Simulation of Computer Systems and Networks

Evaluación de la disponibilidad de los servicios desplegados sobre *Volunteer Computing*

Antonio Escot Praena

Enginyeria Informàtica i Tècnica de Gestió

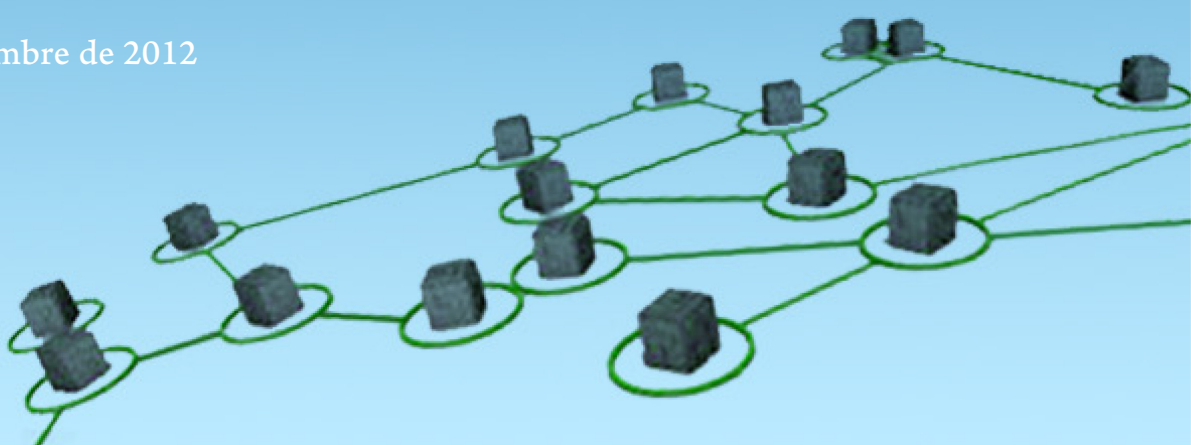
Dirección del TFC

Ángel A. Juan, PhD.

Eva Vallada Regalado, PhD.

Alberto García Villoria, PhD.

30 de diciembre de 2012



Agradecimientos

Curiosamente es al final cuando toca hacer un repaso al tiempo transcurrido desde que se inició este trabajo final de carrera, y es en ese momento que te das cuenta de que o quien lo ha hecho posible. Un trabajo de este tipo requiere muchas cosas pero sin duda alguna en mayor cantidad es el tiempo, por eso he de agradecer a todas esas personas que me han ayudado de tal forma que he tenido el tiempo necesario para poder acabarlo. Mi agradecimiento más sincero a mi familia y mis amigos en los que me he apoyado siempre que los he necesitado, a los consultores Eva Vallada primero y Alberto García después por haberme ayudado a encontrar el enfoque a seguir con sus aportaciones y correcciones, a Guillem por haber compartido conmigo el código del SAEDES y el DiSeASim y sus ideas sobre las posibilidades del algoritmo evaluado que me han permitido centrarme en lo más importante de la implementación, y mi agradecimiento más especial a Ángel A. Juan por haberme permitido iniciarme en el mundo de la simulación.

Resumen

Este trabajo adopta una estructura incremental para exponer la oportunidad que supone poder desplegar servicios de computación intensiva sobre recursos distribuidos cedidos de forma voluntaria y la necesidad de un algoritmo eficiente que ayude a su despliegue seleccionando los mejores nodos de la lista de nodos disponibles minimizando su coste.

La primera parte presenta la *Computación Voluntaria* como una alternativa a la computación distribuida empresarial, basada en la cesión de los recursos excedentes de los equipos de escritorio -CPU, RAM, disco duro, etc.- a proyectos de cálculo intensivo sin coste para los usuarios que participan en estos proyectos. Esta característica, cesión voluntaria, determina en gran medida las problemáticas que aparecen en esta forma de computación colaborativa, equipos con diferentes configuraciones y potencia, sistemas operativos diferentes y con diferentes niveles de aplicación de los parches de actualización, diferentes políticas de seguridad, apagados de la máquina cuando el usuario deja de usarla, cuelgues inesperados, etc. lo que dibuja un escenario muy heterogéneo con recursos geográficamente distribuidos pero que ofrece grandes posibilidades a la computación de alto rendimiento -proyectos como SETI@Home y Folding@Home puestos en marcha por la Universidad de Berkeley son ejemplos de ello- ; el artículo base de este trabajo va un poco más lejos, plantea la posibilidad de desplegar servicios sobre estos recursos de una disponibilidad muy volátil.

En la segunda parte, punto 6.1 y 6.2, se hace una introducción formal del problema. Empieza definiendo el concepto de grafo, $G = (V, A)$, como el par ordenado formado por el conjunto no vacío de vértices V y A el conjunto de aristas [12]. Esta definición nos permite representar un servicio mediante un grafo, donde los vértices se corresponden con los recursos y sus relaciones con las aristas. Para después introducir brevemente los algoritmos de exploración de soluciones vecinas, donde se pone de manifiesto la importancia el concepto de vecindad y como lo utilizan las diferentes heurísticas de búsqueda local para trazar una sucesión de soluciones que nos llevan a un óptimo local. Vemos por tanto, que la búsqueda local establece una relación varios a uno entre las soluciones, S, y las soluciones óptimos locales, S^*, al llevar a toda solución s vecina de s* a un mismo mínimo local. Esta situación nos lleva al concepto de pozo de atracción. Si las heurísticas de búsqueda local son fáciles de implementar y dan soluciones rápidamente, la existencia de estos pozos de atracción nos lleva a soluciones

óptimos locales que no necesariamente son buenos óptimos globales, o están muy alejados de ellos.

Las metaheurísticas hacen uso de heurísticas de base e imponen reglas que permiten escapar de las vecindades que no poseen soluciones de buena calidad o bien guiar la búsqueda hacia espacios de búsqueda más prometedores. Así, la búsqueda local iterada es un proceso recursivo de aplicación de una heurística de búsqueda local que parte de una solución inicial y que va mejorando aplicando mecanismos de perturbación en las soluciones visitadas para escapar de estos pozos de atracción. Estos mecanismos de perturbación o movimientos dependen del problema en cuestión, en nuestro caso, se trata de una sustitución de nodos.

El punto 6.3 introduce la simulación por eventos discretos y los elementos clave que lo componen, el generador de número pseudoaleatorios y las funciones inversas de las distribuciones de probabilidad que nos permiten convertir los números pseudoaleatorios en valores propios de la simulación. Se presentan dos algoritmos basados en estos principios el SAEDES y el DiSeASim que nos permitirán evaluar la disponibilidad del sistema en su totalidad y no sólo de sus componentes por separado.

En el punto 6.4 se introduce el DS-ReliaSim, como un algoritmo que resuelve un problema no resulto en la bibliografía actual, desplegar servicios en entornos de computación voluntaria minimizando el coste del mismo, y pasa a analizar las partes que lo distinguen. Un servicio está definido por su topología y como tal alguno de los nodos que lo componen resultan críticos para su funcionamiento, el algoritmo asigna los nodos de mayor disponibilidad a estos nodos críticos. La selección de la solución inicial tiene un gran impacto en la exploración que realiza el DS-ReliaSim tanto en tiempo de proceso como en la calidad de las soluciones halladas, cabe recordar, que este algoritmo sigue un proceso de búsqueda descendente, parte de una solución que trata de mejorar en cada iteración del mismo. Otro elemento que tiene un gran impacto, especialmente en el problema que tenemos planteado, son las funciones de perturbación propias de las metaheurísticas del tipo búsqueda local iterada como es el caso del DS-ReliaSim, en este trabajo se han analizado dos funciones de perturbación que sustituyen a la versión aportada por el algoritmo, una de ellas estática en el que se aumenta la longitud del salto entre soluciones en cada iteración en una cantidad fija mayor que uno (la versión normal), y otra variable en la que este salto depende de la distancia entre la

disponibilidad de la solución actual y el umbral de disponibilidad deseada, en ambos casos se obtiene una mejora apreciable de la velocidad.

Finalmente se presenta un ámbito de cálculo intensivo donde el algoritmo puede ser aplicado, la bioinformática, específicamente el renderizado de modelos biológicos, donde comunidades no necesariamente muy amplias de ordenadores colaborando para desplegar esta tarea como servicio pueden verse ampliamente beneficiadas.

Índice de contenidos

Tabla de ilustraciones

ILUSTRACIONES

ALGORITMOS

TABLAS

1. Introducción

La motivación de este trabajo hay que encontrarla en un artículo previo del Dr. Angel A. Juan y otros, *"Deploying cost-eficient and highly-available services over distributed systems with heterogeneous and non-dedicated resources"* [1], que presenta un nuevo enfoque para resolver el problema de seleccionar el mínimo conjunto de recursos que permiten desplegar un servicio con un nivel de disponibilidad determinado, y por tanto la posibilidad de desplegar servicios de cómputo públicos basados en redes de dispositivos heterogéneos y no dedicados, es decir, servicios distribuidos bajo el paradigma de redes de voluntarios -*volunteer networking*-.

Actualmente estamos asistiendo a un auge del procesamiento basado en la distribución de servicios y recursos, debido por un lado al crecimiento de las redes de alta velocidad, y por otro al exceso de potencia instalada en los equipos actuales con respecto a los requerimientos de las tareas habituales que constituyen su uso normal, por ejemplo, un equipo doméstico -PC- dotado de procesador i7 con 8 Gb de RAM y 1 Tb de disco duro que se utilice para consultar el correo electrónico, mantenerse al día en las redes sociales, abrir un editor de texto y posiblemente un editor de imágenes, si consultamos el monitor de rendimiento seguramente rara vez veremos un uso de CPU superior al 10% de media.

Los nodos de estas redes se caracterizan por ser muy heterogéneos tanto a nivel del sistema operativo bajo el que corren, como en su coste computacional. Los índices de volatilidad (no disponibilidad) son muy variables al estar fuera del dominio administrativo del gestor de la red, o del usuario que quiere hacer uso de ella, debido a cuelgues inesperados o apagados de la máquina decididos por su propietario.

Desplegar servicios en esta plataforma con un nivel alto de fiabilidad y disponibilidad -QoS-, y con coste mínimo, dependerá del umbral mínimo de recursos requeridos para llevar a cabo las tareas del servicio y de la redundancia necesaria para garantizar su disponibilidad durante el tiempo necesario para completarlas. Así tendremos, que el despliegue de un servicio en redes de estas características con un umbral de disponibilidad mínimo que garantice la realización de la tarea, exigirá un determinado nivel de redundancia que queremos minimizar para mantener ajustado el coste del mismo. La optimización de este problema multi-objetivo proponemos resolverla con la utilización de un algoritmo híbrido, por un lado un algoritmo principal heurístico que

explora el espacio de soluciones del tipo *Iterated Local Search* -ILS-, y un algoritmo de validación de las soluciones basado en simulación de eventos discretos, el SAEDES o bien su versión ligera el DiSeASim.

El objetivo final de este trabajo es implementar el ILS y evaluar su eficiencia para determinar el mejor despliegue del servicio en redes contributivas voluntarias. En último término debe permitir al Gestor del Sistema proponer la mejor asignación de recursos para llevar a cabo la tarea requerida por un usuario, y debe hacerlo en tiempos ajustados a los niveles de interoperabilidad esperados por este.

2. Palabras clave

Grid computing, volunteer computing, discreet events simulation, iterated local search, grid resource availability

3. Objetivos

Este TFC se enmarca dentro de las líneas abiertas en el mencionado artículo [1], centrándose en aspectos de la codificación y optimización del algoritmo principal propuesto, y en la integración del algoritmo de validación y aceptación de las soluciones propuestas por este algoritmo principal -SAEDES-. Los objetivos serán los siguientes:

a. Implementación de la heurística de búsqueda local de soluciones atendiendo a principios de modularidad y generalidad, permitiendo la integración de diferentes funciones de perturbación.
b. Integración del SAEDES dentro del algoritmo principal, ILS, de forma que facilite la implantación del middleware que soporte la experimentación en este tipo de redes.
c. Explorar otros algoritmos de validación de soluciones que mejoren la eficiencia del SAEDES. Dada la centralidad de estos algoritmos en la búsqueda de la mejor solución y su despliegue, el rendimiento será determinante para decantar la elección.
d. Proponer ejemplos realistas que ilustren su aplicación.

Teniendo en cuenta las características de un TFC, a desarrollar durante un semestre lectivo, algunos de los objetivos iniciales puede ser que finalmente se dejen como posibles líneas de trabajo futuras.

4. Trabajos relacionados

En la bibliografía actual, bastante abundante, encontramos estudios relacionados que presentan diferentes algoritmos de búsqueda de soluciones, no todos los autores consideran el coste de los recursos como determinante en la solución propuesta, algo que aquí se considera básico dado que nos movemos en ambientes contributivos voluntarios, donde los recursos pueden llegar a ser escasos. Por otro lado, se propone un algoritmo que estudia la disponibilidad del sistema en su conjunto, al menos del conjunto de nodos que definen el servicio, a diferencia del enfoque de otros autores que centran su análisis en la disponibilidad de cada uno de los nodos del servicio.

