila *i* representa la UB mientras que el índice de columna *j*, corresponde al distrito. En cada celda de Γ se almacena la iteración al final de la cual el estado tabú expira. Si $\Gamma(i,j)$ es menor que el conteo actual de iteraciones, el movimiento correspondiente no es tabú.

Para actualizar una dada celda de $\Gamma(i,j)$ se sigue un procedimiento predeterminado: cuando la UB *i* se transfiere **desde** el distrito *j* a otro distrito, $\Gamma(i,j)$ se asigna como el valor dado por el número actual de iteración + permanencia tabú. Cuandoquiera que se considere a la UB *i* (en posteriores iteraciones) para regresar a *j*, el contador de iteraciones en ese punto

se compara con el valor almacenado en la celda $\Gamma(i,j)$. Si el contador de iteraciones es **menor o igual** que $\Gamma(i,j)$, el movimiento es todavía prohibido (tabú) y por lo tanto no puede ser implementado (a menos que uno o más criterios de aspiración lo hagan admisible).

Para un movimiento Tipo II (*UB_indice_1*, *UB_indice2*, *desde_1*, *hacia_1*) = (*i,k,j,l), dos* celdas son actualizadas cuando se ejecuta el movimiento. Las celdas $\Gamma(i,j)$ y $\Gamma(k,l)$se actualizan de forma similar registrando la iteración en la que el estado tabú expira. Un movimiento Tipo II permanece tabú hasta que *una* (pero no ambas) de las transferencias que componen el movimiento se convierte en admisible.

Tabla S.4. – Ejemplo de Arreglo para el almacenamiento de estados tabú – Bozkaya (1.999)

	1	2	3	4
1			5	
2	7			
3				
4				6
5		9		
6	8		7	

En esta tabla seis movimientos tienen estados tabú, pero cuáles de ellos son efectivamente prohibidos depende del contador actual de iteraciones. Si estamos en la iteración número 7, los movimientos Tipo 1 (2,.,1), (5,.,2), (6,.,1) y (6,.,3) son todavía tabú. Esto es, no podemos transferir la UB 2 de vuelta al distrito 1, la UB 5 al distrito 2, etc. Ahora bien, el movimiento (1,.,3) **no** es tabú, dado que su estado tabú ha expirado al terminar la iteración número 5. Con esto, este arreglo **no** necesita más actualizaciones que el almacenamiento en la celda correcta cuando las UBs son transferidas o intercambiadas.

Para un movimiento Tipo II (1,5,2,3) que intercambia las UBs 1 y 5 entre los distritos 2 y 3, el estatus tabú está determinado al chequear las celdas $\Gamma(1,3)$ y $\Gamma(5,2)$. De la tabla anterior, vemos que el estado del primer movimiento ha expirado ($\Gamma(1,3) = 5$), por lo que el movimiento Tipo II (1,5,2,3) no es más tabú.

Siguiendo a Bozkaya, se ha determinado un mecanismo aleatorio para asignar las permanencias tabú. Esto es, cuando se ejecuta un movimiento, su permanencia tabú se genera aleatoriamente dentro del rango $[\theta_{min}, \theta_{max}]$. Aunque esto incrementa en uno los

parámetros (comparado con el caso determinístico de θ fijo), esta variación permite mejorar la calidad de la solución final.

Memoria basada en la frecuencia (memoria frecuente)

Este tipo de memoria se incorpora al algoritmo de BT para facilitar la diversificación en el espacio de soluciones y permitir que no quede atrapado en óptimos locales[82]. Si bien es cierto que la memoria basada en la inmediatez puede también prevenir que el algoritmo quede atrapado en óptimos locales mediante la aceptación de soluciones no mejores, esta memoria basada en la frecuencia se emplea para reforzar este objetivo.

Con esta memoria, los movimientos más frecuentes son penalizados mediante el agregado de un término de penalización a la función objetivo. El propósito de esto es fomentar movimientos / soluciones que son realizados / visitados en forma menos frecuente. El término de penalización es producto de cuatro variables y parámetros:

1- La frecuencia pasada de un movimiento,
2- El tamaño del problema, medido por $\sqrt{D}$,
3- δ, el mayor cambio en la función objetivo de una iteración a la siguiente,
4- ρ, un factor de escala constante.

El primer factor, la frecuencia de implementación de un movimiento, es la fuerza principal entre los cuatro parámetros anteriores. Típicamente, cuanto más frecuentemente se ha implementado un movimiento en las iteraciones previas, mayor será el valor de este factor. Posee dos componentes que mantienen registro de dos atributos diferentes de un movimiento. El primer componente, denotado por η_{it} almacena la cantidad de veces que la UB *i* ha sido transferida o intercambiada en las anteriores (t – 1) iteraciones. Es decir

$$\eta_{it} = \frac{n_i^b}{t-1} \qquad \text{(S.16)}$$

Donde n_i^b es el número de veces que la UB i fue parte de una transferencia o intercambio en todas las anteriores (t – 1) iteraciones del algoritmo. Para un movimiento Tipo I, n_i^b se incrementa en 1 al final de la iteración en la que el movimiento tuvo lugar. Para un movimiento Tipo II que involucra las UBs *i* y *k*, ambos n_i^b y n_k^b se incrementan en 1. El segundo componente del factor de frecuencia es ν_{jt}, que está asociado con los *distritos* que son sujetos a transferencias o intercambios. En este caso ν_{jt} es la frecuencia de uso del distrito *j* como un distrito *desde* o *hacia* en un movimiento Tipo I o Tipo II en todas las (t – 1) iteraciones previas. Está dado por:

[82] El empleo de múltiples puntos de inicio aleatorios no es apropiado debido a que incrementa excesivamente el esfuerzo computacional y a que no se incorpora fácilmente en el esquema de la Búsqueda Tabú.

$$\nu_{jt} = \frac{n_j^d}{t-1} \qquad \text{(S.17)}$$

donden_j^d es el número de veces que el distrito *j* estuvo involucrado en un movimiento Tipo I o Tipo II en todas las iteraciones previas ($t-1$). Para un movimiento Tipo I (*i,j,l*), n_j^d y n_l^d se incrementan en 1, mientras que para un movimiento Tipo II (*i,k,j,l*), n_j^d y n_l^d se incrementan en 2.

Con esto, el factor de frecuencia se calcula utilizando los dos componentes anteriores, η_{it} y ν_{jt}:

- Para un movimiento Tipo I:

$$\Upsilon_t(i,j,l) = (1+\eta_{it})\left(1+\frac{\nu_{jt}+\nu_{lt}}{2}\right)-1 \qquad \text{(S.18)}$$

- Para un movimiento Tipo II:

$$\Upsilon_t(i,j,k,l) = \left(1+\frac{\eta_{it}+\eta_{kt}}{2}\right)\left(1+\frac{\nu_{jt}+\nu_{lt}}{2}\right)-1 \qquad \text{(S.19)}$$

Las variables n_i^b y n_j^d se almacenan en arreglos unidimensionales de tamaño *C* y *D*, respectivamente. Estos dos arreglos se denotan como Λ^b y Λ^d respectivamente.

El siguiente factor del término de penalización, $\sqrt{D}$ está relacionado con el tamaño del problema. La razón para incorporar el tamaño del problema en el término de penalización es magnificar el efecto de diversificación como resultado de un aumento en el tamaño del problema, de esta manera se tiene en cuenta el aumento de tamaño del espacio de solución. Incluir *C* además de *D* es innecesario dado que un aumento en *C* de un territorio en districtalización a otro, está usualmente asociado con un incremento proporcional en *D*. (Esto asume que el cociente de distrito permanece constante). Por esta razón, *D* se elige arbitrariamente para representar el tamaño del problema. Asimismo, al utilizar la raíz cuadrada se limita el efecto de magnificación como resultado de un incremento en *D*. Varios estudios (Cordeau, Gendreau y Laporte, 1.997) muestran resultados mucho más exitosos con $\sqrt{D}$ que con *D*).

El tercer factor, δ, es la variable que almacena el cambio máximo en la función objetivo de una iteración a la siguiente. La razón para utilizar este factor es escalar la magnitud del termino de penalidad de manera tal que la penalidad total no sea ni demasiado grande ni demasiado pequeña en relación al valor de la función objetivo.

El último factor, ρ, es un parámetro de escala constante. Su rol es proveer un efecto adicional de escala sobre el término de penalización. Cuando el término de penalización es demasiado pequeño de manera que no tiene esencialmente efecto en perturbar el ranking de movimientos, ρ debe ser incrementado. Por otro lado, si el término de penalización es demasiado grande, tanto como para tener un efecto disruptivo en el proceso de búsqueda,

ρdebe ser disminuido. El uso de ρ provee una herramienta adicional para escalar el término de penalización dado que su valor se define mediante el usuario, a diferencia de los otros tres factores que contribuyen a la magnitud de la penalización. Para asegurarse tener un adecuado efecto de diversificación **sin** disrupciones en el proceso de búsqueda, este parámetro se pre-determina y fija en un nivel constante para un determinado problema.

Estos cuatro factores se multiplican entre sí y se agregan a *f*(X) de manera tal de definir una función objetivo modificada:

$$f'(X) = f(X) + \Upsilon_t \delta \varrho \sqrt{m} \qquad \text{(S.20)}$$

Reglas de Parada del algoritmo:

Usualmente se consideran dos reglas de parada dadas por:

1- La principal regla de parada fuerza al algoritmo a terminar cuando se realizan T = [230√m] iteraciones consecutivas sin que se mejoren las mejores soluciones viable e inviable. Este límite superior se designa de manera tal que si el problema crece, se permite mayor cantidad de iteraciones antes de que el algoritmo se detenga. La constante 230 se selecciona de manera tal de obtener un numero grande pero manejable de intentos para salir de un óptimo local.

2- La siguiente regla de parada es un límite superior sobre el total de iteraciones. Cuando se alcanza este límite, el algoritmo se detiene sin importar en que parte del proceso se encuentre. Obviamente este límite debe ser lo suficientemente largo como para no interrumpir el algoritmo en el medio de una línea de búsqueda promisoria. Por ejemplo Bozkaya (1.999) emplea un límite de 30.000, mientras que en nuestro caso hemos utilizado valores de 5.000 con muy buenos resultados.

El Algoritmo de Búsqueda Tabú (Bozkaya, 1.999):

Basándose en las definiciones de movimientos de BT, estructuras de memoria, y reglas de parada, ya se tiene el esquema general del algoritmo. En esta instancia denominaremos al algoritmo como de *Búsqueda Tabú Plana*, dado que comienza con una solución inicial única y realiza luego una pasada del procedimiento iterativo. Más adelante se describirá un algoritmo meta-heurístico que ejecuta múltiples pasadas.

Recordemos que un dado plan de districtalización puede ser **viable** o **inviable,** por lo tanto es necesario mantener un seguimiento de la mejor solución viable al igual que de la mejor solución (viable o no) en forma separada en la medida en que son encontradas durante una corrida de BT. Usaremos F_1 para registrar el valor de la función objetivo para la mejor solución viable, y F_2 para el valor de la mejor solución viable o no. F_1 y F_2 se inicializan con un valor grande cuando el algoritmo inicia, y se actualizan en la medida de lo necesario.

TABU_SEARCH

0. Inicializacion
   - 0.1 Usar el módulo INITSOLN para construir una solución de partida denominada X_0. Hacer $F_2 = f(X_0)$. Si X_0 es viable, hacer $F_1 = f(X_0)$; sino hacer $F_1 = \infty$.
   - 0.2 Setear el contador de iteraciones $t = 0$, pasada = 1, $\alpha = 1$, $\delta = 0$.
   - 0.3 Inicializar la memoria basada en la inmediatez Γ, y las basadas en la frecuencia Λ^b, Λ^d en 0.

1. Repetir hasta que no haya mejora en F_1 y F_2 para cualquier T iteraciones sucesivas:
   - 1.1 Hacer $t \leftarrow t + 1$.
   - 1.2 Construir M_t^1 (el conjunto de los movimientos Tipo I elegibles), y encontrar el mejor movimiento en M_t^1 :
     - 1.2.1 Hacer $M_t^1 = \phi$.
     - 1.2.2 Para cada UBF i de cada distrito j,
       - 1.2.2.1 Construir el conjunto L de todos los distritos vecinos a j a los que la UB i puede ser transferida.
       - 1.2.2.2 Para cada $l \in L$, si ISCONTIGUOUS(i,j,l) devuelve el valor VERDADERO, y el movimiento (i,j,l) satisface una de las siguientes condiciones:
         - $t > \Gamma(i,l)$
         - $t \leq \Gamma(i,l)$, X_{ijl} es viable y $f(X_{ijl}) < F_1$
         - $t \leq \Gamma(i,l)$, X_{ijl} es inviable y $f(X_{ijl}) < F_2$

         Entonces hacer $M_t^1 \leftarrow M_t^1 \cup \{(i,j,l)\}$
     - 1.2.3 Para cada movimiento $(i,j,l) \in M_t^1$ calcular $f'(X_{ijl})$ como:
       - 1.2.3.1 Si $f(X_{ijl}) < f(X_{t-1})$, hacer $f'(X_{ijl}) = f(X_{ijl})$.
       - 1.2.3.2 De otro modo, hacer $f'(X_{ijl}) = f(X_{ijl}) + Y_t \delta \rho \sqrt{m}$

1.2.4 Identificar el movimiento $(i^*,j^*,l^*) \in M_t^1$ como el mejor movimiento Tipo I:

1.2.4.1 Ordenar M_t^1 en orden ascendente de acuerdo con $f'(X_{ijl})$.

1.2.4.2 Tomar el primer movimiento no-tabú o el primer movimiento tabú que mejora F_1 o F_2 y denominarlo (i^*,j^*,l^*).

1.3 Si pasada = 2, construir M_t^2 (el conjunto de todos los movimiento elegibles del tipo II), y encontrar el mejor movimiento en M_t^2:

1.3.1 Hacer $M_t^2 = \phi$.

1.3.2 Para cada par de UBF $i \in D_j$ y $k \in D_l$, para cada par de distritos vecino j y l:

1.3.2.1 Si ISCONTIGUOUS(i,j,l) y ISCONTIGUOUS(k,l,j) **ambos** devuelven VERDADERO, y el movimiento (i,k,j,l) satisface una de las siguientes condiciones:

- $t > \Gamma(i,l)$ O $t > \Gamma(k,j)$
- $t \leq \Gamma(i,l)$ Y $t \leq \Gamma(k,j)$, X_{ikjl} es viable y $f(X_{ikjl}) < F_1$
- $t \leq \Gamma(i,l)$ Y $t \leq \Gamma(k,j)$, X_{ikjl} es inviable y $f(X_{ikjl}) < F_2$

1.3.3 Para cada movimiento $(i,k,j,l) \in M_t^2$, calcular $f'(X_{ikjl})$ de la siguiente forma:

1.3.3.1 Si $f(X_{ikjl}) < f(X_{t-1})$, hacer $f'(X_{ikjl}) = f(X_{ikjl})$.

1.3.3.2 De otro modo, hacer $f'(X_{ikjl}) = f(X_{ikjl}) + Y_t \delta \rho \sqrt{m}$

1.3.4 Identificar el movimiento $(i^*,k^*,j^*,l^*) \in M_t^2$ como el mejor movimiento Tipo II de forma análoga al paso 1.2.4

1.4 Implementar el mejor movimiento encontrado arriba

1.4.1 Hacer $X_t \leftarrow X_{i^*j^*l^*}$, si

- pasada = 1 O
- pasada = 2 y $f(X_{i^*j^*l^*}) \leq f(X_{i^*k^*j^*l^*})$

1.4.2 De otro modo, hacer $X_t \leftarrow X_{i^*k^*j^*l^*}$

1.4.3 Actualizar P_j, P_l, A_j, A_l, R_j y R_l.

1.5 Actualizar la memoria basada en la inmediatez Γ:

1.5.1 Generar aleatoriamente una permanencia tabú θ dentro del rango $[\theta_{min}, \theta_{max}]$.

1.5.2 Si se realizó un movimiento Tipo I, hacer

$$\Gamma(i,j) = t + \theta.$$

1.5.3 De otro modo, hacer

$$\Gamma(i,j) = t + \theta \text{ y } \Gamma(k,l) = t + \theta$$

1.6 Actualizar la memoria basada en la frecuencia Λ^b y Λ^d:

1.6.1 Si se implementó un movimiento tipo I, hacer

$\Lambda^b(i) \leftarrow \Lambda^b(i) + 1$,

$\Lambda^d(j) \leftarrow \Lambda^d(j) + 1$,

$\Lambda^d(l) \leftarrow \Lambda^d(l) + 1.$

1.6.2 De otro modo, hacer

$\Lambda^b(i) \leftarrow \Lambda^b(i) + 1,$

$\Lambda^b(k) \leftarrow \Lambda^b(k) + 1,$

$\Lambda^d(j) \leftarrow \Lambda^d(j) + 2,$

$\Lambda^d(l) \leftarrow \Lambda^d(l) + 2.$

1.7 Actualizar F_2, F_2, α y δ:

1.7.1 Si X_t es viable y $f(X_t) < F_1$, hacer $F_1 = f(X_t)$.

1.7.2 Si $f(X_t) < F_2$, hacer $F_2 = f(X_t)$

1.7.3 Si $t \equiv 0 \pmod{\mu}$, actualizar α:

1.7.3.1 Si por lo menos $\bar{\mu}$ de las ultimas μ soluciones eran inviables, hacer

$\alpha \leftarrow 2\alpha,$

1.7.3.2 De otro modo, si por lo menos $\bar{\mu}$ de las ultimas μ soluciones eran viables, hacer

$\alpha \leftarrow \alpha/2.$

1.7.4 Hacer $\delta \leftarrow \max(\delta, |f(X_t) - f(X_{t-1})|)$

2. Si pasada = 2, PARAR. De otro modo, hacer pasada = 2 e ir al paso 1.

Diversificación e Intensificación Probabilística

Para aumentar la potencia del método de resolución Bozkaya (1.999), Bozkaya *et. al* (2.003) y Bozkaya *et. al* (2.010) utiliza una técnica particular denominada *Diversificación e Intensificación Probabilística*[83] *(DIP)*, desarrollada originalmente por Rochat y Taillard (1.995):

Método DIP[84]

La idea central en DIP es que soluciones de alta calidad (es decir, soluciones *cuasi-óptimas*) tienen componentes de alta calidad y que estos componentes pueden ser utilizados para construir otras soluciones de alta calidad. La presunción principal es que algunos de estos componentes están también presentes en la solución óptima, y que por lo tanto la función objetivo tiene que encontrar tantos de esos componentes como sea posible. El algoritmo opera sobre un "pool" de buenos componentes que luego utiliza para construir una solución inicial para el procedimiento de búsqueda local[85].

[83] Notemos que en trabajos actuales de BT (ver por ejemplo Martí 2003)se han explorado otros elementos más sofisticados que los de memoria reciente y memoria frecuente y se los ha incorporado explícitamente dentro de la BT, por ejemplo: (1) **Movimientos de Influencia**: Son aquellos movimientos que producen un cambio importante en la estructura de las soluciones. Usualmente, en un procedimiento de búsqueda local, la búsqueda es dirigida mediante la evaluación de la función objetivo. Sin embargo, puede ser muy útil el encontrar o diseñar otros evaluadores que guíen a ésta en determinadas ocasiones. Los movimientos de influencia proporcionan una evaluación alternativa de la bondad de los movimientos al margen de la función objetivo. Su utilidad principal es la determinación de estructuras subyacentes en las soluciones. Esto permite que sean la base para procesos de Intensificación y Diversificación a largo plazo. (2) **Oscilación Estratégica:** La Oscilación Estratégica opera orientando los movimientos en relación a una cierta frontera en donde el método se detendría normalmente. Sin embargo, en vez de detenerse, las reglas para la elección de los movimientos se modifican para permitir que la región al otro lado de la frontera sea alcanzada. Posteriormente se fuerza al procedimiento a regresar a la zona inicial. El proceso de aproximarse, traspasar y volver sobre una determinada frontera crea un patrón de oscilación que da nombre a esta técnica. Una implementación sencilla consiste en considerar la barrera de la factibilidad / infactibilidad de un problema dado. Implementaciones más complejas pueden crearse identificando determinadas estructuras de soluciones que no son visitadas por el algoritmo y considerando procesos de construcción / destrucción asociados a éstas. La oscilación estratégica proporciona un medio adicional para lograr una interacción muy efectiva entre intensificación y diversificación. (3) **Elecciones Probabilísticas**: Normalmente TS se basa en reglas sistemáticas en lugar de decisiones al azar. Sin embargo, en ocasiones se recomienda el aleatorizar algunos procesos para facilitar la elección de buenos candidatos o cuando no está clara la estrategia a seguir (quizá por tener criterios de selección enfrentados). La selección aleatoria puede ser uniforme o seguir una distribución de probabilidad construida empíricamente a partir de la evaluación asociada a cada movimiento. (4) **Umbrales Tabú**: El procedimiento conocido como *Tabu Thresholding* (TT) se propone para aunar ideas que provienen de la Oscilación Estratégica y de las Estrategias de Listas de Candidatos en un marco sencillo que facilite su implementación. El uso de la memoria es implícito en el sentido que no hay una lista tabú en donde anotar el status de los movimientos, pero la estrategia de elección de los mismos previene el ciclado. TT utiliza elecciones probabilísticas y umbrales en las listas de candidatos para implementar los principios de TS. (5) **Re-encadenamiento de Trayectorias (*Path Relinking*)**: Este método se basa en volver a unir dos buenas soluciones mediante un nuevo camino. Así, si en el proceso de búsqueda hemos encontrado dos soluciones *x* e *y* con un buen valor de la función objetivo, podemos considerar el tomar *x* como solución inicial e *y* como solución final e iniciar un nuevo camino desde *x* hasta *y*. Para seleccionar los movimientos no consideraremos la función objetivo o el criterio que hayamos estado utilizando hasta el momento, sino que iremos incorporando a *x* los atributos de *y* hasta llegar a ésta. Por eso esperamos que alguna de las soluciones intermedias que se visitan en este proceso de "entorno constructivo" sea muy buena. En algunas implementaciones se ha considerado el explorar el entorno de las soluciones intermedias para dar más posibilidad al descubrimiento de buenas soluciones.

[84] "*Probabilistic Diversification and Intensification*" (PDI) en ingles.

[85] Notemos que una forma sencilla de incorporar la intensificación en la BT implica:

Aplicar la BT con la memoria a corto plazo
Aplicar una estrategia de selección de soluciones elite

En el problema original de Rochat y Taillard, los *componentes* de la solución son los viajes del vehículo. El *pool* inicial de viajes se construye al correr la búsqueda local un número predeterminado de veces y agregar todos los viajes generados al pool. El algoritmo luego construye una solución que utiliza tantos viajes del pool como sea posible, y los provee al algoritmo de búsqueda local como solución de inicio. Luego, la solución se mejora por el algoritmo de búsqueda local y los viajes de la mejor solución encontrada se incluyen en el *pool*.

El algoritmo inicial DIP de Rochat y Taillard es:

DIP_VRP

1. Inicialización
   - 1.1. Generar$I$ soluciones diferentes con la búsqueda local.
   - 1.2. Etiquetar cada viaje con el valor de la solución a la que pertenece.
   - 1.3. Remover los viajes que tienen un único cliente.
   - 1.4. Insertar los viajes restantes en el conjunto V de viajes.
   - 1.5. Ordenar V de acuerdo al valor ascendente de las etiquetas.

