INSTITUTO TECNOLÓGICO
DE CIUDAD MADERO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"Evaluación de Estrategias de Mejora del Desempeño de Metaheurísticos Aplicados a BPP Vía Diagnóstico Visual"

TESIS PARA OBTENER EL GRADO DE:
Maestro en Ciencias en Ciencias de la Computación

PRESENTA:
Ing. Norberto Castillo García

DIRECTOR DE TESIS:
Dra. Claudia Guadalupe Gómez Santillán

CO-DIRECTOR DE TESIS:
Dra. Laura Cruz Reyes

Cd. Madero, Tamaulipas, México. Noviembre de 2011

"2011, Año del Turismo en México"

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

SECRETARÍA DE EDUCACIÓN PÚBLICA

Cd. Madero, Tamps; a 16 de Noviembre de 2011

OFICIO No.: U5.347/11
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

ING. NORBERTO CASTILLO GARCÍA
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"EVALUACIÓN DE ESTRATEGIAS DE MEJORA DEL DESEMPEÑO DE METAHEURÍSTICOS APLICADOS A BPP VÍA DIAGNÓSTICO VISUAL"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"Por mi Patria y por mi Bien"

Ma. Yolanda Chávez Cinco

M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

S.E.P.
DIVISION DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
I T C M

c.c.p.- Archivo
c.c.p.- Minuta

MYCHC 'NMCO ' jar

Ave. 1°. de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Tels. (833) 3 57 48 20, Fax: (833) 357 48 20, Ext. 1002, email: itcm@itcm.edu.mx
www.itcm.edu.mx

Declaración de Originalidad

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derecho de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones.

Además, en caso de infracción a los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y relevo de ésta a mi director y codirector de tesis, así como al Instituto Tecnológico de Cd. Madero y sus autoridades.

Noviembre de 2011, Cd. Madero, Tamaulipas.

_____

Ing. Norberto Castillo García

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Índices de Algoritmos

Capítulo 1

Introducción

La investigación sobre la metodología de experimentación computacional está creciendo rápidamente, y tiene como objetivo promover que los experimentos sean relevantes, correctos, replicables y que produzcan conocimiento para mejorar el desempeño de los algoritmos metaheurísticos [McGeoch 00].

Trabajos recientes en el área de las ciencias computacionales han demostrado que no es suficiente saber que un algoritmo es superior a otro en la solución de un conjunto particular de instancias de un problema. También es de interés contar con explicaciones del comportamiento observado. Los problemas del mundo real son muy variados y las técnicas utilizadas para solucionarlos deben de ser, preferiblemente, adaptadas a la naturaleza del problema [Rivera 09].

Trabajos anteriores analizaron el desempeño de los algoritmos bajo el concepto de caja negra, en el que los aspectos a analizar fueron, principalmente, la adaptabilidad al problema y del algoritmo, y la competitividad de los resultados obtenidos. Sin embargo, se han presentado necesidades en el uso y aplicación de algoritmos complejos a problemas del mundo real; por ejemplo, conocer la naturaleza exploratoria e interpretación confirmatoria de las relaciones presentes entre los elementos que intervienen en el desempeño de los algoritmos metaheurísticos.

Esta situación hace necesario analizar no sólo el problema a resolver, sino también saber qué sucede en el interior del algoritmo; para esto es necesario encontrar relaciones

entre el problema y el comportamiento observado, además de conocer el impacto de estas relaciones en el desempeño obtenido.

Los problemas de asignación, por ejemplo, el Bin-Packing, son difíciles de resolver ya que su solución crece exponencialmente con el tamaño del problema, por esta razón son considerados como NP-Duros [Cruz 04]. Para estos problemas se cree que no existen algoritmos exactos de solución cuyo tiempo de ejecución no aumente exponencialmente con el tamaño del problema. Hay dos formas de resolver los problemas NP Duros: la primera es usar métodos exactos que requieren tiempo computacional exponencial y la segunda, la cual se utiliza en la práctica para los problemas de gran tamaño, es emplear los métodos no exactos llamados metaheurísticos, los cuales producen soluciones en un periodo de tiempo razonable pero no garantizan encontrar siempre el resultado óptimo [Quiroz 09].

Cuando se resuelven problemas complejos como los mencionados anteriormente, el desempeño de algoritmos metaheurísticos depende de muchos factores, por lo que un mal diseño de éstos puede resultar en un desempeño pobre. No existen reglas que indiquen cómo diseñar apropiadamente los metaheurísticos. Es por esto que los diseñadores de estos métodos toman hasta el 90% del tiempo de solución en ajustarlos [Gómez 09].

En trabajos previos [Pérez 07, Quiroz 09] se utilizaron herramientas para visualizar e identificar áreas de mejora en los algoritmos metaheurísticos, pero éstas son de alcance limitado. Entre sus limitaciones se destaca lo siguiente: no satisfacen todos los requerimientos de análisis, además de que el análisis visual no está relacionado con el análisis estadístico, es decir, operan de manera independiente.

Derivado de estas necesidades ha sido propuesto un macro-proyecto que reúne el análisis visual y el análisis estadístico en una herramienta llamada VisTHAA. Esta herramienta de análisis visual trabaja con las instancias del problema y el desempeño del algoritmo metaheurístico buscando disminuir el tiempo que toman los desarrolladores en hacer ajustes a los parámetros.

El presente trabajo de investigación tiene como objetivo incorporar estrategias de mejora en una herramienta de visualización llamada VisTHAA, para permitir al investigador el análisis y la evaluación basado en la caracterización del desempeño utilizando técnicas de visualización de la información. Para lograr esto se desarrollaron diversas metodologías para ayudar al investigador a mejorar el desempeño de los algoritmos. Las principales metodologías son: i) diseñar un método que permita el pre-procesamiento de los datos de entrada, ii) diseñar una metodología para que el investigador pueda crear nuevos índices, iii) diseñar un método de visualización de la superficie de aptitudes en tres dimensiones, iv) implementar una prueba estadística que permita la comparación del desempeño de dos algoritmos, v) implementar un módulo que permita la afinación de parámetros a través del algoritmo de competencias de Hoeffding, entre otras.

1.1. Antecedentes

En la comunidad científica existen dos enfoques principales para el estudio visual de los datos y el análisis de la información obtenida. En las técnicas exploratorias o descriptivas se analizan los datos sin hipótesis previa, basándose en la exploración para la búsqueda de perspectivas que sugieran ideas o hipótesis. Las técnicas confirmatorias tienen como objetivo analizar los datos de tal manera que permitan la verificación de las hipótesis que se han generado acerca de los datos.

Sin embargo, dependiendo de la aplicación y la naturaleza de la información que se analiza, los investigadores hacen uso de una o más de estas herramientas (SAS, R, minitab) para complementar el análisis de los datos, haciendo evidente la necesidad de contar con una aplicación ad hoc para el análisis de datos provenientes de algoritmos.

Desde hace algunos años, el grupo de investigación de la Maestría en Ciencias en Ciencias de la Computación. del Instituto Tecnológico de Ciudad Madero se ha preocupado por llevar a cabo proyectos relacionados con el análisis de los datos resultantes del análisis experimental de algoritmos, así como la validación de resultados del estudio comparativo significativo del rendimiento de los algoritmos.

Esta investigación corresponde a una etapa del macro-proyecto global "***Análisis Teórico y Experimental del Proceso de Optimización de Algoritmos Heurísticos***", el cual se puede ver detallado en la Figura 1.1.

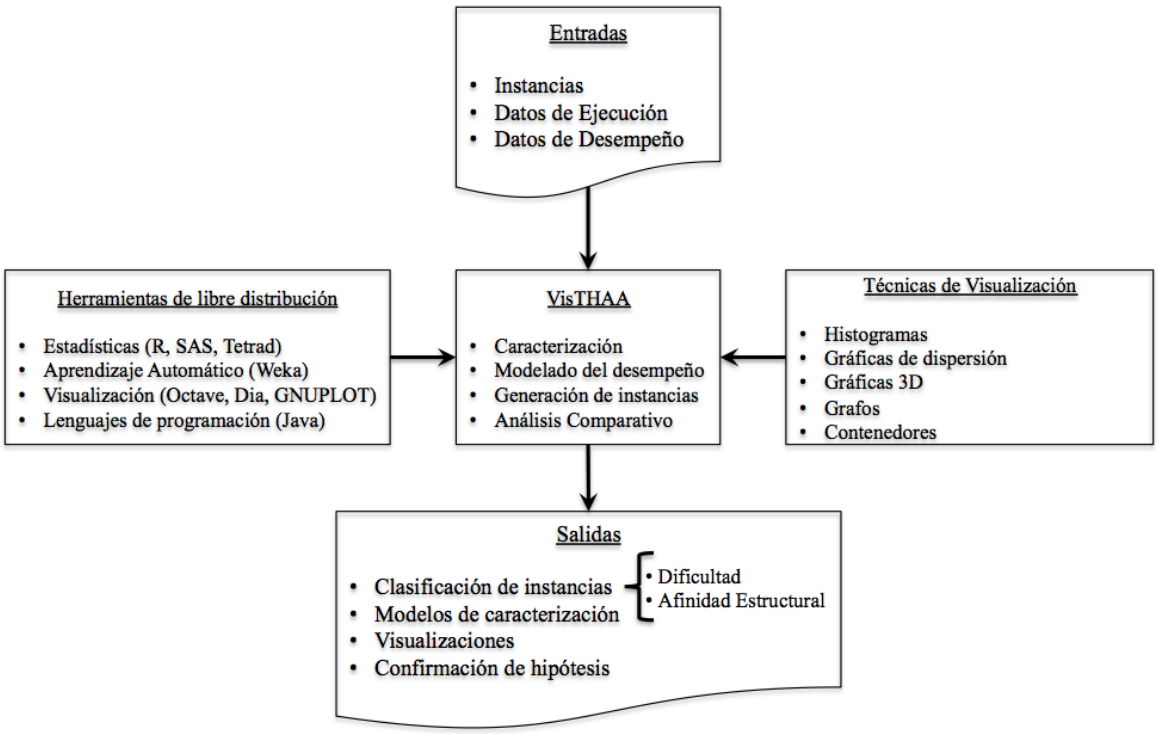


Figura 1.1. Diagrama general del macro-proyecto.

En el marco del macro-proyecto de investigación ya se ha dado inicio desde hace un tiempo con el estudio de la caracterización y el modelado del desempeño, a través de las tesis de Pérez [Pérez 07] y Quiroz [Quiroz 09]. Recientemente se están desarrollando tesis en el área de visualización de la información para integrar los trabajos anteriores con los actuales.

1.2. Descripción del Problema

El presente trabajo está ubicado dentro del *Descubrimiento de Relaciones de Desempeño en el proceso de Optimización de Algoritmos Heurísticos*, a través del desarrollo de índices de descubrimiento de relaciones de desempeño. En la Figura 1.2 se puede observar el

diagrama detallado de VisTHAA, en el cual se identifican las fases involucradas en este proyecto.

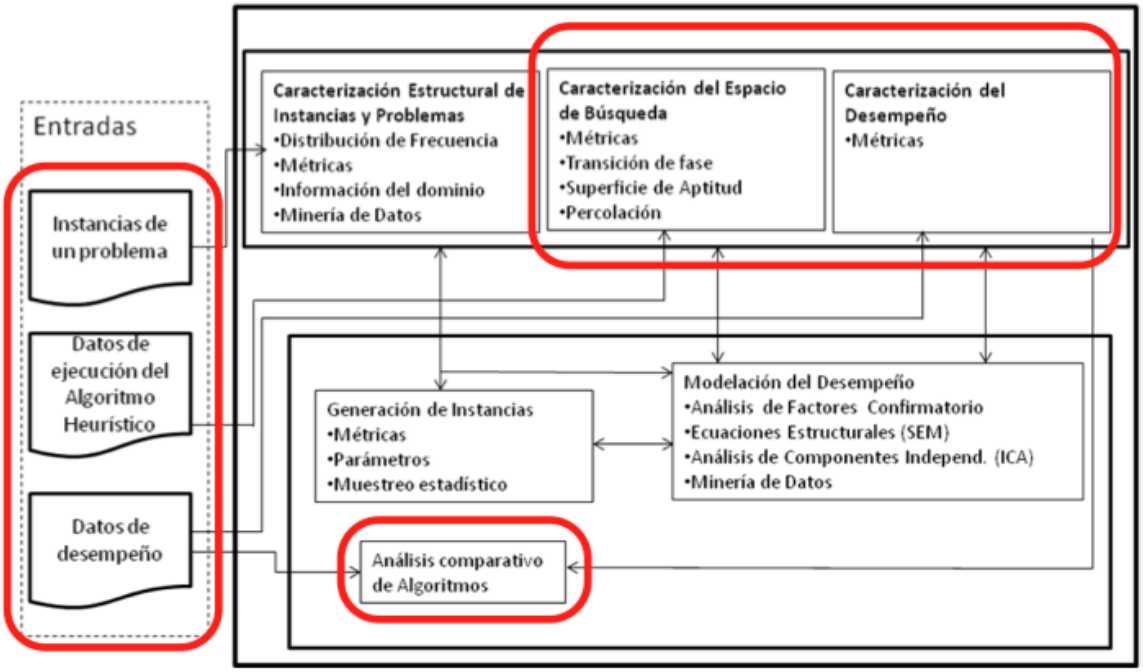


Figura 1.2. Diagrama detallado de VisTHAA.

Las fases identificadas en VisTHAA están relacionadas con la Evaluación de Estrategias de Mejora del Desempeño de Metaheurísticos Aplicados a BPP vía Diagnóstico Visual. El diagrama de este trabajo de tesis se presenta a continuación en la Figura 1.3.

Este trabajo consiste en incorporar estrategias de mejora en VisTHAA que permitan el análisis y la evaluación del algoritmo mediante la caracterización del desempeño y la visualización de la información. Como se observa en la Figura 1.3 los datos de entrada son pre-procesados para que ser ingresados correctamente a VisTHAA. Posteriormente se caracteriza la instancia del problema, el comportamiento del algoritmo durante la ejecución y el desempeño final del algoritmo. Si derivado de la caracterización se detectan áreas de mejora en el algoritmo se realizará el rediseño del algoritmo a través de nuevos índices y/o el ajuste de parámetros.

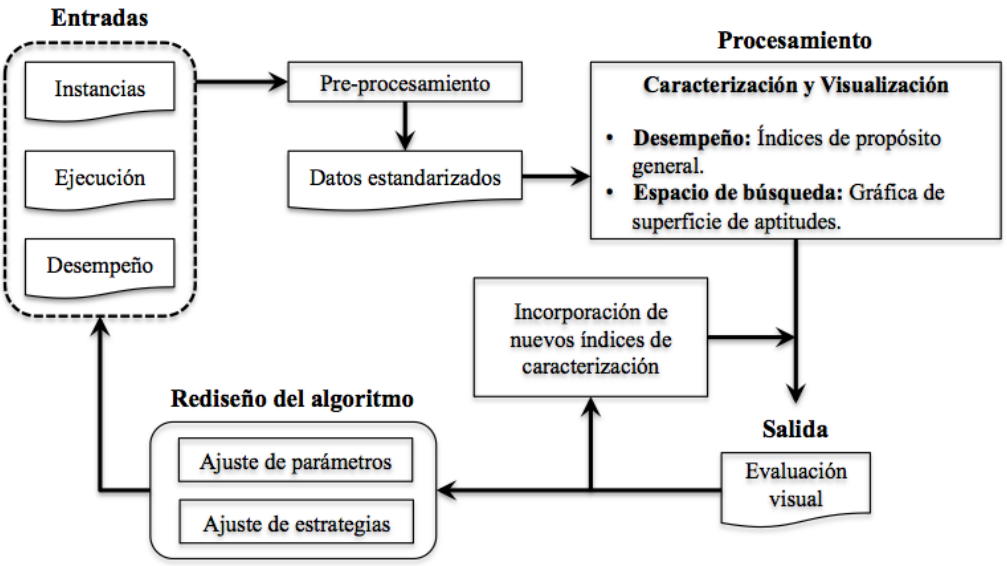


Figura 1.3. Diagrama del problema que aborda este proyecto de tesis.

1.3. Justificación

El análisis gráfico en todas sus etapas es una estrategia poderosa a través de la cual se pueden obtener conclusiones que permiten al investigador ver rápidamente el comportamiento de los datos en profundidad, identificar anomalías, relaciones o patrones, confirmar o rechazar resultados esperados y posiblemente descubrir nuevos fenómenos.

En el diseño de los algoritmos, existen varios factores que pueden conducir a un desempeño pobre, y por consecuencia a un resultado deficiente. El ajuste de los parámetros de los algoritmos es de suma importancia y, debido a la naturaleza multidimensional de éstos, analizar su comportamiento resulta en extremo difícil, ya que se combinan los factores del problema con los del algoritmo.

En este proyecto se propone crear una herramienta que analice tanto los datos previos al proceso algorítmico, como los datos resultantes de dicho proceso mediante técnicas basadas en la estadística. Posteriormente, de forma gráfica se presenten los resultados de dicho análisis para que contribuya tanto en el rediseño de los algoritmos como en el ajuste

de los parámetros. Uno de los beneficios es la reducción del tiempo y recursos invertido por el diseñador en estas tareas, además de la obtención de algoritmos de mejor desempeño.

1.4. Objetivos

1.4.1. Objetivo General

Incorporar estrategias de mejora en VisTHAA que permitan el análisis y la evaluación basados en la caracterización del desempeño utilizando técnicas de visualización de la información.

1.4.2. Objetivos Específicos

- Entender y explicar el análisis del modelado del desempeño.
- Caracterizar el proceso de optimización.
- Visualizar el desempeño de un algoritmo metaheurístico.
- Caracterizar el espacio de soluciones a través del análisis de la superficie de aptitudes.
- Comparar el desempeño de dos algoritmos mediante la aplicación de la prueba de Wilcoxon.
- Diseñar e implementar una metodología para incorporar nuevos índices a partir de los existentes.
- Diseñar e implementar un método que permita la evaluación visual de la información del algoritmo.
- Diseñar e implementar una metodología para el pre-procesamiento de los datos provenientes de las instancias.

1.5. Alcances

El presente proyecto de investigación tiene los siguientes alcances:

- Desarrollo de un módulo de pre-procesamiento de datos que permita tomar las instancias del investigador y transformarlas automáticamente en instancias estandarizadas.
- Caracterización del desempeño y del espacio de búsqueda.
- Análisis visual del desempeño del algoritmo.
- Desarrollo de un modelo que permita agregar nuevos índices por parte del investigador.
- Desarrollo de un método de comparación entre algoritmos, es decir, un método que determine si las diferencias entre los algoritmos son estadísticamente significativas o no.

1.6. Limitaciones

- El caso de estudio será el BPP.
- Los metaheurísticos utilizados serán los algoritmos del estado del arte: HGGA-BP [Quiroz 09] y WABP [Loh 06].
- La caracterización de las instancias se hará mediante el análisis de la superficie de aptitudes, gráficas de frecuencias, gráficas de barras, índices especializados e inclusión de nuevos índices.
- La caracterización del desempeño se hará a través de la inclusión de nuevos índices y de manera visual.
- Para la comparación entre algoritmos, únicamente se trabajará con un método no paramétrico: la prueba de Wilcoxon.