En otros casos, los autores basan sus resultados en un estudio empírico de la disponibilidad de los nodos de la red para proponer un modelo matemático de predicción de la disponibilidad y la carga del sistema [2], los datos fueron recogidos de las trazas de los fallos registrados en la plataforma Condor [3] en los que se recogía cuando y porqué se produce el fallo, registrando hasta cinco estados de disponibilidad, el objetivo es mejorar el rendimiento del sistema implementando un *predictor* del estado de los nodos en el planificador de tareas. En este trabajo sólo consideramos dos estados posibles: disponible y no disponible.

5. La computación distribuida

5.1. Estado del arte

Durante los últimos años estamos asistiendo a un auge sin precedentes en el desarrollo de la computación distribuida, tanto en el número de departamentos de investigación que se dedican a su desarrollo, como en las propuestas que ya están en marcha, convirtiendo en una realidad la posibilidad de ofrecer soluciones escalables y económicas en la computación de alto rendimiento.

Este desarrollo ha sido posible en gran parte por el crecimiento de la infraestructura física en la que se basa, por una lado el crecimiento de la potencia de cálculo de los ordenadores según la Ley de Moore [4] que predice que cada 18 meses se dobla la densidad de transistores en un circuito integrado, incrementando su funcionalidad y prestaciones a la vez que bajan los costes de producirlo, y por otro lado tenemos las consecuencias de la Ley de Gilder [5] en el ámbito de las redes de comunicaciones digitales, ya que predice que el ancho de banda de dichas redes se triplica cada 12 meses. Estas continuas mejoras en las redes informáticas y las tecnologías de los microprocesadores ofrecen a usuarios y empresas nuevos entornos de computación distribuida, en este contexto, la computación en malla -*Grid computing*- surge como un prometedor paradigma para la computación de altas prestaciones que viene a solucionar los crecientes requerimientos de cálculo de los equipos de investigación tanto de organizaciones públicas como privadas, a la vez que surge una nueva concienciación por un mejor aprovechamiento de estos recursos. El Grid pretende integrar estos recursos geográficamente dispersos y heterogéneos, bajo el control de diferentes individuos e instituciones y presentarlos al investigador/usuario como un sistema único de forma transparente y segura [6] [7].

Paralelo a estos desarrollos en los sistemas de cómputo distribuido, han ido apareciendo diferentes entornos de trabajo para el despliegue de aplicaciones -*Middlewares*- tanto en entornos corporativos y privados como en computación voluntaria, que tratan de dar respuesta a las necesidades de flexibilidad y simplificación de las arquitecturas cliente-servidor. Estos entornos han permitido separar la lógica de la aplicación de la infraestructura distribuida en la que se ejecuta, posibilitando la aparición de middlewares que responde a las características propias de la computación voluntaria, sistemas heterogéneos, con una disponibilidad cambiante y que están en el dominio administrativo de un usuario que limita los recursos, en tiempo y espacio, que cede a la aplicación. Como ejemplos dentro de la computación voluntaria podemos mencionar las plataformas Bayanihan [8] y Charlotte [9], que constituyen referencias históricas y la *Berkeley Open Infrastructure for Network Computing* -BOINC- [10] [11], la más ampliamente utilizada con alrededor de una veintena de proyectos desplegados sobre ella.

5.2. Paradigmas en computación distribuida

Atendiendo a las características de los recursos, tipos de administración y control de fallos y posibilidad de acciones malintencionadas, podemos distinguir dos entornos Grid: el Grid computing empresarial o simplemente Grid, y el *Volunteer computing*, también llamado Desktop Grid por algunos autores, dado que mayormente participan en estas redes ordenadores de sobremesa controlados por individuos de forma personal y con unos índices de fallos muy altos y por tanto con muy baja disponibilidad durante largos periodos de tiempo.

Otros sistemas de trabajo en malla lo proporcionan los sistemas P2P, con un modelo de búsqueda de servicios diferente al Grid, y los basados en *Web services* o computación en la nube -*Cloud computing*- sistema este último orientado a servicios empresariales con coste económico añadido normalmente dependiente de los recursos requeridos.

Proyectos como SETI@home basado en el sistema BOINC, Entropia, Bayanihan, etc., se ven como casos de éxito del *Volunteer computing*, paradigma que constituye el centro de atención de este trabajo, dada la oportunidad que ofrecen estos sistemas para el desarrollo de aplicaciones de cálculo intensivo y compartición de recursos sobre las capacidades excedentes de los equipos de sobremesa actuales -CPU multi-núcleo, RAM disponible, procesador gráfico o GPU, disco duro, etc.-

6. Desplegar servicios sobre *Volunteer Computing*

6.1. Planteamiento del problema

Un servicio es cualquier aplicación, ejecutándose en uno o varios ordenadores, capaz de recibir y responder a mensajes procedentes de otros ordenadores. Según los autores del artículo que sirve como base de este trabajo, es posible representar las características de un servicio mediante un grafo no dirigido.

Un grafo G = (V, A) es un par ordenado (V, A) donde V es un conjunto finito no vacío, cuyos elementos son los vértices, y A es un subconjunto del conjunto de parejas no ordenadas (es decir, subconjuntos de dos elementos diferentes) de elementos de V; el conjunto A es el conjunto de aristas [12]. Si asociamos nodos a recursos y aristas a

operaciones o tareas del servicio se hace evidente este paralelismo, el cual nos ofrece una base formal para analizar el problema de desplegar servicios distribuidos.

Las comunidades de redes de voluntarios comparten las características de los sistemas en malla, recursos geográficamente distribuidos, heterogéneos y en general débilmente acoplados, apropiados por diferentes individuos u organizaciones con sus propias políticas de acceso y una gran variabilidad en sus condiciones de disponibilidad y carga. Desplegar servicios en estos entornos nos obliga a proveer medidas de clusterización y redundancia para minimizar el impacto de los fallos imprevistos de recursos que son críticos para completar la función definida en el servicio. Uno de los objetivos de este trabajo es la implementación de un algoritmo híbrido que nos permita encontrar una solución, aprovechando la dualidad establecida, encontrar una topología de los recursos que minimice el coste de los mismos a la vez que garantice la disponibilidad, como sistema, durante el tiempo necesario para finalizar las tareas que componen el servicio con una probabilidad determinada.

Formalmente, sea G = (V, A) un grafo no dirigido, donde V = $\{v_1, v_2, v_3, \ldots, v_n\}$ es el conjunto de los n recursos disponibles para desplegar el servicio y definimos $c(v_i)$ como el coste del recurso v_i. El coste del servicio vendrá dado por la suma de costes de los recursos utilizados,

$$C = \sum_{i=1}^{n} \delta_i c(v_i), \text{ con } \delta_i = \begin{cases} 1, & nodo\ utilizado \\ 0, & nodo\ no\ utilizado \end{cases}$$

La disponibilidad del servicio -A-, vendrá determinada por el estado operacional de los nodos que forman parte, especialmente de aquellos que resultan críticos para que pueda llevar a cabo la tarea asignada. El estado de un recurso, como se ha visto anteriormente, será una función dependiente del tiempo y por lo que respecta al funcionamiento del servicio valdrá 1 si está operativo en el instante t y 0 si no lo está.

Nuestro trabajo se centra en la búsqueda de soluciones que minimizan la función de coste C y a la vez mantienen la disponibilidad del sistema para todo instante de un intervalo de tiempo dado por encima de un umbral A_o.

En general, encontrar la configuración de los recursos que nos permitan desplegar el servicio con efectividad y minimizando el coste es un problema combinatorio de complejidad NP-completo. En la implementación propuesta en este trabajo (continuación de [1]) se propone una resolución aproximada, basada en una heurística de búsqueda de mínimo local y el uso de un algoritmo de validación de la solución

basado en simulación de eventos discretos, el SAEDES [13] o el DiSeASim (una versión ligera del mismo), que estima su índice de disponibilidad.

6.2. Algoritmos de exploración de soluciones vecinas

Un problema de optimización P lo podemos formular como la tripleta (S, Ω, f). S es el espacio de búsqueda (o también llamado espacio de soluciones) definido sobre un conjunto de variables X_i, i=1,…,m; dependiendo de si el dominio de estas variables es discreto o continuo, hablaremos de problema de optimización discreto o continuo. Ω es un conjunto de condiciones que deben cumplir dichas variables. Y $f : S \rightarrow R$ es la función objetivo a optimizar.

El objetivo será encontrar una solución $s \in S$ tal que $f(s) \leq f(s')$, $\forall\ s' \in S$, en el caso de minimizar la función objetivo, o $f(s) \geq f(s')$ en el caso de maximizarla. En los problemas reales, disponer de algoritmos eficaces en problemas de optimización no siempre será posible, y en muchos casos la única posibilidad disponible será utilizar métodos metaheurísticos.

En la literatura podemos encontrar diferentes definiciones del término metaheurística que describen con mayor o menor precisión el concepto. Brevemente se puede definir como un algoritmo de carácter iterativo que guía una heurística base combinando diferentes conceptos de forma inteligente, para explorar y explotar el espacio de soluciones del problema. Vemos que los términos diversificación (exploración) e intensificación (explotación) son elementos que caracterizan la metaheurística. Diversificación se refiere al esfuerzo del algoritmo para explorar el espacio de búsqueda y la intensificación a su capacidad para explotar la información acumulada durante el proceso de búsqueda. En general, una metaheurística mantendrá un balance entre uno y otro; es importante la identificación de regiones prometedoras de soluciones en el espacio, así como también dedicar tiempo a intensificar la búsqueda en esas regiones prometedoras para llegar a buenas soluciones en un tiempo razonable.

En un esquema general del algoritmo, determinadas operaciones son repetidas iterativamente hasta que se verifica una condición de finalización. La aplicación de estas operaciones está guiada por una regla iterativa o trayectoria, el conjunto de estas reglas constituye la vecindad de la solución que se explora. La estructura de la vecindad tiene

un gran impacto en el rendimiento del algoritmo de optimización, si esta no es adecuada al problema la metaheurística fallará en su resolución.

Vecindad. Dada una solución *s* del espacio de búsqueda llamaremos vecindad, V(s), al conjunto de soluciones a las que se puede llegar mediante una regla de transformación (movimiento) a partir de ella,

$$V(s) = \{s' \in S \mid s \xrightarrow{\tau} s' \; donde \; \tau \; es \; la \; regla \; de \; transformación\}$$

Una solución s' en la vecindad de s, s' ∈ V(s), será un vecino de s. En espacios de búsqueda continuos, la vecindad de una solución *s* forma una bola centrada en *s* de radio r>0, de forma que s' es vecino de *s* si la distancia d(s, s') < r, siendo d la distancia euclidiana. Para espacios discretos será necesario definir una distancia que dependerá de la regla de transformación utilizada, por ejemplo para soluciones representadas en un hipercubo, los vecinos de un nodo será el conjunto de nodos adyacentes. Un vecino es generado en la metaheurística por una función de desplazamiento o movimiento. La vecindad queda así caracterizada por su *localidad*, teniéndose que, en una vecindad de fuerte localidad una pequeña transformación de la representación muestra un efecto pequeño en la solución, y por el contrario, si la localidad es débil, se tiene un gran efecto en la solución ante un pequeño cambio en la representación que en último término puede degenerar en una búsqueda aleatoria. La estructura de la vecindad es por tanto dependiente del problema que intenta resolver el proceso de optimización y puede ser, continua o discreta, según sean variables continuas o discretas las que define el espacio de búsqueda del problema.

En general, los algoritmos de optimización pueden ser divididos en exactos y aproximados. Los primeros aprovechan una relación clara entre el problema y las posibles soluciones de forma que, en cada paso de la ejecución del algoritmo existe al menos una forma de continuar, si no, el proceso ha terminado [14], por tanto, aseguran que la solución calculada es óptima, mientras que los segundos, en general, no lo aseguran. Los métodos metaheurísticos son métodos aproximados, los cuales se pueden dividir en deterministas, aquellos que a partir de la misma solución inicial siempre encuentran la misma solución, y los estocásticos, que al hacer uso de número pseudoaleatorios normalmente devolverán soluciones diferentes debido al componente de aleatoriedad de los mismos. Otras clasificaciones se basan en si el algoritmo está

inspirado en la naturaleza o no, en si usa memoria o no, si usa un proceso iterativo o bien construye la solución desde cero, etc.

Si las clasificamos por la forma de encontrar la solución, distinguimos dos tipos:

- **Basadas en trayectorias o en única solución.** Estas metaheurísticas parten de una solución inicial que tratan de mejorar durante el proceso de optimización. Este proceso puede ser visto como recorridos por la vecindad o trayectorias en el espacio de búsqueda del problema, estas trayectorias están marcadas por procedimientos iterativos que transforman la solución actual por otra de dicho espacio. Es decir, aplican iterativa procedimientos de generación de soluciones candidatas por transformación de la solución actual s, $V(s)$, para después seleccionar una de las candidatas, $s' \in V(s)$, que remplazará a la solución en la siguiente iteración. El proceso acaba cuando se cumple alguna condición de contorno o finalización. A este grupo pertenecen los algoritmos, Búsqueda Local, Recocido simulado y Búsqueda Tabú por ejemplo, y son generalmente más adecuados para estructuras discretas de la vecindad.
- **Basadas en poblaciones**. Comparten muchos de los conceptos básicos de las anteriores. En el proceso iterativo se pretende mejorar una población de soluciones a partir de una población inicial, generando una nueva población que será integrada en la nueva solución según un determinado mecanismo de selección. El proceso finaliza cuando se cumple determinada condición. La Búsqueda Dispersa, Algoritmos Genéticos, Colonia de Abejas y Optimización por Enjambre de partículas son ejemplos de este grupo, que se muestran adecuados para problemas continuos.