2. Diversificación e intensificación (repetir hasta que se satisfaga un determinado criterio de parada)
   - 2.1. Hacer V′ := V, $S := \phi$.
   - 2.2. Mientras V′ $\neq\phi$, repetir
     - 2.2.1. Elegir v $\in$ V` en forma probabilística de acuerdo con su evaluación relativa.
     - 2.2.2. Hacer S := S $\cup$ {v}.
     - 2.2.3. Remover de V′ todos los viajes que incluyan uno o varios clientes pertenecientes a v.
   - 2.3. Si algunos clientes no están cubiertos por los viajes de S, construir una solución viable S′, que los incluya, utilizando la solución parcial S.
   - 2.4. Mejorar mediante búsqueda local la solución S′.
   - 2.5. Etiquetar los viajes de la solución mejorada, remover los viajes con un solo cliente, insertar los viajes restantes en V, ordenar V de la misma forma que en los pasos 1.2 a 1.5 de la inicialización.

Hacer *{*
Elegir una de las soluciones elite.
Reanudar la BT con memoria a corto plazo a partir de la solución elegida
Agregar nuevas soluciones a la lista elite cuando fuera aplicable
} ***mientras*** *(iteraciones < limite y la lista de soluciones elite no es vacía)*

La parte principal del algoritmo precedente es construir una solución viable utilizando los viajes en el conjunto *V*. Esto se realiza al elegir los viajes en forma probabilística uno después de otro, basándose en su posición en el ranking de todos los viajes en *V*. Específicamente, el i-esimo viaje desde arriba de la lista tiene una probabilidad de ser elegido dada por:

$$\frac{|V|-i+1}{|V|\cdot(|V+L|)/2}$$

Como resultado, los viajes que pertenecen a mejores soluciones tienen mayor probabilidad de ser seleccionados. Ahora bien, una vez que un viaje se selecciona, todos los viajes con los que este comparte un cliente se remueven de la consideración. Los viajes se seleccionan de manera iterativa hasta que no hay otro viaje disponible para ser elegido. Los viajes seleccionados constituyen entonces la solución factible (parcial) *S*.

Si *S* es de hecho parcial (es decir no cubre a todos los clientes), el paso 2.3 se ejecuta de manera tal de convertirla en una solución completa *S*'. *S*' es entonces mejorado mediante búsqueda local, y los viajes que pertenecen a la solución mejorada se agregan a *V*. En este punto Rochard y Taillard descartan las peores $|V|$ - *L* soluciones en *V*, donde *L* es el tamaño máximo permitido para *V*. Esto se realiza de manera tal de prevenir que el conjunto *V* se expanda en forma continua. Si *I* y *L* son lo suficientemente grandes, esto no implica ningún problema serio, dado que el algoritmo muy probablemente tendrá soluciones representativas de diferentes porciones del espacio de solución. De esta forma, la diversificación tiene lugar como resultado de la habilidad del algoritmo para generar considerablemente diferentes soluciones iniciales para la búsqueda local (especialmente cuando emplea viajes de ranking bajo). Por otro lado, el algoritmo ejecuta la intensificación al seleccionar los viajes de las soluciones de "elite".

Estas ideas pueden modificarse para ser utilizadas en el problema de districtalización. El procedimiento local de búsqueda que aparece en los pasos 1.1 y 2.4 se reemplaza con el algoritmo de BT propuesto. Los viajes en el contexto de Rochat y Taillard se corresponden con los distritos en este contexto, mientras que los clientes que forman parte de los viajes son equivalentes a las UB que construyen los distritos. La única parte del algoritmo que necesita una implementación específica es la construcción de la solución completa *S*' a partir de la solución parcial *S*.

Implementación de la Diversificación e Intensificación Probabilística

Primero se genera una colección *Ds* = {D_1,…, $D_{s\text{-}m}$} de buenos distritos. Para hacer esto se corre el algoritmo BT *s* veces, cada corrida iniciando a partir de una diferente (aleatoria) solución (ver algoritmo INITSOLN). Los distritos que aparecen en todas las *s* soluciones reportadas por el algoritmo BT se incluyen en *Ds*.

Asimismo, sea g_i asociado con cada $D_i \in Ds$, una medida de la "calidad" de D_i medida como el valor objetivo de la solución a la que D_i pertenece. El conjunto *D'* se ordena, todas las veces, en forma ascendente por g_i. Ahora bien, si algún D_i está asociado con una solución inviable, aparecerá en las partes inferiores del ranking. Esto es, *Ds* se ordena primero con respecto a la viabilidad de la solución (las soluciones factibles se prefieren sobre las no factibles) y luego con relación al valor de la función objetivo.

Una vez que se construye *Ds*, el siguiente paso es correr DIP un número predeterminado de iteraciones *M*. En la medida que se identifican soluciones de mejor calidad los distritos que pertenecen a las mismas se incluyen en *Ds*. Con estas ideas:

DIP_DIST

1. Construir una memoria adaptativa inicial de $m \bullet s$ distritos:
   1.1. Ejecutar el algoritmo BT *s* veces para obtener las soluciones $X_1, \ldots, X_s$.
   1.2. Hacer *Ds* = $\{D_1, \ldots, D_m\}$ el conjunto de todos los distritos en todas las X_j (se permiten repeticiones)
   1.3. Ordenar D_i en forma ascendente en base a g_i.

2. Repetir M veces:
   2.1. Hacer *D'* = *Ds*.
   2.2. Construir una solución completa X usando D':
      2.2.1. Hacer L = ϕ
      2.2.2. Repetir hasta que *D'* = ϕ:
         2.2.2.1. Hacer $Pr\{D_i\} = \frac{|D'|-i+1}{|D'|\cdot(|D'|+L|)/2}$, $\forall i = 1, \ldots, |D'|$ (probabilidad de elegir el distrito en la i-esima posición de *D'* en el ranking).
         2.2.2.2. Aleatoriamente elegir un distrito $D_l \in D'$, basándose en las probabilidades $Pr\{D_i\}$. Hacer $L \leftarrow L \cup D_l$.
         2.2.2.3. Remover de *D'* todos los distritos D_i, $i \neq l$, tales que $D_i \cap D_l \neq \phi$.
      2.2.3. Utilizar la colección L para construir X. Si a X le falta alguna UB, r la paso 2.2.4, sino, ir al 2.3.
      2.2.4. Implementar los pasos 1 y 2 de INITSOLN para construir el conjunto de todos los distritos *L'* que cubren solamente las UB faltantes del paso anterior.
      2.2.5. Si $|L \cup L'| > m$, implementar el paso 3 de INITSOLN sobre $L \cup L'$ con $j = |L \cup L'|$. Sea $\bar{L}$ el conjunto resultante de distritos con $|\bar{L}| = m$.
      2.2.6 Usar $\bar{L}$ para construir X
   2.3. Mejorar X usando el algoritmo BT. Hacer X* la solución resultante.

2.4. Actualizar *Ds* con los distritos de X*, si X* es elegible:
2.4.1. Si X* es viable y D_{s-m} pertenece a una solución inviable $\bar{X}$ O
X* es viable, D_{s-m} pertenece a una solución viable $\bar{X}$, y f(X*) <f($\bar{X}$) O
X* es inviable, D_{s-m} pertenece a una solución inviable $\bar{X}$, y f(X*) <f($\bar{X}$)
Reemplazar todos los distritos de $\bar{X}$ en *D* con los distritos de X*. Etiquetar los distritos de X* con f(X*).
2.4.2. Ordenar nuevamente Ds en forma ascendente de acuerdo con g_i.

3. Reportar la solución $\bar{X}$, la solución en la que $D_l \in D$ como la mejor solución encontrada.

Notemos que dentro de DIP_DIST en el paso 2.2.4 empleamos INITSOLN para construir $\bar{L}$ (el conjunto de distritos que cubren las UBs en el territorio, utilizando los distritos en *L*). En este caso INITSOLN se utiliza y el resultado es un conjunto de distritos *L*' que cubren solamente aquellas UB que están faltantes en los distritos en *L*. Por otro lado si el número de distritos en $L \cup L'$ es mayor que *D*, el paso 3 de INITOSLN se utiliza para reducirlo a *D* (2.2.5) en el algoritmo anterior.

Un paso crítico en DIP_DIST es la actualización del conjunto *D*s (paso 2.4). Para mantener el tamaño de *Ds* fijo en *s* – *D*, los distritos de *X**, si es elegible, reemplazan a los peores *D* distritos en *D*s. Dado que ambas *X** y $\bar{X}$ pueden ser inviables, se evalúan las condiciones en el paso 2.4.1. El conjunto *D*s se actualiza si alguna de esas condiciones se satisface.

El último paso que se describe en el algoritmo DIP_DIST es el mecanismo para mantener actualizado el pool de soluciones de alta calidad. Permite que mejores soluciones ingresen a *Ds* al reemplazar las inferiores lo que consecuentemente mejora la calidad general de *Ds* a lo largo del tiempo. El conjunto mejorado se utiliza entonces para generar aun mejores planes de districtalización (en la medida de lo posible) y el conjunto de actualiza nuevamente. Este proceso se repite por M iteraciones y la mejor solución $\bar{X}$encontrada en este proceso es la que se reporta. Obviamente es posible reportar múltiples soluciones para el problema.

4- Plan de Actividades

El esquema de trabajo fue determinado sintéticamente en tres etapas:

a) **Planteamiento inicial**:
   - Elección del tema,
   - Adquisición de la información básica
   - Elaboración del plan de trabajo

b) **Durante el trabajo**:
   - Recolección de datos y bibliografía
   - Ordenamiento e interpretación de los materiales

c) **Redacción Final**:
   - Formulación coherente de los argumentos y diseño algoritmo
   - Implementaciones posibles y testeos con datos reales.
   - Evaluación de la firmeza de las conclusiones
   - Precisión en la distribución final de la exposición

La etapa *a*) se encuadra dentro de la cursada del *Seminario Final de Graduación,* mientras que las etapas *b*) y *c*) se corresponden con el proceso de evaluación del Trabajo Final de Graduación propiamente dicho[86].

El Tiempo para las etapas *b*) y *c*) es de ocho (08) meses, iniciándose en septiembre de 2010 y finalizando en abril 2011.

Consecuentemente, se espera concluir el *TFG* con un adecuado entendimiento de la metodología de districtalización propuesta y una evaluación de su aplicabilidad en el marco del *Departamento de Planeamiento y Mapeo Criminal del Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires.*

Se espera asimismo, haber realizado la determinación de los ajustes a nivel de parámetros que requiere el algoritmo en estudio conforme las particularidades de las unidades espaciales elegidas como *Unidades Base*, al igual que la justificar la recomendación de su utilización por parte del mencionado Departamento y su eventual implementación en el marco de un *Sistema de Información Geográfica*.

[86] *Coloquios*.

5- **Implementación**

Recordemos que básicamente todos los algoritmos de **BT** y **DIP** requieren de un testeo computacional para ajustar la "sintonía fina" de los parámetros, por lo tanto emplearemos los datos de muestra del partido de Berisso de la Provincia de Buenos Aires para ajustar esos parámetros.

a) Datos requeridos:

Los datos básicos para nuestro trabajo están compuestos por la información relativa a las UB, se trata tanto de datos espaciales (geográficos) como demográficos. El componente geográfico incluye el área y perímetro de cada UB, la matriz de adyacencias de las UB, y en la formulación original, la información referente a cuales UB son enclaves de otras. En nuestro caso, hemos solucionado el tema de los enclaves mediante un pre-procesamiento que esencialmente fusiona los enclaves con su UB contenedora, la única salvedad la constituyen potenciales enclaves que presentan niveles de población que los convierten en distritos "por derecho propio", en ese caso se remueven directamente del proceso y solamente se agregan al presentar la solución final. Por otro lado, el componente demográfico incluye datos relativos a variables medidas a nivel de la UB (se trata por lo tanto de "áreas con conteos o tasas").

A modo de testeo, emplearemos los Radios Censales de los partidos de La Plata y Berisso[87] de la Provincia de Buenos Aires, correspondientes al censo 2001 y disponibles como información geográfica en formato digital (*shapefile* © ESRI utilizando la proyección conforme Gauss-Kruger Faja 5 en *Campo Inchauspe*), cada radio censal posee información demográfica asociada, y en este caso emplearemos el *total de población* por radio censal. La similitud con el plan de districtalización existente se realiza empleando las Fracciones Censales del mismo partido y censo disponibles en el mismo formato, sistema y marco de referencia. Y en cuanto al concepto de integridad de comunidades se emplearemos la información provista por el Municipio de La Plata correspondiente a los Barrios del partido, nuevamente en el mismo formato SIG, marco y sistema de referencia:

[87]Dependiendo del *testeo* a realizar y del tiempo de cómputo que demande dicha tarea, elegiremos utilizar los radios de La Plata, o los de Berisso, que si bien son similares, su principal diferencia es la cantidad de elementos comprendidos en cada conjunto. Por lo tanto para las cuestiones que demandan mayor costo computacional, realizaremos los testeos utilizando los radios del partido de Berisso.

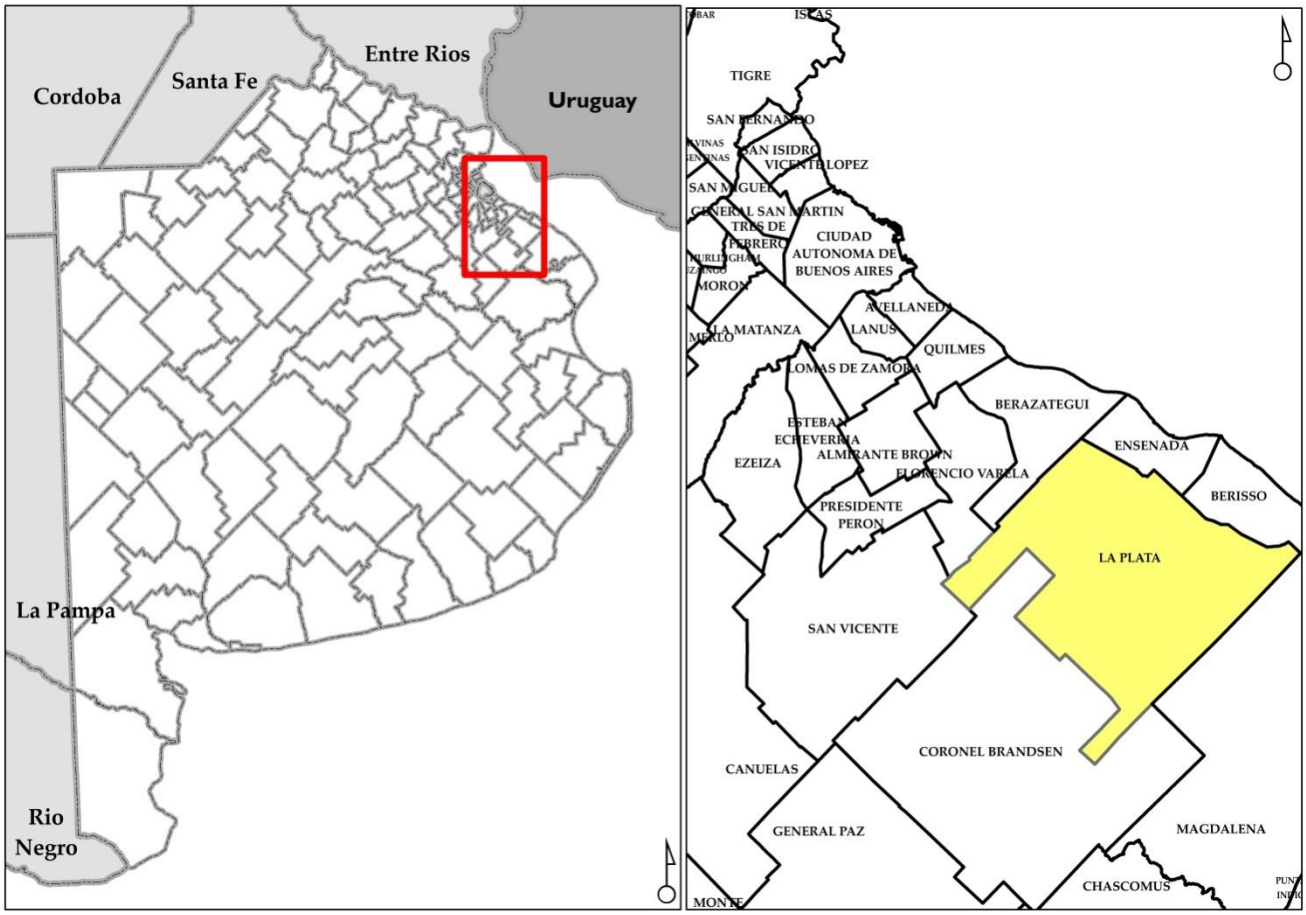


Figura 17 – Provincia de Buenos Aires (partidos) y Partidos de La Plata (en amarillo) y Berisso.-

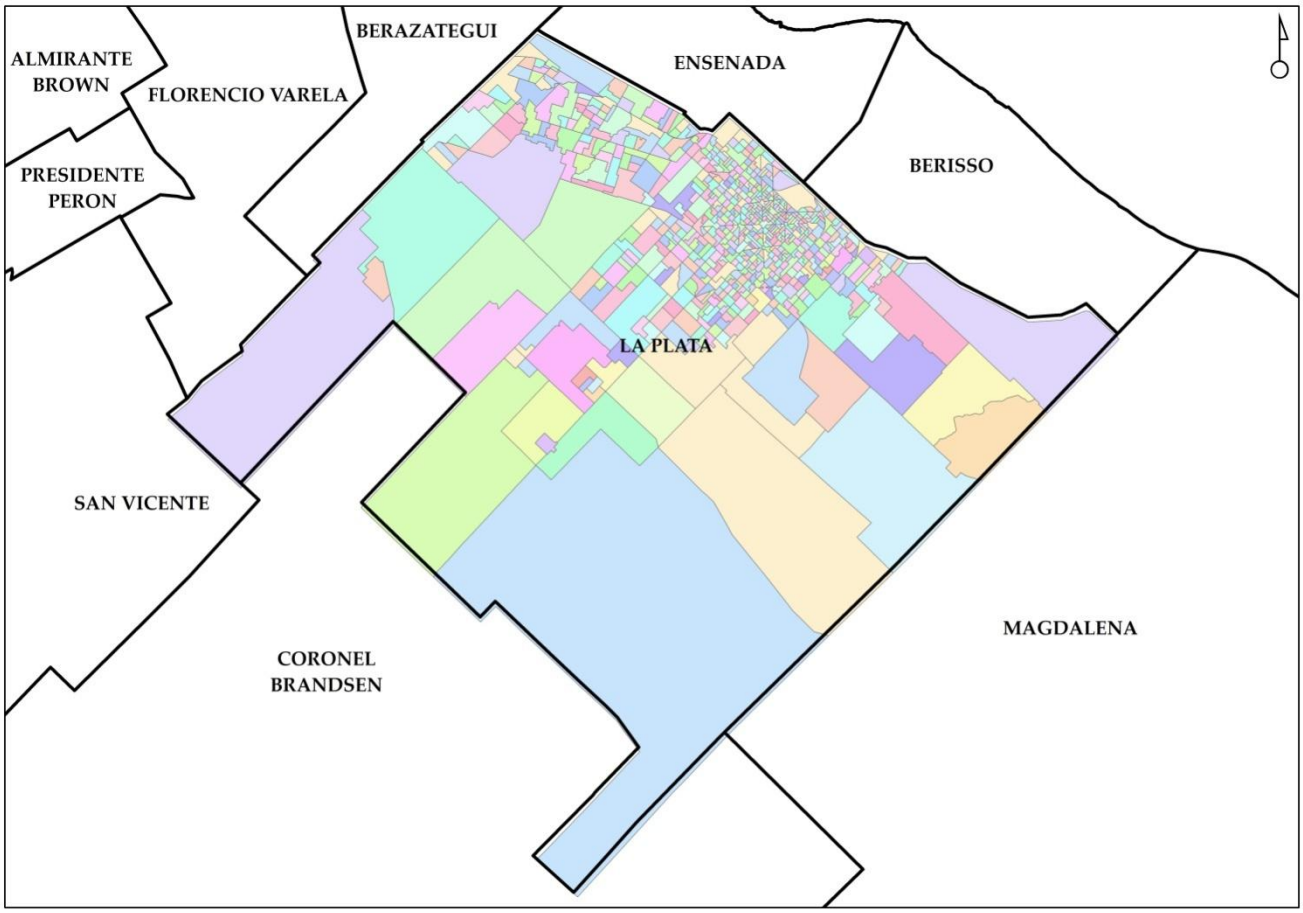


Figura 18 – Radios Censales 2001 del partido de La Plata.-

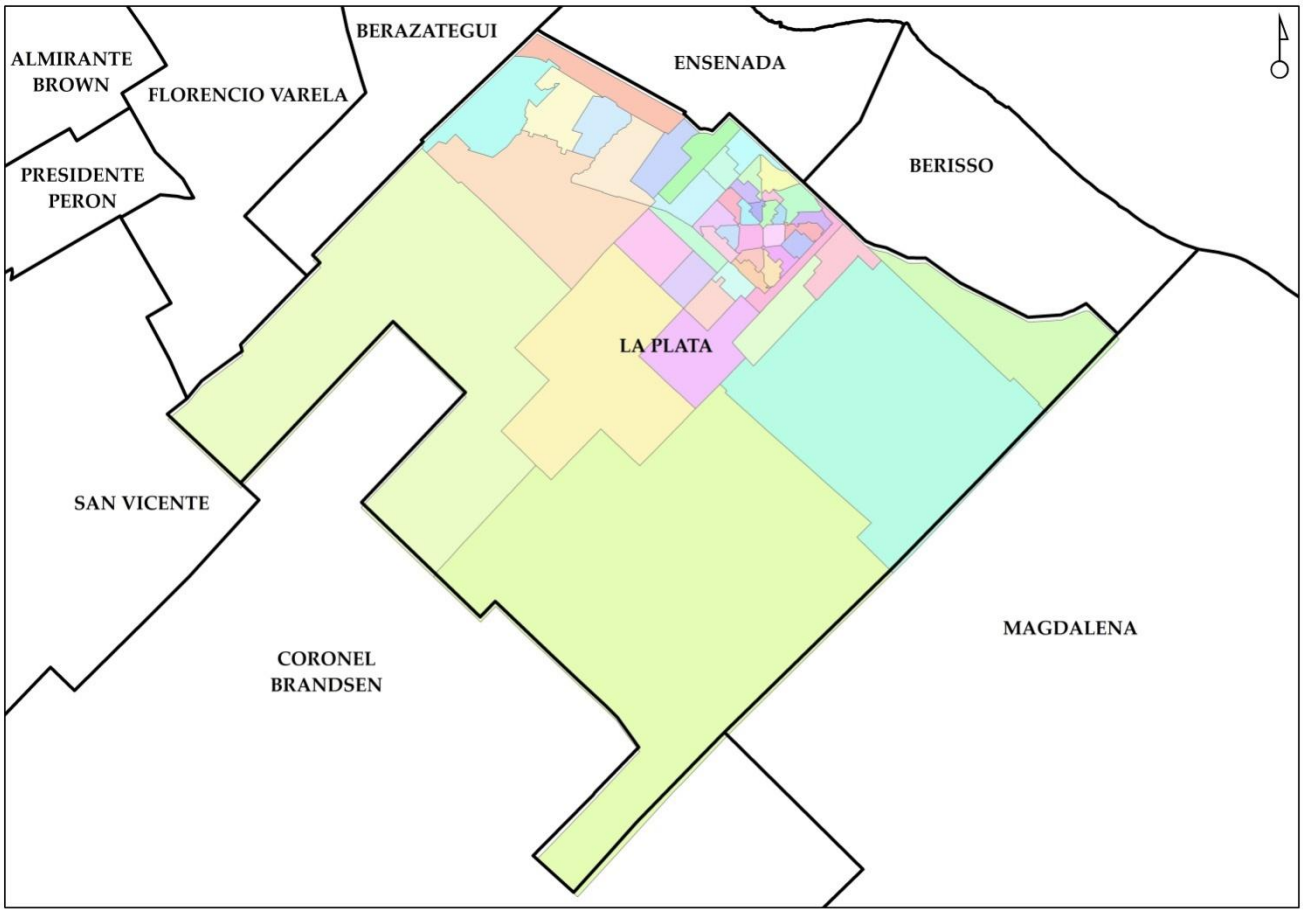


Figura 19 – Fracciones Censales del Partido de La Plata.-

Asimismo, a los fines de facilitar el ajuste de los parámetros del algoritmo, emplearemos los radios censales del Partido de Berisso, lindante con el de La Plata y de gran similitud en cuanto a las características de sus radios censales. De esta forma se podrán testear los parámetros del algoritmo con mayor velocidad que si se ajustaran empleando los radios censales de La Plata (debido esencialmente a la menor cantidad de radios censales en el caso de Berisso, 82 en relación con La Plata, 704).

b) Datos Geográficos, lenguajes e Interfaces:

En principio hemos optado por sugerir la implementación del algoritmo de BT utilizando el lenguaje *Python*[88] de distribución gratuita, haciendo uso de los módulos *NumPy*[89] fundamental en el empleo de Python para "computación científica"[90] y *Pysal*[91]*: Open Source Python Library for Spatial Analytical Functions*. Desafortundamente, la versión actual de *Pysal*, no cuenta con ciertas características requeridas de geometría computacional, *e.g.* en principio no permite unir polígonos ni calcular intersecciones entre polígonos, lo que imposibilita el cálculo correcto de la matriz de adyacencias[92]. Aun así, hemos mantenido contactos con los desarrolladores de *Pysal* en la *Arizona State University*[93] quienes mencionaron estar trabajando en el agregado de estas funcionalidades, y esperan "disponibilizar" las mismas durante el año 2.011.

