GRASP: Greedy Randomized Adaptive Search Procedures

Mauricio G.C. Resende (1), José Luis González Velarde (2)

(1) Algorithms and Optimization Research Department
AT\& T Labs-Research
180 Park Avenue. Florham Park, NJ 07732 USA

(2) Tecnológico de Monterrey
Garza Sada 2501
Monterrey, NL 64849 México

e-mail: mgcr@research.att.com, gonzalez.velarde@itesm.mx

One of a number of successful metaheuristics appearing in the late years of the last century is GRASP, a multi-start method designed to solve hard combinatorial optimization problems. In its basic version, each iteration consists of two phases: a constructive phase whose product is a good but not necessarily locally optimal solution, and a local search procedure, during which, neighborhoods of the solution are examined until a local optimum is attained. The iterations proceed, keeping the best found solution, until a stopping criterion is reached. In this chapter, the basic components of GRASP are reviewed, along with some parameter setting strategies like reactive GRASP. A discussion of the use of path relinking as a method to introduce memory features in the local search phase follows. The chapter concludes with a description of parallel GRASP, with extensive experimentation to compare independent and cooperative implementation.

Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.19 (2003),pp. 61-76
ISSN: 1137-3601. © AEPIA (http://www.aepia.org/revista).

GRASP: Procedimientos de búsqueda miopes aleatorizados y adaptativos

Mauricio G. C. Resende
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07732 USA
mgcr@research.att.com

Jośe Luis González Velarde
Tecnológico de Monterrey
Garza Sada 2501
Monterrey, NL 64849 Mexico
gonzalez.velarde@itesm.mx

Resumen

Una entre las metaheurísticas más exitosas que aparecieron en los últimos aos del siglo pasado es GRASP, un método multi-arranque diseñado para resolver problemas difíciles en optimización combinatoria. En su versión básica cada iteración consiste en dos fases: una fase constructiva cuyo producto es una solución factible buena, aunque no necesariamente un óptimo local, y una búsqueda local, durante la cual se examinan vecindades de la solución, al llegar a un óptimo local la iteración termina. La iteraciones continúan, guardando la mejor solución encontrada en cada una de ellas, hasta que se alcanza un criterio de terminación. En este capítulo se revisan las componentes básicas de GRASP, junto con algunas estrategias para el ajuste de parámetros como es GRASP reactivo. De aquí sigue una discusión acerca del uso de reencadenamiento de trayectorias como una forma de introducir aspectos de memoria dentro de la búsqueda local. El capítulo concluye con una descripción de GRASP en paralelo, con una extensa experimentación para comparar implementaciones independientes contra cooperativas.

1. Introducción

Consideremos un problema de optimización combinatoria definido por un conjunto base finito $E = \{1, \ldots, n\}$, un conjunto de soluciones factibles $F \subseteq 2^E$, y una función objetivo $f : 2^E \rightarrow \mathbb{R}$. En la versión de minimización, buscamos una solución óptima $S^* \in F$ tal que $f(S^*) \leq f(S)$, $\forall S \in F$. El conjunto base E, la función de costo f, así como el conjunto de soluciones factibles F se definen para cada problema específico. Por ejemplo, en el caso del problema del agente viajero, el conjunto base E consta de todas las aristas que conectan las ciudades a ser visitadas, $f(S)$ es la suma de los costos de todas las aristas $e \in S$, y F está formado por todos los subconjuntos de aristas que determinan un ciclo Hamiltoniano.

Un procedimiento de búsqueda miope aleatorizado y adaptativo (GRASP por sus siglas en inglés) [46, 47] es una metaheurística para encontrar soluciones aproximadas (i.e. sub-óptimas de buena calidad, pero no necesariamente óptimas) a problemas de optimización combinatoria. Se basa en la premisa de que soluciones iniciales diversas y de buena calidad juegan un papel importante en el éxito de métodos locales de búsqueda.

Un GRASP es un método multi-arranque, en el cual cada iteración GRASP consiste en la construcción de una solución miope aleatorizada seguida de una búsqueda local usando la solución construida como el punto inicial de la búsqueda local. Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones GRASP se devuelve como la solución aproximada. El seudo-código

```
procedure GRASP

Require: i_máx
  f* ← ∞;
  for i ≤ i_máx do
    x ← GreedyRandomized();
    x ← LocalSearch(x);
    if f(x) < f* then
      f* ← f(x);
      x* ← x;
    end if
  end for
  return x*;
```

Figura 1: Seudo-código GRASP

en la Figura 1 ilustra un GRASP básico para minimización.

En este artículo, nos centraremos primero en las dos componentes más importantes de GRASP. A saber, construcción y búsqueda local. Después examinaremos cómo el reencadenamiento de trayectorias puede ser usado en GRASP como un mecanismo de memoria e intensificación. En seguida se discute GRASP paralelo. El trabajo termina con una lista parcial de aplicaciones exitosas de GRASP.

Reseñas recientes de GRASP pueden encontrarse en [99, 90], una extensa bibliografía comentada está en [54] y una actualización en el URL `http://graspheuristic.org/annotated`.

2. Construcciones GRASP

En esta sección describimos varios mecanismos de construcciones miopes aleatorizadas. Estos procedimientos mezclan miopía con aleatorización de diferentes formas.

Todos los mecanismos de construcción considerados construyen una solución incorporando un elemento a la vez. En cada paso del proceso de construcción, se tiene a la mano una solución parcial. Un elemento que pueda seleccionarse como parte de una solución parcial se llama elemento *candidato*. Consideremos un problema de cubrimiento de conjuntos, donde se tiene una matriz $A = [a_{ij}]$ de ceros y unos, un costo c_j

```
procedure Construcción-C

Require: k, E, c(·);
  x ← ∅;
  C ← E;
  Calcular costo miope c(e), ∀ e ∈ C;
  while C ≠ ∅ do
    RCL ← {k elementos e ∈ C con el
    menor c(e)};
    Seleccionar un elemento s de RCL al
    azar;
    x ← x ∪ {s};
    Actualizar el conjunto candidato C;
    Calcular el costo miope c(e), ∀ e ∈ C;
  end while
  return x;
```

Figura 2: Seudo-código de construcción GRASP: RCL basada en cardinalidad

para cada columna j, y se quiere determinar un conjunto J de columnas con el menor costo total $\sum_{j \in J} c_j$ tal que para cada renglón i, al menos una columna j del conjunto tenga una entrada $a_{ij} = 1$. En este problema, una solución parcial es un conjunto de columnas que no necesariamente forman un cubrimiento. Cualquier columna que no se haya seleccionado previamente es un elemento candidato. El conjunto solución J se construye incorporando un elemento (columna) a la vez hasta que el conjunto J sea un cubrimiento.