Estas dos familias de algoritmos tienen características complementarias (Ilustración 1), mientras las basadas en poblaciones están orientadas a la exploración del espacio de búsqueda, las basadas en solución única están orientadas a la intensificación de la búsqueda en regiones locales.

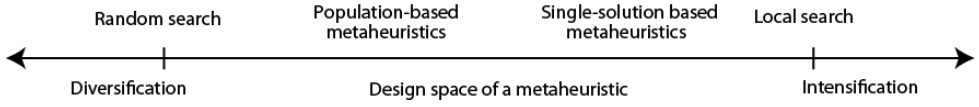


Ilustración 1. Dos conceptos contrapuestos caracterizan las metaheurísticas

En nuestro problema buscamos desplegar un servicio sobre una serie de nodos, a determinar por el algoritmo, sobre un conjunto discreto de recursos disponibles

sometidos a unas determinadas tasas de fallo y recuperación y un coste operacional y económico, y que debe tener un coste mínimo con una disponibilidad superior a un umbral definido en el servicio, por lo que una heurística basada en una búsqueda local que relacione coste y disponibilidad puede ser prometedora. En el artículo [1], se propone un ILS que guía la búsqueda en el espacio de búsqueda mediante la ordenación de los nodos disponibles (por disponibilidad) y los seleccionados (por coste) para proceder iterativamente a probar soluciones sustituyendo nodos caros por nodos más baratos de la lista de disponibles. Como solución inicial escoge la de mayor disponibilidad, que, usualmente, será también la más cara.

6.2.1. La heurística *Local Search* -LS-

LS parte de una solución inicial y en cada iteración busca una solución entre sus vecinas que mejore la solución actual. El proceso continúa hasta que no es posible encontrar una solución mejor entre las soluciones vecinas.

Sea f la función objetivo de nuestro problema de optimización combinatoria que se quiere minimizar -en algunos problemas el objetivo podría ser maximizar f-; y sea S el conjunto de soluciones candidatas s; el proceso de búsqueda de LS puede verse como un recorrido en el espacio de búsqueda que nos lleva de una solución inicial s_0 a un solución óptimo local s*, y por tanto, $f(s^*) \leq f(s), \forall s \in V(s^*)$. Si consideramos el conjunto S* de todo los óptimos locales s*, vemos que LS establece un mapeo de muchos a uno entre las soluciones candidatas S y el conjunto de óptimos locales S*.

En la Ilustración 2 tenemos un gráfico del proceso.

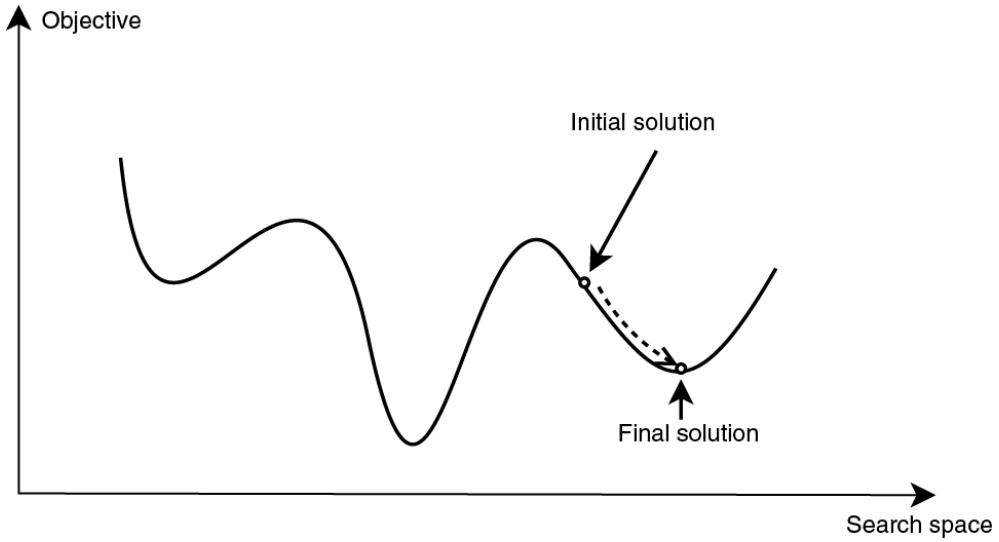


Ilustración 2. Búsqueda local

La eficiencia de la heurística y la calidad de la solución hallada estarán determinadas por la elección de la solución inicial y de la función de selección del entorno de soluciones vecinas [15].

Para la elección de la solución inicial se dispone de dos estrategias principalmente, seleccionarla aleatoriamente o bien utilizar algún algoritmo constructivo -glotón o *greedy*- que nos genere una buena solución de partida.

Seleccionar una solución aleatoria suele ser un proceso rápido en términos computacionales, pero podría requerir un excesivo número de pasos para la convergencia hacia su óptimo local. En estos casos puede ser recomendable usar heurísticas basadas en una selección *greedy* que suelen ofrecer soluciones de mejor calidad además de converger más rápidamente hacia el óptimo local.

Otro elemento a considerar a la hora de mejorar las heurísticas LS, es la estructura de la vecindad del espacio de búsqueda, y ligada a esta estructura, la función de selección de vecinos para la próxima iteración. La existencia de esta estructura permite moverse de la solución actual a otra vecina de una forma más inteligente, no obstante, no siempre tendremos conocimiento a priori de esta estructura al iniciar la optimización. Los métodos de exploración del vecindario más habituales son:

- **El mejor vecino**, para ello se recorren todas las soluciones vecinas y nos quedamos con la mejor.
- **El primer vecino que mejore**, recorremos las soluciones vecinas hasta encontrar la primera que mejore la solución actual.
- **Aleatoria,** se escoge un vecino al azar.

Estas heurísticas permiten encontrar usualmente soluciones rápidamente, es determinista, a partir de una solución inicial siempre encuentran el mismo óptimo local, y pueden hacer uso o no de memoria. Sin embargo tienden a quedarse atrapadas en entornos del espacio de búsqueda -pozos de atracción- con la posibilidad de que el óptimo local obtenido pudiera ser de una calidad mediocre e incluso mala, en la Ilustración 2 se ve gráficamente esta situación.

6.2.2. La metaheurística *Iterated Local Search* -ILS-

La heurística LS es ampliamente utilizada por la sencillez de su implementación y genera buenas soluciones rápidamente. Sin embargo, como se ha visto, se queda atrapada en el

mismo óptimo local sin posibilidad de escapar de él. En la literatura revisada podemos encontrar diferentes estrategias usadas por las metaheurísticas basadas en búsqueda local para escapar de estos mínimos locales. Estas estrategias para ampliar la diversificación de la búsqueda podemos agruparlas en:

- Iterar el proceso partiendo de diferentes soluciones iniciales.
- Aceptar soluciones vecinas que no mejoran la solución actual pero que pueden contener buenas soluciones en su entorno.
- Cambiar la estructura del vecindario de la solución actual, seguramente añadiendo memoria a largo plazo en el algoritmo.
- Cambiar la forma de la función a mejorar o sus datos de entrada de forma dinámica, añadiendo una guía hacia la solución buscada.

El ILS es un algoritmo que sigue la primera estrategia. El principio base de esta meta-heurística es realizar una búsqueda local a partir de una solución inicial, para después aplicar una perturbación a la solución encontrada, y esta nueva solución perturbada es aceptada como solución bajo ciertas condiciones de aceptabilidad para después volver a aplicar una búsqueda local (ver Ilustración 3). Este proceso iterativo acaba cuando se cumple cierta condición de finalización.

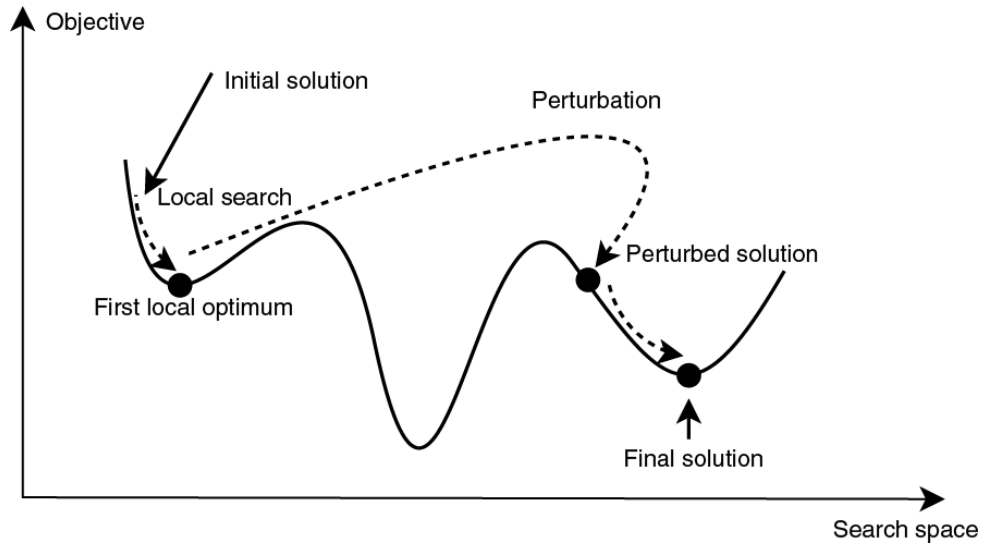


Ilustración 3. Búsqueda local iterada

La heurística LS establece una mapeo entre el conjunto de soluciones S y el conjunto de soluciones óptimas locales S*, un conjunto bastante más reducido de elementos. La metaheurística ILS busca aprovechar esta relación explorando el espacio S* y no S, siguiendo un recorrido que salta de s* a otro vecino también óptimo local. El mecanismo

del ILS para hacer este salto es aplicar una perturbación a la solución actual, de manera que dada s*, la solución actual, se llega a s' en S, entonces se aplica LS a s' de forma que se alcanza la solución óptima local s*', si s*' pasa la condición de aceptabilidad se toma como solución para la siguiente iteración, sino lo pasa volvemos a s* [16]. La eficiencia de este proceso depende de que las perturbaciones aplicadas no sean ni muy grandes ni muy pequeñas. En el primer caso, se cae en un reinicio aleatorio y en el segundo no se escapa del mínimo local.

En Algorismo 1 puede verse el pseudocódigo genérico de este algoritmo, donde aparecen los elementos básicos del ILS:

- **GenerateInitialSolution**, procedimiento que genera la solución de partida, s_0 y que la heurística de búsqueda local mapea a un mínimo local, s_0^*, punto de inicio del proceso iterativo.
- **LocalSearch**, heurística base de la metaheurística que explora el espacio de búsqueda en la vecindad de una solución dada. Aunque normalmente es preferible una búsqueda local determinista y sin memoria, el ILS es una metaheurística general que permite la utilización de heurísticas estocásticas.
- **Perturbation**, función de perturbación del ILS que expande el espacio de búsqueda mediante movimientos o transformaciones de la solución actual, en algunas implementaciones puede hacer uso de la *historia*, la relación de las últimas soluciones visitadas que permite guiar las siguientes soluciones a explorar, y del conocimiento que se tiene del problema. En la práctica se cumple que una buena función de perturbación es aquella que transforma una buena solución en un buen punto de inicio para la búsqueda local.
- **AcceptanceCriterion**, procedimiento/condición para determinar la aceptabilidad de la solución propuesta por la búsqueda local, puede hacer uso o no de *memoria* (soluciones ya evaluadas).
- **EndCondition**, es la condición de finalización que nos sacará del bucle y que en general dependerá de las variables que pretendemos optimizar y del tiempo asignado al proceso.

```
Procedure ILS
  s0 = GenerateInitialSolution();
  s* = LocalSearch(s0);
  Repeat
```

```
      s' = Perturbation(s*, historia);
      s*' = LocalSearch(s');
      if AcceptanceCriterion(s*', s*, memoria) then
            s* = s*';
      endif
   Until EndCondition
End
```

Algoritmo 1. Pseudocódigo del Iterated Local Search

Vemos que finalmente la adaptación del algoritmo a un problema determinado depende de la forma en que se selecciona la solución inicial, de la heurística base LS, de la función de perturbación y del criterio de aceptabilidad aplicado. Esta estructura modular permite que el algoritmo pueda ser mejorado y optimizado de formas diversas, en su totalidad introduciendo información específica del problema combinatorio a optimizar, lo que sin duda le resta generalidad, o bien optimizando cada uno de los componentes que lo forman.

En general, un buen punto de inicio s_0* nos lleva a soluciones de gran calidad rápidamente. Para su elección se tienen dos mecanismos, escoger una solución de inicio de forma aleatoria o bien utilizar un algoritmo *greedy* para construirla. La selección aleatoria puede consumir un número de iteraciones importante antes de que la solución propuesta cumpla los criterios de aceptabilidad.

La función de perturbación permite escapar de los óptimos locales, un simple movimiento aleatorio de la solución actual permite salir de un pozo de atracción y mejorar la calidad de las soluciones así como el rendimiento del algoritmo, sin embargo, son las funciones de perturbación basadas en propiedades del problema son las que generalmente dan mejores resultados.