1.7. Organización del Documento

El documento está organizado de la siguiente manera:

- En el capítulo dos se presentan las definiciones formales de los conceptos que están relacionados con el presente trabajo de investigación.
- En el capítulo tres se presenta el estudio de los trabajos relacionados con el proyecto.
- En el capítulo cuatro se describen las estrategias de mejora implementadas en este trabajo.
- En el capítulo cinco se muestran las implementaciones en VisTHAA de las estrategias de mejora descritas en el capítulo cuatro.
- En el capítulo seis se presenta una descripción de las experimentaciones que se realizaron con la herramienta, así como también los resultados de éstas.
- En el capítulo siete se presentan las conclusiones del presente proyecto, así como también los posibles trabajos futuros.
- En el anexo A se detalla la forma de calcular los estadísticos de uso más frecuente, que a la vez fueron implementados a VisTHAA.
- En el anexo B se detalla la forma de calcular los índices de caracterización de la instancia y del desempeño final de un algoritmo, dichos índices se tomaron del trabajo de Quiroz [Quiroz 09].
- En el anexo C se detalla la forma de calcular los índices para cuantificar la rugosidad de una instancia, dichos índices se tomaron del trabajo de Pérez [Pérez 07].
- En el anexo D se muestran ejemplos de la forma en cómo se realiza la introducción de nuevos índices a la herramienta VisTHAA.
- En el anexo E se presentan ejemplos detallados de la forma en cómo se consiguió realizar la visualización del espacio de búsqueda en dos y tres dimensiones.
- En el anexo F se presenta un ejemplo detallado de la forma en cómo funciona el método de almacenamiento en VisTHAA.
- En el anexo G se presentan algunas recomendaciones para el diseño de interfaces.

Capítulo 2

Marco Conceptual

En este capítulo se presenta una revisión de los conceptos relacionados con el proyecto, tales como el problema del empacado de objetos en contenedores, los algoritmos metaheurísticos y el análisis del desempeño de metaheurísticos, entre otros.

2.1. Introducción

En los trabajos revisados relacionados con el diseño de herramientas gráficas que analizan el desempeño de algoritmos, se enfocan en mostrar el desempeño del algoritmo en forma gráfica únicamente. Sin embargo, trabajos recientes en el área han demostrado la necesidad y utilidad de explicar el desempeño observado en conjunto con información estadística, de manera que se pueda analizar y comprender los factores significantes que afectan o intervienen de manera conjunta en el desempeño obtenido por un algoritmo.

El análisis gráfico y estadístico de los factores muestra la influencia para aquellos casos donde por ejemplo, el desempeño fue adecuado o inadecuado. Independientemente del método elegido para la caracterización de los algoritmos, el objetivo principal es explicar como el desempeño de un algoritmo depende de una serie de factores. El entendimiento adquirido puede conducir a mejores predicciones del desempeño en nuevas situaciones y al descubrimiento de algoritmos mejorados usando la información obtenida de manera gráfica o estadística.

Para realizar el análisis gráfico es necesario contar con un problema y un algoritmo que resuelva ese problema, a continuación se describen formalmente los elementos necesarios inmersos en éstos.

2.2. Problema de Empacado de Objetos en Contenedores

En la vida existen un gran número de problemas de optimización, en los cuales se busca una asignación eficiente de recursos con la finalidad de satisfacer los objetivos planteados. Para contribuir en ello, es necesario resolver estos problemas mediante el uso de herramientas computacionales capaces de adaptarse a diferentes escenarios y obtener buenas soluciones sin consumir grandes cantidades de recursos [Quiroz 09].

Un caso particular de un problema de asignación es el problema del empacado de objetos en contenedores conocido como Bin Packing (*BPP*, por sus siglas en inglés), el cual fue analizado en el presente proyecto de investigación.

El problema de distribución de objetos en contenedores es un problema clásico de optimización combinatoria NP-duro [Cruz 04], en el cual hay una secuencia de *n* objetos $L = \{a_1,\ a_2, \ldots, a_n\}$, cada objeto con un tamaño dado $0 < s(a_i) \leq c$ y un número ilimitado de contenedores, cada uno con capacidad *c*. El objetivo es determinar el menor número de contenedores *m* en los cuales todos los objetos pueden ser distribuidos. La Expresión 2.1 presenta la definición formal de este problema.

dado **(2.1)**

n = número de objetos a distribuir

c = capacidad del contenedor

L = secuencia de n objetos a_i

$s_i(a_i)$ = tamaño de cada objeto a_i

encontrar

una partición de *L* mínima, $L = B_1 \cup\ B_2 \cup \ldots \cup\ B_m$

tal que en cada conjunto B_j la sumatoria del tamaño de cada objeto $s(a_i)$ en B_j

no exceda *c*,

$$\sum_{a_i \varepsilon B_j} s_i(a_i) \leq c \qquad \forall j, 1 \leq j \leq m.$$

En la versión discreta del problema Bin Packing de una dimensión, la capacidad del contenedor es un entero *c*, el número de objetos es *n*, y por simplicidad, el tamaño de cada objeto es s_i, el cual es seleccionado del conjunto $\{1,2,\dots,c\}$.

2.3. Metaheurísticos

Los problemas denominados NP-duros son de gran interés en las ciencias computacionales. Una de las características de los problemas de este tipo es que los algoritmos exactos empleados para resolverlos requieren una cantidad exponencial de tiempo en el peor de los casos [Quiroz 09]. Al resolver problemas NP-duros es necesario conformarse con soluciones "buenas", que en algunos casos pueden ser resultados óptimos. En estas condiciones se hace uso de algoritmos que dan soluciones aproximadas en un tiempo razonable, pero no garantizan obtener la mejor solución, es decir, algoritmos metaheurísticos.

El término metaheurístico fue introducido por primera vez por Fred Glover [Glover 86], con este término quería definir un "*procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local*".

El término metaheurístico se ha usado como referencia para una familia de estrategias de búsqueda bien conocidas. Las cualidades esenciales de cada estrategia están encaminadas a descubrir mejores soluciones en el espacio de búsqueda mediante un enfoque ajustado en buenas soluciones y mejorándolas (intensificación), y a encaminar la exploración del espacio de soluciones mediante un enfoque amplio de la búsqueda de nuevas áreas (diversificación), estas dos cualidades son complementarias y necesarias. Una búsqueda basada puramente en la intensificación no permite aceptar malas soluciones y por lo tanto no puede escapar de un óptimo local; por otro lado, una búsqueda basada puramente en diversificación no tiene control de calidad por el cual se rechazan malas soluciones y se alcanzan buenos resultados [O'Brien 08].

A continuación se muestra una de las clasificaciones de metaheurísticos más comúnmente utilizadas [Blum 03].

Atendiendo a la Inspiración:

- *Natural*: algoritmos que se basan en un símil real, ya sea biológico, social, cultural, entre otros.
- *Sin inspiración*: algoritmos que se obtienen directamente de sus propiedades matemáticas.

Atendiendo al número de soluciones:

- *Poblacionales*: buscan el óptimo de un problema a través de un conjunto de soluciones.
- *Trayectoriales*: trabajan exclusivamente con una solución que mejoran iterativamente.

Atendiendo a la función objetivo:

- *Estáticas*: no hacen ninguna modificación sobre la función objetivo del problema.
- *Dinámicas*: modifican la función objetivo durante la búsqueda.

Atendiendo a la vecindad:

- *Una vecindad*: durante la búsqueda utilizan exclusivamente una estructura de vecindad.
- *Varias vecindades*: durante la búsqueda modifican la estructura de la vecindad.

Atendiendo al uso de memoria:

- *Sin memoria*: se basan exclusivamente en el estado anterior.
- *Con memoria*: utilizan una estructura de memoria para recordar la historia pasada.

Como se observa en la clasificación de los metaheurísticos, es importante identificar el tipo de metaheurístico acorde a las necesidades del problema a resolver. Los Algoritmos Genéticos son algoritmos de tipo poblacional, con inspiración natural y han demostrado su robustez en la solución de problemas como el BPP [Quiroz 09].

2.3.1. Algoritmos Genéticos

Un caso particular de los algoritmos metaheurísticos son los Algoritmos Genéticos (AGs), los cuales fueron presentados por J. Holland. Los AGs representan una metaheurística

poblacional sencilla e intuitiva que, muy probablemente, sea la más utilizada [Duarte 07]. El principio de operación es el siguiente:

Los AGs se basan en una población de soluciones candidatas, que evoluciona por medio de mecanismos genéticos neo-darwinistas de selección, cruza y mutación.

El concepto de recombinación de soluciones supone una de las aportaciones fundamentales de los AGs. Por otro lado es también relevante la diferencia explícita entre la representación del problema (denominada genotipo), que habitualmente viene determinada por cadenas de bits conocidas como cromosomas, y las variables del problema en sí (denominadas fenotipo) [Duarte 07].

La nomenclatura utilizada en los AGs está muy relacionada con su inspiración biológica. En este sentido, es usual denominar "población de individuos" al conjunto de soluciones, de forma que cada individuo se corresponde con una solución. La solución en sí (variables del problema) establece el fenotipo del individuo, y su representación el genotipo. Esta representación recibe el nombre de cromosoma y cada uno suele estar compuesto por unidades discretas llamadas genes [Duarte 07].

Los elementos que deben identificarse para resolver un problema con un algoritmo genético son:

- **Población inicial:** suele estar formada por una generación aleatoria de soluciones al problema dado.
- **Representación:** constituye una correspondencia entre las soluciones factibles (fenotipo) y la codificación de las variables (genotipo). Originalmente las representaciones eran binarias. Sin embargo, actualmente se han utilizado otras representaciones en problemas discretos, de permutaciones y binarios.
- **Función de evaluación:** determina la calidad de los individuos de la población. Habitualmente, es una función monótona creciente que asigna un valor mayor cuanto mejor sea el individuo.

- **Operadores genéticos:** Son métodos probabilísticos que obtienen nuevos individuos. Suelen ser dependientes de la representación. Habitualmente se utilizan los siguientes operadores:
  1. **Cruza:** consiste en la sustitución de un conjunto de genes de un padre por los genes correspondientes del otro padre para generar un nuevo individuo hijo.
  2. **Mutación:** consiste en el cambio aleatorio de parte de un individuo. La mutación se emplea como mecanismo para preservar la diversidad de las soluciones y explorar nuevas zonas del espacio de soluciones.
- **Selección:** es un mecanismo que permite elegir con mayor probabilidad a los individuos que presenten un valor más elevado de la función de evaluación.

Según Hedar [Hedar 04], un AG es un procedimiento que trata de imitar la evolución genética de las especies. Específicamente, un AG simula el proceso biológico que permite a las generaciones siguientes de una población adaptarse a su ambiente. El proceso de adaptación es mayormente aplicado a través de la herencia genética de los padres a los hijos y también a través de una prueba de supervivencia. Los AGs son un método de búsqueda basado en población. A continuación se presenta el algoritmo general de los Algoritmos Genéticos.

Algoritmo 2.1. Algoritmo Genético simple.

1. **Inicialización.** Genera una población inicial P_0. Establece las probabilidades de cruza y mutación $0 \leq P_c \leq 1$ y $0 \leq P_m \leq 1$, respectivamente. Establece el contador de generaciones t:=1.
2. **Selección.** Evalúa la función de aptitud F de todos los cromosomas en P_t (Población actual). Selecciona una población intermedia P_t' de la población actual P_t.
3. **Cruza.** Asocia un número aleatorio dentro del intervalo [0,1] con cada cromosoma en P_t' y agrega este cromosoma al conjunto de padres S_t^P si el número asociado es menor que P_c. Repite los siguientes pasos 3.1 y 3.2 hasta que todos los padres en S_t^P hayan sido cruzados.

3.1. **Elige** dos padres P_1 y P_2 de S_t^P. Cruza p1 y p2 para producir los hijos h1 y h2.

3.2. **Actualiza** el conjunto de hijos S_t^H a través de $S_t^H \coloneqq S_t^H \cup \{h_1, h_2\}$ y actualiza S_t^P a través de $S_t^P \coloneqq S_t^P - \{P_1, P_2\}$.

4. **Mutación.** Asocia un número aleatorio dentro del intervalo $0 \leq I \leq 1$ con cada gen en cada cromosoma en P_t´, muta este gen si el número asociado es menor que P_m, y agrega el cromosoma mutado únicamente al conjunto de hijos S_t^H.

5. **Condiciones de paro.** Si las condiciones de paro son satisfechas, entonces terminar. En cualquier otro caso, selecciona los individuos más aptos para la próxima generación P_{t+1} de $P_t \cup S_t^H$. Establece S_t^H vacío, establece t := t+1, e ir al paso 2.

Como se mencionó anteriormente, los AGs han sido ampliamente aplicados a diferentes tipos de problemas de optimización con resultados exitosos; sin embargo no son los únicos que dan buenos resultados, a continuación se presenta el algoritmo WABP, el cual es una variante del algoritmo metaheurístico conocido como recocido simulado y da solución a BPP.

2.3.2. Algoritmo WABP

Otro caso particular de los algoritmos metaheurísticos es el algoritmo Weight Annealing (WA), el cual fue propuesto por Ninio y Schneider [Loh 06]. El algoritmo WA es un método que permite a una heurística voraz escapar de un óptimo local pobre en el contexto de un problema de optimización combinatoria. La idea clave es cambiar la superficie del problema y hacer uso de la historia de cada ejecución de optimización. Ninio y Schneider cambiaron la superficie a través de la asignación de pesos a diferentes partes del espacio de soluciones. Los pesos pueden ser seleccionados aleatoriamente o haciendo énfasis en aquellas partes del espacio de soluciones que están lejos de la solución óptima [Loh 06]. Ninio y Schneider resumieron su algoritmo WA como sigue en el Algoritmo 2.2.

Algoritmo 2.2. Algoritmo WA.

1. **Empezar** con una configuración inicial de una solución heurística voraz utilizando la superficie original.
2. **Determinar** un nuevo conjunto de pesos basados en las ejecuciones de optimización previas y el conocimiento del problema.
3. **Realizar** una nueva ejecución de la heurística voraz utilizando los nuevos pesos.
4. **Regresar** al paso 2 hasta que se alcance un criterio de paro.

El algoritmo WABP es una extensión de WA, el cual fue adaptado para resolver el problema de empacado de objetos en contenedores por Loh, et al [Loh 06]. WABP comienza con una solución inicial generada por el procedimiento FFD (First Fit Decreasing). Para cada contenedor i en la solución de FFD se computa el porcentaje de llenado del contenedor y su capacidad residual.

Para mejorar la solución, se llevan a cabo operaciones de intercambio con el algoritmo WABP. Un parámetro de temperatura (T) controla la cantidad con la cual un peso puede variar. Al inicio, una alta temperatura (T=1) permite movimientos más frecuentes de cuesta abajo. Como la temperatura es gradualmente enfriada (la temperatura es reducida al final de cada iteración, esto es, $T \times 0.95$), la cantidad de distorsión del objeto decrementa y la superficie del problema se asemeja más a la original.

Se computa un peso para cada contenedor para aplicarlo a cada objeto del contenedor. El proceso de intercambios comienza comparando los objetos en el primer contenedor con los objetos con el segundo contenedor y así sucesivamente, en forma secuencial hasta llegar al último contenedor de la solución inicial y se repite para cada posible par de contenedores. El Algoritmo 2.3 detalla el algoritmo WABP [Loh 06].

Algoritmo 2.3. Algoritmo WABP.

1. **Inicializar** los parámetros: K (parámetro de escala), nloop (máximo número de iteraciones), T (temperatura inicial) y Tred (factor de reducción de temperatura).
2. **Construir** una solución inicial utilizando el procedimiento FFD.
3. **Mejorar** la solución actual iterativamente.
4. **Mostrar** los resultados (número de contenedores utilizados, distribución final de los objetos y la capacidad residual de cada contenedor).

En la práctica, el algoritmo WABP ha logrado obtener buenos resultados en periodos de cómputo relativamente rápidos, lo que demuestra que es una metaheurística eficiente y efectiva, en términos de su desempeño. Para que el investigador pueda apreciar rápidamente el desempeño del metaheurístico o la estructura de la instancia, es necesario recurrir a la visualización, la cual se describe en la siguiente sección.

2.4. Visualización

Los datos procedentes del problema, así como de los algoritmos, pueden ser visualizados para intentar obtener un mayor conocimiento. La visualización es una herramienta para interpretar las imágenes de los datos dentro de la computadora y para generar imágenes de conjuntos de datos multidimensionales complejos [Defanti 1991].

Un análisis completo de datos multidimensionales requiere de un conjunto de herramientas estadísticas: paramétricas, no paramétricas y gráficas. El análisis paramétrico es el más poderoso, el análisis no paramétrico es el más flexible, y el análisis gráfico provee el vehículo para descubrir lo inesperado [Scott 92].

Una rama de la visualización es la ***visualización científica***, la cual apunta a crear algoritmos y métodos que transformen conjuntos de datos científicos masivos en imágenes y representaciones gráficas que faciliten la comprensión y la interpretación. Además, nos

ayuda a extraer información útil de complejos y frecuentemente voluminosos conjuntos de datos a través del uso de gráficos interactivos e imágenes [Padma 96].

Según Spence, la visualización es un proceso en el cual se construye un modelo o una imagen mental acerca de alguna cosa en particular. Es considerada como una actividad cognitiva propia de los seres humanos. Cuando se aplica la visualización en el estudio de representaciones de grandes conjuntos de información recibe el nombre de ***visualización de la información***. Dichas representaciones visuales así como las técnicas de interacción toman ventaja de la facilidad de llegar a la mente a través del sentido de la vista y permiten a los usuarios observar, explorar y entender grandes cantidades de información al mismo tiempo [Spence 07]. Dentro de las técnicas más usuales para crear representaciones de la información y que se utilizaron en el presente proyecto, se pueden encontrar las siguientes:

- ***Gráfica de Barras:*** Es una forma de representación de los datos, consta de un diagrama con barras rectangulares cuyas longitudes (alturas) son proporcionales a los valores que representan.
- ***Gráfica poligonal:*** En este tipo de gráfica se representan valores en los dos ejes del plano cartesiano, se recomienda utilizarlo para representar series de tiempo, i.e., el valor objetivo con respecto a cada iteración.
- ***Gráfica de Frecuencias:*** Este tipo de gráfica se puede utilizar para cualquier tipo de datos, sin embargo la gráfica de frecuencias utilizadas en este trabajo se orientó hacia la visualización de la distribución de los pesos de los objetos con respecto a la capacidad del contenedor de una instancia de BPP [Quiroz 09].
- ***Gráfica de Superficie de Aptitudes:*** Este tipo de gráfica se puede utilizar para cualquier instancia de cualquier problema de optimización. La idea central de esta gráfica es ver a cada individuo (solución) sobre una superficie, e identificar rápidamente si éstos individuos son muy diferentes entre sí (lo cual implica una superficie rugosa) o si, por el contrario, los individuos son parecidos entre sí. En este tipo de gráfica la altitud de un individuo es proporcional a su valor de aptitud, en problemas de maximización un punto alto en la superficie denota a un individuo con un buen valor de aptitud, mientras que en

un problema de minimización, el mismo individuo denotaría a un individuo con un pobre valor de aptitud.