Por otro lado, se han explorado distintas librerías espaciales para Python, incluyendo diferentes módulos espaciales de mayor o menor difusión[94], y el módulo espacial *PyQGIS* del SIG de código abierto *QGIS*[95]. En todos los casos se encontraron problemas que imposibilitan su uso, ya sea por incompatibilidades a nivel binario entre la última versión de Python y la versión de los módulos espaciales, o de herramientas faltantes en estas librerías.

En esta línea de investigación se ha probado la conveniencia de utilizar el módulo *ArcPy*[96] disponible con la instalación de cualquier versión de ArcGIS 10 (© ESRI). Si bien en este caso se trata de software propietario, provee todas las herramientas necesarias para codificar el Algoritmo de Búsqueda Tabú. Y dado que ha sido diseñado por ESRI © para ser su lenguaje de *scripting* a partir de la serie 9.x de sus Sistemas de Información Geográfica , puede ser visto como la herramienta análoga al lenguaje *Avenue* de la serie 3.x de sus SIG de escritorio (*i.e. Avenue* es a ArcView GIS como *Python* es a ArcGIS).

[88] http://www.python.org/
[89] http://numpy.scipy.org/
[90] Especialmente dado por la necesidad de contar con un adecuado soporte para matrices dispersas.
[91] http://geodacenter.asu.edu/pysal
[92] En particular si bien pysal permite el cálculo de una matriz de ponderación espacial, no permite utilizar como ponderaciones la longitud de la frontera común, algo imprescindible en la primer medida de compacidad que analizamos en este trabajo.
[93] En mails privados con el Dr. Sergio (Serge) Rey Professor, School of Geographical Sciences and Urban Planning GeoDa Center for Geospatial Analysis and Computation Arizona State University
[94] Shapely, http://trac.gispython.org/lab/wiki/Shapely[Internet, fecha de ultimo acceso abril 2.011]; WorldMill, http://trac.gispython.org/lab/wiki/WorldMill[Internet, fecha de ultimo acceso abril 2011], GEOJSON, http://trac.gispython.org/lab/wiki/GeoJSON[Internet, fecha de ultimo acceso abril 2.011], etc.
[95] http://www.qgis.org/ [Internet, fecha de ultimo acceso abril 2.011]
[96] © ESRI, ArcGIS 10. Ver por ejemplo http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/What_is_ArcPy/000v000000v7000000/ [Internet, fecha de ultimo acceso marzo 2.011]

Asimismo, para desarrollar, hemos sugerido el empleo de *Enthought Python Distribution x64*[97]versión 6.3-2 gratuita para usuarios académicos, dado que es la última versión de este Entorno de Desarrollo disponible para la versión 2.6.x de Python. (Notemos que ArcPy es dependiente en binario de Python 2.6.x, por lo que no es posible utilizar la última versión[98] disponible del lenguaje (2.7.1 o 3.2 respectivamente)). Por otro lado, y si se optara por utilizar *Pysal*, una vez que *Pysal* incorpore las herramientas necesarias faltantes, es posible actualizar el entorno de desarrollo a la última versión disponible, por ejemplo, al momento de escribir estas líneas (abril 2011), corresponde a *Enthought Python Distribution 7.0* (Python 2.7.x)[99]. Es decir, que si bien el desarrollo en *Python + ArcPy* limita al uso de la versión 2.6.x, si *Pysal* incorpora las funciones requeridas se podría emplear la última versión de Python.

Recordemos asimismo que *Python* es un lenguaje de programación de alto nivel con una sintaxis limpia que favorece la legibilidad del código, es multiparadigma (soporta orientación a objetos, programación imperativa y programación funcional), es interpretado, fuertemente tipado, emplea tipado dinámico y es multiplataforma. Asimismo, es posible emplear Psyco[100] para acelerar el funcionamiento de los programas en Python sin cambiar el código fuente. Y es posible también distribuir los programas en la forma de "*frozen binaries*" que incorpora en un único ejecutable el código *Python* con la máquina virtual necesaria para su funcionamiento.

Es cierto que se podrían haber elegido otros lenguajes de alto nivel (*e.g.* C++, Java), pero hay varias razones por las que nos hemos inclinado por sugerir Python, por un lado, como cita Hetland (2.008) en su introducción del libro **Beginning Python**:

> *A C program is like a fast dance o a newly waxed dance floor by people carrying razors.*
>
> Waldi Ravens.-
>
> *C++: Hard to learn and built to stay that way.*
>
> Anónimo.-
>
> *Java is, in many ways, C++--.*
>
> Michael Feldman.-
>
> *And now for something completely different…*
>
> Flying Circus de Monty Python .-

Por otro lado, en la forma *Pythonica* de pensar, explicito es mejor que implícito (*EIBTI* = "*explicit is better than implicit*") y sencillo es mejor que complejo, con lo que si no hubiera consideraciones fuertes a nivel de performance que impidieran su uso, su legibilidad, sencillez

[97] http://www.enthought.com/
[98] A marzo de 2.011
[99] http://www.enthought.com/products/epd.php[Internet, fecha de ultimo acceso abril 2.011]
[100] http://psyco.sourceforge.net/introduction.html

y herramientas para el tratamiento de diccionarios, listas y matrices lo hacen un excelente candidato (algo que queda claramente demostrado por su utilización por equipos de investigación avanzada en Sistemas de Información Geográfica como ser el GEODA Center[101] de la Universidad de Arizona, o su empleo como lenguaje de scripting en el Sistema de Información Geográfica de mayor difusión en el mundo (ArcGIS © ESRI[102]).

Finalmente, *El Zen de Python* de Tim Peters indica que[103]:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Aunque lo práctico gana a la pureza.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.

[101] http://geodacenter.asu.edu
[102] http://www.esri.com
[103] http://www.python.org/dev/peps/pep-0020/ Zen Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque *nunca* es a menudo mejor que *ya mismo*.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!

Dicho esto, recordemos que el algoritmo de BT necesita esencialmente para funcionar:

i-	Área y perímetro de cada RC (a_i y r_i)
ii-	Área y perímetro de toda el área a "*districtalizar*" (A y R)
iii-	Información de Vecindad (adyacencia), es decir, cuales UBs son adyacentes a cuales otras (o en nuestro caso, cuales radios censales son adyacentes a cuales otros).
iv-	Longitud de la frontera común entre dos UBs adyacentes (t_{ik})
v-	Información referente a los enclaves[104].

Los datos i e ii son rápidamente calculables por cualquier Sistema de Información Geográfica (SIG), mientras que iii y iv no lo son tanto, y requieren algún tipo de codificación para ser obtenidos.

Ahora bien, el *Departamento de Planeamiento y Mapeo Criminal* ha solicitado que en nuestra implementación preliminar utilicemos como muestra de concepto primero una implementación utilizando el lenguaje de programación *Avenue*[105] del software ArcView GIS 3.x de ESRI © debido a la experiencia que sus analistas tienen en esa plataforma y básicamente porque si se prueba que el problema es resoluble de esta forma en esa plataforma entonces es rápidamente *implementable* en Python y ArcPy utilizando la última versión del SIG de ESRI ©. Consecuentemente en el resto de este trabajo haremos continua mención a la codificación de las porciones del algoritmo en *Avenue* junto los resultados obtenidos de su ejecución.

Por otro lado, en la implementación original de Bozkaya (1.999) para la determinación de la matriz de adyacencias y el cálculo de la frontera común, se hace uso de un "truco"

[104] Recordemos que en nuestra implementación la etapa de pre-procesamiento remueve los enclaves.

[105] *Avenue* es un lenguaje de scripting orientado a objetos, no distingue mayúsculas de minúsculas, es compilado y no recursivo. Emplea tres estilos para notificar peticiones a los objetos: postfix, infix y prefix, aunque la notación postfix es la más común (<objeto>.<petición>), la notación infix (<objeto><petición><objeto>) se emplea para operadores aritméticos pero también para cuando se crean intervalos o se se concatenan cadenas, finalmente la notación prefix (<petición><objeto>) se emplea solamente para la negación de operadores. En cuanto a la gramática, *Avenue* emplea la notación Backus-Naur-Form (BNF).

común que implicaba "cargar" dos veces la misma capa y emplear las funciones de algebra de mapas del ArcView para calcular las intersecciones necesarias:

ADJMATRIX

1. Para cada RC i en la capa 1,
   - 1.1 Seleccionar RC 1 en la capa 1.
   - 1.2 Seleccionar aquellos RCs en la capa 2 que intersecan con el RC i de la capa 1.
   - 1.3 De-seleccionar el RC i en la capa 2
   - 1.4 Crear el conjunto índice L de los RCs seleccionados en la capa 2
   - 1.5 Para cada par de RCs i,k donde i es la RC seleccionada en la capa 1, y k ∈ L,
     - 1.5.1 Calcular $R = r_i + r_k$
     - 1.5.2 Combinar las dos RCs en un único polígono y calcular su perímetro r'
     - 1.5.3 Si R > r', indicar que los RCs i y k son adyacentes. Reportar $t_{ik} = R - r'$
     - 1.5.4 Si R = r', indicar que los RCs i y k son adyacentes por puntos.
     - 1.5.5 Deshacer la operación de combinación realizada en el paso 1.5.2
   - 1.6 De-seleccionar todos los RCs de ambos temas.

Como se ve, este algoritmo calcula a la vez t_{ik} (la longitud de la frontera común entre dos UBs adyacentes i y k) y la matriz de adyacencias. Asimismo, define $t_{ik} = 0$ para cualquier par de Radios Censales (RCs) no adyacentes o adyacentes por punto.

El otro componente de información necesario en la formulación original es la referida a los enclaves, para esto Bozkaya (1.999) emplea el siguiente algoritmo:

ENCLAVES

1. Para cada RC i en la capa 1,
   - 1.1 Hacer $E_i = \phi$
   - 1.2 Seleccionar RC i en la capa 1.
   - 1.3 Seleccionar aquellos RCs en la capa 2 que intersecan con el RC i de la capa 1
   - 1.4 De-seleccionar el RC i en la capa 2
   - 1.5 Crear el conjunto L de índices de los RCs actualmente seleccionados en la capa 2
   - 1.6 Usando la matriz de adyacencias ya construida, partir L en L_1, …, L_q de manera tal que cada L_j sea una colección contigua de RCs

1.7 Para cada $L_j \subset L$, chequear si cualquiera de los RCs en L_j son adyacentes a un RC en I \ (L∪ {i}). Si no, hacer $E_i \leftarrow E_i \cup L_j$

1.8 Reportar E_i como el conjunto de RC enclavadas por el RC i.

Ahora bien, como ya hemos señalado varias veces, en nuestro caso, no existen enclaves dentro de los datos a utilizar por el algoritmo de BT, dado que si los hubiera en los datos originales estos han sido removidos en la etapa de pre-procesamiento.

Por otro lado, para la construcción de la matriz de adyacencias, empleamos un algoritmo distinto al de Bozkaya, que en pseudocódigo puede ser descripto como:

```
ADJMATRIX
# X-min, X-max, Y-min, Y-max son los vectores de los valores de los
# "extent" de cada polígono.

For i = 0 to len(Poly_list)-1
For j = i+1 to len(Poly_list)-1
W[i,j] = -1
   Next
Next

For i = 0 to len(Poly_list)-1
W[i,i] = perimeter(Poly_list[i]
Next

For i = 0 to len(Poly_list)-2
   For j = i+1 to len(Poly_list)-1
# Verificar si existe solapamiento en los extent
If ((max(X-min[i], X-min[j]) - min(X-max[i], X-max[j])) >= 0) and
((max(Y-min[i], Y-min[j]) - min(Y-max[i], Y-max[j])) >= 0)
Distancia = dist(Poly_list[i], Poly_list[j])
           If Distancia = 0
              Perimetro = perimeter(union(Poly_list[i],
Poly_list[j]))
If  Perimetro ==  max(W[i,i] + W[j,j])
ENCLAVE !!!!!!!!!!!
              Frontera =  Perimetro - W[i,i] + W[j,j]
W[i,j] = Frontera
W[j,i] = Frontera
           End if
        End if
    Next j
Next i
```

Este algoritmo es dependiente de ciertas funcionalidades de geometría computacional, en particular el cálculo de los *extents* o *bounding boxes* de cada polígono, o funcionalidades de unión e intersección de polígonos.

Por otro lado, en la implementación en *Avenue* efectivamente utilizada, hacemos:

```
'+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
'Generación de matriz de adyacencia
'
for each rec in tabla
  Linea={}
  Poligono1=Tabla.ReturnValue(ShpFld,rec)
  Fraccion=Tabla.ReturnValue(FraFld,rec).asString
  Peri1=Poligono1.ReturnLength
Tabla.SelectByShapes({Poligono1},#VTAB_SELTYPE_NEW)
seleccion=Tabla.getSelection
for each rec1 in Tabla
if (rec=rec1) then Linea.add(Peri1) continue end
if (seleccion.get(rec1)) then
      Poligono2=Tabla.ReturnValue(ShpFld,rec1)
      Peri2=Poligono2.ReturnLength
frontera=((Peri1+Peri2)- (Poligono2.ReturnUnion(poligono1)).ReturnLength)/2
Linea.add(frontera)
else
Linea.add(-1)
end
end
Adyacencia.add(Linea)
Areas.add(Poligono1.returnArea)
Poblacion.add(Tabla.ReturnValue(PobFld,rec))
  Tot_pob= Tot_pob + Poblacion.get(Poblacion.count-1)
end
```

Notemos que nosotros usamos 0 para punto adyacencia, -1 para no adyacencia y t_{ik} para la adyacencia.

Finalmente, atento a las restricciones en tiempo impuestas por el Departamento de Planeamiento y Mapeo Criminal, al igual que la solicitud de presentar en primera instancia el algoritmo de Búsqueda Tabú (BT) utilizando *Avenue*, se ha optado por no codificar la Diversificación e Intensificación Probabilística (DIP) (el componente de memoria adaptativa en el que se encuentra embebida la BT), ahora bien, esto no es una gran restricción, dado que en esencia lo que realiza la DIP es una determinada cantidad de corridas del algoritmo de BT plano y selección de "buenos distritos" que se han encontrado repetidamente, y luego construir el plan final en base a combinaciones de esos *buenos distritos*. Por otro lado, el análisis de sensibilidad realizado por Bozkaya (1.999) y Bozkaya *et. al* (2.003) muestra que la mejora provista por el DIP es de entre un 9 y un 27 % en el criterio de compacidad, por lo que se ha optado por diferir la codificación específica de este componente para la versión definitiva en lenguaje Python que a la fecha se está pensando implementar en el mencionado

Departamento de Planeamiento y Mapeo Criminal del Ministerio de Justicia y Seguridad de la Provincia de Buenos Aires, para fines del año 2.011.

Por otro lado, el tiempo promedio para cada corrida de la BT plana con datos del partido de Berisso es de aproximadamente 30 minutos[106], mientras que para el partido de La Plata, excede las diez (10) horas[107].

[106] Computados sobre una máquina virtual VMWarecon Windows XP ejecutándose sobre un AMD Phenom II X6 1090T y 12 GB de RAM y corriendo la versión 3.3 de ArcView GIS © ESRI.

[107] Más adelante discutiremos ciertas mejoras que es posible realizar a la actual codificación que permitirían acelerar notablemente los tiempos de ejecución del algoritmo. Igualmente notemos que el proceso de districtalización no es un proceso que se realice todos los días y que por lo tanto, cierta espera en la obtención de resultados es tolerable.

6- Testeos y pruebas con datos reales

Comenzaremos rápidamente por presentar algunos resultados obtenidos para el procesamiento del Partido de La Plata, para luego centrarnos en el ajuste de parámetros de la BT realizado utilizando los datos del Partido de Berisso:

El algoritmo en *Avenue* para el cálculo de la matriz de adyacencias de los radios censales del partido de La Plata, produce como resultado una matriz de 704 filas y 704 columnas, donde las primeras columnas y filas son:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
1	*2580.96*	538.537	72.155	-1	-1	-1	-1	-1	-1	-1	-1	-1	274.847	-1	-1	-1	
2	538.537	*1485.63*	383.283	-1	-1	0	268.825	155.622	-1	-1	-1	-1	139.359	-1	-1	-1	
3	72.155	383.283	*1829.66*	66.8426	608.001	193.088	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	
4	-1	-1	66.8426	*2411.42*	1161.59	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
5	-1	-1	608.001	1161.59	*3891.61*	198.088	-1	-1	397.577	-1	-1	-4.55E-13	-1	1496.65	-1	-1	
6	-1	0	193.088	-1	198.088	*944.315*	275.509	-1	277.63	-1	-1	-1	-1	-1	-1	-1	
7	-1	268.825	0	-1	-1	275.509	*1087.92*	274.439	269.147	-1	-1	-1	-1	-1	-1	-1	
8	-1	155.622	-1	-1	-1	-1	274.439	*1441.57*	379.062	-1	-1	-1	570.188	-1	-1	-1	
9	-1	-1	-1	-1	397.577	277.63	269.147	379.062	*2240.18*	343.209	135.943	277.993	-1	0	-1	-1	
10	-1	-1	-1	-1	-1	-1	-1	-1	343.209	*1317.03*	417.03	-1	-1	-1	-1	-1	
11	-1	-1	-1	-1	-1	-1	-1	-1	135.943	417.03	*1303.46*	282.96	-1	-1	-1	-1	
12	-1	-1	-1	-1	-4.55E-13	-1	-1	-1	277.993	-1	282.96	*1121.95*	-1	-4.55E-13	-1	-1	
13	274.847	139.359	-1	-1	-1	-1	-1	570.188	-1	-1	-1	-1	*1419.85*	-1	-1	-1	
14	-1	-1	-1	-1	1496.65	-1	-1	-1	9.09E-13	-1	-1	-4.55E-13	-1	*5957.59*	764.773	574.328	
15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	764.773	*2117.8*	153.882	
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	445.729	-1	*330.482*	
...																	

Figura 20 – Matriz de Adyacencias Radios Censales de La Plata

Notemos que se trata, como era de esperar, de una matriz triangular por lo que en rigor podríamos almacenar solamente la mitad de la misma (algo que se sugiere enfáticamente hacer en la implementación en Python). El valor de -1 se utiliza para señalar UBs no vecinas, y valores de 0 o casi cero[108] para la adyacencia por puntos. Asimismo, notemos que en la diagonal se almacena el perímetro de cada UB.

Por otro lado, esta matriz de adyacencias, nos muestra que en el caso de los radios censales del partido de La Plata, la estructura de contigüidad es tal que:

i)	La menor cantidad de vecinos de una UB es:	2
ii)	La mayor cantidad de vecinos de una UB es:	15
iii)	La moda para la cantidad de vecinos de una UB es:	6 (en 182 casos)
iv)	La mediana para la cantidad de vecinos de una UB es:	6
v)	La media aritmética para la cantidad de vecinos es:	6.33
vi)	La desviación estándar para la cantidad de vecinos es:	1.89

[108] Dependiendo el formato de datos espaciales empleado, es probable que existan ciertos errores en la topología de la capa de Unidades Base y dependiendo también del tipo de datos y precisión empleada es posible que en el caso

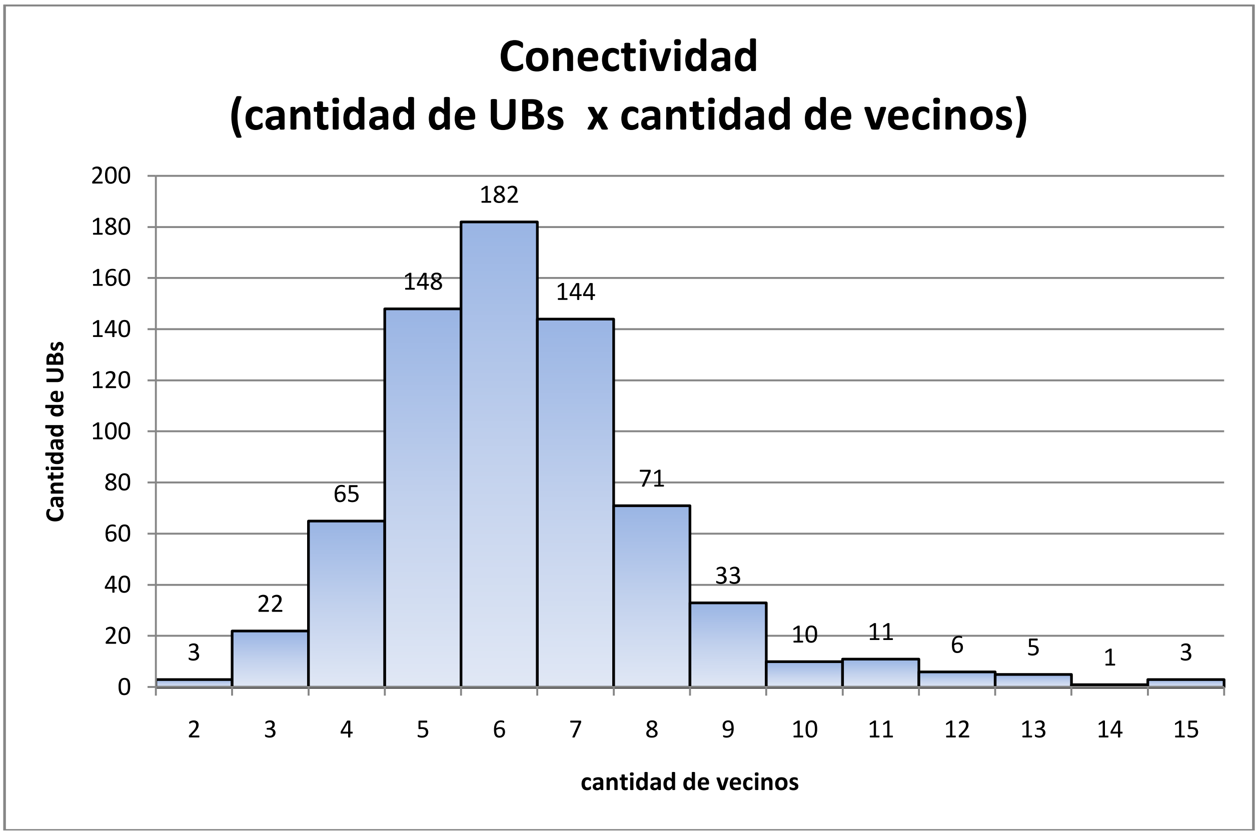


Figura 21 – Conectividad de los Radios Censales de La Plata

Ahora bien, luego del cálculo de la matriz de adyacencias, también se calcula inicialmente las unidades de frontera (que se requieren para el posterior cálculo de la solución inicial)[109]. En nuestro caso, empleamos:

```
'+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
'Lista de UB de frontera
'
UB_frontera={}
for each i in 0..(Cant_UB-1)
suma=0.000
  Fila=Adyacencia.get(i)
for each j in 0..(Cant_UB-1)
if (Adyacencia.get(j).get(i) = -1) then continue end
if (i <> j) then
suma= suma+adyacencia.get(j).get(i)
end
end
if (adyacencia.get(i).get(i)-suma > 0.2) then
    UB_frontera.add(i)
end
end
```

[109]Observemos también, que durante el procesamiento de la Búsqueda Tabú se hace uso de la determinación de las Unidades Base de Frontera al construir los conjuntos de unidades elegibles para participar en los Movimientos Tipo I o Tipo II.