El criterio de aceptación determina el camino que sigue el ILS a discriminar ciertas soluciones propuestas como puntos de partida para la siguiente iteración. Relajar este criterio de forma temporal puede mejorar la exploración de vecinos alejados, que sin ser aceptables tienen óptimos de gran calidad en su vecindad, en estos casos el uso de la *historia* puede ser interesante para no volver a vecinos explorados.

6.3. Evaluación de la disponibilidad

El estudio de la disponibilidad de los sistemas ha sido enfocado principalmente desde dos puntos de vista por los investigadores, el empírico y la simulación; dentro de la simulación encontramos métodos estáticos, caso del método de Montecarlo, en los que

el tiempo no juega ningún papel, y métodos tiempo-dependientes como la simulación de eventos discretos.

El punto de vista empírico ha sido considerado por los investigadores de la infraestructura Condor [17] [18], utilizada por el Network Weather Service -NWS- para el desarrollo de modelos de predicción de carga y disponibilidad [19] que en último término permita generar una agenda -*scheduler*- de los recursos para el mantenimiento de las políticas de uso y carga de la infraestructura.

El método de Montecarlo tiene su origen en la ciudad de Mónaco del mismo nombre, famosa por sus casinos, donde los juegos, como la ruleta y el *blackjack*, muestran un comportamiento aleatorio. Este método en realidad es un conjunto de técnicas usadas para calcular de forma aproximada la probabilidad de ciertos sucesos a partir de la simulación del proceso en estudio mediante variables aleatorias, es decir, mediante la realización de un conjunto de pruebas aleatorias.

La simulación por eventos discretos -SED- es ampliamente utilizada en logística, distribución, fabricación, administración médica, *call-centers*, juegos, etc., por las posibilidades que ofrece tanto en el diseño de soluciones como en la prueba de estos diseños. Relacionada con la teoría de la probabilidad, constituyen sistemas de cálculo para estimar la media o valor estimado de las distribuciones que sirven para modelar el problema [20], y se basa en dos conceptos clave:

1. Los generadores de congruencia lineal -LCG-, que constituyen una forma simple de generar números pseudoaleatorios asociados a una distribución uniforme U [0,1]. Se dicen pseudoaleatorios porque realmente no son aleatorios, ya que disponemos de una regla de generación que permite predecirlos, pero parecen serlo a partir de su distribución. En general los LCG se consideran números pseudoaleatorios de baja calidad por la facilidad con que los test estadísticos muestran que no son aleatorios, sin embargo son fáciles y rápidos de obtener.
2. Y las funciones inversas de la función de distribución acumulada, que permiten convertir los números pseudoaleatorios en distribuciones de interés para el modelo.

1.3.1. El SAEDES

El SAEDES [13] es un algoritmo genérico basado en simulación por eventos discretos, que puede ser utilizado en el análisis de sistemas complejos tiempo-dependientes. En los sistemas reales, los dispositivos que forman parte de ellos están sujetos a fallos,

malfuncionamientos y también a recuperaciones y reparaciones. En estas situaciones, una herramienta que ayude a predecir la disponibilidad del sistema en un tiempo futuro o a proveer de información que mejore los niveles de disponibilidad del mismo resulta interesante para los administradores de estos recursos/dispositivos y para la implantación de políticas de despliegue de servicios sobre ellos.

En muchas situaciones, un sistema puede verse como un conjunto de dispositivos o subsistemas, la disponibilidad del sistema dependerá tanto de la disponibilidad de sus elementos constituyentes como de las relaciones de dependencia entre ellos, su topología. El SAEDES tiene en cuenta estos dos aspectos y evalúa el sistema en su conjunto.

Para la implementación del ILS se disponía de dos versiones del SAEDES en el repositorio, la versión 2a no hace uso de la topología de los nodos en su recorrido y consume mucho tiempo en el análisis de la solución, por lo cual se ha optado por la versión que ha priori podría ofrecer una mayor eficiencia en el cálculo de la disponibilidad.

Esta segunda versión mantiene una colección *Map* con la información de los componentes a evaluar, un objeto *Topology* con la información topológica de los mismos y la lista de recorridos en el grafo del sistema (formes de visitar los nodos del grafo a través de sus aristas [12]) que son críticos para el mantenimiento del mismo, para su disponibilidad. Para el chequeo del estado del sistema, durante la simulación, hace uso del algoritmo *Breadth-first search* (BFS) para encontrar cuando alguno de los recorridos requeridos no está disponible y por tanto, el sistema en su conjunto no estará disponible.

Como generador de números pseudoaleatorios uniformes, el SAEDES, utiliza la clase LFSR113 incluida en el paquete umontreal.iro.lecuyer.rng de la librería SSJ, una librería Java para simulaciones estocásticas desarrollada por Pierre L'Ecuyer de la Universidad de Montreal [21].

El algoritmo asume que la disponibilidad de los nodos ha sido modelada con datos históricos que han permitido asignarles unas distribuciones de probabilidad para los eventos de fallo y reparación que simulan su comportamiento en el tiempo (normalmente distribuciones Weibull, Log-normal, Exponencial y Gamma), y asume

además, que la topología del sistema puede ser representada por un conjunto de recorridos mínimos.

La simulación genera una vida artificial para el servicio (la clase *SaedesSimulation* controla el número de iteraciones (vidas) de la simulación mediante la propiedad *lifeTimes*) que se planifica mediante una cola prioritaria ordenada por tiempo, la cual se inicia con la lista de eventos de fallos programados para los diferentes componentes que forman parte del servicio, un bucle que se repite mientras el tiempo de vida no alcance el límite determinado por la propiedad *endTime* controla la evolución del sistema, durante el cual a cada evento de fallo de un componente se programa un evento de reparación en un tiempo futuro con ayuda de la distribución de reparación asociada al componente y el generador de números pseudoaleatorios, para los eventos de reparación el algoritmo programa un evento de fallo en el futuro calculado con la distribución de probabilidad de fallo del componente. En cada evento de fallo/reparación de un componente se actualiza el estado del sistema para comprobar si está disponible o "caído", el resultado permite actualizar los tiempos de disponibilidad -*timeSysAvail*- o el de no disponibilidad -*timeSysBroken*- dependiendo del estado resultante. Al final de la simulación obtenemos la disponibilidad del sistema para cada una de las iteraciones (vidas) como el cociente timeSysAvail/endTime, el valor medio de este valor para las *lifeTimes* iteraciones será la disponibilidad del sistema durante esta simulación.

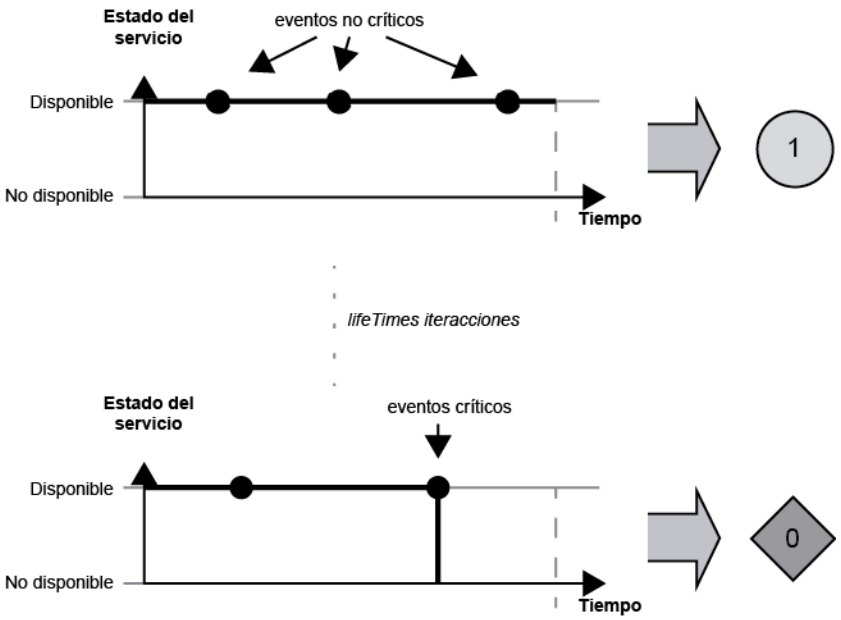


Ilustración 4. Simulación del ciclo de vida

Dado que cada experimento puede ser representado por una variable aleatoria Bernoulli que será igual a 1 si el servicio está disponible durante el tiempo T (*endTime)* requerido e igual a 0 si no está disponible durante ese tiempo (ver la Ilustración 4), se podría estimar la disponibilidad del sistema como el resultado de dividir el número de experimentos en los que el servicio está operativo entre el número total de experimentos realizados como se propone en el artículo original [1] que, como se ha visto, no coincide exactamente con la forma de calcularla en la versión utilizada.

1.3.2. El DiSeASim

Como se ha visto el SAEDES es una herramienta muy potente, no obstante, adolece de algunos problemas de rendimiento por la utilización de algunas estructuras que podrían resultar redundantes o superfluas para nuestros objetivos, o la presencia de un algoritmo BFS para la reconstrucción de la trayectoria en la validación del sistema, por lo cual una versión aligerada de estas cargas puede ser más conveniente para nuestro problema.
El DiSeASim es una implementación basada en el mismo concepto de eventos discretos que el SAEDES en la que se ha prescindido de algunas de estas estructuras redundantes. Utiliza la misma librería de generadores de números pseudoaleatorios, sin embargo, en esta implementación la semilla del generador ha dejado de ser un parámetro del objeto y se obtiene de los milisegundos del reloj del sistema durante la creación del simulador. La función de comprobación del estado del sistema, ya no utiliza un algoritmo BFS, sino que hace una búsqueda de un recorrido con todos sus nodos activos en la lista de los recorridos posibles entre dos nodos dados (conjunto de recorridos mínimos de la topología), si encuentra un recorrido la búsqueda acaba y se marca el sistema como disponible. Ahora disponibilidad se define como el tiempo acumulado durante el cual el sistema está disponible sobre el tiempo total, para su cálculo, el algoritmo simula N tiempos de vida de tiempo T, en cada vida se calcula $Disp=T_{disp}/T$, finalmente la disponibilidad durante la simulación será el valor medio de disponibilidad en cada iteración (vida).

1.3.3. Comparativa y Resultados

Como se ha mencionado, el papel del evaluador de la disponibilidad ocupa una posición central en nuestro algoritmo de búsqueda, su rendimiento tendrá por tanto un gran

impacto en el rendimiento de nuestro ILS ya que será invocado para cada vecino generado.

Los dos algoritmos dependen en gran medida de la topología del servicio que se analiza, aunque hacen un uso diferente de ella en la evaluación del estado del sistema.

También cabe señalar, la gran dependencia de los resultados de ambos, y en general de las simulaciones basadas en distribuciones estadísticas, del modelo estadístico que da soporte al proceso que se pretende simular.

El SAEDES es un algoritmo abierto de uso general lo suficientemente optimizado como para esperar altas tasas de mejora, no obstante el DiSeASim supone una adaptación al problema concreto de calcular la disponibilidad de un servicio representado por una topología de nodos. Los experimentos de comparación del rendimiento no pretende ser exhaustivos, para ser rigurosos, sus resultados no pueden ser totalmente equiparados ya que hacen un uso diferente del concepto de disponibilidad, pero sí nos permiten mostrar esta mejora o no de la eficiencia, necesaria para los objetivos de nuestros experimentos de exploración de soluciones vecinas.

		SAEDES	DISEASIM
Topología	**nº. de nodos**	**Tiempo (s.)**	**Tiempo (s.)**
T131	5	67,45919796	40,357272259
T11213	8	233,02179233	210,36829828
T11511	9	247,40353243	198,70303570
T2433	12	635,38304573	1733,76988787
T13531	13	657,23305091	1568,286166750

Tabla 1. SAEDES vs. DISEASIM

Para la realización de la comparativa, se han diseñado cinco topologías de entre 5 y 13 nodos con diferentes grados de redundancia en uno o varios nodos (Anexo 2). Cada experimento ha repetido 1000 veces una simulación a la que se otorgaba 250 vidas con un tiempo de vida de 1 semana cada una. Los valores recogidos en la Tabla 1 son las medias obtenidas.

Vemos que para topologías con pocos nodos y para topologías con pocas redundancias o concentradas en pocas tareas, el DiseASim tiene un mejor rendimiento que para topologías más complejas donde varias de las tareas presenten un nivel alto de redundancia.

