Discusión y Análisis de la metaheurística SN

Pablo Rodríguez Bocca - Depto. Investigación Operativa, InCo, FI, UdelaR. Reporte Tecnico 03-02, 2003.

Resumen:

Una metaheurística es un conjunto de ideas relacionadas que tienen por objetivo la resolución aproximada de problemas de optimización combinatoria. Esta trabajo describe una relativamente nueva metaheurística llamada SN, que se basa en la idea de dividir un problema de optimización en subproblemas de decisión más fáciles de resolver heuristicamente.

SN surge en junio de 2001, en la tesis de Licenciatura de S Urrutia, a cargo de I. Loiseau, con nombre "SN: Una Nueva Metaheurística". Con su extrema juventud esta metaheurística presenta varios campos no suficientemente explorados.

En este trabajo se analiza y discute las ventajas que presenta esta heurística a nivel general, así como una especialización e implementación para el Problema de Steiner en Grafos. Para esto caso de estudio se recrean y contrastan resultados experimentales con los obtenidos en la tesis original.

Palabras Claves: SN, Metaheurísticas, Problema de Steiner en Grafos (STP)

1. Objetivos

El objetivo principal del trabajo es presentar la metaheurística SN [1], remarcando las principales características que la distinguen y la performance actualmente alcanzada.
Este estudio principalmente es un resumen de [1], sin embargo no solo se pretende interpretar el trabajo realizado, sino reproducir algunos de sus resultados e incorporar nuevos experimentos.
El carácter del trabajo es fuertemente exploratorio, donde se pretende detectar posibles líneas para investigaciones de mayor aliento a futuro, así como los campos mas fructíferos de aplicación.

Este documento comienza con una pequeña introducción al *marco teórico y la motivación* del área. Posteriormente, en el capítulo 3, se explica en detalle la *metaheurística SN*, haciendo hincapié en la idea que hay atrás del método, sin descuidar los detalles para una implementación.
Todos los conceptos de la metaheurística se refinan seguidamente en el capítulo 4 con la aplicación a un problema especifico: el *problema de Steiner en Grafos*. Para este problema se desarrollan y analizan mejoras sobre el método general .
En el capítulo 5 de *trabajo realizado* se detallan los experimentos realizados, así como una breve reseña de los resultados previos conocidos.
Finalmente se termina con *conclusiones* generales y *trabajo a futuro*, en los capítulos 6 y 7.

2. Motivación y Marco Teórico

Un problema de Optimización Combinatoria consiste en encontrar de todos los subconjuntos de un conjunto dado, aquel que cumpliendo determinadas restricciones maximice o minimice cierta función objetivo. Este tipo de problemas siempre se puede resolver evaluando la función objeto para todos los subconjuntos, sin embargo este proceso es muy costoso en tiempo por la cantidad de posibilidades que se presentan en general y por tanto difícilmente practicable.

La teoría de la Complejidad Computacional [5] clasifica los problemas como computables e incomputables, según si tienen o no una solución algorítmica. Los problemas computables son a su vez divididos en clases según la complejidad de la mejor solución algorítmica conocida. La clase *NP* incluye a aquellos problemas para los cuales existe un algoritmo de enumeración completa que los resuelve, cuyo espacio de búsqueda se restrinja a un árbol de profundidad polinomial con respecto al tamaño de la entrada. Dentro de la clase *NP* se encuentra la clase *P* y la clase *NP-Completo.* La clase *P* esta formada por problemas con solución algorítmica polinomial conocida (el tiempo de ejecución crece polinomialmente con respecto a la entrada) y la clase *NP-Completa* esta formada por aquellos problemas de *NP* que son *NP-Difíciles* (un problema es *NP-Difícil* si todo problema *NP* es reductible polinomialmente a él). Por tanto los distintos problemas *NP-Completos* son equivalentes en complejidad, por lo que encontrando una solución polinómica para uno de ellos se podría demostrar que *NP=P*.
Muchos de los problemas de optimización combinatoria son *NP-Difíciles*, por tanto para resolverlos es posible que no exista un algoritmo polinomial, esto justifica la utilización de algoritmos no exactos: heurísticas, si se pretende encontrar la solución en tiempos razonables.

Una Heurística [7] es una técnica que busca soluciones buenas (cercanas al optimo) a un costo computacional razonable, sin garantizar factibilidad u optimalidad (y en muchos casos sin establecer cuan cerca del optimo se haya la solución factible encontrada). En general las heurísticas no recorren todo el espacio de soluciones, pudiendo alcanza en general solo óptimos locales.

Las Metaheurísticas [7] pueden definirse como un proceso iterativo que guía a una heurística subordinada combinando distintos aspectos para explorar y explotar el espacio de búsqueda usando estrategias de aprendizaje para estructurar la información con el objetivo de encontrar eficientemente soluciones cercanas al óptimo.
La mayoría de las metaheurísticas tienen como principal rol escapar a óptimos locales encontrados por la heurística subordinada (o constructiva) con el fin de recorrer de forma basta todo el espacio de soluciones, en contraposición la metaheurística SN intenta simplificar heuristicamente el espacio de soluciones para que la heurística subordinada sea mas efectiva.

3. Metaheurística SN

Inspirada en la búsqueda de un método de simplificación heurística para el espacio de soluciones, SN va transformando sucesivamente el problema a resolver en problemas mas simples (con espacio de soluciones mas pequeños) sobre los cuales se aplica la heurística constructiva. Por tanto además del concepto de solución actual, SN incluye la idea de problema actual.
La clave entonces es buscar un método de simplificación, para esto *"SN divide al problema de optimización original (Q) en varios subproblemas de decisión (q1, q2, ..qn) y los resuelve heuristicamente obteniendo una solución (s1, s2, ..sn) para cada uno de ellos. Establece para cada solución un grado de confianza (gc1, gc2, ..gcn). Este grado de confianza es un valor que indica el nivel de certeza que se tiene sobre la solución heurística hallada para el subproblema. La solución al subproblema cuyo grado de confianza sea máximo, se asume como verdadera. Luego, utilizando la información que da la resolución de este subproblema, el problema original es cambiado por otro equivalente de menor complejidad. Todo el procedimiento se repite hasta que el problema no pueda ser dividido en subproblemas..."* [1].