Notemos aquí, que para definir si se trata de una UB de frontera, en esencia sumamos los elementos distintos de -1 de su fila (es decir, sumamos las longitudes de las fronteras de sus vecinos), si esta suma es menor que su perímetro, se trata de una Unidad Base de Frontera (ya que hay una porción de su frontera que no pertenece a ninguno de sus vecinos, sino al exterior de la zona en estudio).

Por otro lado, antes de iniciar el algoritmo de Búsqueda Tabú se emplea el algoritmo INITSOLN para generar una solución inicial mediante agregación a partir de las unidades de la frontera, hasta que se supera por primera vez la cantidad de población requerida. En nuestro caso, a los fines de mantener consistencia con las fracciones censales del censo nacional de población hogares y vivienda 2001 para los partidos de ejemplo, vamos a pedir la misma cantidad de distritos que de fracciones censales tanto para INITSOLN como para el algoritmo de BT[110]. Asimismo, para los 704 radios censales del Partido de La Plata tenemos para la población:

Total de Población (Suma):	582.957
Cantidad de UBs:	704
Media aritmética:	828
Máximo:	2622
Mínimo:	102
Rango:	2520
Desviación Estándar:	343

[110]En el caso del partido de La Plata se trata de cuarenta y siete (47) distritos y en el caso del Partido de Berisso se trata de siete (07) distritos.

Mientras que para las Fracciones Censales actuales tenemos:

Total de Población (Suma):	582.957
Cantidad de Distritos:	47
Media Aritmética:	12.403,340
Máximo:	34.485
Mínimo:	826
Rango:	33659
Desviación Estándar:	6.169

Por lo tanto a los fines de generar la solución inicial, vamos a utilizar en este caso $\bar{P}$ = 12.403, y el siguiente código en *Avenue*[111]:

```
'+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
'Construcción de las regiones
'
'Si no se asignaron todas las unidades y no hay más unidades de
frontera
'se sigue con cualquiera
While (Asignados.count < Cant_UB)
if (UB_frontera.count = 0) then
for each i in 0..(Cant_UB-1)
if (Asignados.FindByValue(i) = -1) then UB_frontera.add(i) end
end
end
  Pivot= UB_frontera.get(Number.MakeRandom(0,UB_frontera.count-1))
  UB_frontera.RemoveObj(Pivot)
Asignados.add(Pivot)
  Lista_UB={Pivot}
Aux_pob=Poblacion.get(Pivot)
Contiguo= true
  While ((Aux_pob < Umbral) and Contiguo)
    Linea=Adyacencia.get(Pivot)
for each i in 0..(Cant_UB-1)
if ((i = Pivot) or (Linea.get(i) = -1)) then continue end
if (Asignados.FindByValue(i) > -1) then continue end
Aux_pob= Aux_pob + Poblacion.get(i)
Lista_UB.add(i)
Asignados.add(i)
if (UB_frontera.FindByValue(i) > -1) then
        UB_frontera.Remove(UB_frontera.FindByValue(i))
end
if (Aux_pob > Umbral) then break end
end
```

[111] Extracto que muestra el proceso de generación inicial y luego el mecanismo de reasignación cuando se obtiene mayor cantidad de distritos que los necesarios. El proceso para obtener más distritos en caso que en la generación inicial culminara con menos de los necesarios es esencialmente un proceso de partición utilizando el distrito de mayor población para ser dividido a la mitad. Incidentalmente, notemos que en las más de 500 corridas realizadas con el algoritmo, ya sea para el partido de La Plata como para el partido de Berisso, jamás se obtuvo una cantidad inferior de distritos al culminar la primera parte del proceso. Esto se debe esencialmente a las características particulares de las Unidades Base empleadas.

```
if (Aux_pob < Umbral) then
Contiguo=(Lista_UB.FindByValue(Pivot)+1) < Lista_UB.count
if (contiguo) then
        Pivot= Lista_UB.get(Lista_UB.FindByValue(Pivot)+1)
end
end
end
  Pob_reg.add(Aux_pob)
  Lista_UB.Sort(true)
Regiones.add(Lista_UB)
end

Reg_UB={}
for each i in 0..(Cant_UB-1) Reg_UB.add(-1)  end
for each i in 0..(Regiones.Count-1)
for each j in Regiones.get(i)
    Reg_UB.Set(j,i)
end
end


'+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
'Reasignacion de UB para lograr la cantidad deseada de regiones
(código 'para el caso en el que se tienen más regiones que las
necesarias)

While (Regiones.Count > Cant_regiones)
Menor_pob=Pob_reg.get(0)
for each pob in Pob_reg
Menor_pob= Menor_pob.Min(pob)
end
  Menor_reg=Pob_reg.FindByValue(Menor_pob)
  'busco regiones vecinas
vecinas={}
for each i1 in Regiones.get(Menor_reg)
for each i2 in 0..(Cant_UB-1)
if (Adyacencia.get(i1).get(i2) = -1) then continue end
if (Menor_reg <> Reg_UB.get(i2)) then
Vecinas.add(Reg_UB.get(i2))
end
end
end
  Vecinas.RemoveDuplicates
Receptora= Vecinas.get(0)
  Menor_pob= Pob_reg.get(Receptora)
for each i in vecinas
if (Menor_pob > Pob_reg.get(i)) then
      Menor_pob= Pob_reg.get(i)
      Receptora= i
end
end
Regiones.get(Receptora).Merge(Regiones.get(Menor_reg))
Regiones.Remove(Menor_reg)

Pob_reg.Set(Receptora,Pob_reg.get(Receptora)+Pob_reg.get(Menor_reg))
```

```
  Pob_reg.Remove(Menor_reg)
for each i in 0..(Cant_UB-1)
if (Reg_UB.get(i) = Menor_reg) then Reg_UB.Set(i,Receptora) end
if (Reg_UB.get(i) > Menor_reg) then Reg_UB.Set(i,Reg_UB.get(i)-1) end
end
end


Sele_asig=Tabla.getSelection
i=0
for each r in regiones
  Sele_asig.ClearAll
for each UB in r
    Sele_asig.Set(UB)
end
  i= i+1
end
```

A modo de ejemplo, para varios diferentes umbrales de población para las regiones iniciales (que van de $\bar{P}$ a $0.75 \cdot \bar{P}$) los tiempos promedio de cálculo para cada una de las etapas anteriores se muestran en la tabla Im.1 en todos los casos computados sobre una máquina virtual VMWare[112] con Windows XP ejecutándose sobre un Phenom II X6 1090T y 12 GB de RAM:

[112] VMWare ThinApp Suite. http://www.vmware.com/products/thinapp/overview.html [Internet, fecha de último acceso marzo 2011]

Tabla Im1. Tiempos y Cantidad de Regiones para INITSOLN

Corrida	Umbral de Poblacion	Tiempo en segundos				Cantidad de Regiones Iniciales
		Generacion Matriz Adyacencias	Generacion UB de Frontera	Generacion Regiones Iniciales	Ajuste a Regiones Pedidas	
1	$1 \cdot \bar{P}$	13	29	15	2	65
2		13	28	14	1	58
3		12	26	15	1	66
4		12	26	13	2	76
5		11	24	12	3	75
$\bar{x}$		**12.200**	**26.600**	**13.800**	**1.800**	**68.000**
s_x		**0.837**	**1.949**	**1.304**	**0.837**	**7.517**
1	$0.95 \cdot \bar{P}$	14	35	17	3	67
2		15	35	18	3	72
3		15	36	17	3	74
4		15	36	18	4	76
5		10	24	12	2	67
$\bar{x}$		**13.800**	**33.200**	**16.400**	**3.000**	**71.200**
s_x		**2.168**	**5.167**	**2.510**	**0.707**	**4.087**
1	$0.90 \cdot \bar{P}$	13	30	14	2	66
2		13	29	15	3	67
3		12	30	13	5	81
4		12	28	14	3	70
5		12	28	13	4	82
$\bar{x}$		**12.400**	**29.000**	**13.800**	**3.400**	**73.200**
s_x		**0.548**	**1.000**	**0.837**	**1.140**	**7.727**
1	$0.85 \cdot \bar{P}$	13	29	13	5	81
2		13	29	14	4	86
3		11	24	12	4	74
4		12	26	12	4	77
5		12	26	13	3	75
$\bar{x}$		**12.200**	**26.800**	**12.800**	**4.000**	**78.600**
s_x		**0.837**	**2.168**	**0.837**	**0.707**	**4.930**
1	$0.80 \cdot \bar{P}$	11	25	12	4	84
2		11	26	12	5	74
3		12	27	12	5	82
4		12	28	13	5	76
5		12	28	14	5	77
$\bar{x}$		**11.600**	**26.800**	**12.600**	**4.800**	**78.600**
s_x		**0.548**	**1.304**	**0.894**	**0.447**	**4.219**
1	$0.75 \cdot \bar{P}$	14	32	15	11	79
2		14	32	15	11	84
3		14	33	15	9	87
4		14	34	16	9	77
5		14	34	16	8	86
$\bar{x}$		**14.000**	**33.000**	**15.400**	**9.600**	**82.600**
s_x		**0.000**	**1.000**	**0.548**	**1.342**	**4.393**
$\bar{x}_{general}$		**12.700**	**29.233**	**14.133**	**4.433**	**75.367**
$s_{x\_general}$		**1.317**	**3.711**	**1.833**	**2.661**	**7.252**

Ajuste de Parámetros para la Búsqueda Tabú:

Recordemos que en este estudio trabajamos con datos de la ciudad de La Plata, capital de la Provincia de Buenos Aires, específicamente estamos empleando la información correspondiente a Radios Censales (RC) del Censo de Población, Hogares y Vivienda 2001 en formato *shapefile* (© ESRI) de información geográfica:

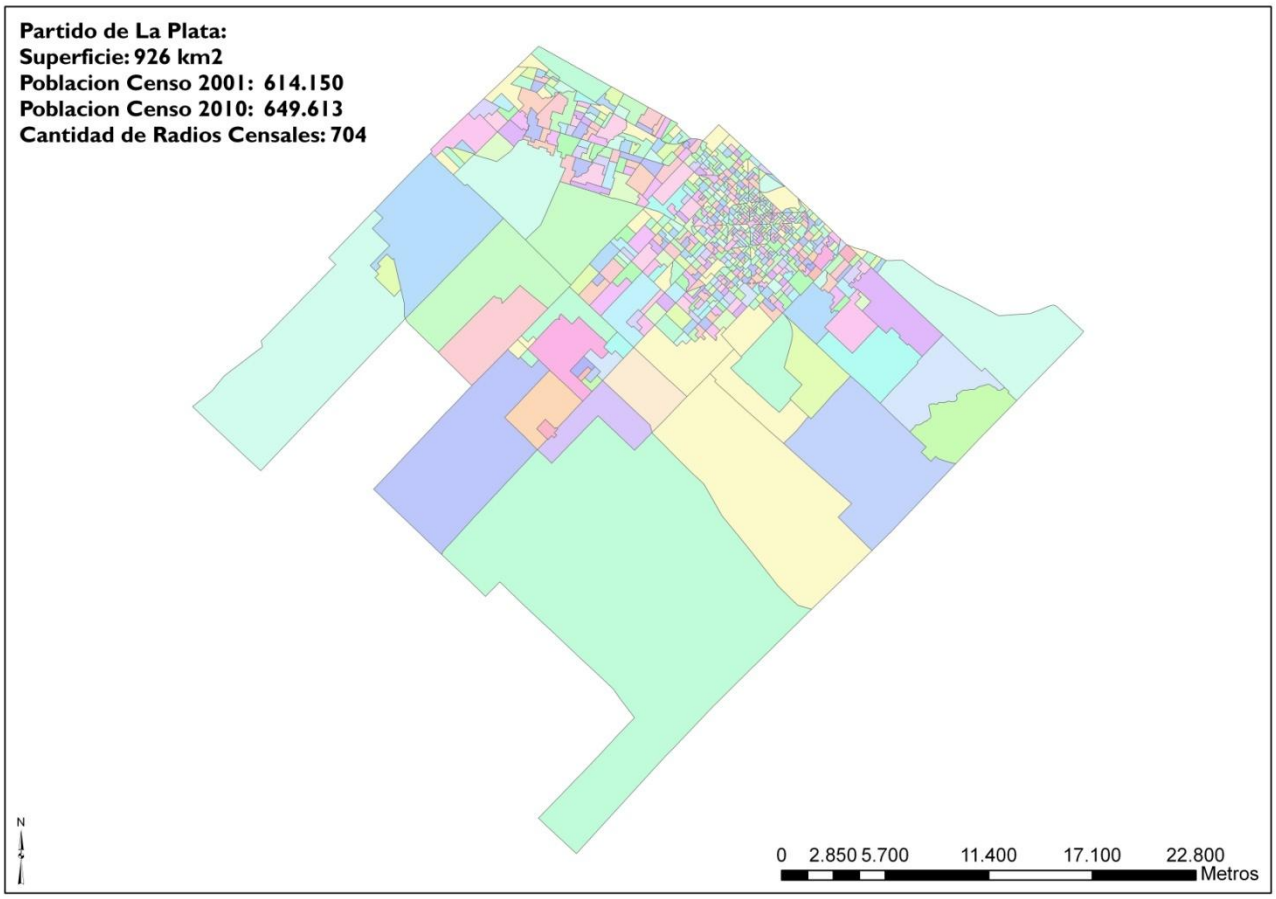


Figura 22 – Radios Censales del Partido de La Plata[113]

Por otro lado, para el partido de Berisso, tenemos:

[113]Notar que la población reportada para el censo 2.001 corresponde en realidad con los datos de proyección al 01 de junio del 2.001 y no exactamente a los 582.957 habitantes relevados en el operativo censal.

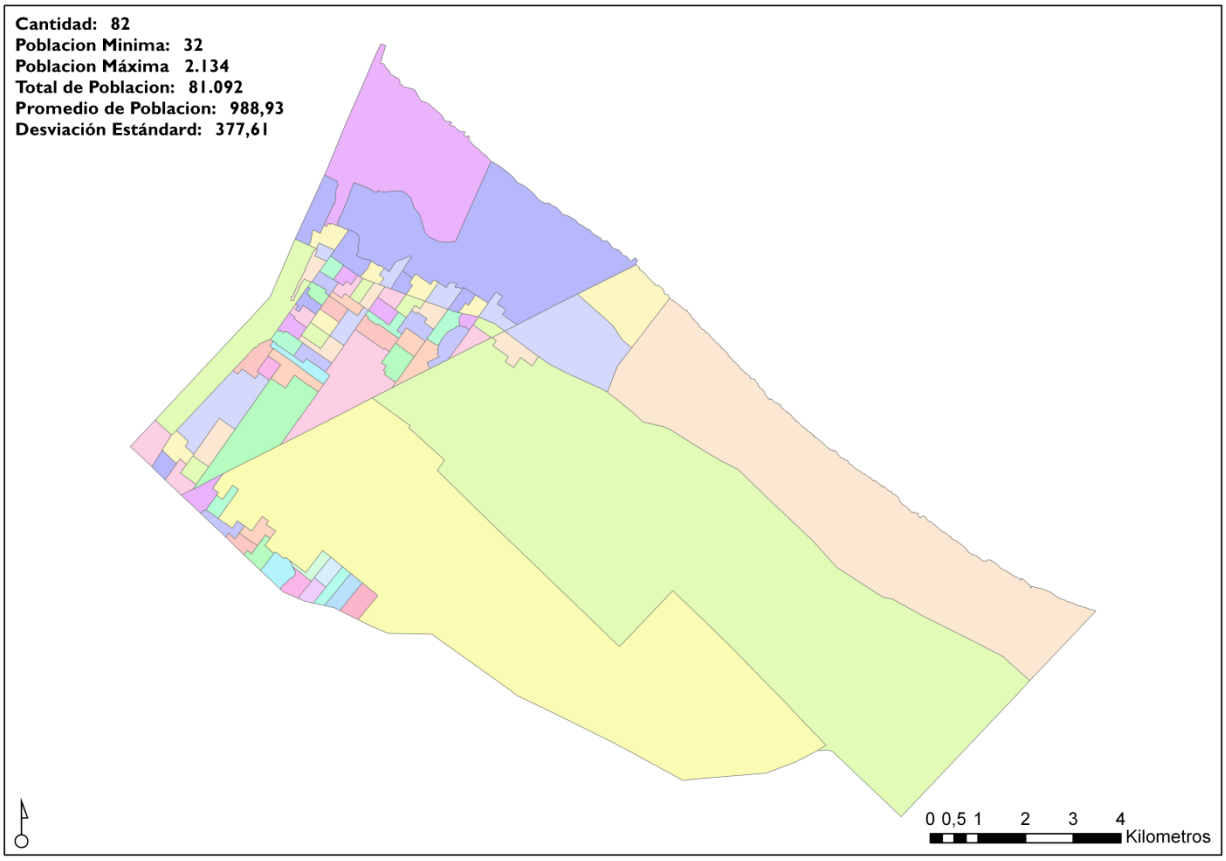


Figura 23 – Radios Censales del Partido Berisso

Asimismo, notemos que cada Radio Censal (RC), se tiene asociado un determinado conjunto de información disponible, de la cual en este trabajo emplearemos solamente:

a- Cantidad Total de Población
b- Superficie
c- Perímetro

Por otro lado, una pieza central de información necesaria para el testeo es el plan de districtalización existente, en este caso correspondiente a las **47** fracciones censales del partido de La Plata y las **7** del partido de Berisso:

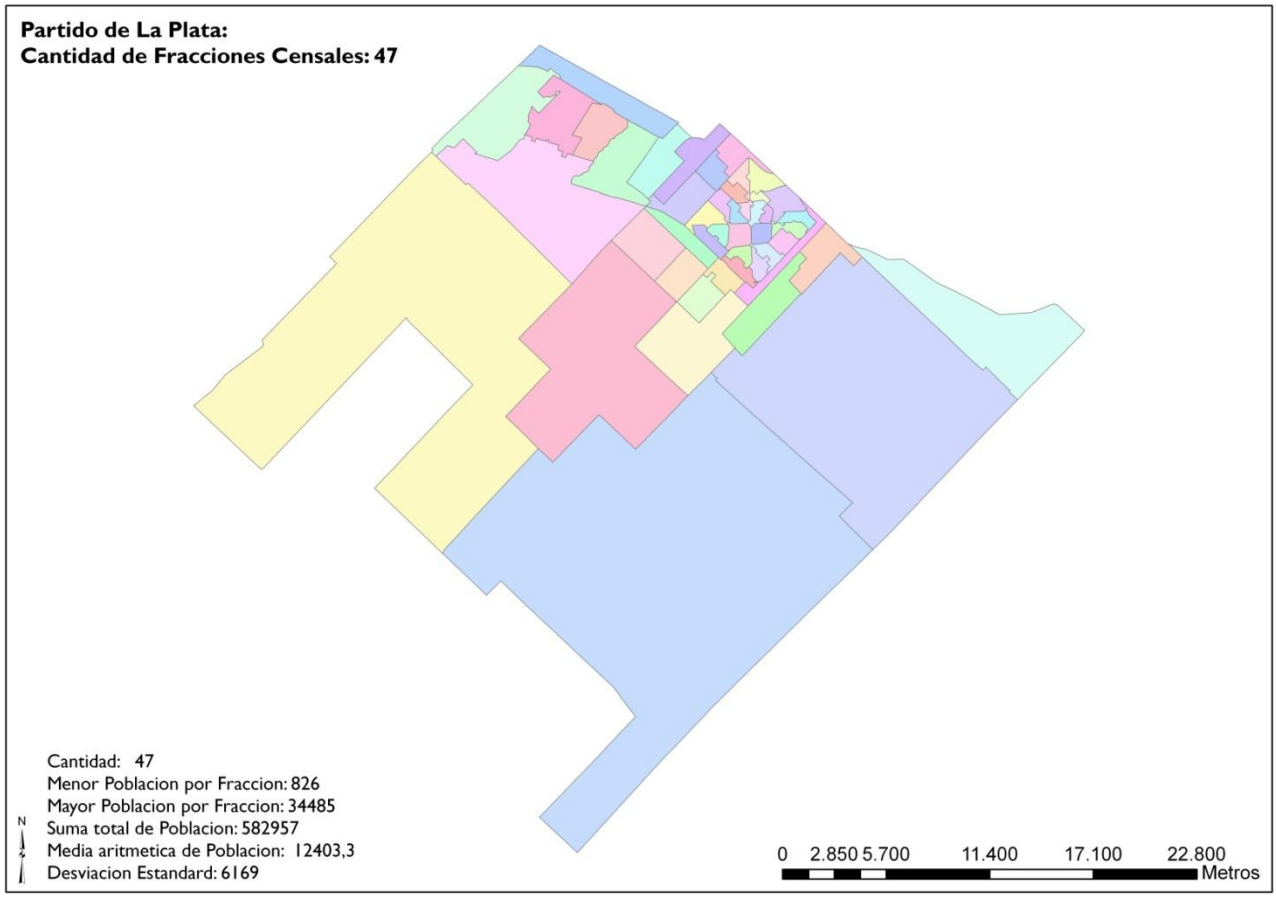


Figura 24 – Fracciones Censales del Partido de La Plata

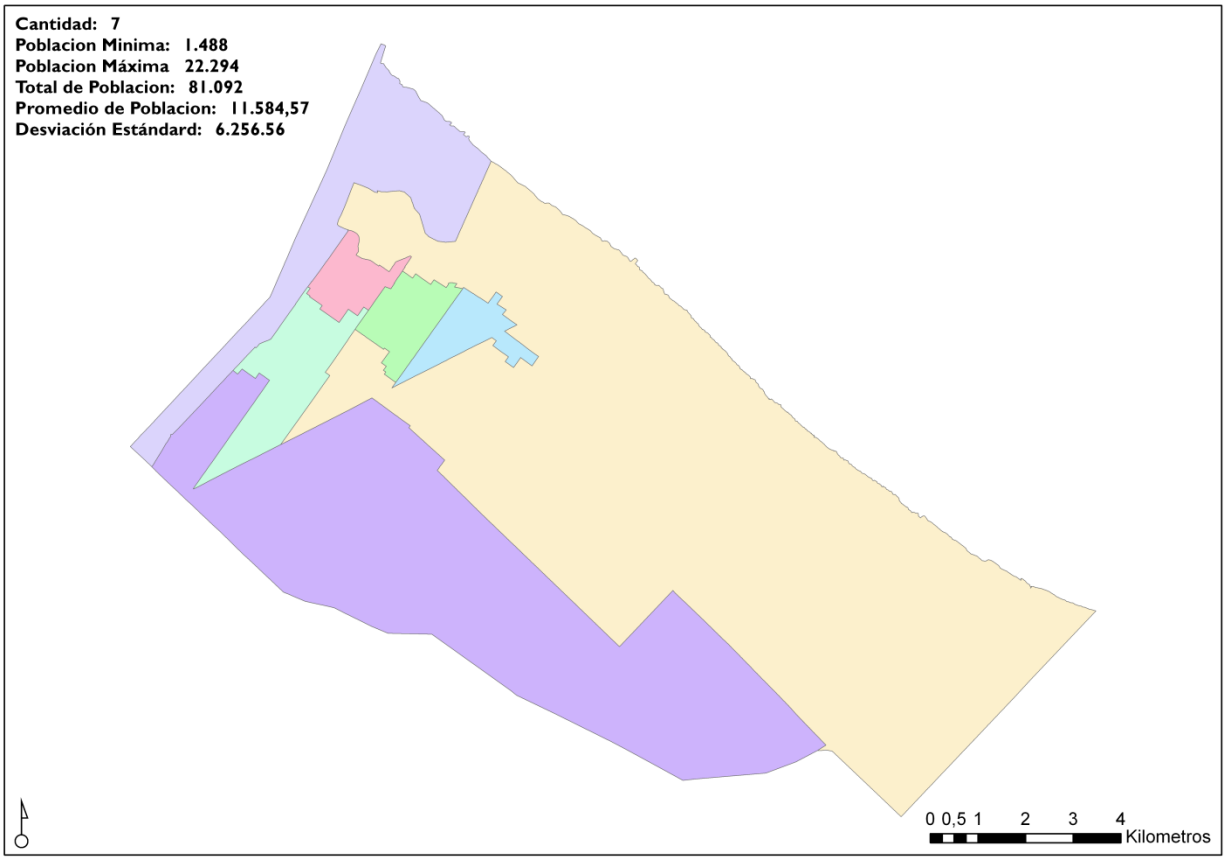


Figura 25 – Fracciones Censales del Partido de Berisso

Notemos que la cantidad total de población a emplear en nuestros cálculos es de 582.957 habitantes para el Partido de La Plata y de 81.092 para el partido de Berisso, por lo que en base a la cantidad de distritos (fracciones) a crear, partimos de una población promedio por fracción objetivo de: 12.403 habitantes en el caso de La Plata y de 11.584 en el caso de Berisso.