2.5. Análisis de la Superficie de Aptitudes

El análisis de la superficie de aptitudes es un enfoque utilizado inicialmente en computación evolutiva [Pérez 07]. Fue propuesto por Wright [Wright 32] y se basa en considerar a la ***evolución*** como el ***flujo de una población sobre una superficie***, en donde la altitud de un punto cuantifica que tan bien se adapta al ambiente un individuo. Bajo este concepto, es posible realizar la visualización del espacio de búsqueda o superficie de aptitudes.

Para dar una definición formal de la superficie de aptitudes, es necesario primero establecer los conceptos de *problema de optimización combinatoria, instancia del problema, espacio de estados, función objetivo, operador de movimiento* y *búsqueda local*.

Se denota a un problema de optimización combinatoria (POC) como Π y una instancia de Π como Ω, la cual es extraída de algún universo (posiblemente infinito) de posibles instancias del problema, que es denotado como U_{Π} [Gendreau 10].

Una instancia Ω de un problema de optimización combinatoria Π es completamente especificado por dos componentes: el ***espacio de estados*** y la ***función objetivo***. El espacio de estados *S* es un finito, aunque típicamente astronómicamente largo, conjunto de posibles soluciones a Ω. La función objetivo F asigna un valor numérico a cada estado $s \in S$. La única restricción formal de F es que debe de existir un orden total del co-dominio, tal que un valor máximo o mínimo esté bien definido. Típicamente $F: S \rightarrow \mathbb{R}^{+}$ o $F: S \rightarrow \mathbb{Z}^{+}$. La función objetivo es comúnmente referida como ***función de aptitud*** [Gendreau 10].

Dado un *espacio de estados S*, la noción de localidad en un algoritmo de búsqueda local es provista por el *operador de movimiento o vecindario N*, el cual define un conjunto de modificaciones permitidas a la solución actual *s* en cualquier iteración dada. Dada una

solución *s*, el conjunto $N(s)$ es conocido como el ***vecindario de s***. Similarmente, si $s' \in N(s)$, entonces se dice que *s'* es vecina de *s* [Gendreau 10]. Las metaheurísticas de búsqueda local frecuentemente emplean operadores de movimiento más simples.

Una vez establecidos los conceptos, se puede presentar la definición formal de la superficie de aptitudes. Dado un metaheurístico de búsqueda local *A*, i.e., Random Walk, y un problema de optimización combinatoria Π, i.e., el *BPP*, es de interés determinar que hace a una instancia en particular $\Omega \in U_{\Pi}$ fácil o difícil para *A*. La dificultad del problema, o equivalentemente el costo de la búsqueda, es dictaminado por la interacción de *A* con el subyacente espacio de búsqueda [Gendreau 10].

Debido al rol central del espacio de búsqueda en la determinación de la dificultad del problema, gran parte de la investigación sobre los modelos de la dificultad del problema para búsqueda local se ha concentrado en identificar las características estructurales del espacio de búsqueda, que puede tener una influencia en el costo de la búsqueda local [Gendreau 10]. Dado un metaheurístico de búsqueda local *A*, el espacio de búsqueda es definido como la combinación del *espacio de estados* ***S***, el *operador de movimiento* ***N*** y la *función objetivo* ***F***. Formalmente, se define el espacio de búsqueda $L = (S, N, F)$ como un grafo dirigido con vértices ponderados $G = (V, E)$ en el cual:

1. $V = S$
2. $\forall v \in V, el\ peso\ w_v\ de\ v\ es\ igual\ a\ F(v)$
3. $E = \{i, j | i \neq j \wedge \exists i, j : i \in N(j)\}$

Dentro de la comunidad de búsqueda local, el grafo ***G*** es conocido como una ***superficie de aptitudes*** [Gendreau 10].

Según Pérez [Pérez 07] la dificultad asociada al espacio de búsqueda de un problema se relaciona con la trayectoria que describen las soluciones de un algoritmo. Es posible que ésta muestre una gran cantidad de óptimos locales, por lo que la dificultad de encontrar buenas soluciones a través de un proceso de optimización dependería de la calidad de las

soluciones y de la distribución y accesibilidad de cada óptimo local. La Figura 2.1 muestra un ejemplo de una superficie de aptitudes.

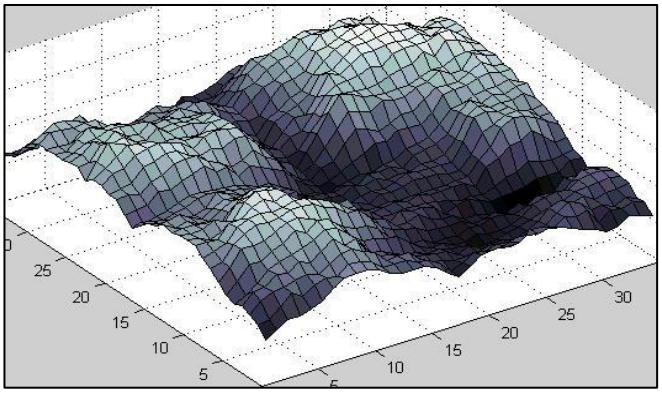


Figura 2.1. Superficie de aptitudes del espacio de búsqueda [Pérez 07].

La superficie de aptitudes es considerada como la visualización del proceso de exploración y explotación del espacio de búsqueda, definido a través de los individuos (soluciones), imaginándolo como una superficie formada por picos, valles, montañas, ríos y planicies. Un valor de aptitud es asociado a cada individuo y una función de aptitud califica en cierto grado si es apto o no [Pérez 07].

Según Fraire [Fraire 10] la idea central del análisis de la superficie de aptitudes en la optimización combinatoria, es ***representar el espacio de búsqueda*** a través de un algoritmo, como una superficie formada por todas las soluciones factibles y su valor objetivo asociado.

Además, Fraire afirma que la información generada con el análisis de superficies es utilizada para ganar conocimiento acerca de:

- Las ***características*** del espacio de búsqueda y su relación con el ***comportamiento*** de la búsqueda local o algoritmos metaheurísticos.
- La posibilidad de ***predecir la dificultad*** de la instancia de un problema.
- Nos indica un ajuste útil de los parámetros en los algoritmos de búsqueda local.

Una superficie es considerada rugosa si hay una baja correlación entre los puntos vecinos. Para medir esta correlación una caminata aleatoria de longitud m es ejecutada en la superficie de aptitudes para interpretar la serie de m puntos resultantes $\{f(x_t), t = 1 \dots m\}$ como una serie de tiempo [Fraire 10].

Índices para Caracterizar la Rugosidad de una Instancia [Pérez 07]

Con el fin de obtener un conocimiento acerca de la rugosidad de una instancia, se han propuesto diversos índices de caracterización, a continuación se describen algunos de ellos en la Tabla 2.1.

Tabla 2.1. Índices para caracterizar la superficie de aptitudes.

Nombre del índice	Descripción
Coeficiente de Autocorrelación	Indica cuan correlacionadas están las soluciones de una instancia.
Longitud de Autocorrelación	Indica cuál es el mayor valor de distancia a la que el conjunto de soluciones se vuelve no correlacionado.
Contenido de información	caracteriza el grado de rugosidad con respecto a las áreas planas de la superficie de aptitudes.

2.6. Proceso de Optimización y Caracterización

Aunado al análisis de la superficie de aptitudes, para lograr comprender el desempeño de un algoritmo sobre un problema, debe realizarse un estudio integral de todo el proceso de solución [Quiroz 09]. El ***proceso de optimización*** puede entenderse como la acción de resolver un problema de optimización (entrada) mediante un algoritmo (proceso), obteniendo una solución final (salida), esto se puede observar en la Figura 2.2.

La entrada consiste en una instancia o caso particular del problema de optimización a resolver, compuesta por un conjunto de parámetros específicos que lo definen. El proceso incluye el conjunto de estrategias utilizadas para dar solución al problema, es decir, el algoritmo utilizado. La salida proporciona la solución óptima del problema, ya sea ésta exacta o aproximada [Quiroz 09].

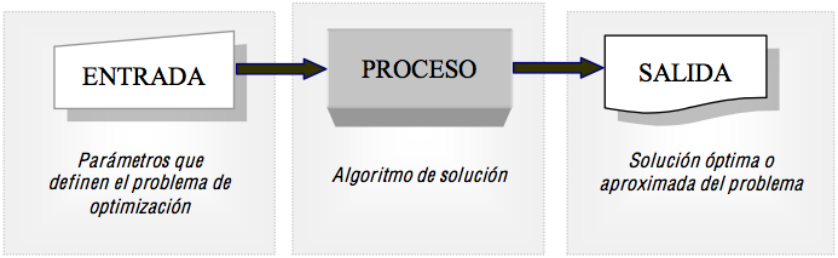


Figura 2.2. Proceso de optimización [Quiroz 09].

El objetivo principal de la ***caracterización*** de algoritmos es entender de que manera el desempeño de un algoritmo es afectado por una serie de factores que lo causan. El conocimiento adquirido puede conducir a mejores predicciones del desempeño ante nuevas situaciones y a la definición de algoritmos mejorados [Quiroz 09]. En palabras más sencillas, la caracterización consiste en obtener las *particularidades* del algoritmo para poder observar áreas de mejora de éste y hacerlo más eficiente y/o eficaz. Sin embargo, la caracterización no es exclusiva del algoritmo, también se puede aplicar a la instancia.

La caracterización del problema y del proceso de solución es una parte esencial en el análisis de desempeño de algoritmos, ya que permite identificar cuáles son los factores que influyen en el comportamiento algorítmico [Quiroz 09]. Un análisis de desempeño de calidad requiere la definición de índices adecuados que cuantifiquen las características indicadoras del desempeño.

2.6.1. Caracterización de la Entrada

Caracterizar la estructura de una instancia de BPP es un punto clave para predecir el comportamiento que tendrá un algoritmo metaheurístico al momento de solucionarla [Quiroz 09]. Es conocido que factores como el número de objetos, la tendencia central de sus pesos y la forma en que se distribuyen, impactan en el grado de dificultad que una instancia pueda tener sobre un algoritmo de solución.

Índices para Caracterizar el Problema [Quiroz 09]

Para poder realizar la caracterización del problema de optimización BPP, a través de sus instancias, Quiroz propone cinco índices que cumplen con este propósito, los cuales se describen a continuación. La Tabla 2.2 muestra estos índices.

Tabla 2.2. Índices de caracterización del problema de optimización.

Nombre del índice	Descripción
Menor	Representa el peso del objeto más pequeño, en relación al tamaño del contenedor.
Mayor	Representa el peso del objeto más grande, en relación al tamaño del contenedor.
Multiplicidad	Caracteriza el número promedio de repeticiones de cada peso de objeto
MaxRepe	Representa la frecuencia máxima con la que un peso se repite en el conjunto de objetos.
Uniformidad	Mide el grado de uniformidad de la distribución de los pesos de los objetos.

2.6.2. Caracterización del Proceso

Los algoritmos metaheurísticos híbridos incluyen estrategias que tratan de ajustar el proceso de solución al problema de optimización que resuelven [Quiroz 09]. Para estos algoritmos, la medición de funciones de caracterización debe estar ligada directamente con las técnicas heurísticas utilizadas.

En el trabajo de Quiroz, se utilizó un algoritmo genético, denominado *HGGA-BP*, el cual es un algoritmo genético híbrido de agrupación para el problema de empacado de objetos en contenedores (Hybrid Genetic Grouping Algorithm for Bin Packing). Este algoritmo incorpora estrategias heurísticas y criterios que, al aplicarse en la creación y evolución de individuos, obtienen buenas soluciones en tiempos cortos.

Índices para Caracterizar el Desempeño Parcial del Algoritmo [Quiroz 09]

La caracterización del comportamiento del algoritmo (desempeño parcial) objeto de estudio, posibilita: observar el impacto de los principales parámetros de control del

algoritmo; analizar la superficie de aptitudes de los individuos en cada generación; identificar la importancia y contribución de cada método que conforma el algoritmo; y evaluar la calidad de las soluciones que conforman la población.

Para caracterizar el proceso de generación de individuos se definieron tres índices , los cuales permiten evaluar el número y la calidad de los individuos nuevos, aceptados en cada generación, como parte de la población. En la Tabla 2.3 se pueden ver dichos índices.

Tabla 2.3. Índices para caracterizar el desempeño parcial del algoritmo.

Nombre del índice	Descripción
Nuevos	Representa el número promedio de individuos, de reciente creación, aceptados en cada generación.
Buenos	Cuantifica el número promedio de individuos nuevos, aceptados por generación, que superaron la aptitud del individuo que sustituyeron.
Malos	Mide el número promedio de individuos nuevos, aceptados por generación, cuya aptitud era menor que la del individuo que sustituyeron.
$Mejor\_Aptitud_g$	Es la aptitud del mejor individuo de la población en la generación g.
$Peor\_Aptitud_g$	Registra la aptitud del peor individuo de la población en la generación g.
$Prom\_Aptitud_g$	Cuantifica el promedio de la aptitud de la población en la generación g.
Mejor_Aptitud	Caracteriza la aptitud promedio de las mejores soluciones obtenidas en cada generación.
$Desv\_Aptitud_g$	Mide la variabilidad en la aptitud de los individuos que conforman la población de la generación g.
Desv_Mejor	Representa la desviación en las aptitudes de los mejores individuos de cada generación.
Mejor_Heurística	Las estrategias que tienen mayor impacto en la obtención de las mejores soluciones del algoritmo son registradas a través de este índice.

2.6.3. Caracterización de la Salida

De acuerdo con Quiroz [Quiroz 09], el desempeño final del algoritmo se mide por tres tipos de información: calidad de la solución, la consistencia de los resultados y el tiempo que le llevó en encontrar la mejor solución.

Calidad de la Solución

La *calidad de las soluciones* obtenidas por el algoritmo, es caracterizada mediante cuatro ***índices***. El objetivo de estos índices es registrar las aptitudes de las soluciones finales alcanzadas en diferentes corridas del programa, así como también el número de contenedores utilizado en dichas soluciones. Un valor de aptitud igual a la unidad, significa que los contenedores que conforman la solución se encuentran completamente llenos, y por lo tanto, se tiene la solución óptima. La Tabla 2.4 describe cada uno de estos índices.

Tabla 2.4. Índices para caracterizar el desempeño final del algoritmo.

Nombre del índice	Descripción
Max_MejorF_{BPP}	Registra la aptitud de la mejor solución obtenida, para una instancia de BPP, en r ejecuciones del algoritmo.
Min_MejorF_{BPP}	Representa la aptitud de la peor solución obtenida, para una misma instancia de BPP, en r ejecuciones del algoritmo.
Prom_MejorF_{BPP}	Calcula el promedio de las aptitudes de las soluciones obtenidas, para el mismo caso de prueba, en r ejecuciones del algoritmo.
Radio_teorico	Es otra medida de calidad, representa la razón de la mejor solución encontrada por el algoritmo (Z_{enc}) y la solución óptima (Z_{opt}).

Consistencia de los Resultados

Para caracterizar la estabilidad del algoritmo, se mide la ***desviación estándar*** de los resultados obtenidos, dicho índice es descrito en la Tabla 2.5.

Tabla 2.5. Índice para caracterizar la consistencia de los resultados.

Nombre del índice	Descripción
Desv_Z_{enc}	Cuantifica la variabilidad de las soluciones encontradas en diferentes ejecuciones del algoritmo, para acomodar la misma instancia de BPP.

Tiempo que tardó en encontrar la mejor solución

El factor tiempo es caracterizado a través de ***dos índices***. El primero mide el tiempo real que requirió la ejecución del algoritmo antes de encontrar la solución al problema. El segundo cuantifica el número de generaciones que el algoritmo tuvo que iterar antes de encontrar la que finalmente fue la mejor solución, este último índice es únicamente aplicable para algoritmos poblacionales, tales como los Algoritmos Genéticos, mientras que los demás se pueden utilizar para cualquier tipo de algoritmo metaheurístico, ya sea población o trayectorial. En la Tabla 2.6 se muestran los índices.

Tabla 2.6. Índices para caracterizar el tiempo requerido para encontrar la mejor solución.

Nombre del índice	Descripción
Tiempo	Cuantifica el promedio del tiempo, expresado en segundos, que el algoritmo tarda en encontrar una solución, para una instancia de BPP.
Generación	Representa la generación promedio en la cual el algoritmo encuentra la mejor solución de una instancia.

Estructura de la Solución Final

Finalmente, para caracterizar la *estructura de la solución final* obtenida, Quiroz propone un índice. En la Tabla 2.7 se puede ver dicho índice.

Tabla 2.7. Índice para caracterizar la estructura de la solución final.

Nombre del índice	Descripción
Objetos_Contenedor	Calcula el número promedio de objetos que fueron almacenados en cada contenedor de la solución alcanzada por el algoritmo.

2.7. Análisis del Desempeño de Metaheurísticos

Para lograr que un algoritmo tenga un mejor rendimiento en la resolución de un determinado problema de optimización, es necesario analizar su desempeño.

El desempeño de un algoritmo está determinado por su ***eficiencia*** y su ***efectividad***. La *efectividad* de un algoritmo se refiere a la calidad de la solución encontrada o a su confiabilidad en la tarea de encontrar soluciones adecuadas. La *eficiencia* por otra parte,

está caracterizada por el comportamiento del algoritmo en tiempo de ejecución. Sin embargo, este análisis no es siempre aplicable, porque los algoritmos son considerados muchas veces como cajas negras cuyo funcionamiento interno no es conocido, además de que la complejidad del problema resuelto dificulta la estimación de la eficiencia.

La efectividad de un algoritmo es altamente dependiente de la estructura del problema y esto tiene que ver directamente con las instancias del problema que se analizan. La eficiencia, por otra parte, está muy relacionada con el conocimiento que se tenga del dominio del problema. Para el análisis del desempeño de los algoritmos, la comunidad científica muestra mayor interés en estudiar los aspectos relacionados con su efectividad [Merz 98]. Este aspecto es el más complicado de modelar debido a que interviene de manera directa la naturaleza del problema.

McGeoch y Barr [McGeoch 00 y Barr 95] sugieren la existencia de tres categorías principales de factores que afectan el desempeño algorítmico: problema, algoritmo y ambiente.

a) ***Factores del problema***: dimensión, distribución de los parámetros que lo definen, estructura del espacio de solución, entre otros.
b) ***Factores del algoritmo***: estrategias heurísticas seleccionadas (procesos de construcción de solución inicial y parámetros de búsqueda asociados), códigos de computadoras empleados, configuración de control interno del algoritmo y comportamiento en la ejecución entre otros.
c) ***Factores del ambiente***: Estos factores se refieren al ambiente físico en el que serán ejecutados los algoritmos como los son: el software (sistema operativo, compilador) y hardware (velocidad del procesador, memoria). Así como también aquellos relacionados con el programador: pericia, habilidad de afinación, lógica.