3.1 La Idea

```
Entrada: problema combinatorio Q
Mientras pueda dividirse Q en subproblemas de decisión q1..qn:
  Para cada subproblema de decisión qi :
       Evaluar heuristicamente las opciones de decisión de qi.
       Determinar la mejor opción y su grado de confianza.
  Fin Para
  Modificar Q con la mejor opción de mayor confianza.
Fin Mientras
Resultado: mejor opción de la última decisión que se pudo tomar.
```

Idea 1. Metaheurística SN Básica.

Es necesario puntualizar que por subproblema de decisión se debe entender una pregunta de alto nivel con dos opciones o respuestas posibles: Sí o No.
En una primera instancia es difícil de imaginar como puede dividirse un problema combinatorio cualquiera en subproblemas de decisión. El hecho es que para cada tipo de problema combinatorio se podrían elegir distintas decisiones a ser tomadas y esto tendría un impacto directo sobre los resultados del algoritmo. Se sigue desarrollando este punto más adelante.
Para asegurar que el algoritmo termina los subproblemas de decisión deben ser verdaderas simplificaciones del problema y una cantidad finita, es decir que disminuyan su complejidad.
También debe notarse que el grado de confianza es un valor que intenta reflejar la seguridad que se tiene sobre la opción elegida en cada decisión. Claramente existe una incertidumbre en dicho valor, porque si se obtuviera un valor determinista sin error, se tendría un método que resuelve en forma polinómica un problema *NP-Difícil*.

3.2 Método y Pseudocódigo

La idea anterior es la base y fundamento de la metaheurística SN, en esta sección se intenta determinar mas claramente el método introduciendo un pseudocódigo.

Por ahora no se ha definido como dividir el problema actual en subproblemas de decisión, entonces suponga que se cuenta con una función *T*, que dado el problema actual *Q*, un identificador de

subproblema de decisión *i* y una opción para dicha decisión *sí* (Sí o No) devuelva como resultado un problema combinatorio *Q*' de menor complejidad, donde si la opción es correcta la solución optima de *Q*' es igual a la de *Q* y si la opción es incorrecta la solución optima de *Q'* es igual o peor que la de *Q*. Esto es Q_{si}*= T(Q, i, SI)* y Q_{no} *= T(Q, i, NO)*.

Lo que resta especificar es como se determina la mejor opción para cada subproblema y su grado de confianza: para esto se modifica el problema *Q* original asumiendo que la solución al subproblema *i* es SÍ, y se ejecuta la heurística constructiva para el problema simplificado Q_{si}. Se repite este procedimiento pero asumiendo que la opción es No. Si el primer resultado es mejor que el segundo la solución heurística del subproblema es Sí, en caso contrario es No. En ambos casos se elige como grado de confianza la diferencia entre ambos resultados.

El pseudocódigo se detalla a continuación, donde H es la heurística constructiva que dado un problema combinatorio devuelve el valor de la función objeto para su solución.

```
solActual = peor solución posible
Mientras pueda dividirse a Q en subproblemas q1..qn hacer
   Para i desde 1 hasta n hacer
       Q' = T(Q, i, SÍ)
       ResSi = H(Q')
       Si ResSi es mejor que solActual entonces solActual = ResSi
       Q' = T(Q, i, NO)
       ResNo = H(Q')
       Si ResNo es mejor que solActual entonces solActual = ResNo
       Si ResSi es  mejor que ResNo entonces s_i = SÍ sino s_i = NO
       gc_i = Abs(ResSi – ResNo)
   Fin Para
   ganador = arg max_{1≤j≤n} {gc_j}
   Q = T(Q, ganador, s_ganador)
Fin Mientras
Resultado = solActual
```

Pseudocódigo 1. Metaheurística SN Básica.

Nota. El método es ampliable a decisiones con más de dos opciones.

3.3 Puntos Clave, Idea Atrás del Método

Existen una serie de puntos clave en el método; el análisis de estos puntos permite entrever las ideas subyacentes y las implicancias que éstas tienen sobre la performance de la metaheurística. A continuación se especifican estos puntos y las interrogantes que cada uno arroja. En las secciones siguientes se estudia cada uno de ellos en detalle.

- ***Subproblema.***

Claramente la clave mas importante es la elección de los subproblemas de decisión: ¿Cómo elijo convenientemente el tipo de decisión para un problema combinatorio dado? ¿Existen algunos tipos de problemas combinatorios que se adaptan más a una división en problemas de decisión? ¿Cómo influye la elección de las decisiones en los tiempos y performance del algoritmo? ¿Qué tan confiable es el grado de confianza?

- ***Transformación del Problema.***

La transformación del problema obtiene a partir del problema actual los distintos subproblemas de decisión. ¿Qué tipo de transformación es valida? ¿Debe cumplir con alguna propiedad?

- ***Heurística Constructiva.***

Para cada subproblema de decisión se ejecuta la heurística constructiva, basando la elección de la simplificación del problema en su resultado. ¿Algunas heurísticas se adaptan mejor al método? ¿Existe alguna relación deseada entre la elección del tipo de decisión y la elección de la heurística? ¿Cuantas veces es ejecutada?

Cada uno de estos puntos merece entonces un estudio mas detallado. El mismo se realizo en las siguientes secciones, donde se responden algunas de estas preguntas.

3.4 Subproblemas

Lo mas importante de los subproblemas de decisión es que son una simplificación del problema original, donde se toma una decisión de alto nivel (Sí o No).

Como ejemplo ilustrativo de lo que puede ser un tipo de decisiones, se introduce el problema : Recubrimiento Mínimo de Ejes. Para este problema se muestra además las implicancias que tiene la elección de dicho tipo de decisiones.

- ***Recubrimiento Mínimo de Ejes***

Una definición informal del problema es: Dado un grafo, elegir el conjunto de nodos de cardinalidad mínima, tal que para toda arista uno de sus nodos incidentes pertenezca a dicho conjunto.

Para fijar ideas se considera la siguiente instancia del problema:

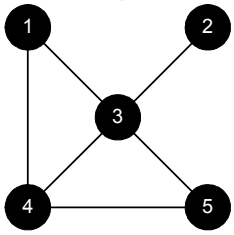

Figura 1. Ejemplo 1. RME.