6.4. Implementación del DS-ReliaSim

Desplegar servicios complejos sobre recursos no dedicados, y a menudo con baja disponibilidad, requiere el desarrollo de metodologías que ayuden a la correcta selección de estos recursos que garantice su disponibilidad durante el tiempo suficiente para llevar a cabo las tareas que lo definen, a la vez que limiten el coste del despliegue eliminando las redundancias no necesarias. Estas metodologías tendrán que tratar con redes de gran escala, topologías complejas y modelos estadísticos de fallo/reparación para los diferentes nodos del servicio. El DS-ReliaSim resuelve este problema con un nuevo enfoque, ya que combina una heurística de búsqueda local con una simulación de eventos discretos, que busca encontrar una solución que garantice la disponibilidad del sistema durante el tiempo requerido utilizando nodos relativamente poco disponibles, en general, más baratos. El coste de un recurso está determinado por el coste de sus componentes, CPU, memoria, disco, etc., pero para los fines del método propuesto aceptamos un modelo simplificado que relaciona disponibilidad del recurso y coste, de forma que los nodos menos disponibles son también los más baratos, y al revés, los más caros son los de una mayor disponibilidad.

```
Procedure DS-ReliaSim
Require: nodes; topology; time; iter; threshold
nodes = sortByAvailability(nodes)
currentSol = buildDummySol(nodes; topology)
sysAvailability = DiSeASim(currentSol; topology; time; iter)
currentNodes = extractNodes(currentSol)
nodes = extractNodes(currentNodes; nodes)
currentNodes = sortByCost(currentSol)
auxSysAvail = sysAvailability
while nodes is not empty and auxSysAvail > threshold do
      inNode = selectNextNode(nodes)
      outNode = selectNextNode(currentNodes)
      while outNode exists and cost(inNode) < cost(outNode) do
            newSol = buildShiftedSol(currentSol; inNode; outNode)
            auxSysAvail = DiSeASim (newSol; topology; time; iter)
            if auxSysAvail > threshold then
                  currentSol = newSol
                  break
            end if
            outNode = selectNextNode(currentNodes)
      end while
end while
return currentSol
```

Algoritmo 2. Pseudocódigo del algoritmo DS-ReliaSim propuesto por los autores del artículo *[1]*

Esta metaheurística es del tipo ILS, con una estructura de la vecindad discreta y basada en trayectorias (secuencia de soluciones del espacio de búsqueda) para encontrar una solución óptima a partir de una solución de partida que cumple los objetivos del problema (de mayor coste).

Como se puede ver en el pseudocódigo del algoritmo, el diseño es modular, lo cual permite la utilización de diferentes heurísticas de búsqueda local, así como la de diferentes funciones de simulación de eventos discretos (SAEDES o DiSeASim).

1.4.1. Descripción del servicio: Topología del servicio

Todo servicio distribuido puede verse como un conjunto de tareas relacionadas que deben ser completadas por un conjunto de recursos. Con esta definición, un servicio puede ser representado por un grafo acíclico.

En un escenario real, un usuario solicitará un servicio que debe ser completado con un conjunto de requisitos, que incluye la definición del mismo, las condiciones de finalización, el umbral de disponibilidad que garantice que el servicio puede ser desplegado y ejecutado satisfactoriamente, y la topología del mismo. Cada tarea del servicio exigirá una determinada manera de distribuir tanto los recursos como el tiempo dedicado a su ejecución. Algunas de ellas resultan críticas para la finalización del servicio y será necesario proveer medidas de redundancia y paralelización para llevarla a cabo en el tiempo asignado, es decir, en la topología tendremos nodos suplementarios para asegurar la disponibilidad del servicio durante ese tiempo.

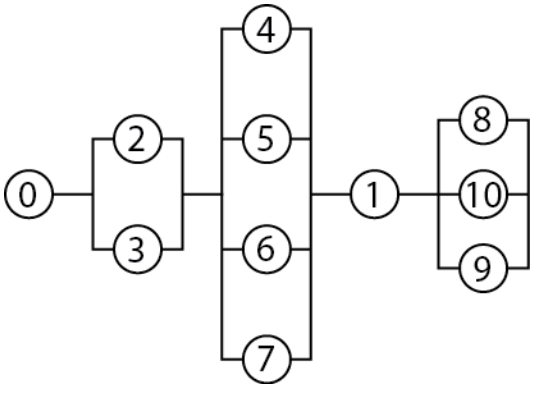


Ilustración 5. Ejemplo de topología de un servicio

En este sentido, la topología describe o representa totalmente al servicio que queremos desplegar. En el algoritmo está representada por un objeto que mantiene la información tanto de los nodos que la constituyen como la de sus relaciones (adyacencias).

La Ilustración 5 muestra el grafo de un servicio con cinco tareas, tres de las cuales tienen un cierto nivel de redundancia lo cual implica la necesidad de utilizar recursos extra para permitir desplegarlo (11 componentes). Los componentes nombrados como 0 y 1, resultan críticos para la funcionalidad del servicio, el fallo de cualquiera de los dos supone la imposibilidad de completar las tareas asignadas al mismo.

1.4.2. Solución inicial

El DS-ReliaSim adopta una aproximación *greedy* para la generación de la solución inicial. El algoritmo empieza ordenando los *n* nodos disponibles por disponibilidad decreciente y construye la solución de inicio seleccionando los *m* primeros nodos de la lista ordenada, los de mayor disponibilidad y más costosos, que son requeridos por la topología del servicio.

La adopción de esta estrategia permite trazar una secuencia descendente en el espacio de búsqueda, que con el tiempo de proceso suficiente, nos lleva a un óptimo local de gran calidad, y resulta, en general, más eficiente que la propuesta de una solución inicial aleatoria que puede necesitar varios reintentos para encontrar una solución de partida que cumpla el criterio de aceptabilidad. La construcción de la solución inicial es en términos computacionales muy rápida y es un buen punto de partida para el proceso de optimización.

1.4.3. Funciones de perturbación

Una función de perturbación puede verse como una operación más o menos aleatoria de la solución actual, de forma que guía la búsqueda a otros pozos de atracción del espacio de búsqueda. En general, será suficiente con que la perturbación nos genere un nuevo estado aceptable desde donde poder aplicar la búsqueda local.

El algoritmo parte de una solución realizable del problema e itera un proceso de búsqueda descendente por el espacio de búsqueda, sustituyendo en cada iteración un nodo de mayor coste por otro más barato de la lista de nodos disponibles. Por tanto, esta función de perturbación se puede considerar estática y de longitud fija, ya que el operador de movimiento que aplica sólo considera el cambio de un determinado número de nodos, 1 en este caso, que es conocido antes de iniciarse el ILS.

Los experimentos realizados muestran que cuando el número de nodos disponibles aumenta significativamente -ver Tabla 3- la convergencia hacia un buen óptimo es muy

lenta, la diversificación propuesta por el operador de perturbación nos lleva a pozos de atracción correspondientes a vecinos muy cercanos, y esta convergencia es aun más lenta cuando buscamos desplegar servicios de baja disponibilidad (bajo coste), debido a que los nodos menos disponibles están al final de la lista de nodos disponibles. En estos casos, incluir funciones de perturbación que tengan en cuenta aspectos aleatorios o componentes adaptativas y que impliquen movimientos más grandes de la solución actual será aconsejable.

¿Cómo ha de ser el cambio en la solución actual provocado por la perturbación? Ya se ha visto que saltos muy grandes en la exploración de vecindades puede llevarnos a reinicios aleatorios y muy pequeños nos devuelve al mínimo local actual, encontrar el valor correcto puede ser difícil y puede depender de otros aspectos del problema combinatorio, como por ejemplo de la condición de aceptabilidad. Sin embargo, una longitud de la perturbación adecuada tiene un impacto directo en la velocidad del algoritmo al permitir la exploración de amplias zonas del espacio de búsqueda con pocas iteraciones.

1.4.4. Tiempo de proceso y nivel de disponibilidad requerido

El tiempo de proceso y las condiciones de aceptación de las soluciones como óptimos, locales y globales, tienen una incidencia sobre la relación intensidad/diversificación de la metaheurística ILS, y por tanto sobre la calidad de la solución.

El DS-ReliaSim itera el proceso de búsqueda mientras las soluciones encontradas tienen una disponibilidad mayor que el umbral requerido, lo que implica limitar la capacidad de exploración del espacio de búsqueda. La estructura de la vecindad usada por la búsqueda local, en la que sólo se substituye un nodo cada vez (d(s, s')=1), y la función de perturbación propuesta, cambiar el nodo más caro por el siguiente nodo disponible, nos lleva a explorar vecindades que sólo difieren en un nodo, refuerza el comportamiento del algoritmo orientado a la intensificación de la búsqueda en detrimento de la diversificación. El algoritmo es eminentemente determinista, a diferencia de los algoritmos basados en poblaciones o de inicio aleatorio, que hacen un mayor esfuerzo exploratorio del espacio de búsqueda, y por tanto más estocásticos.

Estas condiciones del problema tendrán un efecto sobre la convergencia del proceso de iteración y el rendimiento y eficiencia del algoritmo. Un tiempo de simulación bajo nos llevan a soluciones poco significativas o de baja calidad, por otro lado, la búsqueda de

soluciones que tenga una disponibilidad por encima del umbral requerido, no permite la exploración de espacios con malas soluciones pero que podrían tener vecinos que contienen soluciones de buena calidad.

1.4.5. Solución de mínimo coste y alta disponibilidad

El DS-ReliaSim propone una solución inicial y en cada iteración va sustituyendo nodos de la solución por otros más baratos de la lista de nodos disponibles, el proceso acabará cuando no queden más nodos disponibles, o bien, cuando la solución propuesta por el algoritmo al simulador de eventos discretos no alcanza el umbral de disponibilidad deseado -el servicio ha de estar operativo durante el periodo de tiempo especificado con una probabilidad que coincide con este umbral-.

El algoritmo devuelve el conjunto de nodos que permiten desplegar de una forma óptima el servicio, en disponibilidad y coste, o bien nos devuelve la solución inicial si el conjunto de nodos disponibles no permite encontrar una solución de mejor calidad.

Dos elementos controlan las iteraciones que lleva a cabo el algoritmo hasta alcanzar esta solución de mínimo coste y alta disponibilidad, la condición de aceptabilidad, que como se ha visto, depende de la disponibilidad calculada por el simulador y la disponibilidad umbral requerida para el despliegue y que fuerza la búsqueda mientras la disponibilidad es mayor que este umbral, y la condición de finalización que permite que siga el proceso de optimización mientras las soluciones halladas son aceptables y quedan nodos no visitados en la lista de nodos disponibles.

6.5. Experimentos comparativos

Se han realizado básicamente dos tipos de experimentos, uno tendente a determinar los beneficios de partir de una solución inicial construida respecto a la selección aleatoria y en qué circunstancias se obtienen, y otro para comparar las mejoras en la velocidad, posiblemente a costa de una pérdida de calidad en la solución hallada, con el uso de diferentes funciones de perturbación.

Los experimentos han sido realizados sobre un Intel Core i7 con 8GBytes de RAM ejecutando Windows7/64 bits y una máquina virtual Java 6 SE, y disponiendo de los datos modelados a partir del *log* del proyecto SETI (Anexo 1). Se ha repetido cada experimento 50 veces y en cada simulación se han asignado 250 vidas con un tiempo de vida de 1 semana cada una. Las tablas recogen los valores medios obtenidos.

1.5.1. Comparando estrategias de selección de la solución inicial

En estos experimentos se trata de analizar el impacto de la selección de la solución inicial en el rendimiento del DS-ReliaSim. Se han comparado tres estrategias de selección aquí llamadas GREEDY, RANDOM y BINARY.

En la normal o GREEDY se escogen los *m* primeros nodos de la lista de nodos disponibles -los de mayor disponibilidad y mayor coste-, donde *m* son los nodos requeridos por la topología del servicio.

La RANDOM utiliza la selección aleatoria, el algoritmo implementado utiliza un generador de números pseudoaleatorios para escoger el índice en la lista de nodos disponibles a partir del cual se construirá la solución de partida escogiendo los *m* nodos requeridos a partir de este índice, el proceso se repite mientras la solución construida no cumpla el criterio de aceptabilidad.

La BINARY aprovecha la ordenación de la lista de recursos por disponibilidad para escoger un punto de inicio cercano a un buen óptimo utilizando un algoritmo de búsqueda binaria [22]. La búsqueda binaria empieza con los nodos del inicio de la lista, si la solución construida no es aceptable devuelve como solución inicial la construida con los *m* primeros nodos -decae a greedy-, y si es aceptable salta al punto central de la lista, (N-m)/2, donde *N* es el número total de nodos disponibles y *m* los requeridos por el servicio, vuelve a comprobar si la solución construida a partir de este punto es aceptable, si lo es nos quedamos con la mitad inferior de la lista y si no es aceptable nos quedamos con la mitad superior para después saltar al centro de la lista escogida iniciándose de esta manera un proceso iterativo que acaba cuando el número de nodos seleccionados es menor que *m*. Por tanto, en cada iteración el número de nodos elegidos se reduce a la mitad, lo que permite buscar entre cientos de miles de nodos un buen punto de partida para construir la solución inicial con unas pocas iteraciones.

En la Tabla 2 se han recogido los valores medios de Iteraciones, Tiempo y Coste para cada uno de los 3 algoritmos de selección de la solución de inicio:

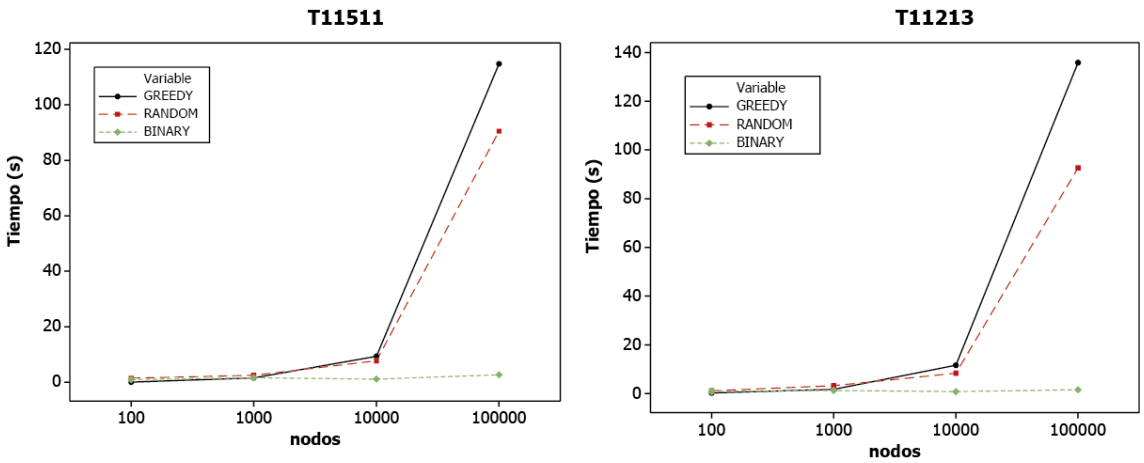
- Iter, número de iteraciones que realiza el ILS en completar la búsqueda, entre paréntesis se han indicado las iteraciones realizadas por el método de selección para encontrar un punto de partida que cumpla la condición de aceptabilidad.
- Tiempo, tiempo total en segundos que tarda cada versión del algoritmo en llegar a una solución óptima.