La caracterización del proceso de optimización además de ayudar a analizar el desempeño de algoritmos, permite obtener un conocimiento a detalle de cada una de las fases de este proceso. El conocimiento de la estructura de dicho proceso, los valores que la

definen y las relaciones entre estos valores, permiten, a su vez, entender la conducta del algoritmo de solución. Este conocimiento puede ser aplicado en áreas como:

- ***Clasificación de instancias***. En el trabajo de Cruz [Cruz 04] se propone el siguiente método de agrupación de instancias: Dados un conjunto de casos de un problema de optimización combinatoria y un conjunto de dos o más algoritmos de solución heurística, se forman grupos de casos por afinidad de desempeño algorítmico. Es así que las instancias se pueden clasificar por su grado de dificultad, distinguiendo entre instancias difíciles y fáciles para todos los algoritmos, y aquellas que tienen que ver con el algoritmo que las soluciona.
- ***Análisis y explicación del desempeño de algoritmos***. El identificar los factores que influyen en el desempeño de un algoritmo y establecer las relaciones que existen entre ellos posibilita el análisis y la explicación del comportamiento de éste al resolver un problema real. Lo anterior permite comprender el funcionamiento del algoritmo ante diferentes tipos de problemas y el porqué de su buen o mal desempeño.
- ***Pronósticos de desempeño***. Dado que no se conoce un algoritmo metaheurístico que sea la mejor opción para todas las posibles situaciones, es necesario elegir el algoritmo más adecuado para un problema específico. Con el conocimiento de las relaciones de desempeño, podría ser posible, al llegar un nuevo caso de un problema, pronosticar el comportamiento de un conjunto de algoritmos y elegir el mejor. Esto permitiría ahorrarse tiempo en la solución de problemas, obteniendo buenos resultados.
- ***Rediseño de Algoritmos***. El entendimiento del comportamiento de un algoritmo, así como de su diseño lógico puede sugerir cambios, tanto en la estructura del algoritmo, como en los parámetros que lo controlan, para mejorar su desempeño ante distintos tipos de problemas.

Los *criterios para medir el análisis del desempeño* de los algoritmos de aproximación dependen de los métodos elegidos para su caracterización, que pueden ser teóricos ó experimentales. En los primeros, para cada algoritmo, se determina matemáticamente la cantidad de recursos necesarios como función del tamaño del caso considerado mejor, peor

o promedio. Los segundos se basan en la experimentación para realizar la caracterización y a diferencia de los anteriores permiten describir el comportamiento de casos específicos.

En los algoritmos de aproximación raramente se estudia su efectividad de manera teórica, esto debido principalmente a la complejidad de los problemas de combinatoria. Asumiendo que no hay algoritmos que resuelvan en tiempo polinomial los problemas denominados NP duros, la única opción disponible es analizar experimentalmente la efectividad de los algoritmos de aproximación.

Para el análisis de desempeño de algoritmos, la literatura propone una gran cantidad de métodos de análisis confirmatorio, sin embargo, profundiza muy poco en las técnicas exploratorias [McGeoch 92, Hooker 94, Cohen 95, Hooker 95, Barr 95, Brglez 07]. Lo anterior puede justificarse por diversas razones. Mucha de la creatividad en la investigación es incluida en la fase exploratoria y la fase confirmatoria es utilizada sólo para validar conocimiento obtenido en la fase exploratoria.

Generalmente, los artículos científicos se concentran en la confirmación experimental de resultados, presentando resúmenes, de sin dar detalles de la forma en la que dichos resultados fueron obtenidos. La tendencia parece ser presentar los resultados adecuándolos de tal forma, que faciliten ciertos análisis, así como las técnicas probadas y resultados erróneos que ocurrieron en el proceso [Gent 99].

2.8. Pruebas de Hipótesis Estadísticas

Una vez que se ha mejorado el desempeño de un algoritmo, no es suficiente decir que se logró una mejora, sino que ésta se debe sustentar en la estadística, para ello se debe hacer uso precisamente de las pruebas de hipótesis estadísticas.

Una hipótesis estadística es una afirmación con respecto a alguna característica desconocida de una población de interés. La esencia de probar una hipótesis estadística es

el decidir si la afirmación se encuentra apoyada por una evidencia experimental que se obtiene a través de una muestra aleatoria [Canavos 88].

Existen dos tipos de pruebas de hipótesis estadísticas: las pruebas paramétricas y las pruebas no paramétricas. Las primeras parten del supuesto que los datos a analizar siguen una determinada distribución de probabilidad, sin embargo, en la práctica no siempre es así, i.e., la distribución podría ser muy plana, tener un pico o estar muy sesgada a la izquierda o a la derecha. Si se viola flagrantemente el supuesto de normalidad requerido para la prueba *t*, es posible que la estadística *t* calculada no siga la distribución *t* de Student. Si esto sucede, no es posible aplicar los valores de *t* tabulados en su tabla, se desconoce el valor correcto de α para la prueba (intervalo de confianza) y la prueba *t* es de dudosa utilidad [Mendenhall 97].

Por otra parte, las pruebas no paramétricas no dependen de la distribución de la población muestreada, por lo que también se les conoce como ***pruebas libres de distribución***. Además, los métodos no paramétricos se concentran en la ubicación de la distribución de probabilidad de la población muestreada, no en parámetros específicos de la población, como la media (de ahí el nombre "no paramétrica") [Mendenhall 97].

Las *pruebas libres de distribución* son pruebas estadísticas que no dependen de supuestos subyacentes relativos a la distribución de probabilidad de la población muestreada [Mendenhall 97].

Existen varias pruebas de libre distribución, entre las que destacan: la prueba de Mann-Whitney, la prueba de tendencias de Wald-Wolfowitz, la prueba del signo y la prueba de rangos de signos de Wilcoxon [Canavos 88] de las cuales las últimas dos se emplean para observaciones por pares, es decir, con muestras pareadas.

La ***prueba del signo*** únicamente basa su funcionamiento en los signos de las diferencias entre las observaciones por pares de dos variables aleatorias. La *prueba del signo* considera, como se mencionó anteriormente, sólo las diferencias en el signo entre

cada par de observaciones e ignora sus magnitudes. Si las observaciones se definen sobre una escala ordinal, las magnitudes de las diferencias tienen poco valor. Pero si las observaciones son magnitudes físicas, la prueba del signo puede ignorar mucha información debido a que no se toman en cuenta las magnitudes de las diferencias. Por otro lado, la ***prueba de rangos y de signos de Wilcoxon*** toma en cuenta tanto el signo como la magnitud de las diferencias entre cada par de observaciones. Por lo tanto, para tener un buen balance, éste es el mejor método no paramétrico por utilizar para observaciones en parejas [Canavos 88].

Prueba de Wilcoxon

La prueba de Wilcoxon es utilizada para responder la siguiente pregunta: **¿Dos muestras representan dos diferentes poblaciones?**. Esta prueba es un procedimiento no paramétrico empleado en una situación de prueba de hipótesis en la que se contrastan dos muestras. La prueba de Wilcoxon detecta diferencias estadísticamente significativas entre el comportamiento de dos algoritmos [García 08]. La hipótesis nula H_0 establece que no existe una diferencia significativa entre las dos muestras, mientras que la hipótesis alterna H_1 determina que sí existe una diferencia significativa. El procedimiento para realizar la prueba de Wilcoxon se presenta a continuación en el Algoritmo 2.4.

Algoritmo 2.4. Procedimiento para realizar la prueba de Wilcoxon.

1. **Obtener** las diferencias entre las unidades experimentales de las dos muestras.
2. **Obtener** los valores absolutos de las diferencias calculadas en el paso número 1.
3. **Ordenar** los conjuntos de datos ascendentemente con base en los valores absolutos de las diferencias calculadas en el paso 2.
4. **Asignar** un orden (número consecutivo) a cada unidad experimental, empezando del número uno y sin tomar en cuenta aquellas diferencias que sean nulas (diferencias con valor cero).

5. **Calcular** un nuevo ranking (orden) considerando los empates en las diferencias absolutas, para lo cual, cuando haya un empate en dichas diferencias, el ranking de cada unidad será el promedio del orden calculado en el paso 4 entre las unidades experimentales empatadas.
6. **Devolver** el signo al nuevo ranking (calculado en el paso 5), es decir, las diferencias que en el paso 1 resultaron negativas, ahora su ranking será negativo; y viceversa.
7. **Calcular** la sumatoria de los rankings con signo positivo (T+) y la sumatoria de los rankings con signo negativo (T-)
8. **Obtener** el valor de T correspondiente a las diferencias con signo más frecuente (la mayor T en valor absoluto); la significación se determina comparando este valor T con el límite superior del correspondiente intervalo de la tabla P, es decir, si T >= Ts(n,alpha) la diferencia entre las muestras es estadísticamente significativa, en caso contrario, no lo es.

En donde:
n es el número de unidades experimentales cuya diferencia no sea nula.
alpha es el nivel de significación (nivel de confianza), en donde la tabla *P* contempla seis: 0.10, 0.05, 0.02, 0.01, 0.005 y 0.001.

Con la prueba de Wilcoxon se puede determinar fehacientemente que las diferencias entre el desempeño de dos algoritmos son estadísticamente significativas con un determinado nivel de confianza, o no lo son. Sin embargo, las pruebas estadísticas no solamente se aplican para propósitos de este tipo, éstas también se pueden aplicar, i.e., para el correcto ajuste de los parámetros de un algoritmo, tal es el caso de la prueba de Hoeffding, la cual se explica a continuación.

2.9. Competencias de Hoeffding para el Ajuste el Parámetros

Los algoritmos de competencia son una familia de algoritmos cuya finalidad es seleccionar, de entre un conjunto de modelos, aquel que se considera ***mejor*** con base en un criterio

establecido. Al ser aplicados al problema del ajuste de parámetros, los algoritmos de competencia seleccionan, de un conjunto de configuraciones (Θ), aquella cuyo costo asociado sea mínimo al solucionar un conjunto de instancias (*I*) [Rivera 09].

La idea principal es que la búsqueda del mejor modelo puede ser acelerada descartando a los candidatos menos prometedores, tan pronto como se cuente con evidencia suficiente para poder retirarlos de la competencia. Dicha evidencia se basa en el uso de alguna prueba estadística que permitiera, con un determinado nivel de confianza, establecer un rango de desempeño esperado para la configuración en Θ.

El algoritmo de competencia original, las competencia de Hoeffding, adoptó una prueba estadística basada en la fórmula de Hoeffding [Rivera 09]. Dicha prueba calcula el valor medio de una variable entera positiva a, cuando su rango es conocido. Lo realiza a partir de una muestra $\Pi = \{a_1, a_2, \dots, a_\pi\}$ de π observaciones tomadas de forma independiente, a partir de las cuales se puede inferir la distribución de probabilidad para a, así como su media.

A partir de la fórmula propuesta por Hoeffding, Maron & Moore proponen la Ecuación 2.2 [Rivera 09]. La adaptación de la fórmula de Hoeffding permite, con un cierto nivel de confianza, calcular el rango de la media real de una variable a partir de la muestra Π. La media de las observaciones es denotada como $\widehat{a(\Pi)}$ y la media real de la variable a como $\bar{a}$. Se espera que $\bar{a}$ se encuentre en el rango $\left[\widehat{a(\Pi)} - \epsilon(\Pi), \widehat{a(\Pi)} + \epsilon(\Pi)\right]$, el cual también se puede denotar como $\left[\widehat{a(\Pi)}_{\downarrow}, \widehat{a(\Pi)}^{\uparrow}\right]$. El error de la media muestral con respecto a la media real, $\epsilon(\Pi)$, puede estimarse como:

$$\epsilon(\Pi) = \sqrt{\frac{B(\Pi)^2 \log(2/\delta)}{2\pi}} \qquad \textbf{(2.2)}$$

Donde de la Ecuación 2.2, δ es el nivel de significancia de la prueba, por consiguiente, su complemento $(1 - \delta)$, es el nivel de confianza. π es la cantidad de

observaciones, y $B(\Pi)$ es la diferencia entre las observaciones con mayor y menor magnitud, lo que se expresa de forma matemática como:

$$B(\Pi) = \max_{a_i \in \Pi}\{a_i\} - \min_{a_j \in \Pi}\{a_j\} \quad \textbf{(2.3)}$$

Aplicadas al ajuste de parámetros, las competencias de Hoeffding no tienen una única variable a, sino que existen θ variables a ser evaluadas, una por cada configuración en Θ.

El error esperado por utilizar la configuración Θ_j, será denotado por $\bar{\Theta}_j$; y el error medio de π ejecuciones del algoritmo utilizando la configuración Θ_j se denotará como $\widehat{\Theta_j(\Pi)}$.

El método comienza con un conjunto de candidatos (Θ). Para cada $\Theta_j \in \Theta$ se realizan ω observaciones, es decir, al inicio $\pi = \omega$. Cada observación se corresponde con el error producido por el algoritmo sobre una instancia $I_i \in I$, utilizando la configuración Θ_j. La elección de I_i depende de $P_I(i)$, que es la probabilidad de que la instancia I_i sea elegida.

Una vez que se obtienen las ω observaciones por candidato, se calcula $\widehat{\Theta_j(\Pi)}$, para cada $\Theta_j \in \Theta$. Después, se calcula el rango $\left[\widehat{a(\Pi)_{\downarrow}}, \widehat{a(\Pi)^{\uparrow}}\right]$ mediante la Ecuación 2.2 para cada $\Theta_j \in \Theta$. A través de este cálculo se conoce el rango de cada $\bar{\Theta}_j$.

Posteriormente se realiza el proceso principal de la competencia, en donde las configuraciones menos prometedoras son eliminadas. Este proceso se expresa como:

$$\Theta^* = \Theta \setminus \Phi \quad \textbf{(2.4)}$$

Donde Θ^* es el conjunto actualizado de candidatos y Φ es el conjunto de candidatos rechazados, el cual puede ser determinado de la siguiente manera:

$$\Phi = \left\{\Theta_j : \widehat{\Theta_j(\Pi)}_{\downarrow} > \min_{\Theta_i \in \Theta}\left\{\widehat{\Theta_i(\Pi)}^{\uparrow}\right\}\right\} \quad \textbf{(2.5)}$$

Donde Φ es el conjunto de configuraciones que deben salir de la competencia extrayéndolas de Θ.

De acuerdo con la Ecuación 2.5 la configuración más prometedora (Θ_i) es aquella que puede generar el menor costo, el cual se encuentra en $\left[\widehat{\Theta_1(\Pi)}_{\downarrow}, \widehat{\Theta_1(\Pi)}^{\uparrow}\right]$. A la parte superior de rango ($\widehat{\Theta_1(\Pi)}^{\uparrow}$) se le conoce como umbral de aceptación, y cualquier configuración que no pueda generar un menor costo con respecto a éste debe ser eliminada.

En la Figura 2.3 se presenta un ejemplo de la forma de evaluación para un algoritmo de competencias. En el ejemplo de la Figura 2.3 la configuración Θ_4 es la más prometedora, esto es, debido a que se espera, con una confianza de $1-\delta$ que, en el peor de los casos, alcance un costo de $\widehat{\Theta_4(\Pi)}^{\uparrow}$. Existen algunas configuraciones ($\Theta_1, \Theta_3, \Theta_6, \Theta_8$ y Θ_{11}) que podrían llegar a ser mejores que Θ_4 si $\overline{\Theta_4}$ llega a alcanzar el valor de $\widehat{\Theta_4(\Pi)}^{\uparrow}$. Las antes mencionadas configuraciones deben permanecer en la competencia, ya que pueden llegar a ser mejores que Θ_4 en el futuro [Rivera 09].

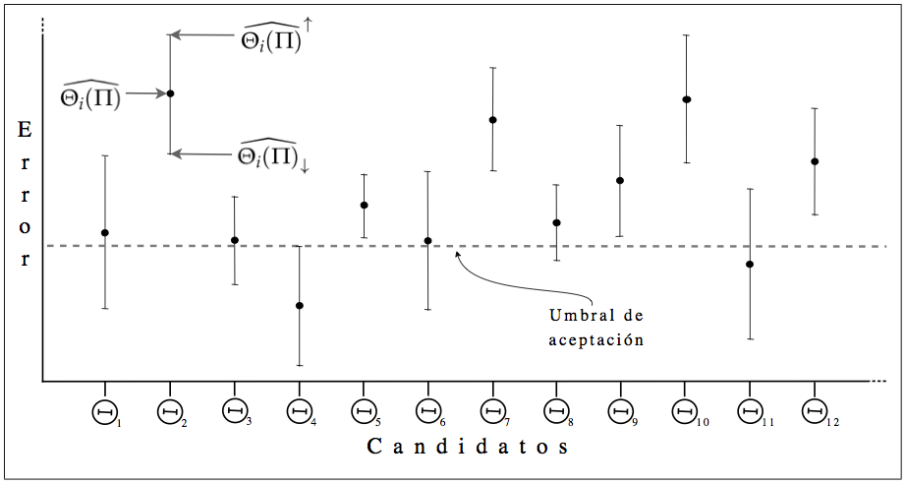


Figura 2.3. Ejemplo de evaluación de candidatos [Rivera 09].

Además, también se puede observar en la Figura 2.XX que existe un conjunto de configuraciones $\Phi = \{\Theta_5, \Theta_7, \Theta_9, \Theta_{10}, \Theta_{12}\}$. Cada $\Theta_i \in \Phi$ se caracteriza por tener una estimación del límite inferior $\widehat{\Theta_i(\Pi)_\downarrow}$ mayor al umbral de aceptación. Lo anterior significa que aunque $\overline{\Theta_4}$ alcanzara su límite superior, ninguna configuración $\Theta_i \in \Phi$, con una confianza de $1 - \delta$, podrá obtener un menor costo que Θ_4. Esta es la razón de que las configuraciones que pertenecen a Φ son eliminadas de la competencia.

En cada iteración, π es incrementado al obtenerse ω nuevas observaciones por cada configuración candidata. Entonces se estiman los límites superior e inferior para cada $\Theta_j \in \Theta$ y el proceso de competencia, expresado en las Ecuaciones 2.3 y 2.4 se realiza. Cada vez que π es incrementado, ϵ se disminuye provocando que el rango para cada $\overline{\Theta_j}$ sea menor. Los algoritmos de competencias siguen iterando hasta que se encuentre sólo una configuración en Θ o hasta que se cumpla un número máximo de iteraciones [Rivera 09].

Capítulo 3

Estado del Arte

En este capítulo se muestra una parte de toda la literatura investigada para la realización de este proyecto. Específicamente, se describieron aquellos trabajos que realizaran el análisis del desempeño de metaheurísticos desde una perspectiva basada en la visualización de la información del proceso de optimización.

3.1. Trabajos Relacionados con Herramientas de Visualización

En particular, las características por las cuales se seleccionaron estos trabajos son que todos ellos realizan un análisis apoyándose en estrategias visuales que aporten información para dar explicaciones del comportamiento de los algoritmos. Además de que algunos de ellos proponen métodos de visualización especializados al problema y/o al metaheurístico, lo cual se relaciona estrechamente con el tema del presente trabajo.