Las preguntas que deben hacerse para buscar las soluciones al problema son: Dado un subconjunto: ¿este subconjunto de nodos recubre los ejes?, y si es así ¿es de cardinalidad mínima? Para poder resolver el problema deberían responderse estas preguntas para todos los subconjuntos posibles.
Sin embargo lo mas probable que un ser humano intente resolver el problema imaginándose que un nodo pertenece a la solución y observando las implicancias que esto trae aparejado, si la opción le convence la adopta y prueba con otro nodo. Más formalmente se hace la siguiente pregunta para cada nodo: ¿pertenece el nodo *n* a la solución óptima?
La metaheurística SN refleja este paradigma y considerando esta pregunta como subproblema de decisión, estima para cada nodos la pertenencia o no a la solución optima, opta por aquella decisión sobre la cual tiene mayor confianza (incluir o quitar el nodo de la solución). Optar significa que la decisión la da por cierta y simplifica el problema. Con los nodos restantes repite el proceso, y sucesivamente va simplificando el problema hasta que el grafo se vacíe, obteniendo una solución.
En este caso la transformación T es:

T(Q, n, SÍ) - Si el nodo *n* pertenece a la solución optima de *Q* entonces el problema *Q'* consistirá en calcular la cantidad de nodos del RME del grafo *Q - {n}* y sumar 1.

T(Q, n, NO) - Si el nodo *n* no pertenece a la solución optima de *Q* entonces el problema *Q'* consistirá en calcular la cantidad de nodos del RME sobre el grafo que queda al borrar de *Q* al nodo *n* y sus nodos adyacentes y sumar la cantidad de nodos adyacentes a *n*.

Considérese que se está evaluando la pertenencia del *nodo 1*, los dos subproblemas que surgen luego de la transformación *T* del problema original son:

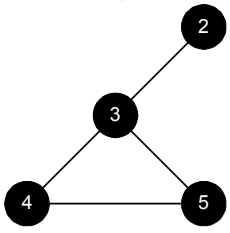

Figura 2. T(Q, 1, SÍ). Sumarle 1 al resultado de la heurística para este grafo.

Figura 3. T(Q, 1, NO). Sumarle 2 al resultado de la heurística para este grafo.

Es necesario sumar 1 o 2 respectivamente a la solución arrojada por la heurística sobre cada grafo transformado para considerar los nodos quitados que pertenecen a la solución.

Pero ¿Cómo transforma cada subproblema de decisión al espacio de soluciones del problema original?
En la decisión de inclusión del nodo 1, puede clasificarse el espacio de la siguiente forma:

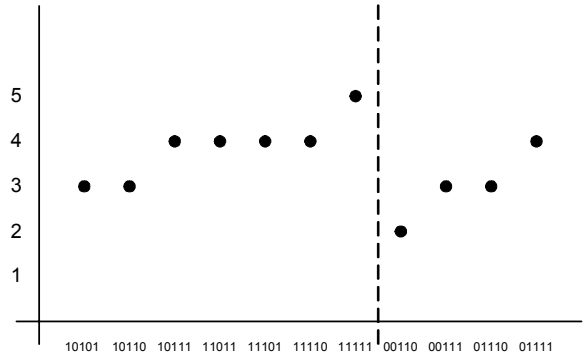

Figura 4. Espacio de Soluciones de RME.

Observándose que cada subproblema de decisión divide el espacio de búsqueda en dos. El tamaño de cada uno de esos subespacios no tiene porque ser el mismo, puesto las soluciones no factibles se dividen arbitrariamente entre los dos subespacios.
Pensando en la idea original este ejemplo presenta la transformación del espacio de soluciones más simple, que es la división en dos grupos.

El procedimiento de simplificación de la metaheurística se basa en asumir la decisión mas confiable como cierta, esta es aquella cuya evolución para cada uno de estos dos subespacios de la heurística constructiva resulte en la mayor diferencia posible. Por tanto si se pretende que el grado de confianza refleje la certeza que se tiene sobre una decisión, se le debe pedir a la heurística constructiva que en promedio ofrezca mejores soluciones para el subespacio que contiene la solución óptima[1]. Es decir, no es necesario esperar de la heurística que ofrezca soluciones cercanas al óptimo, sino que arroje en promedio resultados mejores para subespacios que incluyen mejores soluciones [2].

El razonamiento anterior permite visualizar dos puntos importantes:

- Existe una relación importante entre los subespacios de soluciones obtenidos según un tipo de decisión y el comportamiento deseado de la heurística constructiva (esto se discute más adelante).
- El grado de confianza determina la elección futura, si ocurre una equivocación, es decir se opta por el subespacio que no incluye al optimo, el algoritmo no puede alcanzar el optimo (a menos que exista más de un óptimo). En este sentido es muy importante la certeza del grado de confianza. Se considera que debe refinarse el concepto de grado de confianza, puesto que por ejemplo frente a dos decisiones que ofrezcan el mismo grado de confianza, cual es más confiable en realidad (un criterio podría ser aquella que obtuvo una solución mejor). Ahora ¿porqué ir decidiendo por grado de confianza y no también por aquella decisión que alcanza la mejor solución en la evaluación heurística?

- ***Subproblema Crítico.***

Dependiendo del tipo de decisión adoptada y de la instancia del problema existen algunas clasificaciones útiles de subproblemas.
Un subproblema critico es aquel donde una de sus dos opciones llevan al problema original a no tener solución factible. Existen 3 encares posibles para enfrentar estos subproblemas:

- No discriminarlos. Debe asegurarse que la resolución heurística de estos subproblemas arrojen siempre la solución correcta con alto grado de confianza. Se pide alto grado de confianza para que se opte en forma temprana por esta decisión, reduciendo tiempo de computo innecesarios.
- Eliminarlos mediante reducciones. Es posible eliminar muchos de estos subproblemas mediante una reducción inicial.

[1] Podría pensarse, en búsqueda de ese comportamiento, ejecutar varias heurísticas, o varias veces la misma heurística con parámetros distintos y quedarse con el mejor resultado obtenido. Mas adelante se observará que este método es descartable por el alto costo computacional que ya presenta la metaheurística original.
[2] Obviamente no es posible, dada la complejidad del problema, exigirle a una heurística un comportamiento perfecto en este sentido.