Para determinar qué elemento candidato seleccionar enseguida para incluirse en la solución, generalmente se hace uso de una función miope. Una función miope mide la contribución local de cada elemento a la solución parcial. En el caso del cubrimiento de conjuntos, una función miope plausible es la razón p_j/c_j entre el número p_j de filas sin cubrir, que quedarían cubiertas si la columna j se elige y la contribución c_j al costo total de elegir la columna j para la solución. La elección miope sería agregar la columna con el mejor valor de la función miope.

Existen varias formas posibles de introducir aleatoridad a este procedimiento. Una de las primeras ideas fue el uso de una lista restringida de candidatos (RCL) [46]. Tal lista contiene un conjunto de elementos candidatos con los mejores valores de la función miope. El siguiente candidato a ser agregado a la solución

procedure `Construcción-V`

Require: α, E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Calcular costo miope $c(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    $c_* \leftarrow \text{mín}\{c(e) \mid e \in C\}$;
    $c^* \leftarrow \text{máx}\{c(e) \mid e \in C\}$;
    RCL $\leftarrow \{e \in C \mid c(e) \leq c_* + \alpha(c^* - c_*)\}$;
    Seleccionar un elemento s de RCL al azar;
    $x \leftarrow x \cup \{s\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo miope $c(e)$, $\forall\, e \in C$;
  end while
  return x;

Figura 3: Seudo-código de construcción GRASP: RCL basada en valor

se selecciona al azar de la lista restringida de candidatos. Dicha lista puede consistir de un número fijo de elementos (restricción por cardinalidad) o elementos con los valores de la funcción miope dentro de un rango dado. La Figura 2 muestra un seudo-código para un procedimiento de construcción GRASP basado en restricción por cardinalidad. Por ejemplo, denotemos por c^* y c_* , respectivamente, los valores mayor y menor de la función miope para los elementos candidatos, y sea α un número real tal que $0 \leq \alpha \leq 1$. En una lista restringida de candidatos basada en el valor, la RCL consiste en todos los elementos candidatos e cuyo valor de función miope $c(e)$ es tal que $c(e) \leq c_* + \alpha(c^* - c_*)$. Nótese que si $\alpha = 0$, entonces este esquema de selección es un algoritmo miope, mientras que si $\alpha = 1$, entonces es totalmente aleatorio. La Figura 3 muestra un seudo-código para un procedimiento de construcción GRASP basado en el valor. Más tarde será discutida la forma de determinar valores para α.

Se puede también mezclar una construcción al azar con una construcción miope de la siguiente manera. Elegir secuencialmente un conjunto parcial de elementos candidato al azar y después completar la solución usando un algoritmo miope [101]. La Figura 4 muestra un

procedure `Construcción-RG`

Require: k, E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Calcular el costo miope $c(e)$, $\forall\, e \in C$;
  for $i = 1, 2, \ldots, k$ **do**
    if $C \neq \emptyset$ **then**
      Elegir el elemento e al azar de C;
      $x \leftarrow x \cup \{e\}$;
      Actualizar el conjunto de candidatos C;
      Calcular el costo miope $c(e)$, $\forall\, e \in C$;
    end if
  end for
  while $C \neq \emptyset$ **do**
    $e_* \leftarrow \text{argmin}\{c(e) \mid e \in C\}$;
    $x \leftarrow x \cup \{e_*\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo miope $c(e)$, $\forall\, e \in C$;
  end while
  return x;

Figura 4: Seudo-código de construcción GRASP: construcción aleatoria después miope

procedure `Construcción-PG`

Require: E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Perturbar aleatoriamente los datos del problema;
  Calcular el costo miope perturbado $\tilde{c}(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    $e_* \leftarrow \text{argmin}\{\tilde{c}(e) \mid e \in C\}$;
    $x \leftarrow x \cup \{e_*\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo miope perturbado $\tilde{c}(e)$, $\forall\, e \in C$;
  end while
  return x;

Figura 5: Seudo-código de construcción GRASP: construcción con perturbaciones

seudo-código para tal procedimiento de construcción.

Otro enfoque es mediante perturbación de costos. Aquí, los datos de costos se perturban aleatoriamente y se aplica un algoritmo miope [30]. La Figura 5 muestra un seudo-código para este procedimiento de construcción.

Un ejemplo final de un procedimiento de construcción de GRASP es una variación sobre el enfoque de RCL basado en el valor. En este procedimiento, llamado función de sesgo [28], en vez de seleccionar el elemento de la RCL al azar con iguales probabilidades asignadas a cada elemento, se asignan diferentes probabilidades, favoreciendo elementos bien evaluados. Los elementos de la RCL se ordenan de acuerdo a los valores de la función miope. La probabilidad $\pi(r(e))$ de seleccionar el elemento e es

$$\pi(r(e)) = \frac{\texttt{sesgo}(r(e))}{\sum_{e' \in \text{RCL}} \texttt{sesgo}(r(e'))},$$

where $r(e)$ es la posición del elemento e en la RCL. Se han propuesto varias alternativas para asignar sesgos a los elementos. Por ejemplo,

- sesgo aleatorio: $\texttt{sesgo}(r) = 1$;
- sesgo lineal: $\texttt{sesgo}(r) = 1/r$;

procedure `Construcción-B`

Require: α, E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Calcular costo miope $c(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    $c_* \leftarrow \text{mín}\{c(e) \mid e \in C\}$;
    $c^* \leftarrow \text{máx}\{c(e) \mid e \in C\}$;
    RCL $\leftarrow \{e \in C \mid c(e) \leq c_* + \alpha(c^* - c_*)\}$;
    Asignar rango $r(e)$, $\forall\, e \in RCL$;
    Asignar una probabilidad $\pi(r(e))$ de elegir el elemento e en RCL que favorezca a los candidatos con mejores rangos;
    Elegir el elemento s de RCL al azar con probabilidad $\pi(r(s))$;
    $x \leftarrow x \cup \{s\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo miope $c(e)$, $\forall\, e \in C$;
  end while
  return x;

Figura 6: Seudo-código de construcción GRASP: RCL basada en valores con sesgo

```
procedure BúsquedaLocal
Require: x^0, N(·), f(·);
  x ← x^0;
  while x no es localmente óptimo con respecto a N(x) do
    Sea y ∈ N(x) tal que f(y) < f(x);
    x ← y;
  end while
  return x;
```

Figura 7: Suedo-código de búsqueda local

- sesgo exponencial: $\texttt{sesgo}(r) = e^{-r}$.