Maarten [Maarten 04] propone en su trabajo una plantilla de software basado en agentes artificiales que pueden ser fácilmente creados, además de que la dinámica del sistema de enjambre se visualiza utilizando colores en la distribución de feromona cambiante y el movimiento de los agentes. Los agentes representan el comportamiento colectivo de las hormigas biológicas, dicho comportamiento es la base para modelar muchos de los algoritmos de optimización que el autor propone, a su vez, estos algoritmos pueden ser utilizados para resolver problemas de optimización combinatoria tales como *VRP (Vehicle Routing Problem)* y *SOP (Sequential Ordering Problem).*

La cantidad de la feromona se representa en un grafo, el cual muestra el nivel de

feromonas en el ambiente, es decir, la degradación del color de las aristas indica visualmente la intensidad de las feromonas sobre el grafo. La vista del tour actual es un tipo de visualización cuyo objetivo es resaltar la mejor ruta en la iteración actual del algoritmo elegido, para ello se utiliza un grafo que visualiza la mejor ruta actual.

Otro trabajo relacionado es el trabajo de Lau, et al. [Lau 05] el cual consiste en ajustar los valores de los parámetros del metaheurístico Búsqueda Tabú aplicado al problema *MTP (Military Transport Planning)* mediante una estrategia visual, dicho algoritmo es una estrategia de búsqueda local estocástica. Para realizar el ajuste de sus parámetros, desarrollaron un método de visualización llamado *distancia de radar*.

La distancia de radar utiliza ***soluciones élite***, las cuales consisten en las mejores soluciones encontradas durante la totalidad de la búsqueda. La ***distancia de radar*** consiste en una gráfica dual de dos dimensiones, ***Radar A*** y ***Radar B***, en las cuales el eje *X* representa las soluciones élite recordadas durante la búsqueda y el eje *Y* muestra la distancia entre la solución actual contra cada una de las soluciones élite. Cada gráfica es usada para exhibir la información de la distancia desde diferente perspectiva.

La gráfica del ***Radar A*** despliega las soluciones élite ordenadas de acuerdo a su valor objetivo en orden descendiente. La recencia de la solución élite es denotada por la intensidad de su color (el color más oscuro es el mas reciente). Además únicamente despliega un número visualmente manejable de soluciones élite (usualmente una pequeña fracción con respecto al tamaño del problema) y cualquier otra mejor solución élite encontrada reemplazará a la solución más pobre recordada.

El ***Radar B*** despliega las soluciones élite ordenadas de acuerdo a su recencia en orden decreciente. La calidad de las soluciones élite es representado por la intensidad de su color (los más oscuros muestran los mejores valores objetivos). Típicamente el número de soluciones recordadas es configurado igual que el número de tenencia Tabú.

Los ***puntos de anclaje*** son las soluciones élite (las mejores soluciones que el

programa ha encontrado durante su ejecución), éstas posteriormente se utilizan en el método de la distancia de radar para poder determinar posibles anomalías en la ejecución del algoritmo de búsqueda tabú y si tal es el caso, aplicar acciones remediales.

Ugur, et al. [Ugur 10a] proponen una simulación basada en la web a través de una herramienta llamada TSPAntSim, la cual es un software interactivo que resuelve el problema del *TSP (Traveling salesman problem)* utilizando para ello el algoritmo *ACO (Ant Colony Optimization)* y heurísticas de búsqueda local. Los algoritmos fueron puestos a prueba con problemas del benchmark TSPLIB.

El software proporciona una visualización gráfica del comportamiento del recorrido que realizan las hormigas [Ugur 10b], para ello emplea hormigas virtuales e interacción con gráficos en dos dimensiones. La herramienta consta de tres módulos: simulación, análisis y documentación. En el primer módulo, simulación, se pueden visualizar las rutas generadas por las hormigas en cada iteración. En el segundo módulo se muestran tres gráficas poligonales, las cuales representan la solución actual, la mejor solución y la desviación estándar, cada una de ellas en función del tiempo. Finalmente, en el tercer módulo se muestran los parámetros utilizados por el algoritmo y los estadísticos más usuales, media, desviación estándar, entre otros.

Ugur [Ugur 10c] realiza un trabajo similar al trabajo anterior [Ugur 10a], aquí el autor propone una herramienta de visualización interactiva basada en la web [Ugur 10d]. Esta herramienta está orientada a la resolución del problema *TSP*, para el cual una variedad de algoritmos heurísticos están disponibles para resolver el *TSP* euclidiano y plano. Sin embargo, la optimización sobre un cuboide tiene aplicaciones potenciales en las áreas de planeación de rutas sobre las caras de edificios, cuartos, mobiliario, libros, y productos o en la simulación del comportamiento de insectos. En este trabajo, los autores dirigen una variante del problema *TSP*, en la cual todos los puntos (ciudades) y las rutas (soluciones) están sobre las caras de un cuboide. Los autores desarrollaron un método híbrido efectivo basado en Algoritmos Genéticos y 2-opt, para adaptar el *TSP* euclidiano a la superficie de un cuboide. El método fue puesto a prueba con algunas instancias del benchmark TSPLIB

con resultados satisfactorios [Ugur 10c].

Los autores Pérez, et al. [Pérez 11] proponen una herramienta de visualización gráfica y tabular (estadística) para el apoyo al análisis de los Algoritmos Evolutivos y metaheurísticos aleatorizados en sus diferentes fases de ejecución. El problema al cual está enfocado esta herramienta, es al problema de empacado de objetos en contenedores (*BPP*), ya que el tipo de visualización gráfica que muestra está orientado a dicho problema.

Una de las características de este trabajo es que maneja "código embebido", el cual actúa como intermediario entre la herramienta y la implementación algorítmica.

Otro elemento relevante es que mediante el código embebible se pueden almacenar en una base de datos las soluciones de las ejecuciones generadas por el algoritmo y posteriormente reproducirlas mediante la visualización gráfica y/o tabular.

Esta herramienta tiene la característica de que no se limita a ninguna implementación algorítmica permitiendo que el usuario sea quien determine qué implementación desea visualizar. La herramienta consta de tres módulos principales: código embebible, base de datos y visualizador gráfico, la arquitectura se puede ver en la Figura 3.1.

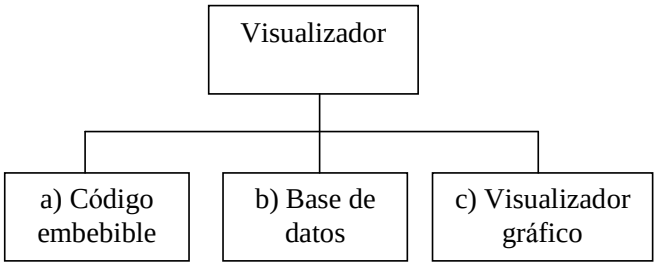


Figura 3.1. Arquitectura de la herramienta.

a) Código embebible, el cual está formado por un conjunto de funciones las cuales podrán ser insertadas en la implementación algorítmica para poder llevar a cabo la interacción con la base de datos.

b) Base de datos, está formada por un conjunto de tablas necesarias para el almacenamiento de las soluciones generadas por la implementación algorítmica.

c) Visualizador gráfico, está formado por tres principales formas de visualizar las soluciones; la primera es de forma gráfica, la segunda es de forma tabular y la tercera es la visualización del desempeño.

En la Figura 3.2 se muestra el modelo conceptual de la herramienta, se observa que es necesario insertar el código embebible (a) en la implementación algorítmica (b) con la finalidad de almacenar los parámetros necesarios para la visualización gráfica, una vez que se tiene información en la base de datos (c) es posible reproducir la ejecución mediante el visualizador, el cual brinda las opciones de reproducir las ejecuciones de forma gráfica, tabular y el desempeño general (d).

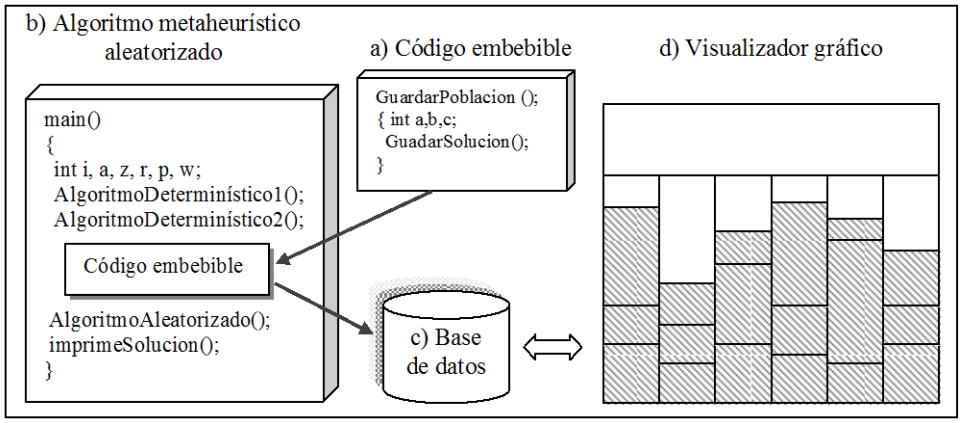


Figura 3.2. Modelo conceptual de la herramienta.

El código embebible es una sección de código especial genérico que actúa como una interfaz entre la herramienta de apoyo al análisis y la implementación del algoritmo, puede ser de tres tipos: almacenamiento de objetos, almacenamiento de soluciones y almacenamiento de parámetros, la arquitectura del módulo código embebible se muestra en la Figura 3.3.

a) Almacenamiento de objetos: esta función tiene como objetivo almacenar en la base de

datos detalles de la solución, el listado de objetos almacenados dentro del contenedor y el número de contenedor al que pertenece el objeto.

b) Almacenamiento de soluciones: esta función tiene como objetivo almacenar en la base de datos el número total de contenedores utilizados en la solución i-ésima.

c) Almacenamiento de parámetros: está función tiene como objetivo almacenar en la base de datos algunos parámetros de la instancia como por ejemplo la capacidad del contenedor.

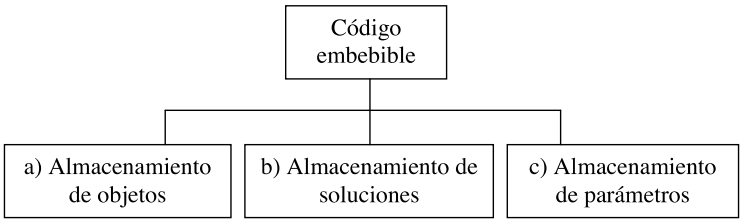


Figura 3.3. Arquitectura general del código embebible.

La base de datos está formada por cuatro tablas cuyo objetivo es almacenar la información a visualizar. La Figura 3.4 muestra el esquema entidad-relación de la base de datos, cuyas tablas se describen a continuación:

a) "*Instancias*", almacena los parámetros de las instancias, está formada por los siguientes campos: "*Instancia*", almacena el nombre de la instancia que es el identificador de la tabla; "*Solucion_internet*", almacena la mejor solución reportada en internet; "*N*", almacena el número total de objetos que contiene la instancia y "*C*", almacena la capacidad el contenedor.

b) "*Descripcion_Solucion*", almacena las soluciones producidas por el algoritmo, formada por los siguientes campos: "*Numero_solucion*", almacena el número de solución generado por el algoritmo; "*Numero_corrida*", almacena el número de ejecución de una instancia dada; "*instancia*", almacena el nombre de la instancia; "*Solucion_encontrada*", almacena el número de contenedores necesarios para almacenar los pesos y "*Id_solucion*", almacena el nombre de la imagen que se generó en esta solución.

c) "*Descripcion_Contenedor*", almacena los pesos guardados en cada contenedor, formada por los siguientes campos: "*Id_solucion*", almacena el nombre de la imagen que se generó en esta solución; "*Peso*", almacena cada uno de los pesos contenidos en una instancia; "*Frecuencia*", almacena la frecuencia de cada peso y "*Numero_contenedor*", almacena el número de contenedor al que pertenece el peso.

d) "*Elementos_Instancias*", almacena los pesos que contienen las instancias así como la frecuencia de cada uno de ellos, está formada por los siguientes campos: "*Instancia*", almacena el nombre de la instancia; "*Peso*", almacena cada uno de los pesos contenidos en una instancia y "*Frecuencia*", almacena la frecuencia de cada peso.

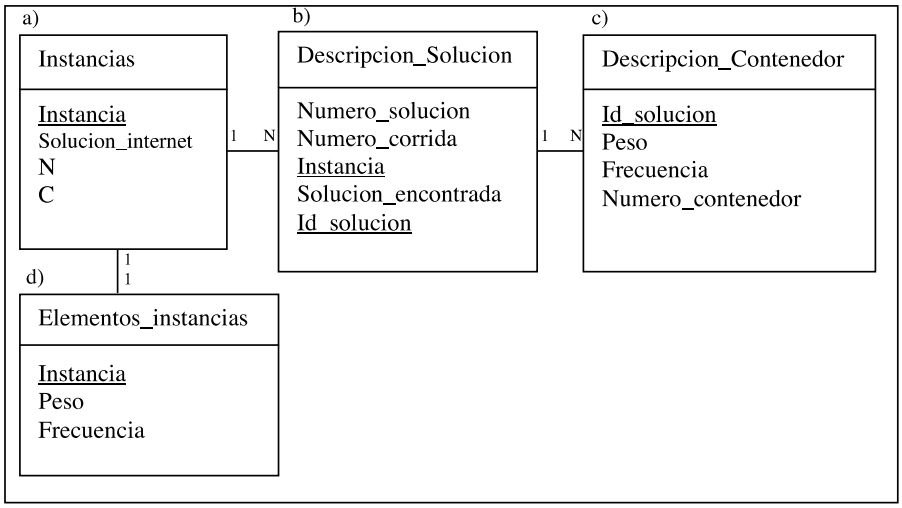


Figura 3.4. Esquema entidad-relación de la base de datos de la herramienta.

El módulo llamado "Visualizador gráfico" permite visualizar las soluciones generadas por el algoritmo, el cual está formado por tres módulos principales: visualización gráfica, visualización tabular y visualizador del desempeño, la arquitectura es mostrada en la Figura 3.5.

a) Visualización gráfica, este módulo permite visualizar de manera gráfica el acomodo de los pesos en los contenedores de cada una de las soluciones que el algoritmo produce de manera intermedia hasta generar la mejor solución. Tiene la opción de guardar la

visualización obtenida como imagen para su posterior análisis.

b) Visualización tabular, este módulo visualiza de manera tabular el acomodo de los pesos en los contenedores, además de mostrar el residuo de cada contenedor.

c) Visualizador del desempeño, visualiza gráficamente el valor que se obtiene para cada una de las soluciones que el algoritmo produce hasta generar la mejor solución.

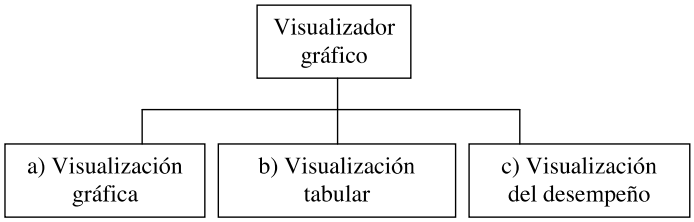


Figura 3.5. Arquitectura general del visualizador gráfico.

Finalmente, después de haber revisado los trabajos relacionados se muestra un resumen de éstos, en comparativa con la herramienta desarrollada en el presente proyecto de investigación. La Tabla 3.1 muestra dicho resumen.

Tabla 3.1. Tabla comparativa del estado del arte.

Investigador	Problema		Algoritmo		Tipo de estructura de almacenamiento	Inclusión de nuevos índices	Pre-procesamiento de datos de entrada	Caracterización del desempeño		Análisis Visual del desempeño	
	Aportado por herramienta	No Aportado por herramienta	Aportado por herramienta	No Aportado por herramienta				Mediante índices	Mediante análisis de superficie de aptitudes	Mediante software comercial	Mediante software propio
Maarten 2004	✓		✓		-			✓			Visual Swarm System
Lau 2005	✓		✓		-			✓			V-MDF
Pérez 2007	-	-	-	-	-			✓		✓	
Quiroz 2009	-	-	-	-	-			✓		✓	
Ugur 2010a	✓		✓		-			✓			TSPAntSim
Ugur 2010c	✓		✓		-						CuboidTSP
Pérez 2011	✓			✓	Base de Datos			✓			✓
Esta tesis		✓		✓	Arreglo dinámico multidimensional	✓	✓	✓	✓		VisTHAA

3.2. Discusión de Trabajos Relacionados

Derivado de la revisión del estado del arte, se observa que la mayoría de las herramientas resuelven un problema específico, por lo cual los métodos de visualización a los que van enfocados son específicos de su determinado problema. En contraste, los métodos de visualización en VisTHAA fueron diseñados para que independientemente del problema éstos se pudieran aplicar.

Además, con excepción de [Pérez 11] las otras herramientas tienen implementado el algoritmo solucionador de sus respectivos problemas de optimización, lo cual las limita.

Otro aspecto a destacar es el tipo de estructura de almacenamiento que utilizan las herramientas analizadas, en la mayoría de los trabajos revisados no se especifica. Sin embargo, en [Pérez 11] se utiliza una base de datos que está pensada específicamente en el problema del empacado de objetos en contenedores. Por otra parte, VisTHAA utiliza como estructura principal un arreglo dinámico multidimensional, el cual fue diseñado de tal manera, que sin importar el tipo de problema, la estructura pudiera almacenar sus datos, es decir, se diseñó un tipo de almacenamiento genérico.

Un aspecto importante de VisTHAA en comparación con las otra herramientas, es que ninguna de éstas permite al investigador introducir sus propios índices, simplemente muestran los estadísticos que tienen implementados por defecto. Además, tampoco realizan un pre-procesamiento de los datos de entrada.

La mayoría de las herramientas realizan la caracterización del desempeño a través de índices, incluyendo a VisTHAA, sin embargo únicamente este trabajo la realiza a través del análisis de la superficie de aptitudes.

Con excepción de los trabajos [Pérez 07, Quiroz 09] los demás proyectos desarrollaron su propia herramienta de visualización, para realizar el análisis visual del

desempeño de los algoritmos. En [Pérez 07] se utilizó TETRAD 4.3.8, Minitab 14 y SPSS 15; mientras que en [Quiroz 09] se utilizó Microsoft© Excel, TETRAD, entre otros.

El trabajo que más se asemeja al presente proyecto de investigación, puesto que trabaja con el mismo problema de optimización (BPP) y además ha desarrollado su propia herramienta, es el de Pérez [Pérez 11]. Sin embargo, a diferencia de este proyecto, en dicho trabajo no se realiza el pre-procesamiento de entradas, no permite al investigador introducir sus propios índices, no se visualiza la superficie de aptitudes en tres dimensiones y no tiene implementada ninguna prueba estadística para el análisis comparativo de algoritmos heurísticos. Otro aspecto importante a señalar de esta herramienta, es que para que pueda funcionar es necesario insertar el código embebido al algoritmo del investigador.

Capítulo 4

Estrategias de Mejora

En este capítulo se describen las estrategias que se desarrollaron en el presente trabajo de investigación, para que el investigador pueda hacer uso de ellas y así mejorar el desempeño de sus algoritmos, dichas estrategias se incorporaron a la herramienta VisTHAA.