- No tratarlos. Es posible redefinir que es un subproblema y que no lo es, para esto se debería evaluar en cada caso si hay solución factible, cosa que en mucho casos es muy costoso en tiempo.

- ***Subproblema Polinomial.***

Otro tipo de subproblema son aquellos que tiene una solución exacta polinomial. Es claro que como un problema combinatorio Q es NP-Difícil, por lo menos uno de los subproblemas en los cuales se divide debe ser también NP-Difícil y entonces no tener solución polinómica conocida.
En general no son comunes los subproblemas polinómicos, pero podría pensarse que luego de algunas transformaciones el problema pueda simplificarse a un problema con solución exacta polinomial (esto se verá mas claramente en el Problema de Steiner para Grafos).

- ***Necesidad de un Problema Caso Base.***

El algoritmo es un proceso iterativo, donde en cada paso se divide el problema actual *Q* en una cantidad finita de subproblemas de decisión y se opta por la mejor decisión. Para que el algoritmo termine es necesario que exista un problema base el cual no pueda seguirse dividiendo.

3.5 Transformación del Problema

En la sección anterior se analizaron los subproblemas de decisión en general, en esa instancia se permitió entrever dos propiedades necesarias sobre la transformación *T*:

- ***Acercamiento al Caso Base***

Aplicar la transformación *T* a un problema combinatorio siempre debe arrojar un subproblema con menor profundidad en su espacio de búsqueda que el original. Esto no es mas que la propiedad de simplificar varias veces mencionada antes.

- ***No perdida del Óptimo***

La transformación debe ser tal que si se decide correctamente entonces la solución optima del subproblema es igual a la solución optima del problema original.

3.6 Heurística Constructiva

La elección de la heurística constructiva tiene un impacto directo sobre el tiempo y performance de la metaheurística:

- ***Tiempo Computacional***

Suponiendo que el tiempo computacional de la heurística *H* es C_H y este se mantiene aproximadamente constante a lo largo de la ejecución del algoritmo se tiene que el tiempo total de la metaheurística es:

$$\frac{n(n-1)}{2} 2C_H$$

Donde en cada iteración i (i:n..1) se divide el problema en i subproblemas y cada subproblema ejecuta dos heurísticas una para cada opción de decisión (Sí y No).
El tiempo de SN es en general bastante superior al de otras metaheurísticas, por esto se eligen heurísticas constructivas livianas, además se realizan ciertas mejoras sobre el algoritmo básico (estas son presentadas mas adelante en el caso de estudio).

- ***Calidad menos importante.***

Debe saber diferenciar la opción que presenta el óptimo, en búsqueda de seguridad en el grado de confianza. Por supuesto que si la heurística encontrara el optimo cada vez, la metaheurística también encontraría el optimo, como esto no se le puede pedir a la heurística es necesario elegir aquella que sepa diferenciar mejor en promedio entre el subespacio que tiene el optimo y aquel que no lo tiene (lo cual es pedirle menos que encontrar el optimo).

3.7 Características Generales

Antes de pasar al caso de estudio es importante remarcar las características mas importantes que distinguen a SN del resto de las metaheurísticas.

- ***Pruebas de Reducción***

Dado que en cada paso de la iteración global se transforma el problema, obteniendo un nuevo problema más simple, es posible ejecutar técnicas de reducción de problema en cada paso.

- ***Construcción Parcial***

Al igual que muchas otras metaheurísticas, SN es un proceso iterativo donde en cada paso se va refinando progresivamente la solución. Esto permite terminar la ejecución antes de la finalización establecida del algoritmo obteniendo no solo la mejor solución hasta el momento sino también un problema más simple (probablemente) equivalente.

- ***Paralelismo***

Dado que en cada paso de la iteración es necesario evaluar una serie de heurísticas totalmente independientes, estas pueden ser resueltas en paralelo por distintos procesadores de forma muy natural y simple.

- ***Alto Tiempo de ejecución.***

Las experiencias realizadas hasta el momento muestran que los tiempos de SN son superiores a los de otras metaheurísticas y que esto se incrementa al crecer el tamaño de la entrada.

4. Caso de Estudio: Problema de Steiner en Grafos

Es necesario implementar el método general en un caso de estudio real, donde puedan hacerse mejoras y analizar su comportamiento. Para esto se opta por el Problema de Steiner en Grafos.

4.1 Definición del Problema

Sean:

$$\begin{cases} G = (V, E) \text{ grafo conexo} \\ c{:}E \to \Re^{+} \text{ función de costo} \\ T \subseteq V \text{ nodos terminales} \end{cases}$$

Problema:

$$\text{Hallar } G^1 = (V^1, E^1) \text{ subgrafo conexo de } G \text{ con } T \subseteq V^1$$
$$\text{con } (V^1 \subseteq V, E^1 \subseteq E) \text{ tal que minimice } \sum_{e \in E^1} c(e)$$

Para fijar idea puede verse la siguiente instancia del problema y su solución como ejemplo, donde los terminales se dibujan en rojo:

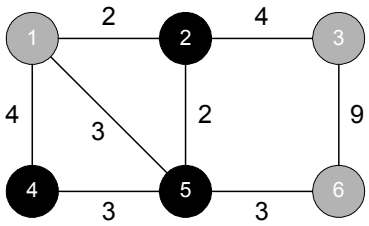

Figura 5. Problema de Steiner (Terminales = grises)

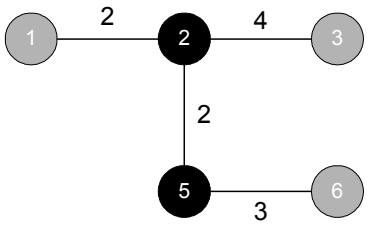

Figura 6. Árbol Óptimo de Steiner

4.2 Pseudocódigo SN Básico para STP

No existen diferencias importantes entre este Pseudocódigo y el presentado para el método general, en las secciones siguientes respectivas se definen la transformación *T* y la heurística constructiva *H*.
Mas adelante se especifican mejoras necesarias para que el algoritmo sea competitivo, redefiniendo partes de este Pseudocódigo.

```
solActual = ∞
Mientras #T < #V hacer
  Para todo i / i ∈ V \ T hacer
    Elegir t ∈ T al azar
    T' = T U { i }
    ResSi = SPH( G, T', t )
    Si ResSi < solActual entonces solActual = ResSi
    G' = G - { i }
    ResNo = SPH( G', T, t )
    Si ResNo < solActual entonces solActual = ResNo
    Si ResSi < ResNo entonces s_i = SÍ sino s_i = NO
    gc_i = Abs(ResSi – ResNo)
  Fin Para
  ganador = arg max_{j∈V\T} {gc_j}
  Si s_ganador = SÍ entonces T = T U { j } sino G = G - { j }
Fin Mientras
Resultado = solActual
```

Pseudocódigo 2. SN Básico para STP.