- Coste, recoge el coste de la solución encontrada. Esta columna nos permitirá comparar la calidad de las soluciones encontradas a partir de cada método.

	GREEDY			RANDOM			BINARY		
Topología	**Iter.**	**Tiempo (s)**	**Coste**	**Iter.**	**Tiempo (s)**	**Coste**	**Iter.**	**Tiempo (s)**	**Coste**
100 COMPONENTES									
T11511	6 [0]	0,02487	8,769	10 [13]	1,43345	8,773	7 [5]	0,89718	8,776
T11213	7 [0]	0,28864	7,789	8 [11]	1,11350	7,823	5 [5]	0,90832	7,841
T131	9 [0]	0,15011	4,860	8 [7]	0,40169	4,863	3 [6]	0,52209	4,869
T13531	10 [0]	1,04985	12,346	11 [9]	4,23980	12,544	19 [4]	3,83808	12,542
T2433	25 [0]	4,99099	9,621	16 [3]	3,48347	10,369	4 [4]	2,94605	10,084
1000 COMPONENTES									
T11511	63 [0]	1,52704	8,876	40 [10]	2,47135	8,873	16 [8]	1,58379	8,855
T11213	71 [0]	1,72748	7,857	45 [16]	3,24636	7,858	6 [9]	1,27070	7,861
T131	90 [0]	1,01671	4,872	49 [13]	1,34658	4,871	4 [9]	0,52992	4,871
T13531	90 [0]	5,98569	12,634	58 [9]	9,66848	12,634	12 [8]	2,83182	12,645
T2433	279 [0]	38,77223	9,703	104 [3]	19,76807	10,164	1 [8]	5,86257	9,607
10000 COMPONENTES									
T11511	748 [0]	9,31221	8,889	395 [11]	7,69269	8,889	16 [12]	1,07194	8,882
T11213	853 [0]	11,64717	7,870	391 [12]	8,36750	7,870	4 [12]	0,81456	7,863
T131	1058 [0]	8,58373	4,879	548 [10]	6,42416	4,880	3 [12]	2,88857	4,872
T13531	1059 [0]	48,20962	12,681	575 [7]	38,53083	12,685	3 [11]	20,45044	12,667
T2433	2964 [0]	472,27431	9,750	932 [2]	195,64479	10,381	1 [11]	8,23021	9,660
100000 COMPONENTES									
T11511	7125 [0]	114,78243	8,892	3826 [18]	90,48473	8,892	17 [15]	2,64473	8,885
T11213	8005 [0]	135,82365	7,874	3986 [11]	92,62886	7,873	2 [15]	1,56827	7,858
T131	9509 [0]	98,40273	4,886	5060 [8]	72,20292	4,885	4 [16]	1,16537	4,873
T13531	9570 [0]	617,48884	12,701	3895 [8]	394,81732	12,699	3 [14]	7,99224	12,667
T2433	28830 [0]	5627,99612	9,803	11008 [2]	2564,83357	10,051	1 [15]	7,49341	9,659

Tabla 2. Resultados de comparar el ILS con inicio greedy, random y binary

Las siguientes gráficas muestran la variación de los tiempos de simulación en función del número de componentes disponibles para cada una de las cinco topologías probadas:



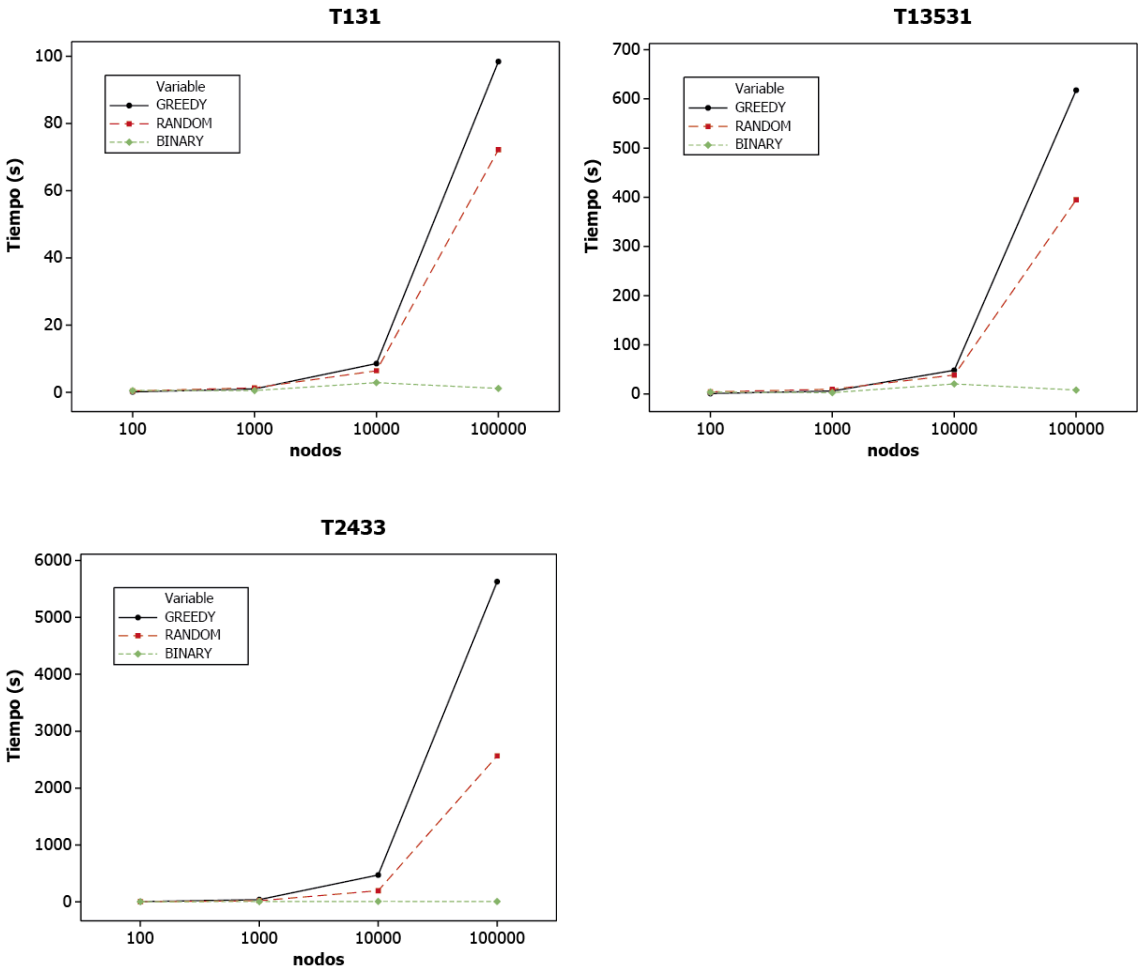


Ilustración 6. Gráficas de las tres estrategias para generar la solución inicial

Los resultados de la Tabla 2 muestran que en general la estrategia de partir de una solución construida es mejor que la de inicio aleatorio, la diferencia es especialmente relevante cuando el número de componentes es pequeño. Para muestras grandes con una disponibilidad uniformemente distribuida la selección aleatoria tiene una probabilidad mayor de partir de una solución cercana al óptimo y por tanto converger más rápidamente hacia él. En cuanto al rendimiento, construir la solución a partir de un punto cercano a un óptimo localizado mediante una búsqueda binaria se sitúa a medio camino entre los dos métodos anteriores. Si nos fijamos en la columna de coste vemos que, para pocos componentes disponibles, las soluciones aportadas por la estrategia propuesta por el DS-RealiaSim son de mejor calidad al permitir el despliegue del servicio con un coste menor, las diferencias en este apartado entre los diferentes métodos se van igualando a medida que aumenta el número de nodos.

1.5.2. Comparando diferentes funciones de perturbación

Se han realizado experimentos con diferentes funciones de perturbación, en la tabla siguiente se recogen sólo los resultados con dos de ellas, junto a los de la versión base, que aplican tipos diferentes de perturbación. Se han nombrado como $P_{DEFAULT}$, P_{FIXED} y $P_{ADAPTIVE}$.

En la implementación normal en cada iteración del proceso se escoge el siguiente nodo más barato de la lista de nodos disponibles para sustituir a alguno de los nodos de la solución actual, de modo que el salto o movimiento provocado por la perturbación en la solución actual es $\delta = 1$ -siendo δ la longitud del salto- .

En el método de perturbación llamado P_{FIXED} se escoge un nodo de la lista alejado un número δ que coincide con el número m de componentes requeridos por la topología del servicio, $\delta = m$, en este sentido constituye un salto constante y fijado antes de iniciarse el proceso de búsqueda como en el caso anterior.

Esta perturbación reduce el número de saltos requeridos para llegar a un buen óptimo en una proporción equivalente a este número m de nodos al aprovechar la ordenación por disponibilidad descendente -y coste descendente- de la lista de nodos disponibles y probar un nodo de cada m. Es de esperar que las topologías que requieran un mayor número de nodos se vean beneficiados por esta estrategia.

El segundo método de perturbación utilizado, $P_{ADAPTIVE}$, recalcula el salto δ en cada iteración pudiendo variar entre 1 y K, siendo K un valor que depende del número de nodos N, de la diferencia entre la disponibilidad de la solución actual y el umbral de disponibilidad requerido y de un parámetro ad-hoc que limita a 5000 las iteraciones cuando N=200000 y a 40 cuando N=200, de esta manera el salto δ aplicado tiende a 1 a medida a que la solución se aproxima a un buen óptimo (decae hacia el comportamiento de la versión por defecto), siendo siempre lo suficientemente pequeño para minimizar la posibilidad de que se aleje demasiado de un buen óptimo (para N=100000 y m=9 el salto en las primeras iteraciones es $\delta = 39$). La fórmula de cálculo es la siguiente:

$$\delta = \max(1.0,\ \min\left(\frac{N}{0.0\widehat{248} * N + 35.\widehat{035}}, (N-2) * \frac{|dif| - 0.01}{2}\right))$$

Donde N es el número de recursos disponibles y $|dif|$ es el valor absoluto de la diferencia entre la disponibilidad de la solución actual y el umbral requerido.

La Tabla 3 recoge los datos obtenidos con cada una de las estrategias, donde las columnas Tiempo y Coste se han de interpretar en el mismo sentido que en el apartado anterior.

	$P_{DEFAULT}$		P_{FIXED}		$P_{ADAPTIVE}$	
Topología	**Tiempo (s)**	**Coste**	**Tiempo (s)**	**Coste**	**Tiempo (s)**	**Coste**
100 componentes						
T11511	0,02487	8,769	0,04088	7,605	0,02259	8,768
T11213	0,28864	7,789	0,03136	7,612	0,27981	7,792
T131	0,15011	4,860	0,01551	4,726	0,14658	4,860
T13531	1,04985	12,346	0,85614	8,855	1,03132	12,346
T2433	4,99099	9,621	1,99979	9,001	4,90872	9,621
1000 componentes						
T11511	1,52704	8,876	0,36047	8,810	0,92106	8,711
T11213	1,72748	7,857	0,08926	7,820	0,06628	7,775
T131	1,01671	4,872	0,23310	4,863	0,06064	4,850
T13531	5,98569	12,634	0,23915	12,179	0,60821	12,033
T2433	38,77223	9,703	2,09159	9,541	0,84953	9,598
10000 componentes						
T11511	9,31221	8,889	0,62708	8,879	0,37929	8,865
T11213	11,64717	7,870	1,39869	7,863	0,28252	7,857
T131	8,58373	4,879	2,42070	4,874	0,49316	4,878
T13531	48,20962	12,681	67,93930	12,628	1,22202	12,574
T2433	472,27431	9,750	78,44469	9,677	32,63122	9,687
100000 componentes						
T11511	114,78243	8,892	22,75518	8,888	5,69924	8,893
T11213	135,82365	7,874	10,74379	7,869	6,99389	7,874
T131	98,40273	4,886	15,43754	4,880	6,02749	4,884
T13531	617,48884	12,701	41,66551	12,682	28,07036	12,698
T2433	5627,99612	9,803	403,47195	9,732	183,9753	9,758

Tabla 3. Rendimiento del ILS con 3 diferentes funciones de perturbación

Estos valores muestran que a partir de unos miles de nodos la mejora de velocidad obtenida por las funciones de perturbación que amplían la vecindad a explorar es muy significativa con respecto a la implementación por defecto, y que entre estas la versión variable a adaptativa ofrece un mejor margen de mejora especialmente para el caso de topologías poco complejas o con pocos nodos requeridos. No obstante su uso restringe la generalidad del planteamiento del algoritmo al introducir correcciones dependientes del problema bajo optimización y requerirá el ajuste de los parámetros que se utilizan para evitar que los movimientos en las soluciones exploradas nos lleven a mínimos malos o poco adecuados para el propósito de nuestro problema.