4.1. Metodología Propuesta

En la Figura 4.1 se presenta un diagrama general, el cual muestra los módulos donde se desarrollaron una o más estrategias de mejora.

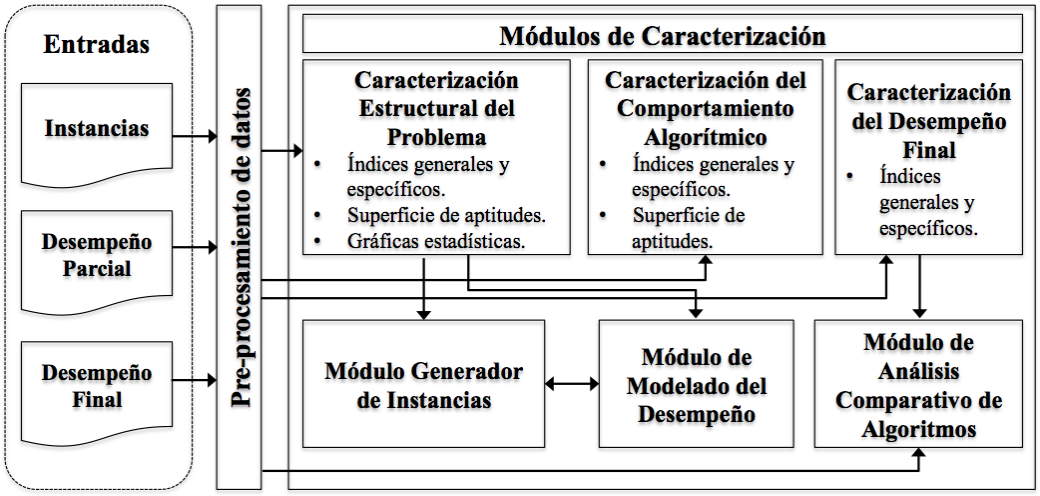


Figura 4.1. Arquitectura de la herramienta VisTHAA.

De la Figura 4.1 se puede observar que la arquitectura de VisTHAA contempla siete módulos, de los cuales en este proyecto se desarrollaron cinco: módulo de pre-

procesamiento de datos, módulo de caracterización estructural del problema, módulo de caracterización del comportamiento algorítmico, módulo de caracterización del desempeño final, y módulo de análisis comparativo de algoritmos. Estos módulos se describirán a lo largo de este capítulo.

4.2. Puntos de vista de los usuarios potenciales de VisTHAA

Antes de explicar cada una de las estrategias desarrolladas en la presente investigación, es necesario primero conocer lo que usuarios potenciales de la herramienta desean de ésta, es por ello que se realizó la recopilación de los puntos de vista de los antes mencionados usuarios potenciales. La Tabla 4.1 muestra dichos puntos de vista.

Tabla 4.1. Sugerencias hechas por usuarios potenciales de VisTHAA.

Problema	Algoritmo	Sugerencia
BPP	ACO HGGA-BP	Que las gráficas tridimensionales posean la capacidad de rotación, para así poder tener una mejor visualización y obtener un mayor conocimiento.
BPP	AG HGGA-BP	Que se pueda variar la escala de cualquiera de las dimensiones en las gráficas.
SQRP	ACO HH_AdaNAS	Que se muestren gráficas de dispersión de dos variables.
SQRP	ACO HH_AdaNAS	Que las gráficas muestren, de manera automática o por selección del investigador, el mínimo, el máximo y el promedio de los datos medidos y graficados.
SQRP	ACO HH_AdaNAS	Obtener la curva de aprendizaje, mediante regresión cuadrática o algo similar, para poder apreciar claramente diferencias entre los desempeños de dos algoritmos.
BPP	AG HGGA-BP	Realizar la gráfica del índice que cuantifica el número de objetos que ocupan un tanto por ciento con respecto a la capacidad del contenedor.

Las sugerencias anteriores, fueron tomadas en cuenta al momento de realizar las implementaciones y seleccionar los índices de caracterización que a los potenciales usuarios de VisTHAA les interesa.

4.3. Introducción de Datos a VisTHAA

Debido a que el investigador, necesita introducir los datos de sus instancias en la herramienta fue necesario desarrollar un módulo que ayude a pre-procesar las instancias de los investigadores. Es decir, se proporciona un módulo que recibe la instancia del investigador y le da el formato para trabajar en la herramienta VisTHAA.

Método para la Introducción de Instancias a VisTHAA

En la versión fuera de línea de VisTHAA, los datos de las instancias son proporcionados por el investigador, utilizando su propio formato. Este método propone pre-procesar las instancias, las cuales son datos referentes a un caso particular del problema, para ajustarse a un formato global.

Motivaciones para la Creación del Método

En la práctica existe una gran variedad de formatos en los archivos de las instancias de los investigadores. Por ejemplo en instancias del Bin Packing, algunos investigadores en su primera línea especifican la capacidad del contenedor, luego la cantidad de objetos y finalmente la lista de los pesos de los objetos; mientras que otros, por ejemplo, especifican el número de objetos en la primera línea, la capacidad del contenedor y la lista de los pesos de los objetos.

Una vez identificadas las características de la instancia, se decide que la herramienta debe tener la capacidad de transformar el formato de entrada del investigador a un formato estandarizado y no que el investigador tenga que adecuar sus archivos de instancias a algún formato específico de VisTHAA.

La forma en la que se propone lograr este objetivo es desarrollando un metalenguaje, es decir, un conjunto de palabras reservadas que VisTHAA reconozca y que a través de ellas el investigador pueda describir sus instancias.

Resulta evidente que para desarrollar el metalenguaje es necesario también desarrollar un autómata que reconozca dicho metalenguaje e incorporarlo en VisTHAA. Para ello este trabajo se soporta en algunos de los principios fundamentales de los compiladores.

Método para el Pre-Procesamiento de las Instancias

Este método consiste en la creación de un conjunto de palabras reservadas (metalenguaje) que VisTHAA reconozca, para que el investigador haga uso de ellas y así describir sus instancias. Una vez hecho esto es necesario, crear un pequeño autómata que identifique las palabras reservadas, para ser incluido en VisTHAA. Además, es necesario contar con un conjunto de estructuras de datos que almacenen el nombre de las variables, su dimensionalidad, sus valores, entre otros.

En este método son necesarios dos archivos de entrada. Los cuales se describen a continuación:

- El primer archivo, denominado ***logbook***, debe de indicar el ***número de instancias*** que va a leer VisTHAA, el nombre y la ruta del ***archivo de instrucciones*** y por último los nombres y las rutas de los ***archivos de las instancias***.
- El segundo archivo, llamado ***metainstance***, debe contener las ***instrucciones*** necesarias para leer correctamente las instancias, haciendo uso del ***metalenguaje***.

Diccionario de Palabras Reservadas

Para el archivo ***logbook*** se han definido algunas ***palabras reservadas***, las cuales se presentan a continuación junto con su descripción.

- **instances(int);**. Le indica al compilador que el número entero que está dentro de los paréntesis corresponde con el número total de instancias que debe leer.

- **instructions_file(“string”);**. Indica que la cadena de texto que se espera dentro de los paréntesis representa la ruta y el nombre del archivo de instrucciones (el que hace uso del metalenguaje) para leer las instancias.
- **instances_names(void) {<instancia_1.txt>,...,<instancia_n.txt>}** . Son los nombres de los archivos de instancias, separados por comas. La cantidad de instancias dentro de las llaves debe de coincidir con el entero dentro de los paréntesis de la palabra reservada *instances*.

La lista de ***palabras reservadas*** que se pueden utilizar en el archivo ***metainstance*** es la siguiente:

- **comments(int);**. Indica A VisTHAA que el valor entero que está dentro de los paréntesis corresponde a la cantidad de líneas de comentarios en las instancias.
- **variables(int);**. Le indica a VisTHAA que el número entero dentro de los paréntesis corresponde a la cantidad de variables consecutivas en el archivo de la instancia. Cuando se utiliza esta palabra reservada, inmediatamente se debe de especificar el tipo de variable que va a ser leído mediante una sintaxis específica, por ejemplo si se trata de un tipo primitivo de dato, de un vector o de una matriz; si es de tipo entero, flotante, caracter, etc. Evidentemente si el valor dentro de los paréntesis es mayor que uno, se debe especificar cada tipo de variable.
- **constants(int);**. Indica que la cantidad de constantes a ser leídas es igual al número entero dentro de sus paréntesis.

Sintaxis de las Palabras Reservadas

Debido a que las instancias generalmente contienen valores de variables, constantes y algunos comentarios de los investigadores, se desarrollaron palabras reservadas con su sintaxis específica.

Una vez definidas las palabras reservadas, se debe definir la sintaxis a ser utilizada para especificar las variables, ya que se reconocen tres tipos de éstas que pueden ser introducidos a VisTHAA: variables simples, vectores y matrices.

Para introducir una ***variable simple***, primero se debe especificar el nombre identificador de dicha variable, seguido de un separador, el cual puede ser un espacio en blanco o un tabulador y finalmente se debe especificar el tipo de dato de la variable.

Si se desea introducir una variable de tipo vector, al igual que con las variables simples, primero se debe especificar el identificador, seguido de un separador, posteriormente se debe escribir la palabra ***vector***, luego un separador, el tipo de dato, de nuevo un separador y finalmente el tamaño del vector (el cual puede ser el nombre identificador de una variable previamente introducida).

Finalmente, si se desea introducir a VisTHAA una variable de tipo matriz, es necesario seguir el siguiente formato: primero, el nombre identificador, luego un separador, acto seguido se debe escribir la palabra matriz, separador, tipo de dato, número de filas y número de columnas. Al igual que en el tamaño del vector, el número de filas y el número de columnas puede estar en función de variables previamente introducidas.

Las sintaxis de las palabras reservadas, pueden ser resumidas en la Tabla 4.2, la cual se muestra a continuación.

Tabla 4.2. Sintaxis requerida por las palabras reservadas.

Variable	**1er token**	**2do token**	**3er token**	**4to token**	**5to token**
SIMPLE	identificador	tipo de dato			
VECTOR	identificador	***vector***	tipo de dato	tamaño del vector	
MATRIZ	identificador	***matrix***	tipo de dato	número de filas	número de columnas

Las Figuras 4.2 y 4.3 muestran un ejemplo de cómo se deben introducir dos instancias de BPP a VisTHAA. La Figura 4.2 muestra la manera de cómo se debe de realizar la descripción de las instancias. Por otra parte, la Figura 4.3 ejemplifica la forma en como se debe crear el *logbook*.

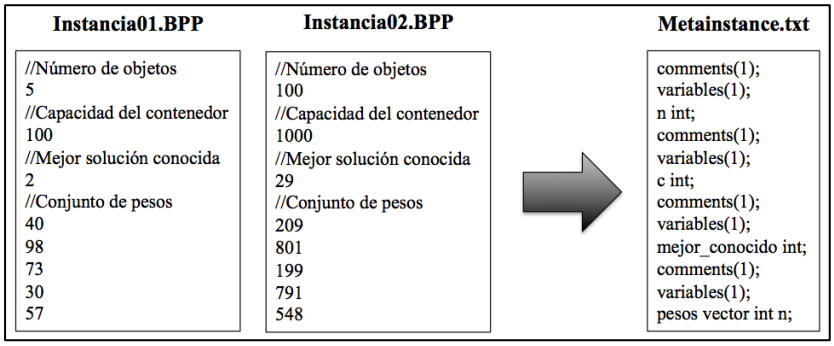


Figura 4.2. Ejemplo de la descripción de una instancia para introducirla a VisTHAA.

Como se puede observar en la Figura 4.2 la descripción de las instancias se realiza en un archivo llamado *metainstance.txt*, utilizando las palabras reservadas descritas anteriormente. Resulta evidente que, como la descripción se realiza una vez para todas las instancias, éstas deben tener el mismo formato. Tal es el caso de las instancias *Instancia01.BPP* e *Instancia02.BPP*, aunque representan dos casos distintos (diferentes valores), ambas tienen los comentarios y las variables en las mismas líneas, i.e., en la primera línea ambas tienen un comentario, luego en la segunda línea tienen el valor de la variable *n*, y así sucesivamente.

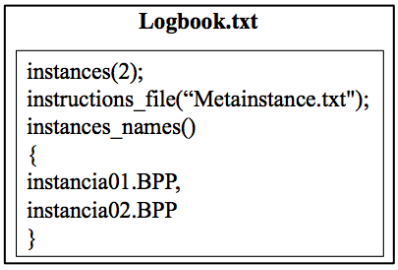


Figura 4.3. Ejemplo de la creación del *logbook* para introducir las dos instancias de BPP.

En la Figura 4.3 se ejemplifica la manera de crearse el archivo *logbook* para introducir las instancias de la Figura 4.2. Se puede ver que lo primero que se especifica es

la cantidad de instancias a introducir, en este caso dos. Luego, el nombre del archivo donde se encuentra la descripción de las instancias, *Metainstance.txt* y finalmente los nombres de cada una de las instancias a ser introducidas. Lo anterior de acuerdo a las palabras reservadas explicadas con anterioridad.

Estructuras de Datos

Aunado a lo anterior se debe contar con estructuras de datos tales como: tabla de símbolos, tipos de variables, tipos de datos y longitud de variables (dimensionalidad). Además de tener a disposición un manejador de errores.

La ***tabla de símbolos*** almacenará los nombres de las variables; la estructura tipos ***de variables*** almacenará los tipos de variables leídas, éstos pueden ser: simple, vector y matriz; la tabla ***tipos de datos*** almacenará los tipos de datos de las variables, por ejemplo: entero, flotante, caracter, doble, cadena de caracteres, entre otros; la estructura ***longitud*** almacenará la dimensionalidad de las variables.

La estructura de datos principal, denominada ***datos***, será una ***arreglo multidimensional*** de cuatro dimensiones de tipo ***double***, en ella se almacenarán los datos extraídos de las instancias para que VisTHAA pueda hacer uso de ellos.

La primera dimensión de la estructura datos corresponde al número de la instancia (ya que se puede introducir más de una instancia), la segunda dimensión corresponde al número de la variable (en orden de lectura). Las últimas dos dimensiones de la estructura son para la dimensionalidad propia de cada una de las variables, por ejemplo, si se trata de una variable de tipo simple, la dimensionalidad de ésta es de uno por uno; si se trata de una variable de tipo vector, su dimensión es de uno por n; mientras que si se trata de una matriz, la dimensión es de *m* por *n*.

La estructura ***datos*** queda representada de la siguiente manera:
datos [numero_instancia] [numero_variable] [numero_fila] [numero_columna]

La Figura 4.4 muestra un ejemplo de cómo interactúan las estructuras explicadas con anterioridad. Para ello se introducirá una instancia de BPP a VisTHAA.

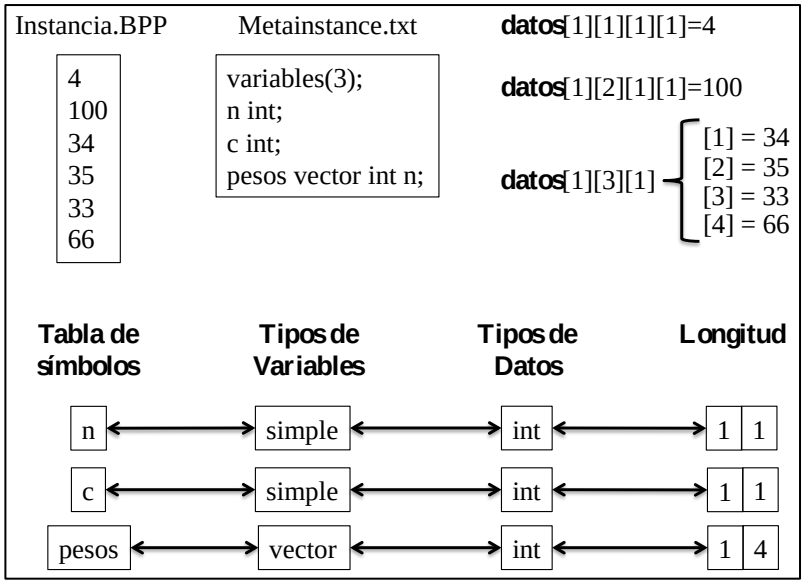


Figura 4.4. Ejemplo completo del almacenamiento de una instancia en VisTHAA.

Como puede verse en la Figura 4.4 para introducir la instancia de BPP a VisTHAA se debe describir ésta como se ejemplifica en la Figura 4.2. Una vez hecho esto, cuando VisTHAA lea el archivo metainstance, relacionará cada variable con su correspondiente posición específica en cada estructura de datos; i.e., para la ***primera*** variable, número de objetos, VisTHAA almacenará el nombre de ésta, *n*, en la ***primera*** posición de la *Tabla de Símbolos*; su tipo en la ***primera*** posición de la estructura *Tipos de Variables*; su tipo de dato en la ***primera*** posición de la estructura *Tipo de Datos*; su longitud en la ***primera*** posición de la estructura *Longitud*; y finalmente su valor en la ***primera*** posición de la segunda dimensión de la estructura principal, ***datos***.

Para las variables restantes se procede de manera similar; sin embargo, en el caso específico de la última variable, *pesos*, al ser de tipo *vector*, en la estructura de datos principal se debe reservar el espacio de memoria suficiente. La manera en que VisTHAA sabe cuánto espacio debe reservar es cuando lee el último token que describe a dicha

variable en el *metainstance*, en este caso el último token es otra variable, *n*, y ya que *n* fue leída con anterioridad a *pesos*, simplemente basta con buscarla en la *Tabla de Símbolos*, una vez encontrada, la posición de *n* en la *Tabla de Símbolos* nos dice en qué posición de la estructura principal ***datos*** estará su valor (ya que de esta forma se relacionan las estructuras de datos). Además del índice de *n* en la *Tabla de Símbolos*, también se debe tener en cuenta de cuál instancia y de qué tipo de variable se trata; en este caso únicamente se introdujo una instancia y el tipo de variable de *n* es *simple*, por lo tanto ya se cuenta con información suficiente para buscar el valor de *n* y así poder saber de cuantos elementos estará constituido el vector *pesos*. En la estructura principal ***datos***, se debe buscar el valor en la primera instancia (ya que sólo hay una instancia) y primera variable (por la posición de *n* en la *Tabla de Símbolos*). Además como se sabe que el tipo de *n* es simple, entonces las dos últimas dimensiones son de uno por uno. De esta manera se sabe que el valor de *n* es 4 (***datos***[1][1][1][1]) y la cantidad de elementos del vector *pesos* también.

Se debe notar que como la descripción de la variable *pesos* en el *metainstance* está en función de otra variable, *n*, el número de elementos de cada variable *pesos* en su respectiva instancia puede variar; es decir, una instancia puede tener cien elementos, mientras que otra puede tener doscientos, y sin embargo VisTHAA siempre reservará el espacio de memoria suficiente para cada una de ellas.