4.3 Subproblemas

La decisión de alto nivel por la cual se optó es si un nodo no terminal pertenece al árbol óptimo de Steiner, o no. Por tanto de forma muy similar al problema RME, cada decisión divide el espacio de soluciones en dos.

Este tipo de decisión genera instancias de subproblemas críticos, donde la no pertenencia de un nodo al árbol de Steiner provoca la generación de problemas con grafos desconexos y por tanto de soluciones no factibles para el problema original. Inicialmente se eliminan algunos de estos casos mediante reducción y durante el proceso se chequea para cada caso, si se está en esta situación se le asigna costo infinito de forma tal de asegurar grado de confianza infinitos y que estas decisiones sean tomadas primero (es decir se opta por no son discriminarlos).

La decisión no genera subproblemas polinomiales, es decir todos se resuelven heuristicamente con el mismo método (excepto el caso base que podría ser resuelto de forma exacta).

El caso base se alcanza cuando todos los nodos del grafo son terminales (*#T=#V*), en este caso se puede usar el resultado de la metaheurística en ese momento o algún método exacto como Prim [8] o Kruskal [6].

4.4 Transformación del Problema

La transformación T es:

T(Q, n, SÍ) - Si el nodo no terminal *n* pertenece a un árbol de Steiner óptimo, entonces el grafo se deja igual pero se agrega el nodo *n* al conjunto de terminales.

T(Q, n, NO) - Si el nodo no terminal *n* no pertenece a un árbol de Steiner óptimo, entonces el nodo es borrado del grafo.

Pueden probarse formalmente las dos propiedades imprescindibles. El acercamiento al caso base puede verse claramente, puesto que luego de cada decisión disminuye en uno la diferencia entre *V* y *T*.

La no perdida del óptimo puede demostrarse por absurdo, no se incluye la prueba por ser poco ilustrativa.

4.5 Heurística Constructiva

Se estudian dos heurísticas constructivas.

En la primera experiencia llevada a cabo por Urrutia [1] se utiliza: SPH, Steiner Problem Heuristic, con mejora de Rayward-Smith y Clare [9]. El tiempo computacional de la metaheurística es del orden de :

$$(n-t)^2 n^2 t \qquad \text{con } C_H = n^2 t \text{ y } t = \#T$$

Con respecto a la calidad, esta heurística asegura que el resultado obtenido nunca es superior al doble del valor óptimo (menor a 2*(1-(1/t)) * óptimo).

Además de los resultados de Urrutia, en este trabajo también se realizan experiencias simples con la heurística: DPP, Dijkstra Plus Prune. Esta heurística a partir de un nodo terminal aleatorio realiza un árbol de cubrimiento, eliminando iterativamente todas las hojas no terminales.

4.6 Mejoras al Algoritmo Básico

Existen una serie de mejoras al algoritmo básico. Principalmente están orientadas a obtener tiempos competitivos en comparación con otras metaheurísticas.
Es posible clasificar estas mejoras como generales y particulares, donde las generales reflejan aquellas que no dependen del problema particular que se quiere resolver (las particulares hay que adaptarlas para cada tipo de problema, en este caso se presentan para el STP).

- ***Generales***

La mejora mas importante, puesto que reduce sustancialmente el tiempo total es la ***Reutilización de la Información***. La idea básica es mantener una base de datos con todas las soluciones encontradas hasta el momento, reutilizando las corridas anteriores de la heurística constructiva, evitando ejecuciones innecesarias.
En cada iteración, cada vez que se va a evaluar heuristicamente alguna de las dos opciones de un subproblema se consulta la base de datos por si existe alguna solución válida; de existir se utiliza la mejor solución guardada, en caso contrario se ejecuta la heurística constructiva y se archiva. En general, antes de la primera iteración, se ejecuta sobre el problema original ***Y*** veces la heurística constructiva con distintos parámetros, de forma de llenar la base de datos inteligentemente.

Otra mejora importante es la ***Evaluación Calificada***. El algoritmo básico evalúa en cada iteración todas las decisiones posibles que quedan por tomar. Ahora si en un paso iterativo se utiliza la información de confiabilidad arrojada en pasos anteriores se podrían evaluar solo las ***X*** decisiones que se preasumen más confiables en lugar de todas las que restan. Este análisis de sólo las decisiones más confiables, puede ser realizado a partir de la segunda iteración si se ordena según la confiabilidad del paso anterior (para la primera iteración se presenta una solución en la sección de mejoras particulares).

- ***Particulares de STP***

En primera instancia se tiene la ***Reducción*** del problema (claramente las propiedades que permiten reducir dependen de cada problema). A pesar de que la estructura de SN permite hacer reducciones del problema en cada iteración, la experiencia muestra que son verdaderamente útiles al comienzo del algoritmo y es allí donde se ejecutan exclusivamente. En este trabajo se redujo el problema usando iterativamente dos propiedades simples, mientras que en Urrutia [1] se plantean cuatro propiedades de reducción (se recomienda leer ese estudio por mas detalles).

La evaluación calificada permite para las iteraciones posteriores a la primera, no revisar todas las decisiones posibles sino aquellas mas probables de ser tomadas. Pero la primera decisión es la más costosa porque implica ejecutar dos veces la heurística constructiva para todas las decisiones posibles. Una ***Primera Ordenación*** según alguna heurística que determine aproximadamente un grado de confianza permite disminuir esta primera costosa iteración. La propuesta en este caso es ordenar los nodos no terminales según su distancia a un nodo terminal al azar, suponiendo que nodos muy alejados posiblemente no pertenezcan a la solución y por eso tengan un alto grado de confianza en no pertenecer a la solución. De esta forma, en la primera iteración no se analizan todas las decisiones, sino sólo las ***X*** primeras (al costo de haber ejecutado un Dijkstra sobre el problema inicial).