Finalmente, a los fines de compatibilizar la *integridad de comunidades* se han utilizado los límites de las localidades del partido de La Plata:

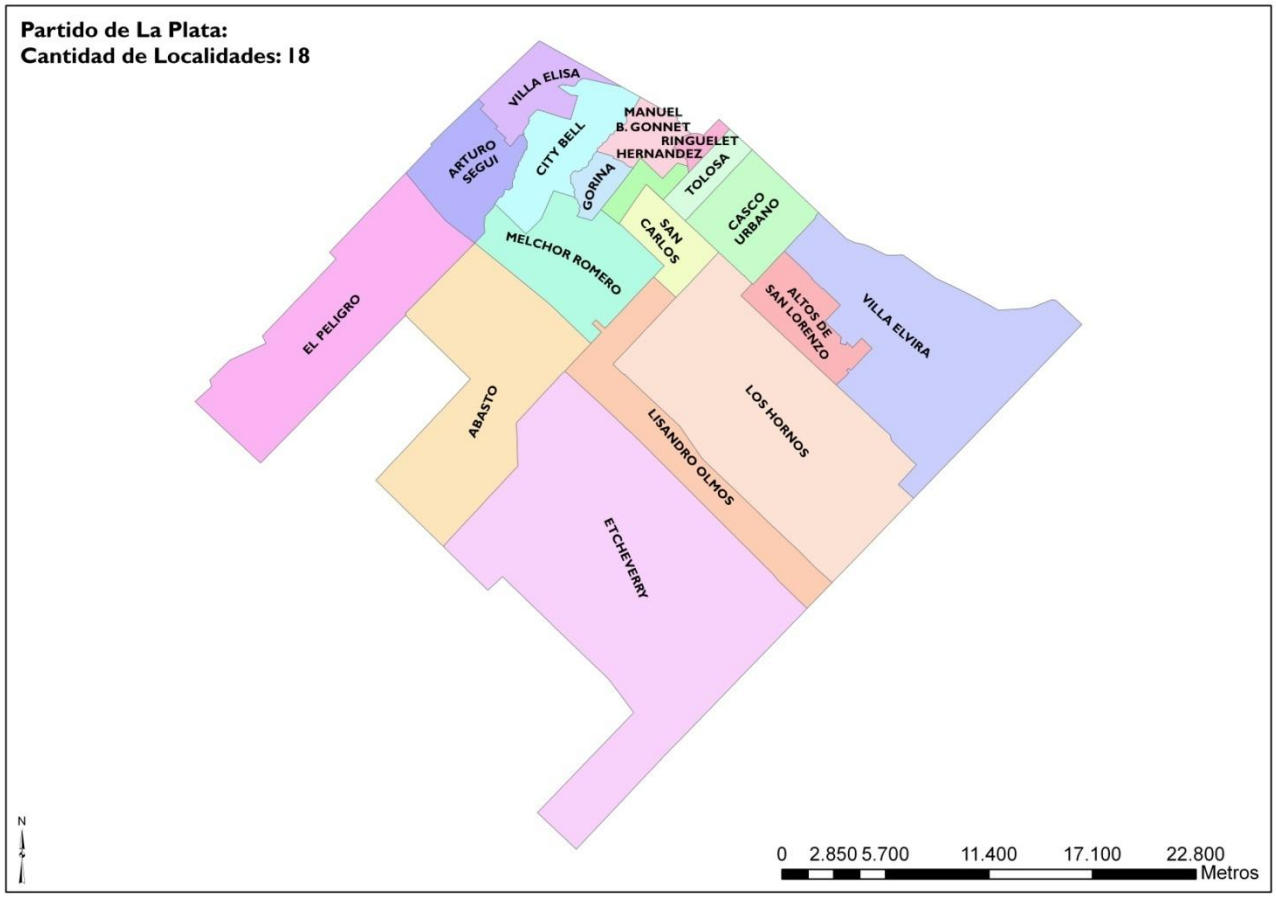


Figura 26 – Localidades del Partido de La Plata

Diseño Experimental para el ajuste de los parámetros:

Bozkaya (1.999)realiza un profundo y pormenorizado estudio computacional de los parámetros del algoritmo (tanto de la BT como de la DIP) y lo realiza en dos etapas:

1- **Etapa 1:** Experimentos necesarios para determinar "buenos" conjuntos de parámetros para el algoritmo, es decir, el ajuste de los parámetros fundamentales del algoritmo de Búsqueda Tabú y Diversificación e Intensificación Probabilística.

2- **Etapa 2:** Experimentos necesarios para determinar "buenos" planes de districtalización bajo diferentes escenarios.

Como ya se ha visto, tanto el algoritmo de BT como el de PDI descriptos precedentemente contienen una importante cantidad de parámetros que deben ser definidos *a priori*. Por lo tanto, es condición previa para intentar resolver problemas de districtalización con los Radios Censales de la ciudad de La Plata o Berisso, asignar valores adecuados a estos

parámetros del algoritmo, lo que requiere obviamente un tiempo de experimentación. Esta es justamente, la etapa 1 del estudio computacional.

Una vez que identifican los valores adecuados en la primera etapa, se empleará el algoritmo para resolver problemas de districtalización en la Ciudad de Berisso[114].

Etapa 1: Ajuste de los parámetros del algoritmo

Previo a iniciar la discusión del tema, es fundamental señalar que el objetivo ***no*** es encontrar un conjunto de parámetros ***genéricos***, sino más bien teniendo en cuenta el territorio en estudio ajustar el algoritmo en consecuencia. Como recuerda Bozkaya (1.999) los procesos de districtalización no se realizan muy frecuentemente, por lo que cuando se realizan es posible prestar exclusiva atención al territorio en el que se aplican.

Los parámetros críticos para la performance de la BT y de la DIP son:

$[\theta_{min}, \theta_{max}]$	el rango permitido para la generación de las permanencias en estado tabú,
α	el valor inicial del coeficiente de penalización para la violación del balance poblacional,
μ	el valor de la frecuencia de actualización de α,
$\bar{\mu}$	el valor mínimo de soluciones factibles / no factibles necesario para actualizar α,
ρ	el valor del factor de escala para la memoria basada en la frecuencia,
T	el valor de la cantidad de sucesivas iteraciones planas de BT permitido sin mejora,
s	el valor de la memoria adaptativa (cantidad de soluciones construidas),
M	el valor de la cantidad de iteraciones de la DIP.

Los primeros seis (06) parámetros están asociados con el algoritmo de BT plano, mientras que los últimos dos (02) están asociados con el algoritmo DIP. Por otro lado, el primer parámetro puede de hecho ser considerado como dos parámetros: 1) $\bar{\theta} = \frac{\theta_{min} + \theta_{max}}{2}$, correspondiente al punto medio del rango de permanencia tabú, y 2) $\theta' = \theta_{max} - \theta_{min}$la amplitud o "ancho" del rango de permanencia tabú.

Dado que el conjunto de parámetros puede ser dividido en aquellos asociados con la BT y aquellos asociados con la DIP, Bozkaya subdivide la Etapa 1 en dos partes: 1) En la primera parte, el objetivo es encontrar buenos valores para los parámetros de la BT, mientras que 2) en la segunda parte se ajustan los valores de *s* y *M* usando los valores ya encontrados en la primera parte, nosotros obviamente emplearemos sólo la primer fase.

Ahora bien, no es computacionalmente práctico en la primera parte testear todas las combinaciones posibles de todos los valores candidatos para los siete parámetros de la BT (recordar que el rango permitido para la generación de las permanencias en estado tabú ha

[114]En este trabajo no realizaremos estudios bajo diferentes escenarios sino más bien compararemos los resultados de la BT simple con los criterios de población y compacidad contra los resultados de las Fracciones Censales del Censo 2001.-

sido descompuesto en dos parámetros). Es más, aun cuando solamente consideráramos dos valores posibles por parámetro tendríamos en este caso 2•2•2•2•2•2•2 = 2^7 = 128 combinaciones posibles. Es por esto que se ha implementado un diseño fraccional factorial en primera instancia. El objetivo es rápidamente identificar los parámetros que son más influyentes en la calidad de la mejor solución encontrada por el algoritmo, posteriormente estos parámetros son sujetos a nuevos testeos computacionales detallados en la primera parte de la Etapa 1.

Un diseño fraccional factorial[115] puede ser definido[116] como "Un experimento factorial en el que solamente una fracción adecuadamente elegida de las combinaciones de tratamiento requeridas para el experimento factorial completo se elige para ser testeada". Aun cuando el número de factores, *k*, en un diseño experimental dado, sea pequeño, las 2^k corridas necesarias en un factorial completo pueden ser crecer rápidamente. La solución a este problema es utilizar solamente una fracción de las corridas especificadas en el diseño factorial completo. Obviamente que corridas realizar y cuáles no realizar es la pregunta de importancia en este caso. Generalmente se elige una fracción tal como ½ o ¼ de las corridas necesarias[117].

Un diseño fraccional factorial adecuadamente elegido para experimentos con dos niveles posee las propiedades deseables de ser balanceado y ortogonal (i.e. balanceado ya que todas las combinaciones de tratamiento tienen la misma cantidad de observaciones, y ortogonal dado que los efectos de cualquier parámetro se equilibran a través de los efectos de los otros factores)

El propósito básico de un diseño fraccional factorial es investigar en forma económica las relaciones significativas de causa y efecto en un dado *seteo* experimental. Sin entrar en los detalles del método, y hablando en términos generales, en diseños de *resolución tres*, y algunas veces *cuatro*, buscamos descartar los pocos efectos importantes principales de los muchos otros menos importantes, y es por eso que a estos diseños se los denomina diseños de efectos principales o diseños de detección.

Existen referencias bibliográficas muy útiles en relación con diseños fraccionales factoriales de dos niveles y hasta 11 factores publicados originalmente en el libro *Statistics for Experimenters* de G.E.P. Box,W.G. Hunter y J.S. Hunter (1.978) y también en el libro *Design and Analysis of Experiments, 5th edition* de Douglas C. Montgomery (2.000). Asimismo en

[115] Ver por ejemplo http://www.itl.nist.gov/div898/handbook/pri/section3/pri334.htm del National Institute of Standards and Technology de los EEUU. [Internet, accedido por última vez en marzo 2011]

[116] ASQC (1983) American Society for Quality Control, actualmente conocida como ASQ (American Society for Quality)

[117] Notemos que en este caso vamos a aplicar un diseño fraccional factorial con dos niveles (un valor mínimo y un valor máximo para cada parámetro), esto es básicamente por ser el diseño más popular en la ciencia de la ingeniería. Si fuere necesario considerar diseños fraccionales factoriales con tres niveles o más, sugerimos consultar también http://www.itl.nist.gov/div898/handbook/pri/section3/pri33a.htm[Internet, accedido por última vez en marzo 2011].

Internet, podemos consultar la cantidad y tipo de corridas necesarias para un dado diseño fraccional factorial de dos niveles en http://www.itl.nist.gov/div898/handbook/pri/section3/pri3347.htm [Internet, accedido por última vez en marzo 2.011].

En este caso y nuevamente siguiendo a Bozkaya utilizaremos un diseño fraccional factorial 2^{7-4} de resolución tres (iii), cuya tabla de corridas está dada por[118]:

[118] http://www.itl.nist.gov/div898/handbook/pri/section3/eqns/2to7m4.txt

```
2**(7-4) FRACTIONAL FACTORIAL DESIGN
NUMBER OF LEVELS FOR EACH FACTOR = 2
NUMBER OF FACTORS                = 7
NUMBER OF OBSERVATIONS           = 8 (A SATURATED DESIGN)
RESOLUTION                       = 3  (CAUTION!  MAIN EFFECTS ARE
                                       CONFOUNDED WITH 2-TERM INTERACTIONS)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 FACTOR    DEFINITION         CONFOUNDING STRUCTURE
   1           1                1 + 24 + 35 + 67 + HIGHER-ORDER INTERACTIONS
   2           2                2 + 14 + 36 + 57 + HIGHER-ORDER INTERACTIONS
   3           3                3 + 15 + 26 + 47 + HIGHER-ORDER INTERACTIONS
   4          12                4 + 12 + 37 + 56 + HIGHER-ORDER INTERACTIONS
   5          13                5 + 13 + 27 + 46 + HIGHER-ORDER INTERACTIONS
   6          23                6 + 17 + 23 + 45 + HIGHER-ORDER INTERACTIONS
   7         123                7 + 16 + 25 + 34 + HIGHER-ORDER INTERACTIONS
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  12                           12 +  4 + 37 + 56 + HIGHER-ORDER INTERACTIONS
  13                           13 +  5 + 27 + 46 + HIGHER-ORDER INTERACTIONS
  14                           14 +  2 + 36 + 57 + HIGHER-ORDER INTERACTIONS
  15                           15 +  3 + 26 + 47 + HIGHER-ORDER INTERACTIONS
  16                           16 +  7 + 25 + 34 + HIGHER-ORDER INTERACTIONS
  17                           17 +  6 + 23 + 45 + HIGHER-ORDER INTERACTIONS
  23                           23 +  6 + 17 + 45 + HIGHER-ORDER INTERACTIONS
  24                           24 +  1 + 35 + 67 + HIGHER-ORDER INTERACTIONS
  25                           25 +  7 + 16 + 34 + HIGHER-ORDER INTERACTIONS
  26                           26 +  3 + 15 + 47 + HIGHER-ORDER INTERACTIONS
  27                           27 +  5 + 13 + 46 + HIGHER-ORDER INTERACTIONS
  34                           34 +  7 + 16 + 25 + HIGHER-ORDER INTERACTIONS
  35                           35 +  1 + 24 + 67 + HIGHER-ORDER INTERACTIONS
  36                           36 +  2 + 14 + 57 + HIGHER-ORDER INTERACTIONS
  37                           37 +  4 + 12 + 56 + HIGHER-ORDER INTERACTIONS
  45                           45 +  6 + 17 + 23 + HIGHER-ORDER INTERACTIONS
  46                           46 +  5 + 13 + 27 + HIGHER-ORDER INTERACTIONS
  47                           47 +  3 + 15 + 26 + HIGHER-ORDER INTERACTIONS
  56                           56 +  4 + 12 + 37 + HIGHER-ORDER INTERACTIONS
  57                           57 +  2 + 14 + 36 + HIGHER-ORDER INTERACTIONS
  67                           67 +  1 + 24 + 35 + HIGHER-ORDER INTERACTIONS
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
DEFINING RELATION       = I = 124 = 135 = 236 = 347 = 257 = 167 = 456
DEFINING RELATION (CONT.)   = 1237 = 2345 = 1346 = 1256 = 1457
DEFINING RELATION (CONT.)   = 2467 = 3567 = 1234567

REFERENCE--BOX, HUNTER & HUNTER, STAT. FOR EXP., PAGE 410, 391, 392.
NOTE--THIS DESIGN IS EQUIVALENT TO A TAGUCHI L8 DESIGN
REFERENCE--TAGUCHI, SYS.OF EXP. DES., VOL. 2.
NOTE--IF POSSIBLE, THIS (AS WITH ALL EXPERIMENT DESIGNS) SHOULD BERUN IN RANDOM ORDER (SEE DATAPLOT'S RANDOM
PERMUTATION FILES).
NOTE--TO READ THIS FILE INTO DATAPLOT--
      SKIP 75
      READ 2TO7M4.DAT X1 TO X7
DATE--DECEMBER 1988
NOTE--IN THE DESIGN BELOW, "-1" REPRESENTS THE "LOW"  SETTING OF A FACTOR
                         "+1" REPRESENTS THE "HIGH" SETTING OF A FACTOR
NOTE--ALL FACTOR EFFECT ESTIMATES WILL BE OF THE FORMAVERAGE OF THE "HIGH" - AVERAGE OF THE "LOW"
X1   X2   X3   X4   X5   X6   X7
   -------------------------
   -1   -1   -1   +1   +1   +1   -1
   +1   -1   -1   -1   -1   +1   +1
   -1   +1   -1   -1   +1   -1   +1
   +1   +1   -1   +1   -1   -1   -1
   -1   -1   +1   +1   -1   -1   +1
   +1   -1   +1   -1   +1   -1   -1
   -1   +1   +1   -1   -1   +1   -1
   +1   +1   +1   +1   +1   +1   +1
```

Tabla lm2 – Diseño Fraccional Factorial 2^{7-4}

Corridas	Parámetros						
1	$\overline{\theta}$	θ'	α	μ	$\overline{\mu}$	ρ	T
2	-	-	-	+	+	+	-
3	+	-	-	-	-	+	+
4	-	+	-	-	+	-	+
5	+	+	-	+	-	-	+
6	+	-	+	-	+	-	-
7	-	+	+	-	-	+	-
8	+	+	+	+	+	+	+

En esta tabla, cada parámetro puede tomar un valor inferior y un valor superior (indicados por los signos – y +, respectivamente), y se observa que solamente se requieren ocho (08) corridas en lugar de evaluar las 128 combinaciones del diseño completo.

Cada corrida se realiza utilizando un β = 25 % y cinco (05) diferentes soluciones iniciales aleatorias, empleando la **primer medida de compacidad**, lo que da un total de cuarenta (40) corridas. En todos los casos la función objetivo se emplea con una ponderación de 10 para el criterio de balance poblacional y de 1 para el criterio de compacidad.

Una vez que se completan estas corridas y se registran los valores de la función objetivo obtenidos para cada combinación, el siguiente paso es encontrar aquellos parámetros que tienen un mayor impacto en la calidad de la solución en respuesta a su testeo con valores "altos" y "bajos". Para cada parámetro se calcula la diferencia entre la suma de valores de la función objetivo para las corridas con el signo "-"con la suma de los valores de la función objetivo para las corridas con el signo "+". Esta diferencia si divide por cuatro (4), que es la cantidad de signos "-" o signos "+" en cada columna. El valor resultante indica la influencia relativa del parámetro en la calidad de la solución con respecto a los otros parámetros. Obviamente este análisis ignora la interdependencia de los parámetros (es decir un *seteo* de un parámetro que consistentemente devuelve buenos valores con otro *seteo* particular de otro de los parámetros).

La siguiente tabla muestra los valores inferiores y superiores testeados para cada uno de los siete (7) parámetros. Notemos que θ' = 0 implica el uso de permanencias tabú determinísticas. Por otro lado, los valores inferiores y superiores para $\overline{\mu}$ se expresan en términos de μ, y se mantiene siempre que $\overline{\mu} > \mu/2$.

Tabla lm3 – Valores máximos y mínimos utilizados en las corridas del diseño fraccional factorial 2^{7-4}

	Parámetros						
	$\overline{\theta}$	θ'	α	μ	$\overline{\mu}$	ρ	T
Valor Inferior (-)	15	0	0.5	15	0.6μ (= 9)	0.05	115$\sqrt{m}$
Valor superior (+)	95	20	2	150	μ	0.2	230$\sqrt{m}$

Los resultados de las corridas para la determinación de los parámetros más críticos muestran:

Tabla lm4 – Resultados de las Corridas para el partido de Berisso

Test	$\bar{\theta}$	θ'	α	μ	$\bar{\mu}$	ρ	*T*	C1	C2	C3	C4	C5	MIN.	PROMEDIO
1	-1	-1	-1	1	1	1	-1	0,706	2,251	1,039	1,810	2,755	1.039	1,912
2	1	-1	-1	-1	-1	1	1	0,945	1,089	2,702	2,006	3,807	0,945	2.110
3	-1	1	-1	-1	1	-1	1	1,352	2,058	2,487	2,939	2,852	1,352	2.338
4	1	1	-1	1	-1	-1	-1	2,029	3,041	0,693	2,501	1,685	0,693	1.990
5	-1	-1	1	1	-1	-1	1	2,771	6,571	4,041	7,802	5,541	2,771	5.345
6	1	-1	1	-1	1	-1	-1	0,735	0,960	2,816	2,709	2,931	0,735	2.030
7	-1	1	1	-1	-1	1	-1	2,165	0,773	4,174	4,934	4,853	0,773	3.380
8	1	1	1	1	1	1	1	2,753	2,707	0,939	0,859	5,781	0,859	2.608
MINIMO	-0,676	-0,453	0.277	0,389	-0.299	-0.484	0,671							
PROMEDIO	-1.059	-0.271	1.253	0.499	-0,984	-0.423	0,772							
VALOR	0.868	0.362	0.765	0.444	0.642	0.454	0.722							

La función objetivo empleada estaba compuesta por el término de población con ponderación 10 y la primer medida de compacidad con ponderación 1, en todos los casos se utilizó β = 0.25. Las columnas C1 a C5 representan el valor de la función objetivo reportado por el algoritmo para cinco soluciones aleatorias iniciales. Las columnas MIN y PROMEDIO son simplemente el menor valor y el valor promedio para cada fila. Las últimas dos filas contienen las estadísticas calculadas utilizando el valor mínimo o el valor promedio de cada test y la fórmula dada más arriba. Por ejemplo, el valor de -0.676 se obtiene haciendo ¼(0.945+0.693+0.735+0.859-1.039-1.352-2.771-0.773). Los valores positivos (negativos) de esta estadística muestran que el valor bajo (alto) del parámetro elegido ha producido menores valores de la función objetivo en promedio, por lo tanto se lo prefiere al valor alto (bajo) de ese parámetro. La magnitud de la desviación absoluta de cero indica la influencia relativa de este parámetro en la calidad de la solución. Dado que hay dos criterios en esa tabla (el valor mínimo y el valor promedio) se ha optado rápidamente por calcular la fila VALOR como el promedio de los valores absolutos de estos dos criterios, de manera de medir el apartamiento del cero. Claramente $\bar{\theta}$ es el parámetro más crítico, por lo que se lo apartará para realizar mayores testeos. Por otro lado, la elección del segundo parámetro crítico no es tan sencilla, Ahora bien, si se combinan los resultados de ambas filas y además se considera que la media aritmética es fuertemente influenciada por valores extremos, se observa que son dos los parámetros candidatos a ser testeados, por un lado *T* que es la cantidad de iteraciones sucesivas que se permiten sin que mejore la solución y que por lo tanto forma parte del criterio de parada, y por otro lado los valores relacionados con la frecuencia de actualización de la penalidad por violación del balance poblacional (α). Dado que en esencia trabajos anteriores han identificado a $\bar{\theta}$ y a μ como los dos parámetros críticos a ser testeados en más detalle nos inclinamos por mantener el mismo esquema en este trabajo.