El seudo-código en la Figura 6 ilustra este procedimiento.

En la siguiente sección, discutimos cómo determinar el valor de α para usarse en los esquemas basados en RCL anteriormente discutidos. Recordemos que si $\alpha = 0$, entonces estos esquemas de selección se vuelven un algoritmo miope, mientras que si $\alpha = 1$, son totalmente aleatorios.

3. Búsqueda local

Un algoritmo de búsqueda local explora repetidamente la vecindad de una solución en busca de una mejor solución. Cuando no se encuentra una solución que mejora la actual, se dice que la solución es localmente óptima. La Figura 7 muestra un seudo-código para un procedimiento de búsqueda local.

La búsqueda local juega un papel importante en GRASP ya que sirve para buscar soluciones localmente óptimas en regiones prometedoras del espacio de soluciones. Esta es la diferenciación de GRASP con respecto del algoritmo semi-miope de Hart y Shogan [59]. Por definición su desempeño nunca será peor que semimiope, y casi siempre producirá mejores soluciones en menos tiempo.

Aunque los algoritmos miopes pueden producir buenas soluciones razonables, su principal desventaja como generador de soluciones iniciales para búsquedas locales es su falta de diversidad. Aplicando repetidamente un algoritmo miope, una sola o muy pocas soluciones pueden generarse. Por otra parte, un algoritmo totalmente aleatorio produce una gran cantidad de soluciones diversas. Sin embargo, la calidad de estas soluciones generalmente es muy pobre y usarlas como soluciones iniciales para búsquedas locales generalmente conduce a una convergencia lenta hacia un mínimo local.

Para beneficiarse de la convergencia rápida del algoritmo miope y de la gran diversidad del algoritmo aleatorio, se acostumbra usar un valor de α estrictamente contenido en el interior del rango $[0, 1]$. Ya que no se conoce a priori qué valor usar, una estrategia razonable es seleccionar al azar un valor diferente en cada iteración GRASP. Esto puede hacerse usando una probabilidad uniforme [97] o usando el esquema de GRASP *reactivo* [91].

En el esquema de GRASP reactivo, sea $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ el conjunto de valores posibles para α. Las probabilidades asociadas con la elección de cada valor se fijan todas inicialmente iguales a $p_i = 1/m,\ i = 1, \ldots, m$. Más aún, sea z^* el valor de la solución incumbente, esto es, la mejor solución encontrada hasta el momento, y sea A_i el valor promedio de todas las soluciones halladas usando $\alpha = \alpha_i,\ i = 1, \ldots, m$. Las probabilidades de selección se reevalúan periodicamente tomando $p_i = q_i / \sum_{j=1}^{m} q_j$, con $q_i = z^*/A_i$ para $i = 1, \ldots, m$. El valor de q_i será mayor para valores de $\alpha = \alpha_i$ que produzcan las mejores soluciones en promedio. Mayores valores de q_i corresponden a valores del parámetro α más adecuados. Las probabilidades asociadas con estos valores más apropiados se incrementarán cuando sean reevaluadas.

En el contexto de GRASP, se han usado esquemas de búsqueda local más elaborados. Por ejemplo, búsqueda tabú [67, 37, 1, 108], recocido simulado [71], vecindades variables [31, 53], y vecindades extendidas [3].

4. Reencadenamiento de trayectorias

Quizás una de las principales desventajas del GRASP puro es su falta de estructuras de memoria. Las iteraciones de GRASP son in-

procedure PR

Require: x_s, x_t;
  Calcular la diferencia simétrica $\Delta(x_s, x_t)$;
  $x \leftarrow x_s$;
  $f^* \leftarrow \text{mín}\{f(x_s), f(x_t)\}$;
  $x^* \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$;
  while $\Delta(x_s, x_t) \neq \emptyset$ **do**
    $m^* \leftarrow \text{argmin}\{f(x \oplus m), \forall m \in \Delta(x, x_t)\}$;
    $\Delta(x \oplus m^*, x_t) \leftarrow \Delta(x, x_t) \setminus \{m^*\}$;
    $x \leftarrow x \oplus m^*$;
    if $f(x) < f^*$ **then**
      $f^* \leftarrow f(x)$;
      $x^* \leftarrow x$;
    end if
  end while
  return x^*;

Figura 8: Seudo-código de reencadenamiento de trayectorias

dependientes y no utilizan las observaciones hechas durante iteraciones previas. Un remedio para esto es el uso del reencadenamiento de trayectorias con GRASP. Reencadenamiento de trayectorias fue propuesto originalmente por Glover [57] como una forma de explorar las trayectorias entre soluciones élite obtenidas por búsqueda tabú o búsqueda dispersa. Usando una o más soluciones élite, se exploran las *trayectorias* en el espacio de soluciones que conducen a otras soluciones élite para buscar mejores soluciones. Para generar trayectorias, los movimientos se seleccionan para introducir atributos en la solución actual que estén presentes en la solución élite guía.

El reencadenamiento de trayectorias en el contexto de GRASP fue introducido por Laguna y Martí [68]. Desde entonces han aparecido numerosas extensiones, mejoras, y aplicaciones exitosas [5, 30, 100, 103, 101, 53]. Ha sido usado como un esquema de intensificación, en el que las soluciones generadas en cada iteración de GRASP se reencadenan con una o más soluciones de un conjunto élite de soluciones, o como en una fase de post-optimización, donde se reencadenan pares de soluciones del conjunto élite.