La reutilización de la Información permite reducir los cálculos del algoritmo sustancialmente, pero luego de varias transformaciones del problema los resultados empieza a no tener total validez y es necesario ***Recalcular la Información***. La propuesta mas simple es que aleatoriamente luego de cada iteración se elija con determinada probabilidad ***p*** una solución para ser recalculada.

4.7 Pseudocódigo SN Mejorado para STP

En esta sección se incorporan las mejoras al algoritmo, citadas anteriormente.

```
sol_actual = ∞
Q=Reducir(Q)
Elegir Y nodos t diferentes/ t ∈ T
Para todo nodo t elegido
   res = SPH(G, T, t)
   Guardar en BD (res, t, nodos que pertenecen a la solución)
Fin Para
Ordenar en Confiables todos los nodos no terminales según sus distancias a algún
nodo terminal.
```

```
Mientras #T < #V hacer
   Para todo i perteneciente a los primeros X nodos de Confiables hacer
       Si existe en BD alguna solución a la cual i pertenezca entonces
              ResSi = Mejor solución válida encontrada
       Si existe en BD alguna solución a la cual i no pertenezca entonces
              ResNo = Mejor solución válida encontrada
       Si no existe en BD alguna solución a la cual i pertenezca entonces
              t = terminal que figura en la solución de la BD para ResNo
              T' = T U { i }
              ResSi = SPH( G, T', t )
              Si ResSi < sol_actual entonces sol_actual = ResSi
              Guardar en BD (ResSi, t, nodos que pertenecen a la sol.)
       Fin Si
       Si no existe en BD alguna solución a la cual i no pertenezca entonces
              t = terminal que figura en la solución de la BD para ResSi
              G' = G - { i }
              ResNo = SPH( G', T, t )
              Si ResNo < sol_actual entonces sol_actual = ResNo
              Guardar en BD (ResNo, t, nodos que pertenecen a la sol.)
       Fin Si
       Si ResSi < ResNo entonces s_i = SÍ sino s_i = NO
       gc_i = Abs(ResSi – ResNo)
   Fin Para
   ganador = arg max_{j∈V\T} {gc_j}
   Si s_ganador = SÍ entonces T = T U { i } sino G = G - { i }
   Con probabilidad p hacer
       Elegir t al azar / t ∈ T
              res = SPH(G, T, t)
       Guardar en BD (res, t, nodos que pertenecen a la sol.)
   Fin Hacer
Fin Mientras
Resultado = sol_actual
```

Pseudocódigo 3. SN Mejorado para STP.

5. Trabajo Realizado y Resultados

Actualmente la única fuente de resultados conocida es [1]. Como complemento se tiene el trabajo sin publicar [3] de Sabiguero. Este trabajo aporta una serie de experimentos con el fin de validar y complementar otros aspectos.

5.1 Experimentos y Resultados Previos

- ***Urrutia, S. [1]:***

Se estudia la implementación mejorada del algoritmo (utilizando la heurística constructiva SPH). Los algoritmos son programados en C++ y compilados en Sun UltraSPARC 1 140. Los problemas de testeo utilizados son las series C y D de la biblioteca OR-Library, Beasley [2].

Como resultados importantes se debe destacar que:
Se utiliza una variación de las reducciones propuestas en GA[4], logrando mejores resultados, especialmente para problemas densos de muchos nodos (los cuales son los mas problemáticos). Por supuesto que esto se logra gracias a un aumento pequeño en el tiempo de procesamiento. Es importante decir que los problemas se reducen mucho, y se considera siempre útil utilizar una de estas técnicas. Para situar ideas las reducciones de la clase D que son en promedio aproximadamente : #V= 1000, #E=8000, #T=200 se reducen en #V= 650, #E=4000, #T= 90 utilizando GA [4] y #V=650 , #E=3500, #T= 70 utilizando SN[1].
La metaheurística cumple su rol de mejorar sustancialmente los resultados obtenidos por la heurística SPH, pasando en promedio de un error del 3% a uno del 1%, al utilizar SN.
Quizás el resultado mas importante es el que surge de la comparación con Tabu Search[10]. Claramente SN es mas lento (en promedio 131 segundos en comparación a 23 segundos del Tabu Search). Y esta diferencia aumenta a medida que el problema crece de tamaño. A su vez los errores relativos son sensiblemente menores en SN, con un promedio de 0.44, en comparación a 0.62 del Tabu Search.

- ***Sabiguero, A. [3]:***

Realiza la implementación básica del algoritmo (utilizando heurística DPP). Se programa en Java, interpretado en varias maquinas Intel (500 Mhz aprox.) en paralelo. Por un tema de tiempos de ejecución se utiliza para el testeo la serie B de la biblioteca OR-Library, Beasley [2].

En este caso los experimentos son orientados a modelar el paralelismo y no se analizan mayores resultados con respecto a SN.

5.2 Experimentos y Exploraciones Propias

Utilizando la implementación base de Sabiguero realizada en Java, se interpreta en Intel (PIII 300 MHz).
Se implementa la versión mejorada del algoritmo SN y se compara con los resultados arrojados por la SN Básica. A su vez se opta por no implementar la heurística constructiva SPH, y utilizar la DPP para evaluar el impacto que tiene la calidad de la heurística en el desempeño general.
Al igual que Sabiguero se utiliza la serie B de la biblioteca OR-Library, Beasley [2].

No es parte del alcance de este trabajo hacer un estudio detallado basado en gran cantidad de experimentos y corridas, sino simplemente captar la esencia de la metaheurística y observar su comportamiento en general. En este sentido se ensayó sólo una vez cada uno de los 18 problemas de la clase B, y en particular, dado el alto tiempo del algoritmo SN Básico no se realizó la prueba para los problemas b14-b18.

La primera observación importante es la diferencia en performance entre SN Básico y Mejorado. Sin notarse grandes diferencias entre el error relativo de ambos algoritmo, el algoritmo Básico presenta una muy mala respuesta de tiempos: con un tiempo promedio de 140 minutos en comparación con 1 minuto para SN Mejorada.