Incidentalmente es interesante notar que a diferencia de lo concluido por Bozkaya (1.999), existe una diferencia apreciable entre el empleo de rangos tabú determinísticos o aleatorios.

Para concluir, cerraremos la fase de determinación de parámetros críticos determinando con mayor justeza los valores para $\bar{\theta}$y μ. Para esto se realizarán un conjunto de corridas del algoritmo BT, donde:

$\theta' = 10, \alpha = 1, \bar{\mu} = \mu, \rho = 0.1\ yT = 230\sqrt{m}$ todos estos valores consistentes con estudios previos e intermedios a los valores máximo y mínimos testeados precedentemente, mientras que por otro lado, testearemos diferentes combinaciones para $\bar{\theta}$ y μ, utilizando como guía estudios los estudios de sensibilidad antedichos.

Habiendo realizado en exceso de 300 simulaciones para testear combinaciones para $\bar{\theta}$y μ, encontramos que la mejor combinación está dada por: **μ = 40** y **θ_{min} = 50,θ_{max} = 60**.

Con estos parámetros se corrieron entonces 20 simulaciones para determinar la mejor districtalización para el partido de Berisso (β = 0.25, función objetivo controlada por población y compacidad (primer métrica), y una ponderación de 10 para la población y 1 para la compacidad). Con estos parámetros, la mejor solución encontrada tiene una función objetivo de **0.693**.Notemos que la districtalización provista por las fracciones censales tiene una función objetivo de **2.590.** Es más, notemos que $\bar{P}$ = 11.584, P_{max} = 14.481 y P_{min} = 8.688 (β = 0.25), entonces:

Tabla lm5 – Datos de Población para las Fracciones de Berisso

ID	Población	$\bar{P}$ - P_{max}	P_{min} -$\bar{P}$	Población fuera de Rango
1	9750,0000	-4730,7143	-1061,57142857143	0,0000
2	13620,0000	-860,7143	-4931,57142857143	0,0000
3	13418,0000	-1062,7143	-4729,57142857143	0,0000
4	1488,0000	-12992,7143	7200,42857142857	7200,4286
5	22294,0000	7813,2857	-13605,57142857140	7813,2857
6	14900,0000	419,2857	-6211,57142857143	419,2857
7	5622,0000	-8858,7143	3066,42857142857	3066,4286

Es decir, el 60 % de los distritos tiene población fuera del rango solicitado. Mientras que:

Tabla lm6 – Datos de Población para la Districtalización Optima vía BT

ID	Población	$\overline{P}$ - P_{max}	P_{min} - $\overline{P}$	Población fuera de Rango
1	12906	-1574,7143	-4217,57142857143	0,0000
2	10933	-3547,7143	-2244,57142857143	0,0000
3	11802	-2678,7143	-3113,57142857143	0,0000
4	11051	-3429,7143	-2362,57142857143	0,0000
5	11314	-3166,7143	-2625,57142857143	0,0000
6	11878	-2602,7143	-3189,57142857143	0,0000
7	11208	-3272,7143	-2519,57142857143	0,0000

Por lo que **todos** los distritos tienen población dentro del rango solicitado. Asimismo, la métrica de compacidad es de **0,993** para las fracciones censales y de **0.693** para la districtalización óptima por BT. Es decir el algoritmo provee una solución que mejora **ambos criterios**.

Figura 27 – Distritos Óptimos por Búsqueda Tabú

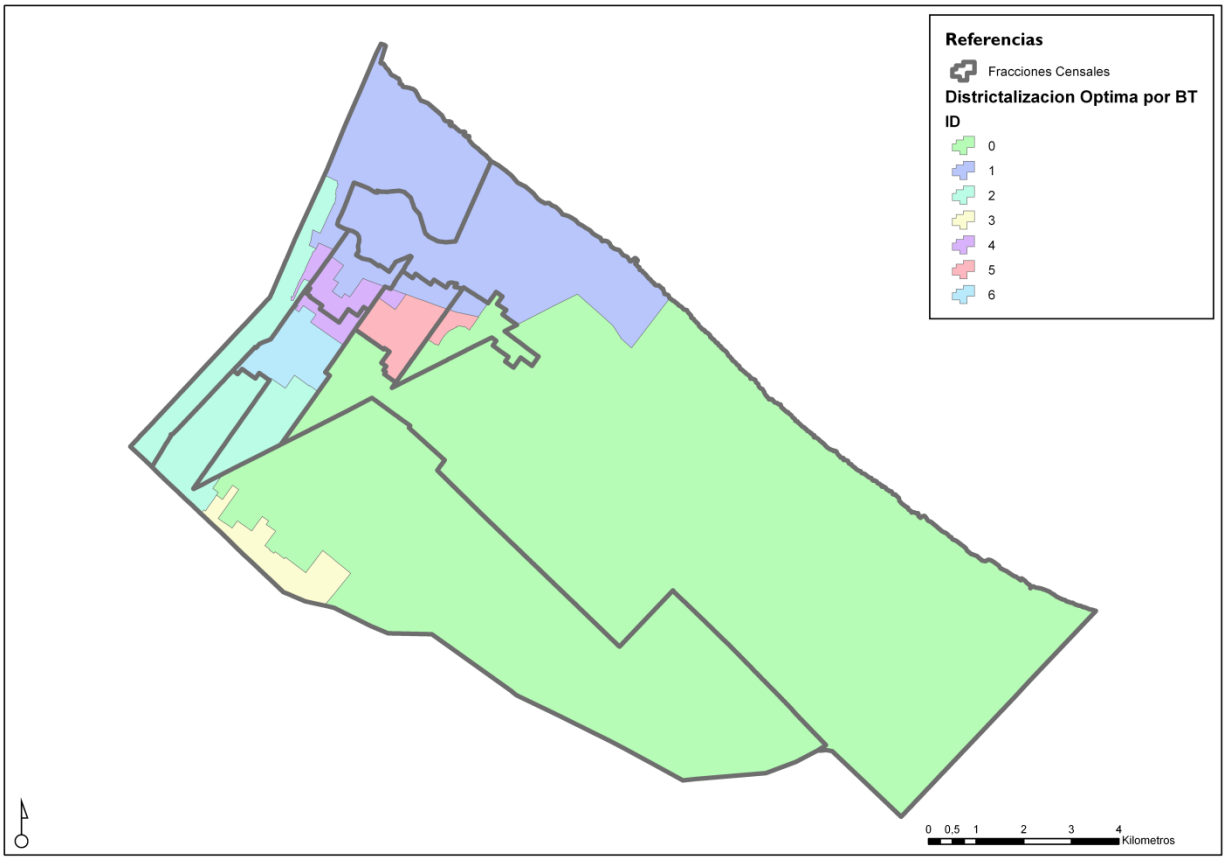


Figura 28 – Distritos óptimos por Búsqueda Tabú y Fracciones Censales para el partido de Berisso

Comentario final sobre la codificación del Algoritmo:

Del testeo computacional realizado, ha surgido la necesidad de realizar ciertas optimizaciones en la codificación empleada, en particular dos son las principales sugerencias que se consideran promisorias para su exploración en la siguiente etapa (implementación en Python):

1- Durante el proceso de búsqueda de soluciones vecinas, es decir soluciones alcanzables a partir de la actual y empleando un movimiento tipo i o tipo ii, la implementación actual busca en cada iteración **todas** las soluciones alcanzables. Ahora bien, una vez realizado un dado movimiento, el espacio de soluciones alcanzable a partir de esa nueva solución no es totalmente distinto de la iteración anterior, es decir, no es necesario calcular nuevamente todo el espacio de solución, sino más bien actualizar la búsqueda anterior para incorporar la última modificación local. Esta modificación tiene un fuerte impacto en la velocidad del proceso, dado que en esencia el mayor costo está en la búsqueda de las soluciones vecinas y su posterior cálculo del valor objetivo.

2- Por otro lado, ciertos investigadores sugieren que no es necesario evaluar **todo** el espacio de soluciones alcanzables, sino que en principio puede ser suficiente realizar un muestreo aleatorio de tamaño adecuado del mismo. Nuevamente, esta posibilidad acelera rápidamente el proceso de evaluación de los valores de la función objetivo sin impactar negativamente en la lógica de la BT. Esta última posibilidad ha sido incorporada en la actual versión en *Avenue* del algoritmo, permitiendo que se defina el tamaño de la muestra aleatoria a emplear utilizando una variable de proporción para controlar esta propiedad (obviamente una proporción unitaria implica la evaluación completa de todas las soluciones alcanzables).

Ejemplo final de comparación utilizando la librería BARD[119][120] y el ambiente estadístico R[121]:

A modo de comparación final, se utilizó la librería BARD y el ambiente estadístico R para calcular redistrictalizaciones para el partido de La Plata, a tal fin se realizaron los siguientes pasos:

a) Importación de los radios censales del partido de La Plata.

lp.map <- importBardShape("ruta al archivo shapefile", "POLY_ID")[122]

b) Construcción de una solución con balanceo poblacional.

rpplan<- createRandomPopPlan(lp.map, 47, predvar = "TP_T")[123]

c) Reparación de la contigüidad de los distritos (de manera de asegurar la contigüidad de cada distrito, algo que la solución de balanceo poblacional no garantiza).

rpplan1 <- fixContiguityPlan(rpplan, score.fun = myScore[124])

d) Luego se utilizó la rutina de *refineTabuPlan(plan, score.fun, displaycount, historysize, dynamicscoring, tracelevel, tabusize, tabusample)*, donde:

- *plan* := plan de districtalización a ser refinado en este caso el resultado del paso c.

- *score.fun* := la función que acepta el plan y devuelve un valor único como puntaje (función objetivo). En nuestro caso empleamos una función con el componente de balance poblacional y compacidad medida en base a los perímetros interiores (esencialmente análoga a nuestra medida de compacidad 1).

- *displaycount* := se utiliza generalmente para demostraciones y actualiza un gráfico con el último *score* obtenido por el algoritmo.

- *tracelevel* := provee trazabilidad al algoritmo, y sus valores posibles son 0 para silencio, 1 para información mínima, 2 para mayor información y 3 para información máxima.

[119] Altman y Mcdonald (2.009).
[120] http://bard.sf.net/[Internet, fecha de último acceso abril 2.011]
[121] http://www.r-project.org/[Internet, fecha de último acceso enero 2.011]
[122] Ejemplo de sentencia en R para definir la importación del archivo geográfico en formato shapefile © ESRI, POLY_ID es el ID de enlace (clave primaria de los polígonos) que se emplea para unirlo con el archivo .gal de contigüidad espacial.
[123] Ejemplo de sentencia en R para crear distritos aleatorios con igualdad de población, en este caso se solicitan 47 distritos y se indica que el campo con los datos de población se denomina "TP_T"
[124] Ejemplo de sentencia en R para reparar la contigüidad de una solución no contigua. La función myScore fue definida en otro contexto, y por ejemplo puede ser una función de actualización dinámica que incluya balance poblacional y compacidad, *e.g.*:
myScore >- function(plan, ...) combineDynamicScores(plan, scorefuns = list(calcPopScore, calcLWCompactScore))

- *dynamicscoring* := hace uso de funciones de *scoring* incrementales, que guardan información sobre los planes candidatos y solamente puntúan los cambios incrementales. Dado que se evalúan múltiples candidatos a la vez, se la sugiere utilizar solamente si la función objetivo es costosa computacionalmente.

- *hystorisize* := Mantiene una historia de puntajes de tamaño *n*, de manera de evitar re-evaluar puntajes para planes ya visitados. Nuevamente se sugiere su utilización cuando el costo de calcular la función objetivo es elevado.

- *tabusize* := El tamaño de la lista tabú

- *tabusample* := El tamaño de la muestra aleatoria a ser calculado en cada iteración.

```
tabu1 <-RefineTabuPlan(rpplan1, score.fun = myScore, displaycount = NULL, historysize = 0, dynamicscoring = TRUE, tracelevel = 1, tabusize = 100, tabusample = 500)
```

La rutina de *refineTabuPlan* se corrió dos veces, una con *historysize* = 0 , la otra con *hystorisize* 50. En ambos casos con un *tabusize* = 100 y *tabusample* = 500. Por otro lado el valor de la función objetivo para el plan a refinar era de **14.86687**. Los resultados de ambas corridas muestran:

a) **Corrida 1**: la rutina obtuvo su regla de parada en la iteración 853. Desde la Iteración 148 el valor de la función objetivo no mostró una mejora apreciable[125]. El tiempo de cómputo fue de 01:53:44 hs., utilizando la versión de 64 bits de R corriendo sobre un procesador AMD Phenom II X6 1090T con 12 Gb de RAM. El valor de la función objetivo alcanzado al momento de finalización del algoritmo fue de **5.20456**.

a) **Corrida 2**: la rutina alcanzó su condición de parada en la iteración 849, y nuevamente desde la iteración 144 la función objetivo no mostró una mejora apreciable[126]. El tiempo de cómputo fue de 14:49:40 y el valor de la función objetivo alcanzado al momento de finalización del algoritmo fue de **5.20206**.

Ahora bien, al analizar las fracciones censales del partido de La Plata con relación a estas dos corridas (que en esencia son muy similares a pesar de lo dispar de su tiempo de

[125]Nótese que el valor de *T* que no puede ser modificado por el usuario, parece ser igual a la cantidad de UB (704 en este caso).
[126]Nótese nuevamente el valor de *T* idéntico al caso anterior.

cálculo), encontramos su valor de la función objetivo es sensiblemente inferior a estos valores[127].

Es interesante notar el tiempo de cálculo que le demanda a un software diseñado específicamente para trabajar con grandes volúmenes de datos (y optimizado a tal fin) como es el caso de R, al igual que la relativa baja calidad de la solución final obtenida. Aun así es importante observar que la propia rutina tabú de BARD mejora con creces el valor de la función objetivo del plan original (un decremento del 65 % en su valor). Asimismo, es interesante notar que la propia implementación de BARD restringe el valor de *T* a la cantidad de UBs y utiliza muestreo aleatorio para restringir el espacio de soluciones vecinas a visitar.

[127]Dado que no es sencillo incorporar las fracciones censales para su cálculo de la función objetivo equivalente a las de estas corridas, se procedió a calcular para cada una de estas districtalizaciones el valor de la función objetivo calculada por nuestro algoritmo:
a- F(fracciones censales) = 9.754
b- F(plan no refinado) = 20.289
c- F(tabu1) = 18.558
d- F(tabu2) = 17.639

7- Conclusiones

Se ha explorado el problema de districtalización automática empleando un algoritmo de Búsqueda Tabú (BT), y se ha probado su implementación exitosa utilizando el lenguaje *Avenue* del Sistema de Información Geográfica (SIG) ArcView GIS 3.x de ESRI ©. Asimismo, se ha explorado su posterior implementación en lenguaje *Python* en el contexto del SIG *ArcGIS v.10* de ESRI ©.

Debido a las particularidades del problema en estudio, se analizó la complejidad computacional del mismo como justificación de la necesidad de utilizar métodos heurísticos para su resolución y se probó su aplicabilidad con datos reales de nuestro entorno local.

Por otro lado, se han testeado los parámetros específicos del algoritmo BT teniendo en cuenta las características de las Unidades Base con las que se trabaja en el área metropolitana de Buenos Aires obteniéndose resultados compatibles con los de Bozkaya *et al.* (2.003) en relación a los factores de escala y con la misma metodología de análisis de sensibilidad.

El algoritmo descripto en este trabajo, puede resumirse sintéticamente como:

Paso 1 (inicialización): Construir una solución inicial x_0 (por ejemplo empleando el algoritmo de construcción de Vickrey (1.961). Definir la mejor solución conocida como la solución inicial y definir su valor de función objetivo. Definir el contador de iteración inicial en 0.

Paso 2 (búsqueda de soluciones vecinas, primer pasada): Repetir lo que sigue hasta se cumpla la regla de parada:

- Incrementar el contador de iteraciones,
- Actualizar la penalidad para la violación del balance poblacional,
- Identificar todos los vecinos de la solución actual con relación a movimientos Tipo I[128]. Evaluar cada función objetivo candidata; penalizar el valor objetivo si el candidato es una solución no mejor. Ordenar todos los candidatos en orden no decreciente de su valor objetivo y elegir el primer movimiento no tabú, o el primer movimiento tabú que mejora el mejor valor objetivo.
- Ejecutar el movimiento y actualizar la lista tabú.
- Actualizar la mejor solución conocida, mejor valor de la función objetivo y los demás parámetros.

Paso 3 (búsqueda de soluciones vecinas, segunda pasada): Este paso es idéntico al paso 2, pero ahora incluye también los movimientos Tipo II[129]. El algoritmo para cuando no

[128] Es decir, soluciones a las que se alcance mediante la transferencia de una unidad base de un distrito a otro distrito vecino **SIN** crear una solución no contigua.
[129] Es decir, soluciones vecinas a las que se alcance mediante el intercambio de dos unidades básicas en dos distritos vecinos.

hay mejora en la función objetivo de la mejor solución factible y la mejor solución (factible o no) puede ser obtenida una determinada cantidad de veces (iteraciones).

El algoritmo puede mejorarse embebiéndolo dentro de un proceso de memoria adaptativa como el propuesto por Rochat y Taillard (1.995). Este meta procedimiento construye una solución inicial a partir de un pool de soluciones tomando componentes de alta calidad (distritos de alta calidad) de ese pool y completándolos luego de manera de obtener una solución factible. El algoritmo tabú se ejecuta entonces sobre esta solución inicial, y los distritos de la solución mejorada se emplean para actualizar el pool. El proceso de memoria adaptativa para luego de una cantidad fija de meta iteraciones.

A partir de los hallazgos de Bozkaya (1.999) y Bozkaya *et al.* (2.003 y 2.010), no hemos realizado en este trabajo una implementación efectiva del módulo de *Diversificación e Intensificación Probabilística (DIP)*, correspondiente al proceso de memoria adaptativa mencionado precedentemente, específicamente por su costo computacional y la disponibilidad de recursos informáticos para las etapas de testeo e implementación de este trabajo. Aun así, la mejora reportada en los estudios antedichos oscila entre un 9 y un 27 %, y esperamos que al momento de implementar el algoritmo definitivo en *Python* para el Departamento de Planeamiento y Mapeo Criminal, se incluya el componente de memoria adaptativa.

Este trabajo ha probado que la metodología expuesta permite definir con claridad un conjunto de criterios para un proceso de districtalización automatizado, y en particular que puede adaptarse rápidamente a las necesidades de districtalización policial de una agencia de protección de la ley como la Policía de la Provincia de Buenos Aires, que tradicionalmente emplea los radios censales para construir sus comisarías y sectores. Es decir este trabajo tiene el potencial de convertirse en una plantilla para la guía y definición de los criterios de districtalización a utilizar por la agencia policial al igual que el proceso para la generación de planes de districtalización de *buena calidad* y definir la misma en base a valores cuantitativos de la función objetivo.

Asimismo, se ha probado que es posible implementar mecanismos de districtalización **multicriterio**, haciendo que el proceso pueda realizarse aún con recursos computacionales escasos (notemos que la implementación fue realizada específicamente utilizando lenguaje *Avenue* por esa restricción), y que es fácilmente traducible a implementaciones de mayor performance (*e.g.* Python + *ArcPy* o bien *Python* + *Pysal* en nuestro trabajo).

Por otro lado, la aproximación algorítmica presentada, permite considerar un marco de trabajo estructurado para la resolución de problemas de districtalización y permite caracterizar rápidamente la calidad de la solución presentada en base a los puntajes obtenidos en la función objetivo definida con antelación.

Una de las principales fortalezas de la Búsqueda Tabú es su adecuación a problemas en los que las restricciones son difíciles de tratar matemáticamente (como el caso de la contigüidad), y su elección no ha sido caprichosa, sino que puede rastrearse inclusive a estudios en las que se compara su performance en la resolución de problemas de optimización combinatoria con otras técnicas[130].

Finalmente es importante resaltar que la metodología expuesta en este trabajo es aplicable a esencialmente cualquier problema de districtalización en la medida que se puedan operacionalizar adecuadamente los diferentes criterios y componerlos en una función a ser evaluada. La mecánica del algoritmo es siempre la misma (inicia con una partición que posteriormente es modificada analizando sus soluciones vecinas alcanzables mediante determinados tipos de movimientos y sujeta a un mecanismo diseño para evitar ciclos) y en ello reside su fortaleza y adaptabilidad.

[130] Ver por ejemplo: Gendreau, Hertz y Laporte(2.005).

ANEXO I

Demostraciones de intratabilidad de la Districtalización (Micah Altman)[131]

La forma más común para mostrar que un problema es NP-completo es mostrar que puede ser polinomialmente reducido a otro problema que ya se sabe es NP-completo. De la misma forma, la manera más común de demostrar que un problema es NP-difícil es mostrar que su resolución requiere resolver un problema que es NP-completo.

Los problemas de districtalización son NP-difíciles.