4.4. Índices en VisTHAA

Una parte importante de la herramienta, es que el investigador puede hacer uso de ésta para realizar cálculos que puedan ser importantes para él. Es por ello que se realizó la implementación de algunos de los estadísticos de uso más frecuente, así como también se implementaron otros índices reportados en la literatura para un problema específico, en este caso el BPP. Sin embargo, una de las aportaciones más importantes en este trabajo, es que se le permite al investigador introducir sus propios índices (ecuaciones) a la herramienta, de manera tal que se pueda obtener un mayor conocimiento.

Índices por defecto en VisTHAA

Al igual que la mayoría de las herramientas revisadas del estado del arte, a VisTHAA también se le integraron algunos índices por defecto, los cuales son para la caracterización de instancias de BPP [Quiroz 09, Pérez 07], para la caracterización del desempeño final del un algoritmo [Quiroz 09] y algunos de los estadísticos de uso más frecuente. En las siguientes secciones se describen cada uno de ellos.

Estadísticos de uso más frecuente

VisTHAA pretende ser una herramienta de visualización y a la vez estadística, es por ello que se implementaron algunos de los estadísticos más utilizados por la mayoría de los investigadores, la Tabla 4.3 muestra los estadísticos implementados:

Tabla 4.3. Estadísticos de uso más frecuente implementados en VisTHAA.

Estadístico	Descripción
Media	Obtiene el promedio de un conjunto de datos. Véase la sección A.1 para su definición y un ejemplo.
Mediana	Obtiene la mediana de un conjunto de datos. Véasela sección A.2 para su definición y un ejemplo.
Moda	Obtiene la moda de un conjunto de datos. Véase la sección A.3 para su definición y un ejemplo.
Varianza	Obtiene la varianza de un conjunto de datos. Véase la sección A.4 para su definición y un ejemplo.
Desviación estándar	Obtiene la desviación estándar de un conjunto de datos. Véase la sección A.5 para su definición y un ejemplo.
Búsqueda del máximo	Obtiene el mayor valor de un conjunto de datos. Véase la sección A.6 para su definición y un ejemplo.
Búsqueda del mínimo	Obtiene el menor valor de un conjunto de datos. Véase la sección A.7 para su definición y un ejemplo.
Sumatoria	Obtiene la suma de todos los elementos de un conjunto de datos.

Índices de Caracterización del Desempeño Algorítmico

Por otra parte, los índices de la literatura para el problema BPP implementados, se resumen en la Tabla 4.4.

Tabla 4.4. Resumen de los índices de caracterización implementados en VisTHAA.

Caracterización	Índice	Propósito
Instancia	Menor	Permite conocer la estructura de una instancia de BPP y predecir el comportamiento que tendrá el metaheurístico.
	Mayor	
	Multiplicidad	
	MaxRepe	
	Uniformidad	
Desempeño Final	Max_MejorF$_{BPP}$	Permite conocer y cuantificar la efectividad de un algoritmo metaheurístico.
	Min_MejorF$_{BPP}$	
	Prom_MejorF$_{BPP}$	
	Radio_Teórico	
	Desv_Z$_{enc}$	
	Tiempo	
	Generación	
	Objetos_Contenedor	
Rugosidad	Coeficiente de Autocorrelación.	Permite conocer cuán correlacionadas están las soluciones de una instancia.
	Longitud de Autocorrelación.	
	Contenido de información.	

Introducción de Índices a VisTHAA

En algunas ocasiones los índices que las herramientas estadísticas y/o de visualización tienen por defecto no son suficientes para los fines del investigador, ya que pueden llegar a ser muy generales y por tanto no contribuir de una manera significativa en la investigación.

Una parte importante de la aportación de este trabajo, es otorgarle a VisTHAA la capacidad para que permita a los investigadores introducir sus propios índices, es decir, sus

propias ecuaciones y así obtener mayor conocimiento de su problema o del comportamiento que tiene su algoritmo.

Descripción general del método que permite la introducción de índices

Primero es necesario que el investigador introduzca la ecuación (índice) que quiere evaluar, esto se puede realizar de manera asistida, es decir, con ayuda de una paleta de opciones y las variables disponibles; o bien de manera no asistida, a través de una caja de texto en donde el investigador debe escribir la ecuación en notación infija.

Posteriormente, se tiene que realizar la transformación de la ecuación en notación infija a notación polaca inversa (notación postfija) para que pueda ser evaluada la expresión.

Finalmente, se evalúa la expresión ya en notación polaca inversa sustituyendo las variables de la tabla de símbolos. La Figura 4.5 ilustra la forma en cómo funciona el método para introducir nuevos índices.

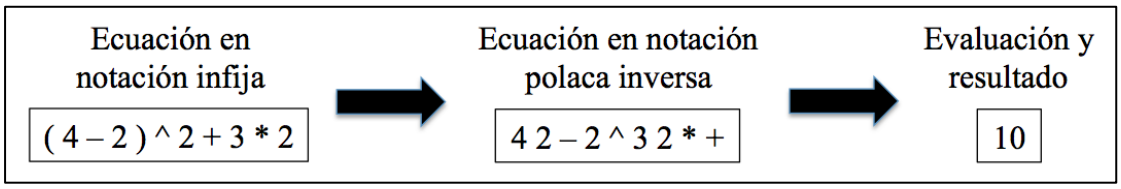


Figura 4.5. Ejemplificación del método para introducir nuevos índices.

El principal objetivo de realizar la conversión entre notaciones es que resulta mucho más sencillo evaluar la expresión en notación polaca inversa, ya que se realiza la lectura secuencialmente de izquierda a derecha y las operaciones se encuentran debidamente jerarquizadas, es decir, no es necesario emplear el uso de paréntesis o algún otro método para agrupar operaciones con prioridad mayor.

Algoritmo Shunting yard

Una vez que el investigador de alguna de las dos maneras (asistida o no asistida) ha introducido la ecuación en notación infija, es necesario realizar la transformación entre notaciones, de tal manera que pueda ser fácilmente calculada por la computadora. Para lograr esto, se utilizó el algoritmo *shunting yard*, desarrollado por Edsger Dijkstra en 1961 [Dijkstra 61].

El algoritmo *shunting yard* funciona con dos estructuras de datos: una cola de salidas y una pila de operadores. En el algoritmo 4.1, de la línea 1 a la línea 1.7.4 el algoritmo lee una cadena de caracteres completa, en cada paso pregunta qué tipo de carácter está leyendo y en base a eso decide en cual estructura guardar dicho caracter. De la línea 2 a la 2.1.2 el algoritmo vacía los caracteres contenidos en la pila de operadores a la cola de salida, la cual resulta en la expresión postfija deseada. El algoritmo 4.1 muestra en detalle el algoritmo *shunting yard*.

Algoritmo 4.1. Algoritmo *shunting yard* [Dijkstra 61].

1. ***Mientras*** haya tokens por ser leídos, hacer:
1.1. ***Leer*** token
1.2. ***Si*** el token es un número o una variable, entonces: ***Agregarlo*** a la cola de salida
1.3. ***Si*** el token es un token de una función (logaritmo, seno, etc.), entonces: ***Ponerlo*** en la pila de los operadores
1.4. ***Si*** el token es un separador de argumento de función (i.e., una coma), entonces:
1.4.1. ***Mientras*** que el tope de la pila de operadores no sea un paréntesis de apertura, hacer:
1.4.1.1. ***Quitar*** de la pila los operadores y agregarlos a la cola de salida
1.4.2. ***Si*** no se encontró el paréntesis de apertura, entonces: **Error**: Separador mal colocado o paréntesis mal emparejados
1.5. ***Si*** el token es un operador (oper1), entonces:
1.5.1. ***Mientras*** haya un operador (oper2) en el tope de la pila de operadores (excluyendo al paréntesis de apertura) **&&** oper1 es asociativo izquierdo y su precedencia

(prioridad) es menor que la de oper2 **||** oper1 es asociativo derecho y su precedencia es menor que la de oper2, entonces:

1.5.1.1. ***Retirar*** de la pila de operadores oper2 y agregarlo a la cola de salida

1.5.2. ***Poner*** oper1 en el tope de la pila de operadores

1.6. ***Si*** el token es un paréntesis de apertura, entonces: ***Ponerlo*** en la pila de operadores

1.7. Si el token es un paréntesis de cerradura, entonces:

1.7.1. ***Hasta*** que el tope de la pila sea un paréntesis de apertura, hacer:

1.7.1.1. ***Retirar*** los operadores de la pila y ***agregarlos*** a la cola de salida

1.7.2. ***Quitar*** el paréntesis de apertura de la pila, pero ***no*** agregarlo a la cola de salida

1.7.3. ***Si*** el tope de la pila es un token de función, entonces: ***Agregarlo*** a la cola de salida

1.7.4. ***Si*** la pila se termina sin encontrar un paréntesis de apertura, entonces: ***Error:*** hay un paréntesis sin pareja

2. ***Si*** no hay más tokens por leer, entonces:

2.1. Mientras todavía haya operadores en la pila, hacer:

2.1.1. ***Si*** el token del operador en el tope de la pila es un paréntesis de apertura, entonces: ***Error:*** Hay un paréntesis sin pareja

2.1.2. ***Quitar*** el operador del tope de la pila y ***agregarlo*** a la cola de salida

Evaluación de la expresión postfija

Una vez realizada la transformación entre notaciones, es necesario evaluar la expresión en notación polaca inversa y así obtener el resultado. Para tal efecto se deben seguir estas tres sencillas reglas [Borbón 06]:

1) Si lo que sigue en la expresión es un número, se agrega a la pila de números.

2) Si sigue una operación que ocupa dos números (como la suma y la resta) se sacan los dos últimos números de la pila, se realiza la operación y se introduce el resultado en la pila.
3) Si es una función que ocupa un sólo número (como seno o coseno) entonces se saca un número de la pila, se evalúa y se guarda el resultado.

4.5. Estrategias de Visualización de la Información

En las siguientes secciones se describen las estrategias de visualización de la información desarrolladas para que el investigador pueda obtener un beneficio de ellas y así obtener un mayor entendimiento de su problema o del comportamiento de su algoritmo.

Visualización del problema

El objetivo de la visualización de un problema de optimización, es el encontrar relaciones, patrones o tendencias en los datos, permitiendo esto desarrollar estrategias que pudieran mejorar el desempeño algorítmico a partir de los datos observados.

Método para visualizar una instancia de BPP

Se desarrolló un método que permite la visualización de una instancia del problema BPP, para ello se tomaron en consideración los siguientes puntos:

1. Los objetos serán representados a través de barras en una gráfica de barras bidimensional, en donde el eje de las abscisas representará el número del objeto, mientras que el eje de las ordenadas será el peso de los objetos.
2. VisTHAA buscará automáticamente la variable cuyo tipo sea *vector* y en base a lo encontrado graficará los valores correspondientes.

Método para visualizar una instancia de BPP ordenada ascendentemente

También se desarrolló un método que consiste en visualizar los pesos de los objetos, de igual manera que el descrito en la sección anterior, sin embargo en este método se ordenan los datos en forma ascendente, para así poder apreciar mejor las tendencias en la distribución de los pesos de los objetos. Este método respeta los puntos del método arriba citado.

Método para Visualizar la Superficie de Aptitudes en Dos Dimensiones

En trabajos de la literatura ya hay métodos para realizar la visualización del espacio de búsqueda, específicamente en [Pérez 07] se propone uno que visualiza el espacio en dos dimensiones, a través del cálculo parcial del índice contenido de información (véase la sección C.3). Este trabajo retoma el método de Pérez y lo agrega a VisTHAA.

Para lograr la visualización de la superficie de aptitudes, es necesario contar con los valores de aptitud, en [Pérez 07] se especifica que los valores de aptitud utilizados corresponden a los valores obtenidos tras ejecutar un algoritmo de búsqueda tabú (algoritmo central en su trabajo). Sin embargo, como este trabajo pretende caracterizar la instancia de BPP y no el comportamiento de un determinado algoritmo, se optó por implementar un algoritmo de caminata aleatoria. La razón principal por la cual se seleccionó dicho algoritmo sobre cualquier otro, ya sea metaheurístico o no, es porque al ejecutar el algoritmo de caminata aleatoria, éste extrae una muestra que resulta ***representativa*** de la población (instancia), esto debido a que la caminata aleatoria asigna la misma probabilidad de elección a cada una de las soluciones del espacio de búsqueda [Gómez 09]. Por otro lado, los algoritmos metaheurísticos, buscan obtener los mejores resultados posibles, por lo que dan preferencia a zonas del espacio con potencial prometedor, el resultado es que la muestra que se extrae ***no es representativa*** de la instancia.

Una vez obtenidos los valores de las funciones de aptitud, lo que sigue es la construcción de una cadena $S(\varepsilon) \in \{\bar{1}, 0, 1\}$. Finalmente, se construye la gráfica bidimensional conforme a lo descrito en la sección C.3.

Método para Visualizar la Superficie de Aptitudes en Tres Dimensiones

La visualización de la superficie de aptitudes se llevó al siguiente nivel, visualizarla en las tres dimensiones espaciales. La idea general del método es extraer una muestra representativa de la instancia a través de un algoritmo Random Walk. Luego, ese conjunto de datos unidimensional convertirlo en un conjunto bidimensional sin alterar de ninguna manera los valores de aptitud calculados. Finalmente, ya con tres variables por cada solución (valor de fila, valor de columna y valor de aptitud) es posible visualizar dichos puntos en el espacio.

Descripción General del Algoritmo Random Walk

La idea general del algoritmo *Random Walk* es obtener una solución inicial factible, y a partir de ella generar soluciones vecinas iterativamente hasta un cierto número de pasos predefinidos.

En cada paso de la caminata aleatoria una nueva solución es evaluada, sin embargo no es de interés que el algoritmo reporte la mejor solución alcanzada durante su ejecución, ya que únicamente se empleará para caracterizar el espacio de búsqueda de una instancia de BPP. El algoritmo implementado se detalla en el Algoritmo 4.2:

Algoritmo 4.2. Algoritmo de caminata aleatoria que resuelve el BPP.

1. **Generar** una solución inicial factible
2. **Computar** los valores de la función objetivo y la función de aptitud
3. **Para** i = 1 hasta *N* - 1, hacer:
3.1. **Generar** una solución vecina de la solución actual
3.2. **Computar** los valores de la función objetivo y la función de aptitud
4. **Almacenar** los resultados en un archivo de texto plano

donde:
N es el número de pasos de la caminata aleatoria.

Representación de la Solución Candidata

El problema del empacado de objetos en contenedores (BPP por sus siglas en inglés) consiste en acomodar objetos de diversos pesos en contenedores de capacidad limitada, la cantidad de contenedores que se pueden utilizar es ilimitada.

De manera natural, uno puede pensar en una solución de BPP como una permutación, en donde cada elemento de dicha permutación esté representado por un objeto.

Por ejemplo, dado el siguiente vector de *pesos* una solución candidata puede representarse de la siguiente manera:

pesos = {10, 15, 22, 30, 11, 16, 21, 10}
solución candidata:

30	22	16	10	21	11	15	10

Nótese que la representación de la solución candidata se asemeja a la representación de la cadena cromosomática de un Algoritmo Genético.

Cómputo de la Función Objetivo y de la Función de Aptitud

Una vez definida la forma de representación de una solución candidata, lo que resta por hacer es obtener de alguna manera la cantidad de contenedores que requiere esa permutación, así como también el valor de aptitud asociado a ésta. Para el cálculo del ***valor objetivo*** se procede conforme al algoritmo 4.3.

Algoritmo 4.3. Algoritmo para calcular el valor objetivo de una solución de BPP.

1. **Hacer** *i* = 1 y *j* = 1.
2. **Introducir** el objeto *i* al contenedor *j*.
3. **Para** *i* = 2 hasta *N*, hacer:
3.1. **Si** la suma del llenado actual del contenedor y el objeto actual es menor o igual a la capacidad del contenedor, entonces:
3.1.1. **Introducir** el elemento actual (*i-ésimo* elemento) al contenedor actual (*j-ésimo* contenedor).
3.2. En **caso contrario**, hacer:
3.2.1. Hacer j = j + 1.
3.2.2. **Introducir** el *i-ésimo* elemento al *j-ésimo* contenedor.
4. **Hacer** *Numero_de_contenedores* = *j*.

donde:
N es la cantidad de objetos (cardinalidad del vector pesos).

Para ver un ejemplo de cómo se calcula el valor objetivo bajo esta representación de una solución candidata, véase la sección E.1.

Para calcular la ***función de aptitud*** de una solución candidata de BPP, se utilizó la Ecuación 4.1, la cual fue propuesta por Falkenauer [Quiroz 09] para discriminar entre soluciones que son ***aparentemente iguales***. Dicha ecuación evalúa el promedio de llenado de los contenedores que conforman una solución [Quiroz 09].

$$F_{BPP} = \frac{\sum_{i=1}^{m} \left(\frac{S_i}{c}\right)^2}{m} \tag{4.1}$$

donde:
m es el número de contenedores utilizados en la solución.
S_i es la suma de los tamaños de los objetos en el i-ésimo contenedor (cantidad de llenado).
c es la capacidad del contenedor.

Para ver un ejemplo detallado del procedimiento para realizar el cálculo de la función de aptitud, véase el anexo E, sección E.2.

Técnica para generar soluciones vecinas

Se dice que una ***solución vecina*** es aquella que se genera al realizar ***un solo*** movimiento o intercambio a una solución dada.

Como se mencionó con anterioridad, la representación de la solución se asemeja al denominado cromosoma de los Algoritmos Genéticos (AGs), razón por la cual la forma de generar una solución vecina a partir de una solución dada, es una técnica de mutación de los AGs cuando la representación del cromosoma es una permutación, dicha técnica de mutación es conocida como ***mutación por intercambio recíproco***, la cual describe Coello en [Coello 08].

La forma en cómo funciona la mutación por intercambio recíproco es sencilla, a continuación se especifica en el Algoritmo 4.4.

Algoritmo 4.4. Algoritmo de mutación por intercambio recíproco [Coello 08].

1. **Seleccionar** dos elementos distintos de la cadena cromosomática (representación de la solución) al azar.
2. **Intercambiar** los elementos seleccionados de posición.

Para ver un ejemplo de cómo es que funciona el procedimiento para obtener una solución vecina a partir de una solución dada, véase la sección E.3.

Descripción del Método Propuesto

Una de las aportaciones más importantes en este proyecto de investigación, referente a la visualización, es el ***diseño*** de un método que permite la visualización del espacio de búsqueda en tres dimensiones.

Con el fin de apegarse lo más posible a la visión de Wright en [Wright 32], este trabajo propone el siguiente método para **visualizar** la superficie de aptitudes de cualquier instancia de cualquier problema en **tres dimensiones**.

1. Realizar una caminata aleatoria de *m* pasos sobre la instancia de optimización y obtener las funciones de aptitud de las *m* soluciones.
2. Una vez obtenidos los valores de aptitud (en un conjunto unidimensional), realizar varias particiones del conjunto unidimensional, para obtener un conjunto bidimensional de tamaño $k \times l$. Esto con el fin de que la superficie sea lo más cuadrada posible.
3. Ahora debemos considerar la superficie como una cuadrícula sobre un plano en el espacio tridimensional, en donde el tamaño de la cuadrícula será precisamente de $k \times l$.
4. Para darle altitud a los puntos de la cuadrícula, consideramos al conjunto bidimensional, cada intersección de la cuadrícula será un individuo y su valor de aptitud, previamente calculado en el primer paso, será la altura de ese individuo.
5. Finalmente, lo que resulta es la gráfica tridimensional de la superficie de aptitudes de la instancia de optimización.