Consideremos dos soluciones x_s y x_t en las cuales queremos aplicar reencadenamiento de trayectorias desde x_s hacia x_t. La Figura 8

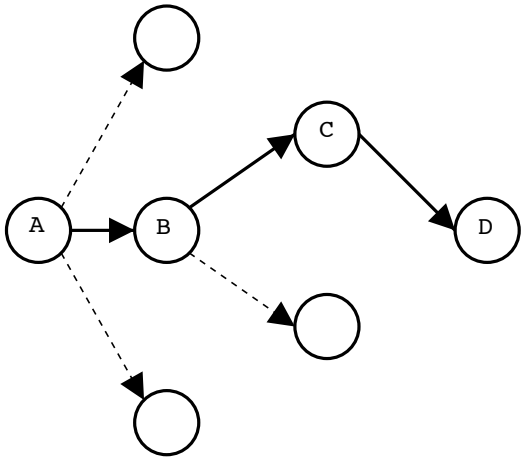

Figura 9: Ejemplo de reencadenamiento de trayectorias.

ilustra el procedimiento de reencadenamiento de trayectorias con su seudo-código. El procedimiento se inicia calculando la diferencia simétrica $\Delta(x_s, x_t)$ entre las dos soluciones, i.e. el conjunto de movimientos necesarios para alcanzar x_t desde x_s. Se genera entonces una *trayectoria* de soluciones encadenando a x_s con x_t. El algoritmo devuelve la mejor solución encontrada en esta trayectoria. En cada paso, el procedimiento examina todos los movimientos $m \in \Delta(x, x_t)$ desde la solución actual x y elige aquel que resulta en la solución menos costosa, i.e. aquel que minimiza $f(x \oplus m)$, donde $x \oplus m$ es la solución resultante de aplicar el movimiento m a la solución x. Se efectúa el mejor movimiento m^* produciendo $x \oplus m^*$ y el movimiento m^* se elimina de la diferencia simétrica de $\Delta(x \oplus m^*, x_t)$. En caso necesario, la mejor solución x^* se actualiza. El procedimiento termina cuando se alcanza x_t, i.e. cuando $\Delta(x, x_t) = \emptyset$.

La figure 9 ilustra el reencadenamiento de trayectorias. En el grafo de esta figura, los nodos corresponden a soluciones, y los arcos corresponden a movimientos que permiten que una solución se alcance a partir de otra. Supóngase que dos soluciones, A y D, se reencadenan. Sea A la solución inicial y D la solución meta, y supóngase que la diferencia simétrica $\Delta(A, D) = 3$. Existen tres posible movimientos partiendo de A. Se elige el mejor movimiento, el cual produce la solución B. En este punto, la diferencia simétrica $\Delta(B, D) = 2$ y por lo tanto existen dos movimientos posibles. De nuevo,

se elige el mejor movimiento, el cual produce la solución C. Finalmente, en este punto hay un solo movimiento posible, el cual conduce a la solución meta D. Este esquema de reencadenamiento de trayectorias produjo una "trayectoria" $A \rightarrow B \rightarrow C \rightarrow D$ la cual puede ahora ser evaluada.

El reencadenamiento de trayectorias mantiene un conjunto P de soluciones élite halladas durante la optimización [55]. Las primeras $|P|$ soluciones distintas que se encuentran se insertan en el conjunto élite. Después de eso, una solución candidato x^* se agrega a P si su costo es menor que el costo de todas las soluciones del conjunto élite, o si su costo es mayor que el mejor, pero menor que la peor solución élite y es suficientemente diferente de todas las soluciones del conjunto élite. Si se acepta su entrada al conjunto élite, la nueva solución reemplaza a la solución más similar a ella entre el conjunto de soluciones élite con un costo peor que ella [101]. El conjunto élite puede ser renovado periódicamente [4] si no se observan cambios en el conjunto élite durante un número especificado de iteraciones GRASP. Una forma de hacer esto es fijar en infinito los valores de la función objetivo de la peor mitad del conjunto élite. De esta forma se crearán nuevas soluciones del conjunto élite.

Se han propuesto varios esquemas alternativos para el reencadenamiento de trayectorias. Ya que este procedimiento puede ser demandante en recursos computacionales, no necesita ser aplicado después de cada iteración GRASP. Es más conveniente hacerlo periódicamente. Usualmente las trayectorias desde x_s hasta x_t y desde x_t hasta x_s son diferentes y ambas pueden explorarse. Ya que las trayectorias pueden ser largas, no es necesario seguir la trayectoria completa. Se puede restringir siguiendo una trayectoria truncada que inicie en x_s y otra que inicie en x_t.

5. GRASP paralelo

La mayoría de las implementaciones en paralelo para GRASP siguen la estrategia de *tramas de ejecución independientes* [112], la cual distribuye las iteraciones sobre los procesadores [4, 8, 9, 48, 70, 76, 78, 82, 87, 88]. En general, cada trama de búsqueda tiene que efectuar $i_{\text{máx}}/p$ iteraciones, donde p es el número de procesadores e $i_{\text{máx}}$ es el número total de iteraciones. Cada procesador almacena una copia del algoritmo secuencial, los datos del problema, y una semilla distinta para generar una serie de números seudo-aleatorios. Una sola variable global se usa para guardar la mejor solución encontrada por todos los procesadores. Uno de los procesadores es el amo, el cual lee y distribuye los datos del problema, genera las semillas usadas por los generadores de números pseudo-aleatorios en cada procesador, distribuye las iteraciones, y recoge la mejor solución encontrada por cada procesador. Ya que las iteraciones son independientes y se intercambia una pequeña cantidad de información, pueden obtenerse aceleraciones lineales fácilmente si el desbalanceo de cargas presente no es muy grande.