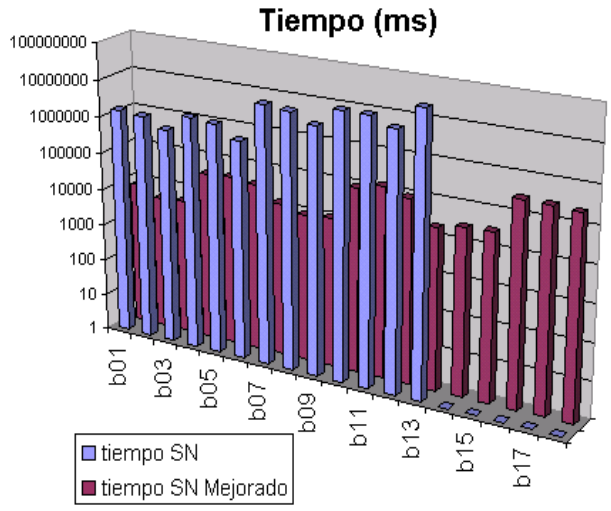

Figura 7. Tiempo (ms) en escala logarítmica, para las dos versiones de metaheurística según la instancia de problema.

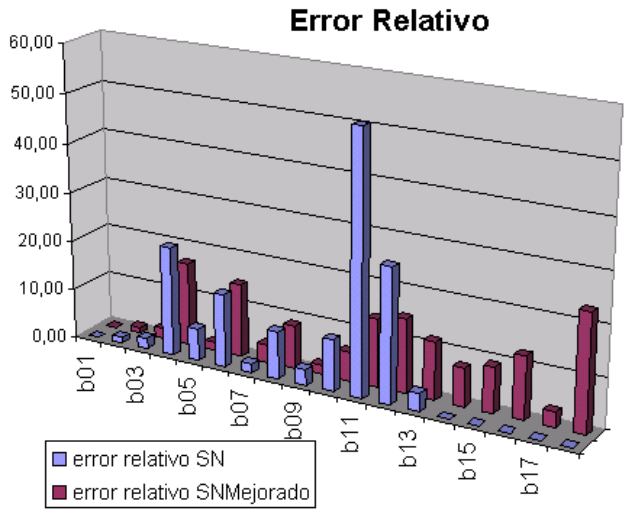

Figura 8. Error Relativo (diferencia de óptimo / óptimo) para las dos versiones de metaheurística según la instancia de problema.

Considerando que SN Mejorada tiene problemas de velocidad al compararla con otras metaheurísticas conocidas, este resultado descarta completamente la posibilidad de utilizar de forma practica SN Básica, teniendo que pensarse en imprescindibles las mejoras presentadas en la sección de STP.

Ahora, intentando evidenciar porque son tan distintos los tiempos entre ambas propuestas, se grafica en las figuras 9 y 10 el tiempo (ordenadas) que le lleva a cada metaheurística tomar cada decisión (abscisas) para todos los problemas analizados.
La tendencia en la toma de decisión, muestra que en la metaheurística mejorada cada decisión es tomada mucho mas rápida que la anterior, mientras que para la metaheurística básica esto es bastante mas lento. Esto se debe a que cada vez es mas probable utilizar datos almacenados en la base de datos y no tener

que calcular la heurística, evidenciándose que la reutilización de información es la principal mejora para el desempeño del algoritmo mejorado.
También puede verse que el paso inicial es mas costoso en promedio para la metaheurística mejorada, porque es en ese paso donde se hace un relleno inicial de la base de datos.

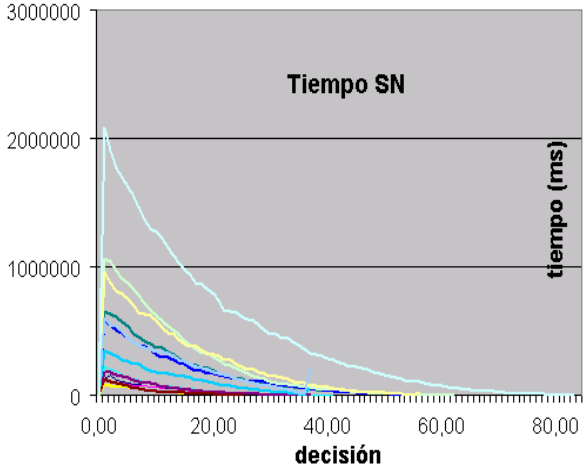

Figura 9. Tiempo (ms.) para cada iteración de decisión. Aplicación de SN Básico para todos los problemas.

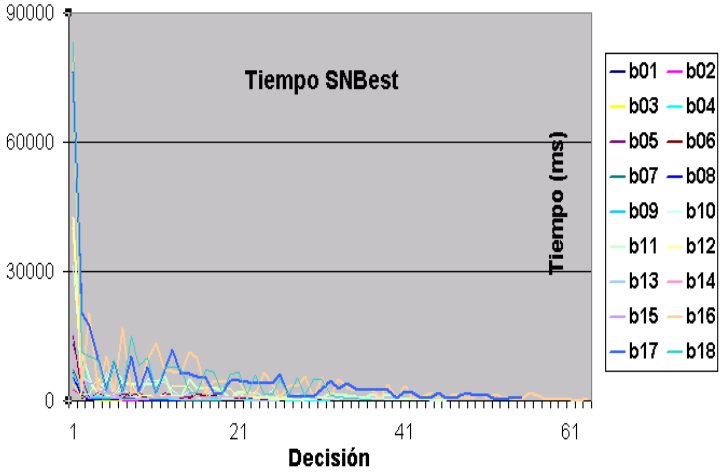

Figura 10. Tiempo (ms.) para cada iteración de decisión. Aplicación de SN Mejorado para todos los problemas.

Por ultimo existe una diferencia muy importante entre los errores relativos obtenidos en [1] (menores al 1%) y los presentados en este trabajo (aproximadamente 10%). Esto puede deberse a que se utilizaron distintas heurísticas constructivas: SPH y DPP, especulando que el desempeño del algoritmo es muy dependiente de la calidad de la heurística constructiva, cosa que se había considerado de poca importancia en todos los trabajos (inclusive este).

Finalmente se incluyen una tabla con los experimentos realizados.