- Usaremos $\mathbf{x}_i$ para referirnos a la i-ésima UB. Estas UB son **vectoriales** y presentan valores de población, geografía, etc.
- Usaremos $\mathbf{d}_i$ para referirnos al i-ésimo distrito. Un distrito es un conjunto de UBs: $\mathbf{d}_i = \{\mathbf{x}_j, \mathbf{x}_k, \ldots, \mathbf{x}_n\}$
- Usaremos $\mathbf{p}_i$ para referirnos a un plan en particular. Un plan es una partición del conjunto de todas las UB en un conjunto de distritos $\mathbf{p}_i = \{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$

1 – Crear Distritos con igualdad de población es NP-difícil

Asumiendo que la población en cada UB es fija y positiva, necesitamos primero definir una medida de igualdad de población. Podemos calcular la población en un distrito sumando sobre todos sus UBs:

$$pop(\boldsymbol{d_i}) = \sum_{x_i \in \boldsymbol{d_i}} pop(\boldsymbol{x_i})$$

Usaremos una medida simple de la desigualdad de un plan, $V(\mathbf{p}_i)$, como la diferencia entre la población de su distrito más grande y su distrito más pequeño[132]:

$$V(\boldsymbol{p_i}) = \sup pop_{\boldsymbol{d_i} \in \boldsymbol{p_i}}(\boldsymbol{d_i}) - \inf pop_{\boldsymbol{d_i} \in \boldsymbol{p_i}}(\boldsymbol{d_i})$$

Demostración: Dado un conjunto particular de UB, etiquetemos el conjunto de todos los planes posibles **P**. Para determinar distritos tan cerca como sea posible

[131] Altman (1.997).

[132] El problema de districtalización permanece NP-difícil en tanto y en cuanto $V(\mathbf{p}_i) = 0$ si y sólo si tenemos absoluta igualdad de población.

del objetivo, debemos encontrar un plan **p*** que minimice V sobre todos los planes posibles usando esas UB, y con un número fijo de distritos *k*:

$$\boldsymbol{p}^* = \underset{\{\boldsymbol{p}_i \in \boldsymbol{P_1} || \boldsymbol{p}_i| = k\}}{\arg \inf} V(\boldsymbol{p}_i)$$

Supongamos que producimos un algoritmo que encuentra **p*** para cualquier conjunto de UB. Dado **p*** es trivial decidir sobre la pregunta "¿Existe un plan tal que **V**(**p***) = 0?". Ahora bien, si podemos responder esta pregunta, podemos resolver un problema NP-completo. Para k = 2, responder esta pregunta es equivalente a resolver el siguiente problema NP-completo, simplemente re-etiquetemos nuestras UB *a*, re-etiquetemos nuestros dos planes *A*' y (*A* – *A'*), y re-etiquetemos nuestra medida de población como *s:*

Partición de conjuntos

Instancia: Conjunto Finito *A* y de tamaño $s(a) \in Z^+$ para cada $a \in A$.

Problema: ¿Existe un subconjunto $A' \subseteq A$ tal que $\Sigma_{a \in A'} s(a) = \Sigma_{a \in A - A'} s(a)$?

Este problema es NP-completo, pero no fuertemente NO-completo. Si necesitamos crear un plan con más distritos, entonces el problema es fuertemente NP-completo. Si podemos decidir si V(**p***) = 0 para $k \geq 2$, entonces podemos responder el siguiente problema NP-completo, al realizar un re-etiquetado similar:

Partición en 3

Instancia[133]*:* Sea A de *3m* elementos, una cota $B \in Z^+$ y un tamaño $s(a) \in Z^+$ para cada $a \in A$ tal que $B/4 < s(a) < b/2$ y tal que $\Sigma_{a \in A} s(a) = mB$.

Pregunta: ¿Puede A ser particionado en *m* subconjuntos disjuntos tales que para $1 \leq i \leq m$, $\Sigma_{a \in A} s(a) = B$.

[133]Dado que un problema de Partición en 3 asume más restricciones para sus elementos que las que pusimos sobre las UB, esto muestra que nuestro problema es NP-difícil, pero no es suficiente para mostrar que es NP-completo.

2 – Crear un plan de districtalización de compacidad máxima es NP-difícil

Asumiendo que medimos la compacidad de un distrito como la distancia máxima entre los centros de cualquier par de UB en el distrito[134], y que medimos la compacidad de un plan de districtalización a través de su distrito menos compacto:

$$V(\boldsymbol{d}) = \sup\left(distancia\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right)(\mathbf{x}_i, \mathbf{x}_j \in \mathbf{d})$$

$$V(\boldsymbol{p}) = \sup\left(V\left(\boldsymbol{d}_i, \boldsymbol{d}_j\right)\right)(\mathbf{d}_i, \mathbf{d}_j \in \mathbf{p})$$

Demostración: Nuevamente supongamos que tenemos un algoritmo que encuentra **p*** que resuelve el problema para cualquier conjunto de UB. Dado **p*** es entonces trivial responder la pregunta (para una dada constante D): "¿Existe un plan tal que V(**p***) < D?"
Pero si podemos responder esa pregunta podemos resolver un problema NP-completo, ya que es equivalente a resolver el siguiente problema que es (fuertemente) NP-completo para cualquier k > 2:

Partición por distancia-d de puntos en un Plano
Instancia: Conjunto finito $\mathbf{P} \subset Z \times Z$ de puntos con coordenadas enteras en el plano, enteros positivos D y k.
Problema: ¿Existe una partición $\mathbf{P} = \mathbf{P}_1 \cup \ldots \cup \mathbf{P}_k$ tal que si *u* y *v* son dos puntos distintos en algún conjunto P_i de la partición, la distancia Euclídea d(u,v) < D?

[134] De esta manera un gran valor de V indica que el distrito es poco compacto.

3 – Crear un plan de districtalización contiguo con igualdad de población es NP-difícil

Usaremos la misma medida de población que en la prueba 1. Asimismo, si dos UBs i,j, son contiguas, las conectaremos mediante un único borde $e_{i,j}$. Nuestro problema es en este caso, encontrar una partición en k distritos, tal que los bordes completamente contenidos en el distrito formen un grafo conectado y de manera tal que se minimice V(**p**) como en el problema 1.

Demostración: Nuevamente supongamos que tenemos un algoritmo que encuentra **p*** que resuelve nuestro problema para cualquier conjunto de UB. Dado **p*** es ahora trivial responder a la pregunta (para una dada constante D): ¿Existe un plan tal que todos los distritos son contiguos y V(**p***) < D? Si podemos responder esto, podemos resolver un problema NP-completo, ya que responder esa pregunta es equivalente aq resolver el siguiente problema que es fuertemente NP-completo:

Cortar en componentes conexos de peso acotado

Instancia: Grafo **G** = (**V,**E), tamaño s(v) $\in Z^+$ para cada a.

Problema: ¿Existe una partición de **V** en conjuntos disjuntos $\mathbf{V}_1$y $\mathbf{V}_2$ tal que $\sum_{v \in \mathbf{V}1} s(v) \leq K$ y $\sum_{v \in \mathbf{V}2} s(v) \leq K$ y tanto $\mathbf{V}_1$como $\mathbf{V}_2$ inducen subgrafos conectados de **G**?

ANEXO II

Búsqueda Tabú en su formulación original (Fred Glover)[135]

La búsqueda tabú (BT) es una estrategia para resolver problemas de optimización combinatoria. Se trata de un procedimiento adaptativo con la habilidad de utilizar otros métodos, tales como algoritmos de programación lineal o heurísticas especializadas, que dirige a sobreponer las limitaciones de los óptimos locales. En lo que sigue expondremos las principales consideraciones de Fred Glover en sus dos *papers* sobre el tema:

1- **Notación**

En este caso representamos el problema de optimización combinatoria como

(P) Minimizar $c(x)$: $x \in X$ en R_n

La función objetivo c(x) puede ser lineal o no lineal y la condición $x \in X$ se asume que restringe componentes específicos de x a valores discretos. En algunos casos (P) puede representar una forma modificada de algún problema original, mientras que X es el super-conjunto de vectores que normalmente califican como factibles, mientras que c(x) es una función de penalización diseñada para asegurar que soluciones óptimas para (P) lo sean también para el problema del que (P) deriva.

Una gran cantidad de procedimientos, heurísticos y óptimos, para resolver varios problemas susceptibles de ser escritos en la forma de (P) pueden ser caracterizados convenientemente mediante referencias a secuencias de *movimientos* desde una *solución de prueba* (elegida $x \in X$) a otra. Glover define un movimiento *s* como un mapeo definido en un subconjunto X(s) de X:

$s: X(s) \rightarrow X$

El conjunto S(x) está asociado con $x \in X$, y consiste de todos aquellos movimientos $s \in S$ que pueden ser aplicados a x. Es decir:

$S(x) = \{s \in S: x \in X(s)\}$, por lo que también podemos escribir

$X(s) = \{x \in X: s \in S(x)\}$

[135] Glover, 1.989.

El conjunto S(x) puede ser visto como una "función de vecindad"

2- Una forma sencilla de Búsqueda Tabú

Hay dos características claves en la BT:

i- Restringe la búsqueda al clasificar ciertos movimientos como prohibidos (tabú).

ii- Libera la búsqueda mediante una función de memoria a corto plazo que provee "olvidos estratégicos".

Glover comienza por referenciar a la clase de aproximaciones heurísticas denominadas *hill climbing* (trepar la colina) que toman el progreso en forma unidireccional a partir de un punto de partida y hacia un óptimo local (en el contexto de minimización esto se invierte y en realidad se "baja" de la montaña).

HEURISTICA de SUBIDA para (P) (*HILL CLIMBING*)

1. Elegir una $x \in X$ inicial.
2. Elegir algún $s \in S(x)$ tal que
$$c(s(x)) < c(x)$$

Si no existe tal s, x es un óptimo local y el método se detiene. De otro modo,

3. Hacer x := s(x) y volver al paso 2.

A pesar que conceptualmente es simple, esta heurística puede tener útiles y a veces sutiles características. Si X es el conjunto de puntos extremos factibles para un programa lineal y S(x) es el conjunto de movimientos que llevan de x a un punto extremo adyacente, entonces el método simplex es una de estas heurísticas.

La principal limitación de este procedimiento en un problema combinatorio es que el óptimo local obtenido en su condición de parada cuando no hay movimientos que mejoren la situación, puede no ser el óptimo global. La BT guía a esta heurística a continuar la exploración sin confundirse por la ausencia de mejores movimientos y sin caer en óptimos locales de los que ya ha emergido. Dada su habilidad para incorporar y guiar otro procedimiento, la BT puede ser vista como una meta-estrategia para la resolución de problemas combinatorios.

En forma simplificada, el procedimiento puede ser descripto de la siguiente manera, se crea un subconjunto *T* de *S* cuyos elementos se denominan *movimientos tabú*. Los elementos

de *T* se determinan por una función no Markoviana que utiliza información histórica del proceso de búsqueda, extendiéndose hasta *t* iteraciones en el pasado, donde *t* puede ser fija o variable dependiendo de la aplicación o de la etapa en la búsqueda. La membrecía en *T* se realiza mediante una lista itemizada o mediante referencias a un conjunto de *condiciones tabú* expresadas indirectamente en términos de una solución de prueba actual *x*; por ejemplo haciendo que *T* tome la forma *T(x)* = $\{s \in S$: *s(x)* viola las condiciones tabú$\}$.

Para una adecuadamente determinada *T* y una función de evaluación denominada ÓPTIMO, el procedimiento puede describirse como:

BUSQUEDA TABU SENCILLA

1. Elegir una $x \in X$ inicial, y hacer $x^* := x$. Poner el contador de iteraciones en cero, k = 0 y comenzar con T vacío.
2. Si S(x) – T es vacío, ir al paso 4. De otro modo, hacer k := k + 1 y elegir $s_k \in$ S(x) – T de manera tal que $s_k(x)$ = OPTIMO(s(x): $s \in$ S(x) – T)
3. Hacer x := $s_k(x)$. Si c(x) < c(x*), donde x* indica la mejor solución actualmente encontrada, hacer x* := x.
4. Si ha pasado un número dado de iteraciones en total o desde que x* fue mejorada, o si S(x) – T = $\varnothing$ al llegar a este paso directamente del paso 2, PARAR. De otro modo, actualizar T y volver al paso 2.

Hay tres aspectos de esta versión que merecen resaltarse:

i- El uso de T provee el elemento de "búsqueda restringida" y por lo tanto la solución generada depende críticamente de la composición de T y de la forma en que se actualiza en el paso 4;

ii- El método no hace ninguna referencia a la condición del óptimo local, excepto implícitamente donde un óptimo local mejora la mejor solución previamente encontrada;

iii- El *mejo*r movimiento (en lugar de un movimiento que mejore) se elige en cada paso, empleando el criterio embebido en la función OPTIMO.

Consideremos ahora algunas formas de la función ÓPTIMO y del conjunto tabú T. Una elección natural para la función ÓPTIMO está dada por la elección de $s_k(x)$ de manera tal que

$c(s_k(x)) = \min (c(s(x)): s \in S(x) - T)$

En casos en los que la exclusión de T puede ser expresada como un requerimiento de satisfacer un conjunto de restricciones dadas por desigualdades (tales como cotas sobre las variables), y el conjunto S(x) puede ser caracterizado de forma similar, la solución s(x) obtenida de definir OPTIMO de esta forma puede representar el resultado de resolver un problema de optimización auxiliar.

Con la forma dada para OPTIMO, cada ejecución del Paso 2, se mueve del x actual a un s(x) que genera la mayor mejora – o, si no hay posibilidad de mejora, el menor empeoramiento – en la función objetivo c(x) sujeta a la restricción que solamente se permiten movimientos no tabú. En aquellos casos en los que el conjunto S(x) – T puede ser muy grande, y procesado por itemización en lugar de por una solución auxiliar, es apropiado que OPTIMO se base en una estrategia de mapeo de la región, encogiendo S(x) – T para el propósito de identificar el mínimo c(s(x)). Elegir el primer s tal que c(s(x)) < c(x), si es que existe, provee una instancia de una estrategia de muestreo de este tipo, pero la BT generalmente procede de forma más agresiva. Esto contrasta con métodos que progresan lentamente a un óptimo local como los métodos de *simulated annealing*, que se basan en la idea de que un descenso lento, adecuadamente regulado, hará del óptimo local más cercano al global.

La justificación sobre la orientación agresiva de la BT se deriva de dos consideraciones: Primero que muchos problemas de optimización pueden ser resuelto óptimamente tomando el mejor movimiento disponible en cada paso. Segundo, en lugar de pasar proporcionalmente mayor cantidad de tiempo en regiones cuyas soluciones son menos atractivas, la BT dedica la mayor parte de su esfuerzo a explorar regiones con buenas soluciones.

La sencilla regla que selecciona el mínimo c(s(x)) sujeto a restricciones tabú ha sido probada exitosamente en una gran variedad de aplicaciones. Cuando el mínimo es costoso de calcular es posible elegir una aproximación al mismo como una alternativa para reducir la región de muestreo.

Otra forma tan sencilla pero efectiva para determinar el conjunto T está dada por:

$T = \{s^{-1}: s = s_h \text{ para } h > k - t \}$

donde k es el índice de la iteración y s^{-1} es la inversa del movimiento s; es decir $s^{-1}(s(x)) = x$. En otras palabras, T es el conjunto de movimientos que “revierten” o “deshacen” uno de los movimientos hechos en las más recientes t iteraciones del proceso de búsqueda. Por lo tanto, el proceso de actualizar este tipo de T en el Paso 3 de la BT, consiste en hacer T : $T - s^{-1}_{k-t} + s_k^{-1}$ (el signo menos y el signo más indican las operaciones de agrega y remover elementos al conjunto). Por convención cuando $k \leq t$, la referencia a s^{-1}_{k-t} se desecha.

Esta forma para el conjunto tabú se basa en la presunción que la probabilidad de entrar en un ciclo, es decir una secuencia de movimientos que llevan nuevamente a una solución ya visitada en el pasado, está inversamente relacionada con la distancia de la

solución de prueba actual x en relación con la solución previa. Si la distancia se mide en término del número de movimientos que toma (número de iteraciones hechas) desde que la solución previa fue visitada, sujeto a la estipulación que ninguno de sus movimientos intervinientes puede revertir uno de sus predecesores, entonces T está diseñado para contrarrestar el ciclo de acuerdo a la presunción estipulada.

Ahora bien, dentro de la estructura de ciertas reglas de elección y condiciones tabú, el objetivo es en forma más general, evitar retornar a *estados de solución* previos (por ejemplo, a una solución previamente visitada, donde el mejor movimiento disponible no tabú para alejarse de esta solución es el mismo que antes. En este caso, T toma en realidad la forma $T = T_1 \cup T_2$ donde $T_1 = \{s^{-1}_h : h > k - t_1\}$ y $T_2 = \{s_h : h > k - t_2\}$).

Por lo tanto, al prevenir la elección de un movimiento que representa la reversa de alguno realizado durante una secuencia de t iteraciones, el procedimiento se mueve progresivamente lejos de todos los estados de solución de las previas t iteraciones (en cierto sentido determinado por la naturaleza de los movimientos en S)., y para t adecuadamente grande, la probabilidad de regresar efectivamente desaparece. Pero no se deduce de aquí que el objetivo de la BT es elegir t grande. Desde una lógica de competencia, cuanto más pequeño sea el valor de t, mayor es la amplitud de elección que el método tiene para acercarse a lo que la función ÓPTIMO considera preferible. Un descubrimiento empírico realizado en base a la aplicación de la BT es que t tiene un rango relativamente estable de valores que a la vez evitan los ciclos mientras que permiten acercarse a soluciones especialmente buenas.

En la práctica, T rara vez toma la forma dada precedentemente por dos razones:

i- En ciertas situaciones, cuando el movimiento s se elige de S(x), las condiciones tabú que previenen que s^{-1} sea elegido también previenen una gran cantidad de movimientos dominados por s^{-1}.

ii- Debido a cuestiones de conservación de memoria y facilidad de procesamiento, es muchas veces aconsejable guardar menos que la totalidad de atributos requeridos para caracterizar un movimiento, o la solución a la que se aplica, y por lo tanto se almacena un conjunto parcial de atributos (que potencialmente pueden ser compartidos por otros movimientos o soluciones). Bajo estas circunstancias, una lista tabú no consiste solamente de movimientos s^{-1}_h para $h > k - t$, sino una colección C_h de movimientos, donde cada C_h define sus miembros en base a ciertos atributos, en base a sus condiciones tabú, y esto hace que contenga no solo a s^{-1}_h sino también a otros movimientos que satisfacen esas mismas condiciones. Por lo tanto, y en forma más general, T toma la forma de:

$$T = \cup C_h : h > k - t \quad (\text{donde } s^{-1}_h \in C_h)$$

Asimismo, es posible que el conjunto C_h contenga otros elementos además de s^{-1}_h para regiones estratégicas relacionadas con la prevención de ciclos.

Otro aspecto importante en relación con la administración de T es el siguiente: en los casos donde S(x) – T es vacío en el Paso 2 del procedimiento de BT sencillo, la actualización de T en el Paso 4 se aleja de la actualización de T implícita en la discusión precedente (que de hecho descarta la restricción tabú registrada t iteraciones atrás) al crear una priorización equivalente a basar la elección de descartar el menor número de elementos de T, en secuencia, del más antiguo al más nuevo, de manera tal de permitir que algunos movimientos obtengan el estado no tabú. Se puede demostrar que esta priorización lleva a un óptimo en un número finito de pasos en situaciones sencillas, si se permite que t crezca con k (numero de iteraciones) a pesar que este esquema de priorización potencialmente permite movimientos superfluos que pueden ser evitados mediante refinamientos. Ahora bien, en la práctica, nunca han sido necesarios estos refinamientos en las prioridades ni tampoco un crecimiento en el valor de t.

Ejemplo:

Crear una partición óptima de un conjunto de elementos E en conjuntos E_i, $i \in N = \{1, \ldots, n\}$. Cada elemento e en E tiene una ponderación w(e), y cada conjunto E_i tiene un peso a obtener W_i. Definimos $w(E_i)$ como el peso de los elementos de E asignados a E_i, es decir, $w(E_i) = \Sigma(w(e): e \in E_i)$, donde $w(E_i) = 0$ si E es vacío. Con estas consideraciones, el objetivo es minimizar la suma de desviaciones absolutas de los pesos $w(E_i)$ en relación con los pesos pretendidos W_i:

Minimizar $\Sigma (|w(E_i) - W_i|: i \in N)$

Un razonable conjunto de movimientos para incluir dentro de la BT parte de la idea de iniciar con alguna asignación arbitraria de elementos en los conjuntos y utilizar intercambios para intercambiar un elemento e_i actualmente en el conjunto E_i con un elemento e_j actualmente en el conjunto E_j. Se permiten también el caso especial en el que ya sea e_i o e_j es un elemento nulo de peso cero, lo que permite un intercambio parcial, es decir la transferencia de un elemento de un conjunto hacia otro.

La función ÓPTIMO puede tomar la forma de prescribir el intercambio que lleva al cambio más favorable en el objetivo de minimización, restringiendo la atención a movimientos que no califiquen como tabú. En este mismo nivel de simpleza, el conjunto tabú tiene el rol de prevenir los movimientos de reversión hechos durante las más recientes t iteraciones.

Para conseguir la prevención de movimientos de reversión e implícitamente definir T, utilizamos dos arreglos LISTATABU y ESTADOTABU. LISTATABU(p), p = 1, …, t registra los atributos seleccionados que describen los movimientos asociados, si bien existen diferentes niveles de detalle posibles para describir estos movimientos, en este caso podemos restringirnos meramente a dos pares ordenados (i, $w(e_i)$) y (j, $w(e_j)$), que es suficiente para identificar el hecho que el intercambio asociado consistía en la transferencia de un elemento de peso $w(e_i)$ del conjunto E_i al conjunto E_j y un elemento de peso $w(e_j)$ del conjunto E_j al E_i.

Es interesante notar que los elementos exactos e_i y e_j no se identifican como parte de estos atributos, solamente su peso. Este nivel reducido de especificidad puede tener valor estratégico. En particular, prevenir el intercambio de pesos es más apropiado en este contexto que prevenir el intercambio de elementos. De hecho, una implementación más cuidadosa de la BT también requerirá en forma adicional, evitar movimientos tales que $w(e_i) = w(e_j)$. Estas observaciones forman parte de principios generales que involucran movimientos **equivalentes** y **deficientes**.

Los atributos (i, $w(e_i)$) y (j, $w(e_j)$) se utilizan en este ejemplo para codificar las restricción tabú que impide un movimiento si agrega ya sea un elemento de peso $w(e_i)$ al conjunto E_i o un elemento $w(e_j)$ al conjunto E_j. Es importante observar que el mismo conjunto de atributos puede ser utilizado para codificar otras restricciones tabú. Por ejemplo (i, $w(e_i)$) y (j, $w(e_j)$) puede ser utilizado para prohibir movimientos que quitan un elemento de peso $w(e_j)$ del conjunto E_i o un elemento de peso $w(e_i)$ del conjunto E_j. Asimismo puede ser usados para prevenir movimientos que a la vez agreguen o quiten los elementos indicados para crear un movimiento de reversa. Esta última condición tabú es menos restrictiva (es decir prohíbe menos movimientos) que las mencionadas anteriormente.

El arreglo ESTADOTABU tiene el rol de hacer que las restricciones tabú sean más fáciles de implementar. En nuestro ejemplo, supongamos que los elementos a ser particionados tienen r pesos distintos, w_q, q = 1, …, r. Entonces hacemos que ESTADOTABU tome la forma de una matriz de n × r ESTADOTABU(i,q), donde ESTADOTABU(i,q) tiene un valor positivo si un elemento de peso w_q no puede ser agregado al conjunto E_i, y ESTADOTABU(i,q) tiene un valor cero en cualquier otro caso. Un valor positivo de ESTADOTABU(i,q) nombra la posición más reciente p en LISTATABU donde el atributo (i,q) está registrado (con el propósito de prohibir que un elemento de peso w_q se agregue al conjunto E_i).