Martins et al. [78] implementaron un GRASP paralelo para el problema de Steiner en grafos. La paralelización se logra mediante la distribución de 512 iteraciones entre los procesadores, con un valor del parámetro α de la RCL elegido al azar en el intervalo $[0{,}0, 0{,}3]$ en cada iteración. El algoritmo se implementó en lenguaje C en una máquina IBM SP-2 con 32 procesadores, usando para la comunicación la biblioteca MPI. Se usaron los 60 problemas de las series C, D, y E de la OR-Library [23] en los experimentos computacionales. La implementación en paralelo obtuvo 45 soluciones óptimas sobre las 60 instancias de prueba. La desviación relativa con respecto al valor óptimo nunca fue mayor a 4 %. Se observaron aceleraciones cuasi-lineales para 2, 4, 8, y 16 procesadores con respecto a la implementación secuencial lo cual se ilustra en la Figura 10.

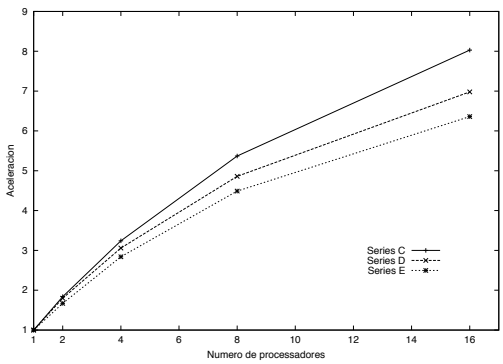

Figura 10: Aceleraciones promedio en 2, 4, 8, y 16 procesadores.

El reencadenamiento de trayectorias se puede usar también en forma conjunta con implementaciones paralelas de GRASP. En el caso de la implementación de las tramas de ejecución independientes descrita por Aiex et al. [5] para el problema de asignación en 3 índices, cada procesador aplica reencadenamiento de trayectorias a parejas de soluciones élite almacenadas en una pila local. Los resultados computationales usando MPI en una computadora SGI Challenge con 28 procesadores R10000 mostraron aceleraciones lineales.

Alvim y Ribeiro [8, 9] mostraron que los enfoques de tramas de ejecución independientes para la paralelización de GRASP se pueden beneficiar de las técnicas de balanceo de cargas cuando se usan procesadores heterogéneos o si la máquina paralela se comparte simultáneamente entre varios usuarios. Se pueden obtener aceleraciones cuasi-lineales con una distribución heterogénea de las iteraciones sobre los p procesadores en $q \geq p$ paquetes. Cada procesador comienza ejecutando un paquete de $\lceil i_{\text{máx}}/q \rceil$ iteraciones e informa al amo cuando ha concluido con su paquete de iteraciones. El amo detiene la ejecución de cada procesador esclavo cuando ya no hay más iteraciones que realizar y recoge la mejor solución encontrada. Los procesadores más veloces o menos cargados realizarán más iteraciones que los otros. En el caso del GRASP paralelo implementado para el problema de asignación de tráfico descrito en [91], este tipo de estrategia de balanceo de carga dinámica permitió reducciones en los tiempos de hasta 15 % con respecto a los tiempos observados para la estrategia estática, en los cuales las iteraciones fueron distribuidas uniformemente sobre los procesadores.

La eficiencia de las implementaciones de tramas de ejecución independientes de metaheurísticas, basadas en ejecutar copias múltiples del mismo algoritmo secuencial, ha sido abordado por varios autores. Un valor meta dado τ para la función objectivo se transmite a todos los procesadores, los cuales ejecutan independientemente el algoritmo secuencial. Todos los procesadores se detienen immediatamente después de que uno de ellos encuentra una solución con un valor al menos tan bueno como τ. La aceleración está dada por la razón entre los tiempos necesarios para encontrar una solución con valor al menos tan bueno como τ, usando respectivamente el algoritmo secuencial y la implementación en paralelo con p procesadores. Es necesario asegurar que ningún par de iteraciones inicien con idénticas semillas para el generador de números aleatorios. Estas aceleraciones son lineales para un número de metaheurísticas, incluyendo recocido simulado [39, 84], algoritmos de búsqueda local iterados para el problema del agente viajero [41], búsqueda tabú (suponiendo que la búsqueda se inicie desde un óptimo local [22, 109]), y WalkSAT [107] sobre problemas aleatorios duros de 3-SAT [61]. Esta observación puede explicarse si la variable aleatoria *tiempo para encontrar una solución al menos tan buena como algún valor meta* se distribuye exponencialmente, como indica la siguiente proposición [112]:

Proposición 1: Sea $P_\rho(t)$ la probabilidad de no haber encontrado una solución con un valor meta dado en t unidades de tiempo con ρ procesos independientes. Si $P_1(t) = e^{-t/\lambda}$ con $\lambda \in \mathbb{R}^+$ (correspondiente a una distribución exponencial), entonces $P_\rho(t) = e^{-\rho t/\lambda}$.

Esta proposición sigue de la definición de la distribución exponencial. Implica que la probabilidad $1-e^{-\rho t/\lambda}$ de encontrar una solución dentro de un valor meta dado en tiempo ρt con un algoritmo secuencial es igual a la probabilidad de encontrar una solución al menos tan buena como aquella en tiempo t usando ρ procesadores paralelos independientes. De aquí que, es posible, en promedio, lograr aceleraciones lineales en el tiempo para encontrar una solución dentro de un valor meta mediante múltiples procesadores independientes. Una proposición análoga puede establecerse para una distribución exponencial biparamétrica (desplazada) [6]:

Proposición 2: Sea $P_\rho(t)$ la probabilidad de no haber encontrado una solución con un valor meta dado en t unidades de tiempo con ρ procesadores independientes. Si $P_1(t) = e^{-(t-\mu)/\lambda}$ con $\lambda \in \mathbb{R}^+$ y $\mu \in \mathbb{R}^+$ (correspondientes a una distribución exponencial biparamétrica), entonces $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Análogamente, esta proposición es consecuencia de la definición de la distribución exponencial biparamétrica. Implica que la probabilidad de encontrar una solución dentro de un valor meta dado en tiempo ρt con un algoritmo secuencial es igual a $1 - e^{-(\rho t-\mu)/\lambda}$, mientras que la probabilidad de encontrar una solución al menos tan buena como aquella en tiempo t us-