Problema					SN							SN Mejorado						
Name	V	E	T	Opt	V	E	T	Sol	mseg.	dpp	error	V	E	T	Sol	mseg.	dpp	error
b01	50	63	9	**82**	50	63	9	**82**	1514988	1365	0,00	16	28	8	**82**	8108	15	0,00
b02	50	63	13	**83**	50	63	13	**84**	1374383	1223	1,20	19	31	11	**84**	4531	13	1,20
b03	50	63	25	**138**	50	63	25	**141**	745572	565	2,17	22	34	16	**141**	5136	10	2,17
b04	50	100	9	**59**	50	100	9	**72**	2151081	1591	22,03	40	90	9	**69**	38721	20	16,95
b05	50	100	13	**61**	50	100	13	**65**	1827256	1297	6,56	43	96	13	**62**	42191	34	1,64
b06	50	100	25	**122**	50	100	25	**140**	876671	597	14,75	46	95	25	**140**	35851	19	14,75
b07	75	94	13	**111**	75	94	13	**113**	9724783	3189	1,80	30	48	12	**115**	14642	18	3,60
b08	75	94	19	**104**	75	94	19	**114**	8667266	2754	9,62	29	46	16	**113**	9444	13	8,65
b09	75	94	38	**220**	75	94	38	**227**	4647917	1197	3,18	34	53	29	**224**	10765	13	1,82
b10	75	150	13	**86**	75	150	13	**95**	15455302	3553	10,47	57	132	13	**91**	95701	42	5,81
b11	75	150	19	**88**	75	150	19	**134**	14945665	3056	52,27	64	138	19	**100**	137043	40	13,64
b12	75	150	38	**174**	75	150	38	**221**	8603826	1382	27,01	65	138	37	**200**	93539	16	14,94
b13	100	125	17	**165**	100	125	17	**171**	39199586	5676	3,64	39	62	14	**184**	22037	17	11,52
b14	100	125	25	**235**	100	125	25	-	-			42	67	21	**253**	30117	23	7,66
b15	100	125	50	**318**	100	125	50	-	-			51	76	41	**347**	32859	14	9,12
b16	100	200	17	**127**	100	200	17	-	-			80	180	17	**143**	298091	53	12,60
b17	100	200	25	**131**	100	200	25	-	-			79	179	24	**135**	285787	50	3,05
b18	100	200	50	**218**	100	200	50	-	-			83	183	46	**269**	255379	26	23,39

Tabla 1. Resultados Comparativos SN Básica y Mejorada. Intel PIII 300Mhz, Linux Redhat 7.2, VM de www.sun.com.

6. Conclusiones

Este trabajo resume las principales características de una nueva metaheurística recientemente propuesta en [1].

Se detecta que hay problemas que se adaptan más a SN, estos son aquellos basados en decisiones tipo asignación, para los problemas de ordenamiento no parece ser tan natural el método.

Existe un fuerte vínculo entre la heurística constructiva y el tipo de subproblema de decisión. Según la forma en que se transforma el subespacio de soluciones según la decisión, una heurística puede ser mas competente que otra.

Con respecto a Java es de notar que permite rápido desarrollo, pero los tiempos de ejecución son prohibitivos.

Claramente existen muchas cuestiones abiertas y a ser mejoradas en la metaheurística. Esto se discute con mas cuidado en la sección de trabajo a futuro.

7. Trabajo a Futuro

7.1 Líneas de Trabajo Inmediato

- Hacer un modelo matemático simple que exprese el comportamiento del algoritmo, de forma tal de formalizar la idea del método.
- No se ha analizado de forma profunda la aplicación de este método a otros problemas. Su utilización parece muy natural para problemas de asignación o elección, no quedando claro como se implementaría para problemas de ordenamiento.
- Explorar otros grados de confianza, comparando resultados estadísticos. Por ejemplo agregar varianza, agregar ponderación del valor de la solución (no solo de la diferencia).
- Cuantificar la utilidad (tiempo versus error relativo) de realizar técnicas de reducción durante la ejecución del algoritmo.
- Definición de las constantes del algoritmo mejorado: X,Y y p, determinar un criterio de calibración según la instancia del problema. Estas constantes podrían ser dinámica en la ejecución. Es posible mejorar el criterio de recalculo p basándose en las soluciones ya calculadas.
- Ensayar un método que analice si hubo decisiones previas erróneas, y de ser así implementar una vuelta atrás en decisiones. Puntos clave son asegurar finalización del algoritmo y evitar reelección de decisiones. Una opción es considerar por ejemplo las 3 opciones con mayor grado de confianza en lugar de solo la primera; explorarlas como decisiones tomadas y luego de una cantidad de pasos de decisión tomados configurable descartar las peores cuando se tenga mas seguridad.
- Como se podría trabajar en base a varios tipos de decisiones en paralelo y utilizar esos resultados para elegir la simplificación de forma mas exitosa.

8. Bibliografía

[1] Urrutia, S., Loiseau I. A new metaheuristic and its application to the Steiner Problems in graph. Tesis de Licenciatura, FCEyN, Universidad de Buenos Aires, 2001.

[2] Beasley, J. E. OR-Library: Distributing test problems by electronic mail, Journal of the Operational Research Society 41 (1990), 1069 - 1072.

[3] Sabiguero, A. Modelado de clusters de PCs para computación paralela. Tesis de Maestría, InCo, UDELAR, reporte tecnico, 2002.

[4] Esbensen, H. Computing near-optimal solutions to the Steiner problems in a graph using a genetic algorithm, Networks, 26 (1995), 173-185.

[5] Garey M. R., Johnson D. S. Computers and intractability. A guide to the theory of NP-Completeness, Bell Laboratories, Murray, New Jersey. (1979)

[6] Kruskal Jr., J. B. On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Amer. Math. Soc., 7 (1956), 48-50.

[7] Osman, I. H. An introduction to Meta-heuristics, Operational Research Tutorial Papers Series, Annal Conference OR37 - Canterbury (1995).

[8] Prim, R. C. Shortest connection networks and some generalizations, Bell System Tech. J., 36 (1957), 1389-1401.

[9] Rayward-Smith, V. J., Clare, A. On finding Steiner vertices, Networks 16 (1986), 283-294.

[10] Ribeiro, C. C., De Souza, M. C. Improved tabu search for the Steiner porblem in graphs, Working paper, Catholic University of Rio de Janeiro, Department of Computer Science, (1998).