A continuación se tienen las gráficas de tiempos para las distintas funciones de perturbación y topologías probadas que muestran lo comentado:

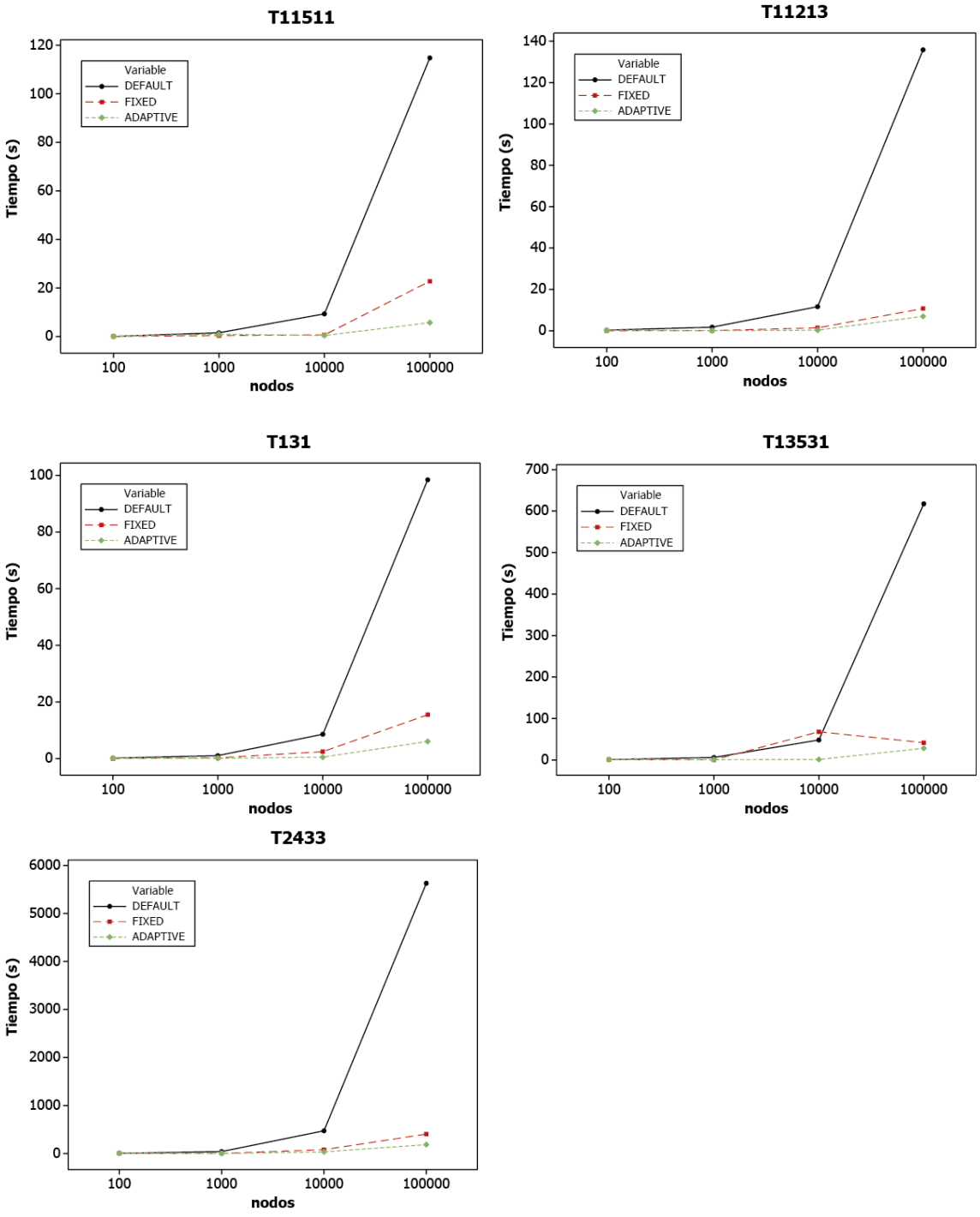


Ilustración 7. Gráficas comparativas de las tres funciones de perturbación

6.6. Alternativas al DS-ReliaSim

A modo de conclusión, podemos afirmar que el DS-ReliaSim es una implementación de una metaheurística que es conceptualmente simple a la vez que sencilla y de propósito

general. La estructura modular del algoritmo permite múltiple adaptaciones y optimizaciones, cambiando la heurística de búsqueda local o el simulador de eventos discretos que evalúa la disponibilidad, por ejemplo.

Algunas mejoras importantes del rendimiento del algoritmo se obtienen cuando cambiamos la función de perturbación permitiendo saltos mayores, exploración de vecinos alejados por más de un nodo. Esta característica es interesante cuando la lista de nodos disponibles es muy grande comparada con la lista de nodos requeridos por la topología del servicio, no obstante, las soluciones suelen ser de menor calidad -más alejadas de un óptimo global- por la pérdida de intensidad en la exploración. El ILS hace un muestreo correcto del espacio de búsqueda cuando estas perturbaciones no son muy grandes ni muy pequeñas. Si son muy pequeñas no nos aportan nuevas soluciones de partida y si son muy grandes se cae en una búsqueda aleatoria. En algunas situaciones puede ser necesaria la incorporación de información específica del problema en las funciones de perturbación.

La Tabla 3 recoge los tiempos consumidos por el DS-ReliaSim implementado con tres funciones de perturbación distintas en encontrar una solución óptima para las cinco topologías en estudio en las que se requería una disponibilidad del 95%.

Los resultados confirman plenamente lo comentado, el algoritmo puede ser optimizado en cuando a su rendimiento y velocidad utilizando funciones de perturbación que mejoren su capacidad de diversificación en la exploración, sin perder calidad en las soluciones, la columna de costes muestran un buen ajuste entre las 3 variaciones.

El ILS con búsqueda binaria aprovecha la ordenación de los componentes por disponibilidad para encontrar una solución inicial muy cercana o igual a un buen óptimo.

7. Ejemplos de aplicación

El campo de la computación voluntaria abre una opción a la computación de alto rendimiento a diferentes tipos de comunidades, grandes y pequeñas, sin la necesidad de las grandes inversiones económicas que supone contratar servicios *Cloud* o la compra de servidores dedicados. La propuesta de reutilizar los recursos de los que ya se dispone en los equipos de escritorio ha atraído ya a una gran comunidad de interesados por las

posibilidades que ofrece a las aplicaciones que requieren de una gran potencia de cálculo. Campos como la cosmología, donde ya están en marcha proyectos como SETI@Home centrado en la búsqueda de vida extraterrestre, o el Einstein@Home que busca ondas gravitacionales, o el proyecto Folding@Home centrado en el análisis del plagado de las proteínas en el campo de la biología, aglutinan ya a grandes comunidades de usuarios voluntarios que demuestran las grandes posibilidades de la computación de alto rendimiento en este paradigma, donde la anexión de un nuevo recurso supone una aumento de la capacidad de cálculo del sistema, por tanto, no es necesario grandes comunidades (100000 usuarios o más) de voluntarios para beneficiarse de estas capacidades de cálculo.

7.1 *3D Rendering Service* para bioinformáticos

El campo de la bioinformática y la medicina ofrece grandes posibilidades para el *Volunteer Computing*, donde algunos de los problemas que tienen en sus laboratorios los grupos de investigación, como la aplicación de nuevas terapias en el tratamiento del cáncer, el alineamiento de las diferentes proteínas que forman las heteroproteínas y el alineamiento estructural de las mismas, la expresión génica, las interacciones proteína-proteína, etc. requieren de sistemas de alta computación para analizar o llevar a cabo las simulaciones que permitan extraer información o modelar el proceso.

Estos equipos de trabajo, en general no muy numerosos, pero con grandes necesidades de cómputo se verían beneficiados de servicios de tratamiento de imágenes de muy alta resolución o de renderizado de modelos 3D, tanto para obtener proyecciones de alta resolución como animaciones del proceso.

Como ejemplo de aplicación del despliegue de servicios en estos entornos distribuidos se propone el *3D Rendering service*, su puesta en marcha supone una serie de pasos:

1. Un investigador, usuario de la comunidad, carga el modelo en el sistema, para lo cual hace servir un lenguaje de descripción de objetos estándar o bien solicita la inclusión de objetos de la base de datos -la base de datos contiene modelos de proteínas ya utilizados anteriormente-, y solicita un determinado nivel de cómputo o hilos de ejecución -paralelización del cálculo-, es decir, define la topología requerida.
2. El Manager del sistema propone una solución inicial a partir de la lista de recursos disponibles en la comunidad, y procede a optimizar la selección de nodos.

3. El Manager despliega la solución optimizada, asigna tareas a los nodos del renderizador y coordina su ejecución. El resultado del proceso se deposita en la base de datos distribuida del sistema, y queda a disposición del servicio web para ser consultada por el usuario.
4. El servidor web es el encargado de recibir las peticiones de consulta del usuario y de enviar los resultados al mismo.

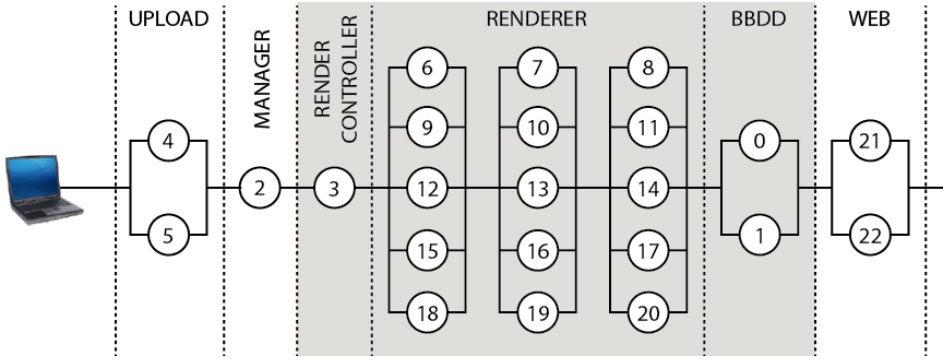


Ilustración 8. Topología del 3D Render Service

Asumimos que el usuario incluye la topología del servicio -parte de sombreada de la Ilustración 6-, fija un umbral para el nivel de disponibilidad del servicio, el 95%, así como la lista de componentes críticos del servicio -el Render Controller- cuya indisponibilidad provocará el fallo del servicio. Además, para que el servicio pueda completar el proceso de renderizado, al menos un componente de cada columna de nodos del Renderer ha de estar operativo hasta el final del proceso.

8. Conclusiones

En el artículo base de este TFC se propone un algoritmo híbrido, el DS-ReliaSim, que utiliza una heurística de búsqueda local innovadora y un algoritmo de validación de soluciones basado en simulación de eventos discretos para resolver el problema planteado de desplegar servicios complejos a coste eficiente sobre recursos distribuidos no dedicados, heterogéneos y propios de los equipos de sobremesa.

El algoritmo ofrece soluciones que responde a las necesidades del problema, la estrategia de la heurística se ha demostrado acertada, ya que parte de una solución de alto coste y alta disponibilidad y va construyendo soluciones cada vez más baratas mientras se

mantenga la disponibilidad calculada por encima del umbral requerido en la descripción del servicio.

La selección de la solución inicial tiene un gran efecto en el rendimiento del algoritmo cuando el número de nodos es grande, ya que aumenta significativamente el número de comparaciones que realiza el algoritmo hasta llegar a la de mínimo coste. Los experimentos comparativos muestran que para un número por encima de miles de nodos usar reinicio aleatorio o la búsqueda binaria reduce de forma apreciable el tiempo de proceso al construir la solución de inicio a partir de nodos más baratos pero que cumplen la condición de aceptabilidad. Cabe destacar las mejoras obtenidas por la búsqueda binaria en estos casos.

La transformación de la solución en cada iteración afecta a un solo nodo, se sustituye el nodo de mayor coste en la solución por el siguiente nodo más barato disponible, lo que lo convierte en un algoritmo con un gran esfuerzo de intensificación en la búsqueda, lo que por un lado garantiza soluciones de muy alta calidad, próximas a óptimos globales cuando no coincidentes, y por otro lado, limita el esfuerzo de diversificación, lo cual tiene en un gran impacto en la convergencia hacia la solución, especialmente cuando el número de nodos disponibles es muy alto, en estos casos usar otras funciones de perturbación que impliquen saltos más grandes en la selección del nodo sustituto parece recomendable. Un método adaptativo en la perturbación parece ser muy prometedor, aunque podría requerir incluir información específica del problema.