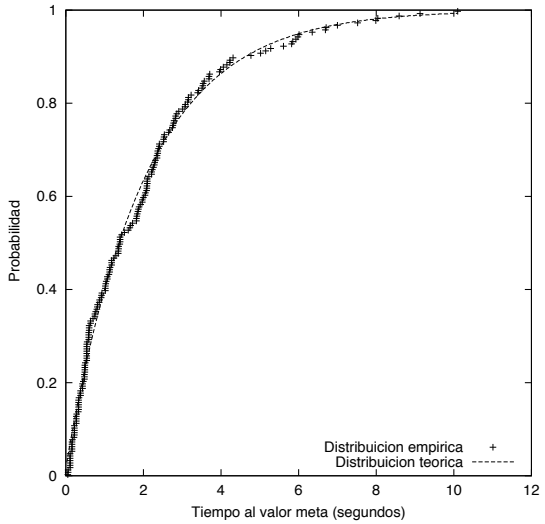

Figura 11: Distribuciones empírica y teórica superimpuestas (tiempos hasta valores meta medidos en segundos en una computadora SGI Challenge con 28 procesadores).

ando ρ procesadores paralelos independientes es $1 - e^{-\rho(t-\mu)/\lambda}$. Si $\mu = 0$, entonces ambas probabilidades son iguales y corresponden a la distribución exponencial sin desplazamiento. Más aún, si $\rho\mu \ll \lambda$, entonces las dos probabilidades son aproximadamente iguales y es posible alcanzar aproximadamente aceleraciones lineales en el tiempo para encontrar una solución dentro de un valor meta usando múltiples procesadores independientes.

Aiex et al. [6] han mostrado experimentalmente que los tiempos de solución para GRASP también tienen esta propiedad, demostrando que se ajustan a una distribución exponencial biparamétrica. La Figura 11 ilustra este resultado, desplegando superpuestas las distribuciones empírica y teórica observadas para uno de los casos estudiados durante los experimentos computacionales reportados por los autores, los cuales involucraron 2400 ejecuciones de GRASP para cada uno de cinco problemas diferentes: máximo conjunto independiente [48, 94], asignación cuadrática [70, 95], planarización de grafos [98, 102], máxima satisfactibilidad con peso [97], y cubrimiento máximo [92]. El mismo resultado sigue siendo cierto cuando GRASP se implementa conjuntamente con un procedimiento de reencadenamiento de trayectorias como post-optimizador [5, 4].

En el caso de estrategias de *tramas de ejecución cooperativas*, las tramas de búsqueda que se ejecutan en paralelo intercambian y comparten la información recabada a lo largo de las trayectorias investigadas por ellas. Se espera no sólo que se acelere la convergencia a la mejor solución si no, además, que se encuentren mejores soluciones que aquellas encontradas por estrategias de tramas independientes. El aspecto más difícil de establecer es la determinación de la naturaleza de la información a ser compartida o intercambiada para mejorar la búsqueda. Se necesita poner mucha atención para no usar memoria o tiempo adicionales en demasía. Se pueden implementar estrategias de tramas cooperativas usando reencadenamiento de trayectorias, combinando soluciones élite almacenadas en una pila central con los óptimos locales encontrados por cada procesador al final de cada iteración GRASP. Canuto et al. [30] usaron reencadenamiento de trayectorias para implementar un GRASP paralelo para el problema del árbol de Steiner con recolección de premios. Su estrategia es verdaderamente cooperativa, ya que pares de soluciones élite de una única pila centralizada se distribuyen a los procesadores los cuales ejecutan reencadenamiento de trayectorias en paralelo. Los resultados computacionales obtenidos con una implementación de MPI corriendo sobre una red de 32 procesadores Pentium II a 400-MHz mostraron aceleraciones lineales.

Aiex et al. [4] compararon GRASP paralelo independiente y cooperativo con implementaciones de reencadenamiento de trayectorias para programación de job shop. Ambas implementaciones usaron MPI en una computadora SGI Challenge con 28 procesadores R10000. El GRASP paralelo cooperativo mantiene una pila de soluciones élite en cada procesador. Cuando cualquier procesador encuentra una nueva mejor solución, esa solución se envía a todos los otros procesadores para que la inserten en sus respectivas pilas de soluciones élite. La Tabla 1 ilustra que mientras que para el GRASP paralelo independiente se observaron aceleraciones sub-lineales, para la implementación cooperativa se lograron aceleraciones super-lineales.

Cuadro 1: Aceleración con respecto a la implementación con un solo procesador. Los algoritmos son: implementaciones independiente y cooperativa de GRASP con reencadenamiento de trayectorias. Las instancias son `abz6`, `mt10`, `orb5`, y `la21`, con valores meta 943, 938, 895, y 1100, respectivamente.

GRASP paralelo independiente

	procesadores			
problema	2	4	8	16
`abz6`	2.00	3.36	6.44	10.51
`mt10`	1.57	2.12	3.03	4.05
`orb5`	1.95	2.97	3.99	5.36
`la21`	1.64	2.25	3.14	3.72
promedio:	**1.79**	**2.67**	**4.15**	**5.91**

GRASP paralelo cooperativo

	procesadores			
problema	2	4	8	16
`abz6`	2.40	4.21	11.43	23.58
`mt10`	1.75	4.58	8.36	16.97
`orb5`	2.10	4.91	8.89	15.76
`la21`	2.23	4.47	7.54	11.41
promedio:	**2.12**	**4.54**	**9.05**	**16.93**

6. Aplicaciones

La primera aplicación de GRASP fue en cubrimientos de conjuntos [46] en 1989, y, a partir de entonces, ha sido aplicado a un amplio rango de tipos de problemas. Referimos al lector a Festa y Resende [54] y el URL `http://graspheuristic.org/annotated` para una extensa bibliografía comentada de GRASP. Concluimos este artículo con una lista parcial de aplicaciones de GRASP, mostrando su amplia aplicabilidad.

- enrutamiento [12, 15, 20, 32, 65];
- lógica [38, 88, 93, 96];
- cubrimiento y partición [11, 13, 46, 56, 58];
- localización [1, 37, 62, 110, 111];
- árbol mínimo de Steiner [31, 76, 77, 78, 103];
- optimización en grafos [2, 48, 66, 86, 92, 98, 102, 52, 69, 35];
- assignación [45, 55, 70, 71, 79, 82, 87, 91, 89, 106, 60];
- horarios, programación, y manufactura [17, 18, 19, 24, 36, 40, 42, 43, 44, 49, 50, 64, 104, 105, 114, 7];
- transporte [12, 42, 45, 72, 21];
- sistemas de potencia [25, 26, 16];
- telecomunicaciones [2, 14, 62, 71, 91, 92, 100, 29, 83, 113, 63, 73];
- dibujo de grafos y mapas [51, 68, 98, 102, 74, 75, 27];
- lenguaje [34];
- estadística [80, 81];
- biología [10];
- programación matemática [85];
- empaquetado [33]; y
- VLSI [11], entre otras áreas de aplicación.