La implementación completa de la BT en este ejemplo es la siguiente. LISTATABU(p), que en nuestro ejemplo almacena dos pares ordenados para cada valor de p = 1, …, t se trata como una lista circular. Para hacer esto, el valor p se actualiza por p := p + 1 cuando el contador de iteración se actualiza por k := k + 1, excepto que p se resetea a 1 cuando su valor debiera incrementarse de t a t + 1. Los atributos registrados en la posición p de LISTATABU automáticamente borran los atributos almacenados ahí previamente. Cuando este borrado ocurre, el arreglo ESTADOTABU (que comienza con todos sus valores en 0) se modifica de

manera tal que las entradas (i, q) donde ESTADOTABU(i, q) = p se resetean a ESTADOTABU(i,q) = 0.

Los pares ordenados (i,q) que identifican aquellas entradas de ESTADOTABU que están marcadas para ser reseteadas a 0, son precisamente los dos pares ordenados registrados como atributos en LISTATABU(p). Hay una única excepción: es posible que luego de ajustar ESTADOTABU(i,q) = p (cuando(i,q) se registra en LISTATABU(p)), una iteración posterior puede involucrar quitar un elemento de peso w_q del conjunto E_i, y por lo tanto reseteando ESTADOTABU(i,q) a un nuevo valor. Por lo tanto es suficiente controlar si el par (i,q) almacenado en LISTATABU(p) lleva a ESTADOTABU(i,q) = p y si no, la operación de asignar ESTADOTABU(i,q) a 0 se evita.

Mediante estar reglas de actualización, en cada punto donde se deba evaluar el intercambio de los elementos e_i y e_j como candidatos para el mejor movimiento durante la actual iteración, los elementos del arreglo ESTADOTABU(i, $w(e_i)$) y ESTADOTABU(j, $w(e_j)$) indican al momento si el movimiento es tabú (conforme si alguno de esos dos elementos es positivo). El control del arreglo ESTADOTABU para ver si un movimiento es tabú se pospone hasta luego de que el primer mínimo local se consigue (aunque este tipo de salvedad sobre el estado tabú se consigue automáticamente utilizando criterios de **aspiración**).

Queda por responder la actualización que ocurren cuando todos los movimientos disponibles en la iteración actual son tabú (es decir, cuando el Paso 4 del procedimiento de BT simple se alcanza inmediatamente luego de descubrir que S(x) – T es vacío en el Paso 2). La priorización de movimientos tabú mencionados, puede conseguirse mediante referencia a valores de sus entradas en el arreglo ESTADOTABU, donde el movimiento prescripto se convierte el que produce el mejor cambio en la función objetivo sujeto a encontrarse en el conjunto de mayor prioridad. Cada entrada no nula de ESTADOTABU implícitamente identifica un valor asociado de iteración k, y por lo tanto al permitir que entradas asociadas con valores de k más pequeños tengan mayor prioridad, el efecto puede ser obtenido de elegir el mejor movimiento sujeto a descartar solamente las más antiguas restricciones tabú.

3- **Usos de niveles de aspiración**

Un elemento importante de la BT es la incorporación de una *función de nivel de aspiración A(s,x)*, cuyos valores dependen de un movimiento específico *s* y/o del vector *x*. Se dice que el nivel de aspiración se *consigue* si:

$$c(s(x)) < A(s,x)$$

El rol del *A(s,x)* es proveer mayor flexibilidad a la elección de buenos movimientos al permitir que el estado tabú de un movimiento sea anulado si se obtiene el nivel de aspiración pretendido. El objetivo es hacer esto al mismo tiempo que mantener la habilidad de evitar los ciclos.

Para discutir esto, Glover[136] utiliza en la discusión una definición restringida de la palabra movimiento, como una instancia particular de un mapeo, es decir, habla de un movimiento de "*x* a *s(x)*". Este tipo de movimiento, cuya identidad depende tanto de *x* como de *s*, lo denomina *movimiento específico a la solución*.

Hay tres niveles estratégicos para evitar los ciclos que involucran prevenir los movimientos específicos a la solución de x a s(x):

1) Si s(x) ya ha sido visitado con anterioridad;
2) Si el movimiento s ya ha sido aplicado a x con anterioridad;
3) Si el movimiento s^{-1} ha sido aplicado a s(x) con anterioridad.

A pesar que (1) es el único criterio para prevenir un movimiento específico a la solución que asegure completamente que no se producirá un ciclo, el proceso de chequear si el estado tabú de un movimiento puede ser anulado sobre la base de (1) generalmente requiere más memoria y mayor esfuerzo que el que es conveniente utilizar. Si un movimiento tabú se permite provisto que sea que falle solamente la condición (2), entonces es posible revertir un movimiento tan pronto como se realiza, volviendo a una solución recién visitada. Varios experimentos han demostrado que las listas tabú diseñadas de esta manera para prevenir la repetición en lugar de la reversión de movimientos, por lo general no funcionan adecuadamente (por razones que no son difíciles de imaginar). Ahora bien, usando el control (3) para evitar los ciclos y permitiendo realizar un movimiento tabú que lleve de x a s(x) a menos que haya ocurrido con anterioridad un movimiento específico a la solución de s(x) a x, se mejora más efectivamente el intento de prevenir el regreso a una solución ya generada que si se utilizara la condición (2). Por otro lado, si se agrega la condición (2) a la (3) se fortalece la aproximación de conseguir las condiciones de la condición (1).

Tomemos nuevamente un ejemplo. Definamos A(s,x) = el mejor (más pequeño) valor de c(x') que puede ser conseguido al *revertir* un movimiento especifico a la solución previo, de algún x' a algún s(x') tal que c(s'(x')) = c(s(x)). Supongamos que todos los posibles valores de c(x) para diferentes vectores x están dados por los enteros q = 1, 2, …, U, y sea MEJOR(q) = mínimo (mejor) valor de c(x) que puede ser conseguido al revertir un movimiento previo que produjo c(s(x)) = q. Inicialmente hagamos MEJOR(q) = U + 1. Luego, cuando se realiza un movimiento específico a la solución de x a s(x) actualizamos MEJOR(q) para q =c(s(x)) usando

[136] Glover (1.989).

la regla MEJOR(q) = Min(MEJOR(q), c(x)). En este contexto el nivel de aspiración puede permitir anular un movimiento tabú que lleve de (un diferente) x a s(x) si

c(s(x)) < MEJOR(c(x))

ésta forma particular de aspiración es una instancia del control de la condición (3).

La forma correspondiente para (2) en este contexto está dada por definir A(s,x) = el mejor (más pequeño) valor de c(s'(x')) obtenido para todos los anteriores movimientos específicos a la solución de x' a algún s'(x'), donde c(x') = c(x). Esta forma alternativa de función de nivel de aspiración puede ser obtenida inicializando MEJOR(q) de la misma forma que antes, pero cuando se realiza un movimiento específico a la solución de x a s(x), MEJOR(q) se actualiza por q = c(x), haciendo MEJOR(q) = Min(Mejor(q), c(s(x)). Y nuevamente la condición para anular la restricción tabú de un movimiento que lleva de (una diferente) x a s(x) es la misma que antes:

c(s(x)) < MEJOR(c(x))

por lo tanto las condiciones (2) y (3) se combinan rápidamente haciendo *ambas* actualizaciones de MEJOR(q) como se indicó precedentemente, ya que el control para decidir la anulación de la restricción tabú es el mismo en ambos casos.

Es posible utilizar diferentes funciones MEJOR de manera tal de aumentar la probabilidad de que un movimiento tabú escape su estado tabú mediante el requerimiento de que por lo menos una de las inecuaciones asociadas se mantenga, aunque esto demanda mayor esfuerzo para el chequeo y almacenamiento. Ahora bien, una alternativa interesante es definir A(s,x) solamente en relación con movimientos actualmente tabú.

Las restricciones tabú y los criterios del nivel de aspiración para la BT juegan un rol dual en restringir y guiar el proceso de búsqueda. Las restricciones tabú permiten considerar a un movimiento admisible si las mismas **NO APLICAN**, mientras que los criterios de aspiración, permiten considerar a un movimiento admisible si las mismas **APLICAN** (se cumplen o satisfacen). Esta complementariedad de las nociones subyacentes en la restricción tabú y en los criterios de aspiración permite que ambas se integren en el mismo marco de funcionamiento. En honor a la síntesis omitiremos ese análisis, dejando el detalle del mismo al trabajo ya mencionado de Glover (1989).

Por otro lado, la **dominación** y la **equivalencia** operan de formas diferentes en la BT de lo que lo hacen en el contexto usual de la optimización, requiriendo una referencia a la naturaleza del movimiento empleado, pero nuevamente dejamos esta discusión a Glover.

4- Memorias intermedias y de largo plazo

Las funciones de memoria intermedia y de largo plazo se emplean dentro de la BT para conseguir **intensificación regional y diversificación global** de la búsqueda. Combinadas con la memoria a corto-plazo que se materializa por las listas tabú, las memorias intermedia y de largo plazo proveen el juego necesario entre "aprender" y "des-aprender".

La memoria intermedia opera al registrar y comparar elementos de un número seleccionado de "mejores" soluciones generadas durante un periodo particular de la búsqueda. Características que son comunes a todas o a una gran mayoría de estas soluciones (como ser valores recibidos por ciertas variables particulares) se consideran como atributos **regionales** de las buenas soluciones. El método busca entonces soluciones que exhiben esas características, al restringir o penalizar movimientos disponibles durante un periodo subsiguiente de intensificación regional de la búsqueda.

La memoria a largo plazo, cuyo objetivo es diversificar la búsqueda, emplea principios que son groseramente inversos a los de la memoria a corto plazo. En lugar de inducir a que la búsqueda se focalice más intensamente en regiones que contienen (o que pueden ser extrapoladas de) buenas soluciones previamente encontradas, la función de memoria a largo plazo guía el proceso a regiones que marcadamente contrastan con aquellas examinadas hasta el momento. **NO** se trata de buscar la diversidad al generar una serie de puntos de inicio aleatorios, sino que el objetivo es crear un criterio de evaluación que puede ser utilizado por la heurística de búsqueda de manera tal de generar un nuevo punto de búsqueda NO aleatorio. Los criterios de evaluación penalizan los elementos que la memoria a largo plazo encuentra como prevalentes en las ejecuciones previas del proceso de búsqueda.

5- Consideraciones Finales

Está claro que este anexo es una breve y somera aproximación a la búsqueda tabú detallando sus principales líneas generales sin ahondar en profundidad en sus complejidades. Los detalles particulares de la implementación puntual quedan asociados al problema en estudio y al esquema de solución heurística propuesto que incorpora los elementos de la búsqueda tabú y de la memoria adaptativa.

ANEXO III

Órdenes de Magnitud[137]

Los símbolos que se emplean para comparar tasas de crecimiento de funciones son esencialmente: "o", "O", "Θ", "$\sim$" y"Ω", para definirlos, sean $f(x)$ y $g(x)$ dos funciones de x, cada uno de los símbolos anteriores se emplea para comparar la rapidez de crecimiento de f y g. Si decimos que $f(x) = o(g(x))$, entonces estamos informalmente diciendo que f crece más lento que g cuando x es "muy grande". Si decimos que $f(x) = O(g(x))$ estamos informalmente diciendo que ciertamente f no crece a una tasa más rápida que g (puede crecer a la misma tasa o más lento). Más formalmente:

i- Decimos que $f(x) = o(g(x))$ ($x\to\infty$) si el $\lim_{x\to\infty} f(x) / g(x)$ existe y es igual a 0.
Por ejemplo:
$x^2 = o(x^5)$
$\sin x = o(x)$
$1/x = o(1)$

ii- Decimos que $f(x) = O(g(x))$ ($x\to\infty$) si $\exists c, x_0$tal que$|f(x)|<cg(x)$ ($\forall x>x_0$).
Por ejemplo:
$\sin x = O(1)$

iii- Decimos que $f(x) = \Theta(g(x))$ si existen constantes $c_1> 0$, $c_2> 0$, x_0 tales que para todo $x>x_0$ es cierto que $c_1g(x)<f(x)<c_2g(x)$.
Por ejemplo:
$(x + 1)^2 = \Theta(3x^2)$
$(1 + 3/x)^x = \Theta(1)$

iv- Decimos que $f(x)\sim g(x)$ si el $\lim_{x\to\infty} f(x) / g(x) = 1$.
Por ejemplo
$x^2 + x \sim x^2$
$\sin 1/x \sim 1/x$

Notemos que, mientras que es cierto que $2x^2 = \Theta(x^2)$, **no** es cierto que $2x^2\sim x^2$

v- Decimos que $f(x) = \Omega(g(x))$ si existe un $\varepsilon> 0$, y una secuencia $x_1, x_2, x_3, \ldots \to\infty$ tal que $\forall j$: $|f(x_j)|>\varepsilon g(x_j)$

[137] Ver por ejemplo: Wilf(1994 Internet Edition).

Notemos que Ω es la negación de *o*. Es decir, si decimos que $f(x) = \Omega(g(x))$, estamos diciendo que **no** es cierto que $f(x) = o(g(x))$.

Por otro lado:

a- Una función que crece más rápidamente que x^a, para cada constante *a*, pero que crece más lentamente que c^x para cada constante $c > 1$, se dice que tiene un crecimiento moderadamente exponencial. O más formalmente, $f(x)$ tiene un crecimiento moderadamente exponencial si para cada $a > 0$ tenemos $f(x) = \Omega(x^a)$ y para cada $\varepsilon > 0$ tenemos $f(x) = o((1 + \varepsilon)^x)$.

b- Una función *f* es de crecimiento exponencial[138] si existe $c > 1$, tal que $f(x) = \Omega(c^x)$ y existe *d* tal que $f(x) = O(d^x)$.

[138] Más allá de las funciones de crecimiento exponencial hay funciones que crecen tan rápidamente como se quiera, por ejemplo $n!$, que crece más rápidamente que c^n para cada constante fija *c*; y 2^{n*n} que crece aún más rápido que $n!$. En el otro extremo, tenemos las funciones de crecimiento lento (o de crecimiento logarítmico).

Referencias:

Altman M. (1.997).Is automation the answer: The computational complexity of automated redistricting.*Rutgers Computer and Law Technology Journal*. 23(1):81–141. Disponible en: http://maltman.hmdc.harvard.edu/papers/complexv1_1.pdf.

Altman, M. & McDonald, M.P.(.2009). BARD: Better Automated Redistricting.*Journal Of Statistical Software*, VV(II). Disponible en: http://www.jstatsoft.org/.

Balas E, y Toth P. (1.985), Branch and Bound Methods for the Traveling Salesman Problem.*The Traveling Salesman Problem*.G. Lawler, J.K. Lenstra, A. H. G. Rinnooy Kan, D. Shmoys, eds., *TheTraveling Salesman Problem*, J. Wiley, Chichester.

Box, G.E.P., Hunter W.G., Hunter J.S. (1.978) *Statistics for Experimenters*. New York, John Wiley & Sons.

Bozkaya B. Political Districting: A Tabu Search algorithm and Geographic Interfaces**.** *University of Alberta*. 1999. (PhD. Thesis).

Bozkaya, B., Erkut E. y Laporte, G. (2.003). A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144, 12–26.

Bozkaya, B., Erkut E., Haight, D. y Laporte, G. (2010). Designing new electoral districts for the city of Edmonton. *Aceptado para publicación.* Obtenido directamente de los autores. Ver: http://research.sabanciuniv.edu/13317/

Cook, S.A., (1.971). The Complexity of Theorem Proving Procedures. *Proceedings of the Third ACM Symposium on Theory of Computing*. pp. 151-158.

Cordeau, J.-F., Gendreau, M. y Laporte, G., (1.997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2), p.105-119. Disponible en: http://doi.wiley.com/10.1002/%28SICI%291097-0037%28199709%2930%3A2%3C105%3A%3AAID-NET5%3E3.3.CO%3B2-N.

Curtin,K. M., K. L. Hayslett-McCall, and F. Qiu, (2.010). Determining Optimal Police Patrol Areas with Maximal Covering and Backup Covering Location Models. *Networks and Spatial Economics*. DOI 10.1007/s11067-007-9035-6. Vol. 10, No. 1, 125-145. Disponible en: http://www.utdallas.edu/~ffqiu/published/2010CurtinHayslett-McCallQiu2010.pdf

D'Amico, S. J., Wang S., Batta R. y Rump C.M. (2.002). A simulated annealing approach to police district design. *Computers & Operations Research* 29, 667–684.

de Werra, D., Hertz, A. (1.989). Tabu search techniques: A tutorial and an application to neural networks. *OR Spectrum* (1989-09-01), p. 131-141. Disponible en: http://dx.doi.org/10.1007/BF01720782

ESRI (C). **Using Avenue. Customization and Application Development for ArcView GIS**. (1996). ESRI PRESS.

Faigle, U., Kern, W. (1.992) Some Convergence Results for Probabilistic Tabu Search. *INFORMS JOURNAL ON COMPUTING*. PP 32-37. V 4 N 1. Disponible en http://joc.journal.informs.org/cgi/content/abstract/4/1/32). (INFORMS Journal on Computing, ISSN 1091-9856, was published as ORSA Journal on Computing from 1989 to 1995 under ISSN 0899-1499).

Fotheringham, Stewart (Editor); Peter Rogerson (Editor) (1.994) *Spatial Analysis and GIS (Technical Issues in Geographical Systems)*. Taylor & Francis Ltd.

Friden, C., Hertz, A., y de Werra, D. (1.989). STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing 42*, (1989-03-16), pp. 35-44. Disponible en: http://dx.doi.org/10.1007/BF02243141.

Friden, C., Hertz, A. y de Werra, D. (1.990) TABARIS : an exact algorithm based on tabu search for finding a maximum independent set in a graph. *Computers & operations research*, 17(5), pp.437-445. Dispnible en: http://cat.inist.fr/?aModele=afficheN&cpsidt=5040357.

Garey, M.R. y Johnson, D.S., (1.979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman.

Gendreau, M., Hertz, A., y Laporte, G. (2005). A New Tabu Search Heuristic for the Site-Dependent Vehicle Routing Problem. *Management Science*, *40*, 107–119. Springer

Glover, F., (1.986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5), p.533-549.
Disponible en: http://linkinghub.elsevier.com/retrieve/pii/0305054886900481.

Glover, F. (1.989) Tabu Search — Part I. *ORSA Journal on Computing*. 1: 3, 190-206.

Glover, F. (1.990) Tabu Search — Part II. *ORSA Journal on Computing*. 1990 2: 1, 4-32.

Glover, F. y Laguna, M., (1.997). *Tabu Search.* Kluwer Academics Publisher.

Glover, F., (2.002). Tabu search and finite convergence. *Discrete Applied Mathematics*, 119(1-2), pp.3-36.
Disponible en: http://linkinghub.elsevier.com/retrieve/pii/S0166218X01002633.

Goldreich O. (2.006). *Computational Complexity: A conceptual Perspective*. Department of Computer Science and Applied Mathematics. Weizmann Institute of Science, Rehovot. Israel.

Gonzalez, T.F, (2.007). *Handbook of Approximation Algorithms and Metaheuristics.* (Cap. 23). Chapman & Hall/CRC (Taylor & Francis Group). Disponible en http://leeds-faculty.colorado.edu/glover/TS%20-%20Principles%20of%20Tabu%20Search%20-%20Glover,%20Laguna%20and%20Marti.pdf

Hertz, A. & De Werra, D., (1.987). Using tabu search techniques for graph coloring. *Computing*, 39(4), p.345-351.
Disponible en http://www.springerlink.com/index/Y766J232466L1674.pdf

Hertz, A.; Taillard E., de Werra D. *A TUTORIAL ON TABU SEARCH***.** Université de Montréal, Centre de Recherche sur les Transports, Montréal, Canada H3C 3J7.

Hertz, A., Taillard, E.; de Werra, D., (1.990). A tutorial on tabu search. *Interfaces*, 20(i), p.74-94.
Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.521&rep=rep1&type=pdf.

Hetland, M. L., *Beginning Python, From Novice to Professional*, Second Edition. Apress 2008.-

Johnson, D.S., (1.984). The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3, 182-195 (1982). Disponible en: http://www2.research.att.com/~dsj/columns/col3.pdf.

Karp, R.M., (1.972). Reducibility among combinatorial problems. R. E. Miller & J. W. Thatcher, eds. *Complexity of Computer Computations*. Plenum Press, pp. 85-103. Disponible en: http://www.jstor.org/stable/2271828?origin=crossref.

Li M y P M B Vitanyi (1.992) Average case complexity under the universal distribution equals worst-case complexity, 42 *Information Processing Letters* 145 (1992).

Li Z., Wang R y Wang Y. (2.007). A Quadratic Programming Model for Political Districting Problem. *The First International Symposium on Optimization and Systems Biology (OSB'07) Beijing, China, August 8–10, 2007* Copyright © 2007 ORSC & APORC §pp. 427–435. Disponible en: http://www.aporc.org/LNOR/7/OSB2007F49.pdf

Macmillan W., Pierce T. (1.994). Optimization modeling in a GIS framework: the problem of political redistricting. *Spatial Analysis and GIS*, Stewart Fotheringham y Peter Rogerson Ed. Taylor & Francis Ltd.

Martí, R. (2003). Procedimientos Metaheurísticos en Optimización Combinatoria. *Matemàtiques* 1(1), 3-62. Disponible en: http://www.uv.es/rmarti/paper/docs/heur1.pdf.

Mehrotra A., Johnson E.L., Nemhauser G.L. (1.998) An Optimization Based Heuristic for Political Districting**.** *Management Science archive* Volume 44 , Issue 8 (August 1998) Pages: 1100 - 1114 1998 ISSN:0025-1909.

Montgomery D.C. (2.000). *Design and Analysis of Experiments, 5th edition.* New York, John Wiley & Sons.

Nelson, A. Data reduction and low dimensional representation of complex spatial databases. *CIAT 2000. International Center for Tropical Agriculture*.

Openshaw, S., (1.977). Optimal zoning systems for spatial interaction models. *Environment and Planning A,* 9, 169-184.

Openshaw, S., (1.978), An empirical study of some zone design criteria. *Environment and Planning A,* 10, 781-794

Openshaw, S., (1.978) An optimal zoning approach to the study of spatially aggregated data. I. Masser and P.J.B. Brown *Spatial Representation and Spatial Interaction* Martinus Nijhoff, Leiden 93-113.

Openshaw, S., Rao L. (1.994). *Re-engineering 1991 census geography: serial and parallel algoriths for unconstrained zone design***.** School of Geography, University of Leeds. Disponible en:http://www.geography.leeds.ac.uk/papers/95-3/

Openshaw, S., Rao L. (1.995). Algorithms for reengineering 1991 Census Geography. *Environment and Planning* A, 27, pp 425-446

Ricca, F., Scozzaria A y Simeone B. (2.008). Weighted Voronoi region algorithms for political districting**.** *Mathematical and Computer Modelling Volume 48*, Issues 9-10, November 2008, Pages 1468-1477

Ricca, F., y Simeone, B. (2.008). Local search algorithms for political districting. *European Journal Of Operational Research*, *189*(3), 1409-1426

Rochat, Y. E. Taillard (1.995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1(1), pags. 147-167.

Tavares-Pereira F., Figueira J., Mousseau V. y Roy B. *Multiple Criteria Districting Problems, The Public Transportation Network Pricing System of the Paris Region*. Departamento de Matemática, Universidade da Beira Interior.

Vickrey, W. (1961) On the Prevention of Gerrymandering**.** *Political Science Quarterly,* 76, pp. 105-110.-

Wilf, Herbert S. (1.994 Internet Edition). *Algorithms and Complexity*. University of Pennsylvania. Disponible en: http://www/cis.upenn.edu/wilf.

Yamada, T. (2.009). A mini-max spanning forest approach to the political districting problem. *International Journal of Systems Science*, *40*(5), 471-477.