9. Trabajos futuros

La computación voluntaria está rápidamente creciendo tanto en el número de equipos que participan en ella como en las capacidades de almacenamiento y cálculo que poseen estos equipos. No obstante, la disponibilidad de estos equipos esta sujeta a altos niveles de variabilidad debida a las características propias de este paradigma. La heterogeneidad de los recursos que la forman, su distribución geográfica, las políticas de acceso y mantenimiento de sus propietarios, etc. determina esta variabilidad y conocer o modelar su comportamiento y disponibilidad resultan claves para el despliegue de servicios sobre ellos. El usuario que requiere el servicio espera una respuesta, el gestor del sistema debe seleccionar la mejor combinación de nodos que permitan la disponibilidad del mismo

durante el tiempo necesario para llevar a cabo todas las tareas, para lo cual necesitará de la información correcta del estado actual de los nodos y de su comportamiento futuro. Esta información se puede obtener de los informes de estado que recoge el sistema -log- y modelarla estadísticamente asignando funciones de distribución de fallo/reparación, y debería tener en cuenta además otros estados del nodo, no sólo activo o no activo como en nuestra aproximación.

Otro tema abierto es el de la asignación del coste a los recursos, en este trabajo se ha adoptado la simplificación de equiparar disponibilidad y coste, de forma que los equipos de mayor disponibilidad son también los más caros -y al revés-. Seria interesante la utilización de funciones de coste de los nodos más realistas que tuvieran en cuenta la CPU, la memoria, el ancho de banda, el disco duro, etc. La metodología propuesta por el DS-ReliaSim permite el uso de diferentes funciones de coste.

Este TFC se centrado en la implementación de un algoritmo que selecciona el mejor conjunto de nodos y de menor coste para desplegar un servicio representado por una topología con una disponibilidad fijada -normalmente el 90% o 95%-. Sin embargo, en función del conjunto de nodos disponibles no siempre será posible alcanzar este nivel umbral de disponibilidad requerido debido principalmente a la complejidad del servicio o de sus redundancias y a la baja disponibilidad individual de los nodos disponibles. Una línea de trabajo interesante seria encontrar la topología mínima de un servicio que permite desplegarlo en una determinada lista de nodos.

La conclusión final es, que el DS-ReliaSim no sólo responde al problema planteado aplicando una nueva metodología, hasta donde alcanza mi conocimiento, sino que además permite que pueda ser adaptado a cualquiera de estos escenarios abiertos.

Bibliografía

[1] P. Ángel A. Juan, «Deploying cost-eficient and highly-avaible services over distributed systems with heterogeneus and non-dedicated resources,» *Varias,* 2012.

[2] B. R. a. M. J. Lewis, «Multi-State Grid Resource Availability Characterization».

[3] University of Wisconsin-Madinson, «Condor - High Throughput Computing,» [En línea]. Available: http://www.cs.wisc.edu/condor/description.html. [Último acceso: 10 11 2012].

[4] Moore's Law, «Moore's Law and Intel Innovation,» 2003. [En línea]. [Último acceso: 3 Mayo 2012].

[5] G. Gilder, Gilder's law on network performance. Telecosm: The World After, Touchstone Books, 2002.

[6] F. Berman, G. C. Fox and a. A. J. G. Hey, Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2003.

[7] M. Baker and R. B. y. D. Laforenza, "Grids and Grid Technologies for wide-area distributed computing," *Software:Practice and Experience (SPE),* vol. 32, no. 15, pp. 1437-1466, 2002.

[8] L. F. G. Sarmenta, «Bayanihan: Web-Based Volunteer Computing Using Java,» Computer Architecture Group, Cambridge, 1997.

[9] A. Baratloo, M. Karaul y Z. M. K. a. P. Wyckoff, «Charlotte: Metacomputing on the Web,» Bell Communications Research, New York, 1998.

[10] D. P. Anderson, C. Christensen y B. Allen, «Designing a Runtime System for Volunteer Computing».

[11] C. Christensen, T. Aina y D. Stainforth, «The Challenge of Volunteer Computing With Lengthy Climate Model Simulations,» Department of Atmospheric Physics University of Oxford.

[12] R. Masià, J. Pujol, J. Rifà i M. Villanueva, «Fonaments de grafs,» de *Matemàtica Discreta*, FUOC, 2006.

[13] J. Faulin, Á. A. Juan, C. Serrat y V. Bargueño, «Predicting Availability Functions in time-dependent complex systems with SAEDES simulation algorithms,» 2007.

[14] T. Weise, "Global Optimization Algorithms -Theory and Application-," http://www.it-weise.de, 2009.

[15] E.-G. Talbi, METAHEURISTICS From Design To Implementation, University of Lille: A John Wiley & Son, Inc., Publication, 2009.

[16] H. R. Lourenço, O. C. Martin y T. Stützle, Iterated Local Search, 2001.

[17] P. Jermini, «Building a Condor Desktop Grid,» *Manager,* 2009.

[18] X. Ren, S. Lee, R. Eigenmann y S. Bagchi, «Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation,» de *J Grid Computing*, Springer, 2007, p. 23.

[19] B. Rood y M. J. Lewis, «Multi-State Grid Resource Availability Characterization,» Departament of Computer Science State University of New York, Binghamton, NY, 13902.

[20] T. T. Allen, Introduction to Discrete Event Simulation and Agent-based Modeling, Columbus: Springer, 2011.

[21] P. L'Ecuyer, «SSJ : A Java library for a stochastic simulation - API specification,» Université de Montréal, [En línea]. Available: http://www.iro.umontreal.ca/~simardr/ssj/doc/html/overview-summary.html. [Último acceso: 14 11 2012].

[22] A. Drozdek, Estructura de Datos y Algoritmos con Java, Cengage Learning Editores, 2007.

[23] P. Harold Castro, «Grid computing. promesa de los sistemas distribuidos,» *Revista Sistemas,* pp. 45-57, 2010.

[24] E. V. a. M. Caserta, «Metaheuristics: Intelligent Problem Solving,» de *Hybridizing Metaheuristics and Mathematical*, Springer.

[25] C. Blum, M. J. B. Aguilera, A. Roli y M. Sampels, Hybrid Metaheuristics An Emerging Approach to Optimization, Springer, 2008.

Anexos

Anexo 1. Tabla de datos (39 primeros registros)

Id	Distribución fallos		Distribución recuperación		Coste
100416828	Exponential	115625,45880500	Exponential	272354,09417500	0,298019465
100075173	Exponential	3046,68900003	Exponential	670986,76158800	0,004520086
100422568	Exponential	4270,26764800	Exponential	43684,12751870	0,089048514
100408768	Exponential	256506,03309600	Exponential	71561,32933330	0,781870014
100224447	Exponential	13359,43663570	Exponential	46150,21004850	0,22449195
100203236	Exponential	256217,86037900	Exponential	22372,82320630	0,919692852
100233682	Exponential	2308,91252096	Exponential	17080,28775200	0,119082401
100205414	Exponential	7391,69583061	Exponential	2817,91438563	0,723993931
100252332	Exponential	623060,25420000	Exponential	3567,71094736	0,994306493
100052112	Exponential	17397,33685300	Exponential	18890,13992540	0,479430878
100117943	Exponential	1082,07977825	Exponential	8295,56195797	0,115389328
100427449	Exponential	21231,93753360	Exponential	62916,57760340	0,252315059
100389873	Exponential	471639,80027500	Exponential	14765,11389750	0,969644398
100317715	Exponential	1347678,70191000	Exponential	59939,66819050	0,957417671
100021662	Exponential	278743,41202800	Exponential	521108,94874300	0,348493579
100055671	Exponential	638046,65160000	Exponential	285048,32831000	0,691203685
100355392	Exponential	13944,25589740	Exponential	97517,20423680	0,125103833
100067629	Exponential	4164,64019879	Exponential	10391,01236520	0,286118412
100309685	Exponential	16206,74124960	Exponential	48278,03818150	0,251326614
100327214	Exponential	203760,84725400	Exponential	5798,24592481	0,972331213
100284574	Exponential	242995,51616000	Exponential	521,59458333	0,997858078
100265331	Exponential	3853,57851298	Exponential	5024,79711673	0,434040941
100208668	Exponential	50521,55500000	Exponential	19335,02857150	0,723218234
100422926	Exponential	2500,98976897	Exponential	6393,93298228	0,281170488
100413314	Exponential	37023,89045600	Exponential	23906,20465620	0,607645374
100019206	Exponential	6291,19310615	Exponential	3971,37535509	0,613023253
100066658	Exponential	343,08671803	Exponential	38,86823669	0,89823869
100347816	Exponential	504351,14244900	Exponential	5579,30583825	0,989058693
100094952	Exponential	20603,01130570	Exponential	42140,33023720	0,328369685
100220664	Exponential	2511,86933332	Exponential	10232,37112500	0,197098394
100238799	Exponential	3660,70383621	Exponential	51002,32552270	0,06696855
100101910	Exponential	60449,50510820	Exponential	26263,84141430	0,697118812
100135148	Exponential	50005,98822480	Exponential	7418,43334527	0,870813965
100212088	Exponential	8468,91440370	Exponential	3254,21733146	0,722410581
100030591	Exponential	57495,82661750	Exponential	27407,54555500	0,677191319
100325958	Exponential	36418,22924000	Exponential	22125,13246940	0,622072737
100315281	Exponential	79772,87750010	Exponential	113254,84080000	0,413271618
100249151	Exponential	11163,68585110	Exponential	9541,16263550	0,539182205
100417317	Exponential	7055,05783332	Exponential	11068,58854550	0,389273642

Anexo2. Topologías utilizadas

Para las pruebas de comparación y análisis se han diseñado las 5 topologías de servicio de la ilustración inferior. Responden a diferentes complejidades tanto en el número de tareas implicadas en el servicio como en el nivel de redundancia que presentan algunos de los nodos.

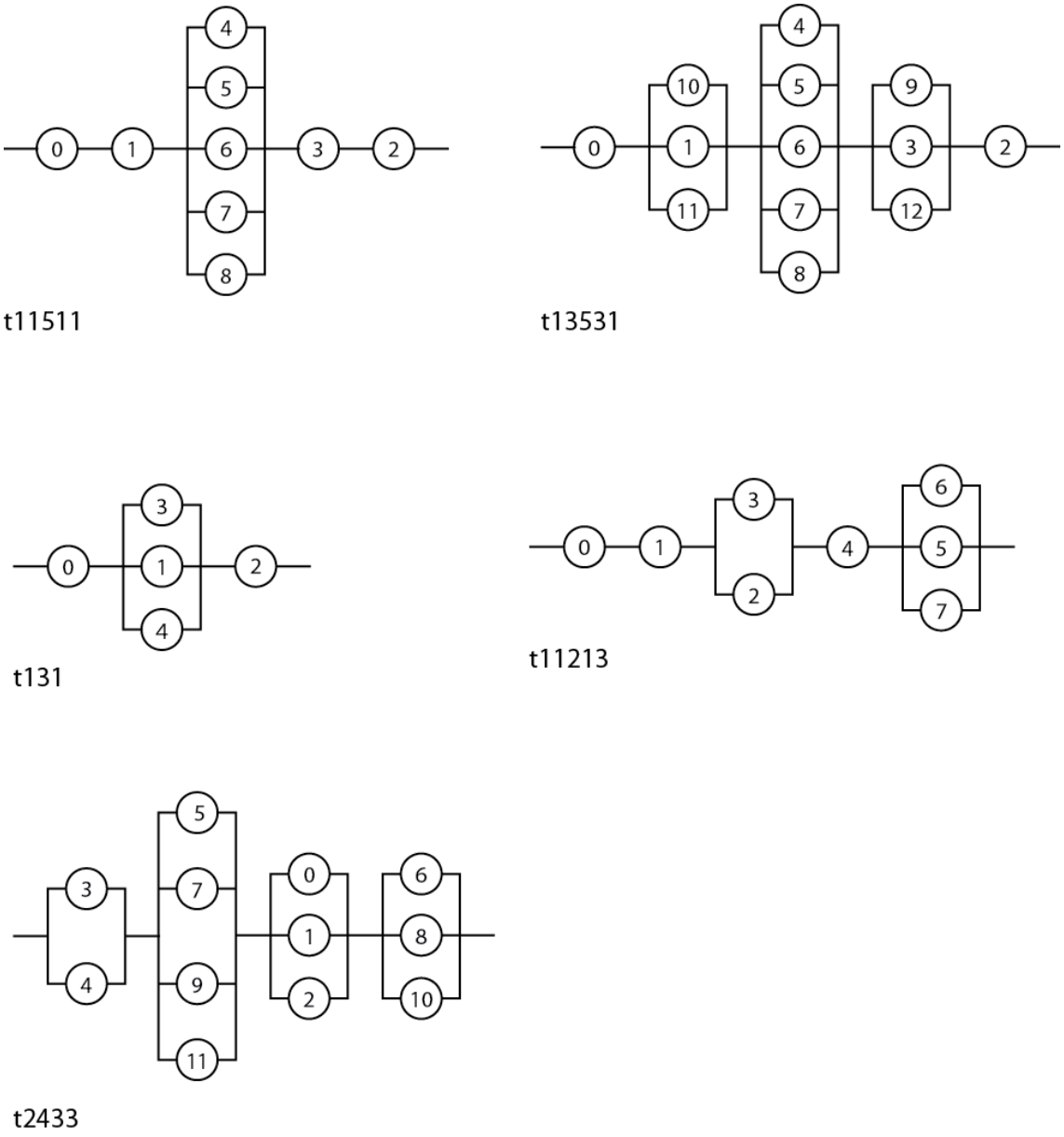


Ilustración 9. Grafos de las topologías analizadas