Referencias

[1] S. Abdinnour-Helm and S.W. Hadley. Tabu search based heuristics for multi-floor facility layout. *International Journal of Production Research*, 38:365–383, 2000.

[2] J. Abello, P.M. Pardalos, and M.G.C. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, *External Memory Algorithms and Visualization*, volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 199–130. American Mathematical Society, 1999.

[3] R.K. Ahuja, J.B. Orlin, and D. Sharma. New neighborhood search structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.

[4] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

[5] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. Technical report, AT&T Labs-Research, 2000. To appear in *INFORMS J. on Comp.*

[6] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

[7] M.S. Akturk and D. Ozdemir. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, 135:394–412, 2001.

[8] A.C. Alvim. Estratégias de paralelização da metaheurística GRASP. Master's thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brasil, 1998.

[9] A.C. Alvim and C.C. Ribeiro. Balanceamento de carga na paralelização da meta-heurística GRASP. In *X Simpósio Brasileiro de Arquiteturas de Computadores*, pages 279–282, Búzios, Brasil, 1998.

[10] A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002.

[11] S. Areibi and A. Vannelli. A GRASP clustering technique for circuit partitioning. In J. Gu and P.M. Pardalos, editors, *Satisfiability Problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 711–724. American Mathematical Society, 1997.

[12] M.F. Argüello, J.F. Bard, and G. Yu. A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, 1:211–228, 1997.

[13] M.F. Argüello, T.A. Feo, and O. Goldschmidt. Randomized methods for the number partitioning problem. *Computers and Operations Research*, 23:103–111, 1996.

[14] M. Armony, J.C. Klincewicz, H. Luss, and M.B. Rosenwein. Design of stacked self-healing rings using a genetic algorithm. *Journal of Heuristics*, 6:85–105, 2000.

[15] J.B. Atkinson. A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *Journal of the Operatinal Research Society*, 49:700–708, 1998.

[16] L. Bahiense, G.C. Oliveira, and M. Pereira. A mixed integer disjunctive model for transmission network expansion. *IEEE Transactions on Power Systems*, 16:560–565, 2001.

[17] J.F. Bard and T.A. Feo. Operations sequencing in discrete parts manufacturing. *Management Science*, 35:249–255, 1989.

[18] J.F. Bard and T.A. Feo. An algorithm for the manufacturing equipment selection problem. *IIE Transactions*, 23:83–92, 1991.

[19] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for scheduling printed wiring board assembly. *IIE Transactions*, 28:155–165, 1996.

[20] J.F. Bard, L. Huang, P. Jaillet, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science*, 32:189–203, 1998.

[21] J.F. Bard, G. Kantoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36:250–269, 2002.

[22] R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and tabu. *Microprocessors and Microsystems*, 16:351–367, 1992.

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

[24] S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and

P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2002.

[25] S. Binato and G.C. Oliveira. A reactive GRASP for transmission network expansion planning. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 81–100. Kluwer Academic Publishers, 2002.

[26] S. Binato, G.C. Oliveira, and J.L. Araújo. A greedy randomized adaptive search procedure for transmission expansion planning. *IEEE Transactions on Power Systems*, 16:247–253, 2001.

[27] C. Binucci, W. Didimo, G. Liotta, and M.Ñonato. Labeling heuristics for orthogonal drawings. In *Proceedings of GD'98 – Symposium on Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 139–153. Springer-Verlag, 2002.

[28] J.L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, Portland, 1996.

[29] M. Brunato and R. Battiti. A multistart randomized greedy algorithm for traffic grooming on mesh logical topologies. Technical report, Department of Mathematics, University of Trento, Trento, Italy, 2001.

[30] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

[31] S.A. Canuto, C.C. Ribeiro, and M.G.C. Resende. Local search with perturbations for the prize-collecting Steiner tree problem. In *Extended Abstracts of the Third Metaheuristics International Conference*, pages 115–119, Angra dos Reis, July 1999.

[32] C. Carreto and B. Baker. A GRASP interactive approach to the vehicle routing problem with backhauls. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 185–199. Kluwer Academic Publishers, 2002.

[33] P. Chardaire, G.P. McKeown, and J.A. Maki. Application of GRASP to the multiconstraint knapsack problem. In E.J.W. Boers et al., editor, *EvoWorkshop 2001*, pages 30–39. Springer-Verlag Berlin Heidelberg, 2001.

[34] H. Fraçois and O. Boëffard. The greedy algorithm and its application to the construction of a continuous speech database. In *Proceedings of LREC-2002*, volume 5, pages 1420–1426, May 29-31 2002.

[35] A. Corberán, R. Martí, and J.M. Sanchís. A GRASP heuristic for the mixed chinese postman problem. *European Journal of Operational Research*, 142:70–80, 2002.

[36] P. De, J.B. Ghosj, and C.E. Wells. Solving a generalized model for con due date assignment and sequencing. *International Journal of Production Economics*, 34:179–185, 1994.

[37] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem. *INFOR*, 37:194–225, 1999.

[38] A.S. Deshpande and E. Triantaphyllou. A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. *Mathematical Computer Modelling*, 27:75–99, 1998.

[39] N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

[40] A. Drexl and F. Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102:193–214, 1997.

[41] H.T. Eikelder, M. Verhoeven, T. Vossen, and E. Aarts. A probabilistic analysis of local search. In I. Osman and J. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 605–618. Kluwer Academic Publishers, 1996.

[42] T.A. Feo and J.F. Bard. Flight scheduling and maintenance base planning. *Management Science*, 35:1415–1432, 1989.

[43] T.A. Feo and J.F. Bard. The cutting path and tool selection problem in computer-aided process planning. *Journal of Manufacturing Systems*, 8:17–26, 1989.

[44] T.A. Feo, J.F. Bard, and S. Holland. Facility-wide planning and scheduling of printed wiring board assembly. *Operations Research*, 43:219–230, 1995.

[45] T.A. Feo and J.L. González-Velarde. The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Science*, 29:330–341, 1995.

[46] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[47] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[48] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

[49] T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers and Operations Research*, 23:881–895, 1996.

[50] T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a difficult single machine scheduling problem. *Computers and Operations Research*, 18:635–643, 1991.

[51] E. Fernández and R. Martí. GRASP for seam drawing in mosaicking of aerial photographic maps. *Journal of Heuristics*, 5:181–197, 1999.

[52] P. Festa, P.M. Pardalos, and M.G.C. Resende. Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. *ACM Transactions on Mathematical Software*, 27:456–464, 2001.

[53] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods & Software*, 7:1033–1058, 2002.

[54] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[55] C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11:198–204, 1999.

[56] J.B. Ghosh. Computatinal aspects of the maximum diversity problem. *Operations Research Letters*, 19:175–181, 1996.

[57] F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[58] P.L. Hammer and D.J. Rader, Jr. Maximally disjoint solutions of the set covering problem. *Journal of Heuristics*, 7:131–144, 2001.

[59] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

[60] M. Hasan, I. Osman, and T. AlKhamis. A meta-heuristic procedure for the three dimension assignment problem. *International Journal of Applied Mathematics*, 8:365–380, 2002.

[61] H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

[62] J.G. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

[63] J.G. Klincewicz. Enumeration and search procedures for a hub location problem with economies of scale. *Annals of Operations Research*, 110:107–122, 2002.

[64] J.G. Klincewicz and A. Rajan. Using GRASP to solve the component grouping problem. *Naval Research Logistics*, 41:893–912, 1994.

[65] G. Kontoravdis and J.F. Bard. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7:10–23, 1995.

[66] M. Laguna, T.A. Feo, and H.C. Elrod. A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research*, 42:677–687, 1994.

[67] M. Laguna and J.L. González-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

[68] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

[69] M. Laguna and R. Martí. A GRASP for coloring sparse graphs. *Computational Optimization and Applications*, 19:165–178, 2001.

[70] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

[71] X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A GRASP for frequency assignment in mobile radio networks. In B.R. Badrinath, F. Hsu, P.M. Pardalos, and S. Rajasejaran, editors, *Mobile Networks and Computing*, volume 52 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 195–201. American Mathematical Society, 2000.

[72] H. Ramalhinho Lourenço, J.P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus-driver scheduling problem. *Transportation Sciences*, 35:331–343, 2001.

[73] P. Mahey and C.C. Ribeiro. Modeling modern multimedia traffic. *Annals of Operations Research*, 110:107–122, 2002.

[74] R. Martí. Arc crossing minimization in graphs with GRASP. *IIE Transactions*, 33:913–919, 2001.

[75] R. Martí and V. Estruch. Incremental bipartite drawing problem. *Computers and Operations Research*, 28:1287–1298, 2001.

[76] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, *Randomization Methods in Algorithmic Design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

[77] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

[78] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

[79] T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the biquadratic assignment problem. *European Journal of Operational Research*, 105:613–621, 1998.

[80] M.C. Medeiros, M.G.C. Resende, and A. Veiga. Piecewise linear time series estimation with GRASP. *Computational Optimization and Applications*, 19:127–144, 2001.

[81] M.C. Medeiros, A. Veiga, and M.G.C. Resende. A combinatorial approach to piecewise linear time analysis. *Journal of Computational and Graphical Statistics*, 11:236–258, 2002.

[82] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel Processing of Discrete Problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.

[83] A. Myslek. Greedy randomised adaptive search procedures (GRASP) for topological design of mpls networks. In *Proceedings of the 8th Polish Teletraffic Symposium*, 2001.

[84] L. Osborne and B. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3:213–225, 1991.

[85] G. Palubeckis and A. Tomkevicius. GRASP implementations for the unconstrained binary quadratic optimization problem. *Information Technology and Control*, 24:14–20, 2002.

[86] P. M. Pardalos, T. Qian, and M. G. C. Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2:399–412, 1999.

[87] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems– Irregular'94*, pages 115–133. Kluwer Academic Publishers, 1995.

[88] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

[89] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate solutions to the turbine balancing problem. *European Journal of Operational Research*, 130:147–155, 2001.

[90] L.S. Pitsoulis and M.G.C. Resende. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.

[91] M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

[92] M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

[93] M.G.C. Resende and T.A. Feo. A GRASP for satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

[94] M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

[95] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

[96] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, *Satisfiability Problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

[97] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

[98] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.

[99] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search pro-

cedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

[100] M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41(1):104–114, 2003.

[101] M.G.C. Resende and R.F. Werneck. A GRASP with path-relinking for the p-median problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002.

[102] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:342–352, 1999.

[103] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

[104] R.Z. Ríos-Mercado and J.F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, pages 76–98, 1998.

[105] R.Z. Ríos-Mercado and J.F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup costs. *Journal of Heuristics*, 5:57–74, 1999.

[106] A.J. Robertson. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Computational Optimization and Applications*, 19:145–164, 2001.

[107] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, Seattle, 1994. MIT Press.

[108] D. Serra and R. Colomé. Consumer choice in competitive location models: Formulations and heuristics. *Papers in Regional Science*, 80:439–464, 2001.

[109] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 7:443–455, 1991.

[110] T.L. Urban. Solution procedures for the dynamic facility layout problem. *Annals of Operations Research*, pages 323–342, 1998.

[111] T.L. Urban, W.-C. Chiang, and R.A. Russel. The integrated machine allocation and layout problem. *International Journal of Production Research*, pages 2913–2930, 2000.

[112] M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–65, 1995.

[113] J. Xu and S. Chiu. Effective heuristic procedure for a field technician scheduling problem. *Journal of Heuristics*, 7:495–509, 2001.

[114] J. Yen, M. Carlsson, M. Chang, J.M. Garcia, and H.Ñguyen. Constraint solving for inkjet print mask design. *Journal of Imaging Science and Technology*, 44:391–397, 2000.