La Figura 4.6 muestra el proceso del funcionamiento del método propuesto. Supongamos que queremos visualizar la superficie de aptitudes del siguiente conjunto: *F*={2.2, 2.6, 2.1, 2.2, 2.6, 1.6, 1.3, 0.9, 0.8, 1.1, 1.1, 2.2, 2.2, 1.8, 0.8, 0.7}, lo primero que se debe hacer es obtener el conjunto bidimensional a partir de *F* (inciso a). Posteriormente, cada elemento de la matriz debe imaginarse como una intersección sobre una cuadrícula, donde la altitud de un punto viene dado por el valor del elemento de la matriz (inciso b). Finalmente, lo que resta por hacer es unir cada punto (rombos de color rojo del inciso b) vecino sobre la cuadrícula (inciso c).

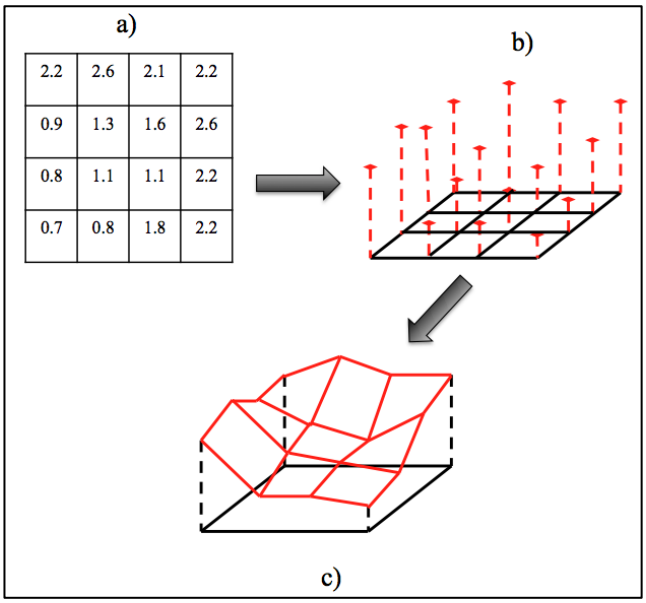


Figura 4.6. Método para visualizar la superficie de aptitudes en tres dimensiones.

Una de las grandes ventajas de este método es que indistintamente del problema de optimización que el investigador aborde, podrá realizar la visualización de su instancia, ya que únicamente como datos de entrada pide un conjunto unidimensional de los valores de aptitud.

Otra característica importante del método propuesto, es que no necesariamente se debe utilizar un algoritmo *Random Walk*, el investigador puede hacer uso de su propio algoritmo y posteriormente utilizar este método para visualizar la porción del espacio de búsqueda que su algoritmo ha logrado visitar.

Cabe mencionar que este método no garantiza que los puntos que se sitúen en una misma columna serán vecinos entre sí, sin embargo sí garantiza que los puntos que se sitúen en una misma fila lo serán.

Criterios utilizados en el método propuesto

Para poder llevar a cabo el método propuesto, se establecieron algunos criterios referentes a la forma de calcular los valores de la partición del conjunto unidimensional y a la forma de distribuir éstos en el nuevo conjunto bidimensional.

Forma de Obtener los Valores de *k* y *l*

Para obtener los valores de los de *k* y *l*, se procede a través de las siguientes reglas:

1. Si el número a descomponer en sus factores tiene raíz cuadrada exacta, entonces el valor de *k* y *l* será la raíz cuadrada del número.

Por ejemplo:

$$m = 25$$
$$\sqrt{m} = 5$$
$$\therefore k = 5, l = 5$$

2. Si el número a descomponer en sus factores es un número primo, el valor de *k* será el del antes mencionado número primo, mientras que el valor de *l* será la unidad.

Por ejemplo:

$$m = 11$$
$$\therefore k = 11, l = 1$$

3. Si al realizar la descomposición factorial, únicamente se obtuvieron dos factores, entonces el valor de k será el primer factor y el valor de l el segundo.

Por ejemplo:

$$m = 14$$

14	2
7	7
1	

$$\therefore k = 2, l = 7$$

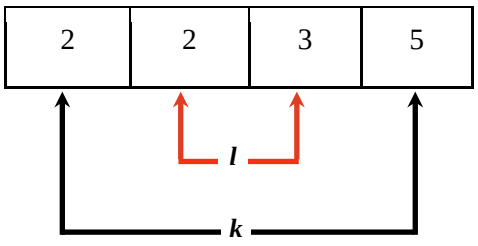
4. Si al realizar la descomposición factorial, el número de factores es par y más de dos, entonces se deberá ir multiplicando los factores en el extremo, es decir, el primero con el último y el producto resultante será parte del valor de k, luego se multiplica el segundo con el penúltimo y el producto será parte del valor de l, y así sucesivamente se van alternando los resultados en *k* y *l*.

Por ejemplo:

$$m = 60$$

60	2
30	2
15	3
5	5
1	

Entonces: el número 60 se descompone en cuatro factores:



Luego,

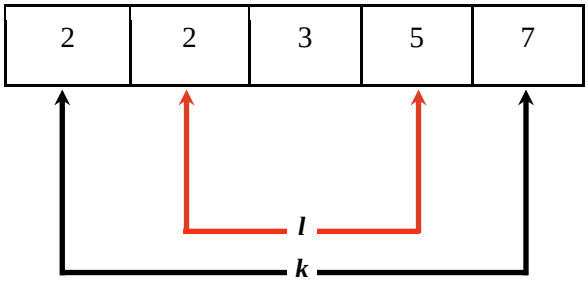
$$k = (2)(5)$$
$$l = (2)(3)$$
$$\therefore k = 10, l = 6$$

5. Si al realizar la descomposición factorial, el número de factores es impar, entonces se procede exactamente como en el paso 3, pero al llegar al factor del centro éste se le incluirá (como producto) al menor de los valores entre *k* y *l*.

Por ejemplo:

$$m = 420$$

420	2
210	2
105	3
35	5
7	7
1	



Luego:

$$k = (2)(7) = 14$$
$$l = (2)(5) = 10$$
$$l < k$$
$$\Longrightarrow l = (l)(3) = 30$$
$$\therefore k = 14, l = 30$$

Forma de Distribuir los Elementos del Conjunto

La forma en la que los valores de aptitud se deben distribuir en el nuevo conjunto bidimensional ***F´*** tiene que respetar el hecho de que los elementos consecutivos del conjunto unidimensional ***F*** son vecinos entre ellos, por lo cual se propone lo siguiente:

1. Se empieza la distribución en la primera fila y primera columna.
2. La primera partición se adiciona a la primera fila en orden de izquierda a derecha.
3. La segunda partición se adiciona a la segunda fila de derecha a izquierda, de tal manera que el último elemento de la primera fila será el vecino del último elemento de la segunda fila.
4. Se va alternando este proceso hasta concluir de acomodar los elementos.

La Figura 4.7 muestra la forma en cómo se deben distribuir los elementos del conjunto *F* para formar el nuevo conjunto *F'*.

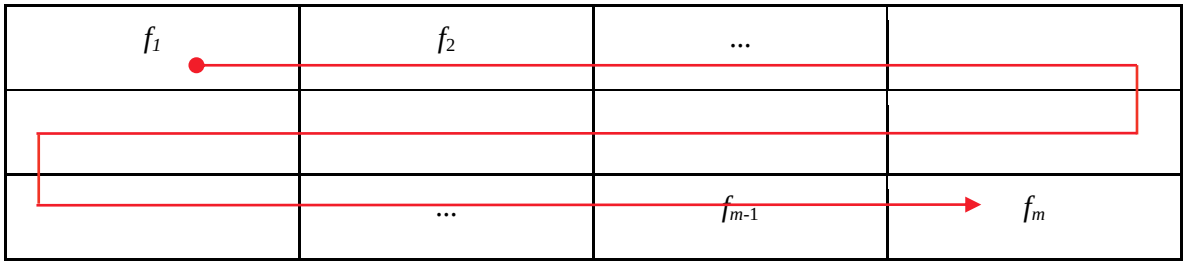


Figura 4.7. Forma de distribuir los elementos de *F* en *F*'.

4.6. Algoritmo de Carreras para Afinación de Parámetros

En esta sección se describe a detalle el algoritmo de carreras implementado para la afinación de parámetros de algoritmos metaheurísticos.

Afinación de Parámetros

Las dos etapas principales en la aplicación de un metaheurístico a un problema específico son el esquema de representación y la determinación de la función objetivo [Rivera 09]. Estos dos elementos forman el puente entre el contexto original del problema y el algoritmo solucionador. Al desarrollar cada uno de estos elementos se seleccionan valores de entre un conjunto para cada uno de sus parámetros, de tal manera que se logre una mejor representación del problema y una mayor eficiencia en el proceso de resolución del mismo [Adenso 06].

Objetivo del Algoritmo de Carreras

Determinar la mejor configuración de valores para los parámetros de cualquier algoritmo metaheurístico.

Idea General

Dados un algoritmo metaheurístico implementado y compilado (.jar, .exe, etc), un conjunto de instancias a resolver, un archivo de configuraciones, un archivo de salidas del algoritmo y una especificación para los posibles valores que puede tomar cada parámetro; obtener el total de posibles combinaciones de valores de los parámetros, actualizar el archivo de configuraciones del algoritmo, e invocar la ejecución de éste, tantas veces como configuraciones totales haya. En la primera iteración del algoritmo F-Race, el número de instancias a resolver por el metaheurístico debe ser preferentemente una fracción del total de instancias.

Al término de cada ejecución del algoritmo F-Race, el resultado de éste es guardado. Una vez que se han evaluado todas y cada una de las posibles configuraciones iniciales, se aplica el método de competencias de Hoeffding (véase la sección 2.10), el cual eliminará las configuraciones estadísticamente menos prometedoras. Si el método de Hoeffding conservó sólo una configuración, quiere decir que dicha configuración es la mejor sobre ese

determinado conjunto de instancias, en caso contrario se debe repetir nuevamente el proceso anterior, pero esta vez únicamente con las configuraciones restantes e incrementando el número de instancias, de ser posible. El número de veces que se debe repetir el método de competencias de Hoeffding se puede fijar en un valor determinado (máximo número de iteraciones) o hasta que sólo quede una configuración (criterio de convergencia). El Algoritmo 4.5 muestra el algoritmo F-Race implementado.

Algoritmo 4.5. Algoritmo F-Race implementado en VisTHAA.

1. **Leer** el porcentaje inicial de instancias.
2. **Leer** el porcentaje de incremento.
3. **Leer** el número de Iteraciones máximas.
4. **Leer** la ruta del Algoritmo ejecutable.
5. **Leer** la ruta del archivo de configuraciones de los parámetros.
6. **Leer** la ruta del archivo de salidas.
7. **Leer** la ruta del archivo de instancias.
8. **Leer** los rangos de parámetros.
9. **Generar** todas las posibles configuraciones de los parámetros.
10. **Leer** el total de las instancias.
11. **Inicializar** los mejores conocidos en -1.
12. **Mientras** (configuraciones restantes > 1 **or** Iteración actual <= Iteraciones máximas)
    12.1. **Actualizar** el tamaño de la muestra (π) de instancias a evaluar.
    12.2. **Actualizar** el número total de configuraciones restantes.
    12.3. **Seleccionar** π instancias aleatoriamente (con la misma probabilidad).
    12.4. **Escribir** la cantidad y los nombres de las instancias seleccionadas en el archivo de instancias.
    12.5. **Para** cada *i*=1,2,…,*actual número de configuraciones*, **hacer**:
        12.5.1. **Escribir** la configuración actual en el archivo de configuraciones.
        12.5.2. **Invocar** la ejecución del Algoritmo con la configuración actual.
        12.5.3. **Esperar** hasta que la ejecución del Algoritmo termine.

12.5.4. **Leer** el archivo de salidas y agregar los resultados a la Tabla de Resultados.
12.6. **Fin** Para
12.7. **Si** los Valores de Aptitud no están normalizados, **entonces**:
12.7.1. **Normalizar** la Tabla de Resultados.
12.8. **Calcular** la media muestral y su error.
12.9. **Calcular** el umbral de aceptación utilizando la media muestral y su error.
12.10. **Eliminar** las configuraciones que quedan fuera del umbral calculado.
13. **Fin** Mientras.

Generar todas las posibles combinaciones

El método inicia generando todas las posibles combinaciones de parámetros, a diferencia de CALIBRA [Adenso 06] que únicamente permite optimizar hasta 5 parámetros, en este proyecto de investigación se le da la libertad al investigador de optimizar ***n*** parámetros, cada uno de ellos con sus propios niveles.

CALIBRA utiliza el diseño experimental factorial fraccional de Taguchi acoplado con un procedimiento de búsqueda local, por lo cual los mejores valores encontrados no están garantizados a ser los óptimos.

En los algoritmos de carreras se exploran todas las posibles combinaciones de valores de los parámetros. A continuación, en el Algoritmo 4.6 se muestra el algoritmo que se ***propone en este trabajo*** para generar todas las posibles combinaciones de parámetros posibles.

Algoritmo 4.6. Algoritmo que genera las combinaciones de valores de parámetros.

1. **Obtener** el número total de combinaciones posibles, c, para ello simplemente se realiza el producto entre las cardinalidades de cada conjunto de valores de los parámetros: $|Parámetro_1| \times |Parámetro_2| \times \cdots \times |Parámetro_n|$.

2. **Obtener** la primera combinación de valores, θ_1, la cual siempre se forma con los primeros valores de los parámetros de cada conjunto, es decir, $\theta_1 = \{Parámetro_{1,1}, Parámetro_{2,1}, \cdots Parámetro_{n,1}\}$.
3. **Declarar** *Posición_actual*[*n*] como arreglo entero de *n* elementos.
4. **Para** *i*=2 hasta *c*, **hacer**:
4.1. **Para** *j*=1 hasta *n* **hacer**:
4.1.1. **Declarar** *k* := 1
4.1.2. **Mientras** *k* < *n* **hacer**:
4.1.2.1. **Si** *Posición_actual*[*k*] < $|Parámetro_j|$ **hacer**:
4.1.2.1.1. *Posición_actual*[*k*] := *Posición_actual*[*k*] + 1.
4.1.2.1.2. $\theta_{i,j} = Parámetro_{j,Posición\_actual[k]}$
4.1.2.1.3. **Para** *l* = *j*+1 hasta *n* hacer:
4.1.2.1.3.1. $\theta_{i,j} = Parámetro_{l,Posición\_actual[l]}$
4.1.2.1.4. **Fin** Para
4.1.2.1.5. *k* := *n*
4.1.2.1.6. *j* := *n*
4.1.2.2. **Fin** Si
4.1.3. **Fin** Mientras
4.1.3.1. **Si no, hacer**:
4.1.3.1.1. *Posición_actual*[*k*] := 0
4.1.3.1.2. $\theta_{i,j} = Parámetro_{j,Posición\_actual[k]}$
4.1.3.1.3. *j* := *j* + 1
4.1.3.2. **Fin** Si no
4.1.3.3. *k* := *k* + 1
4.1.4. **Fin** Mientras
4.2. **Fin** Para
5. **Fin** Para

Capítulo 5

Estrategias Implementadas en VisTHAA

En este capítulo se muestran las implementaciones de las estrategias explicadas en el capítulo 4, y como se debe hacer uso de ellas en la herramienta VisTHAA.

5.1. Interface Principal de VisTHAA

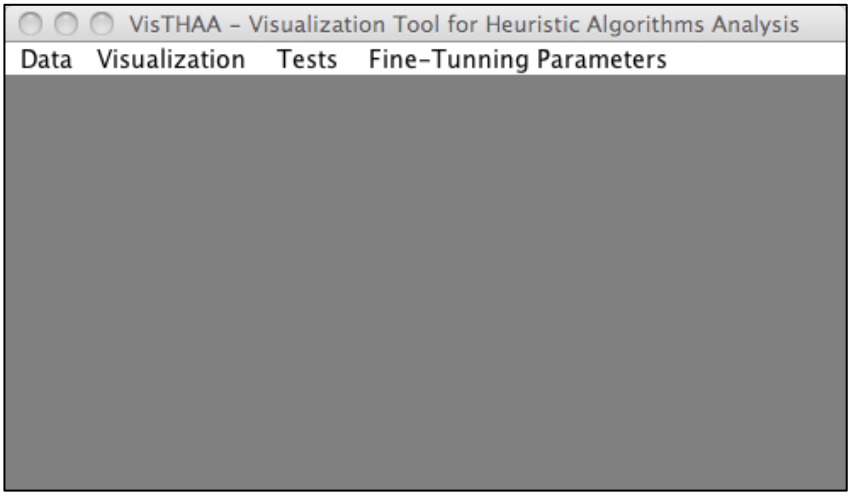


Figura 5.1. Interfaz principal de la herramienta VisTHAA.

La Figura 5.1 muestra la interfaz principal de la herramienta VisTHAA actualmente, sin embargo la herramienta está diseñada en dos capas: la capa de datos y la capa visual, por lo cual, si se desea se puede cambiar el aspecto de VisTHAA sin que esto afecte a las demás funciones en lo más mínimo.

5.2. Lectura de Instancias

Para leer las instancias del investigador, se implementó el método descrito en la sección 4.2.1, el cual permite leer correctamente la instancia del investigador con cualquier formato de ésta, simplemente describiendo dicha instancia. Para acceder al módulo de carga de instancias se debe seguir la ruta mostrada en la Figura 5.2.

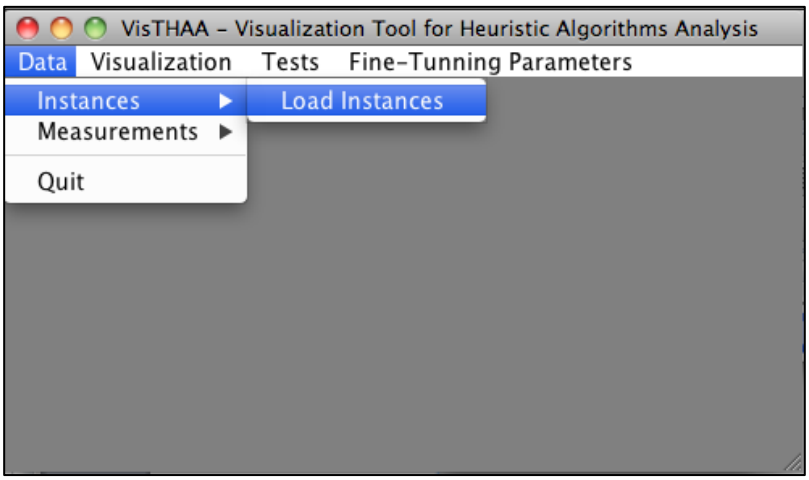


Figura 5.2. Ruta para acceder al módulo que permite cargar las instancias a VisTHAA.

En la ventana principal de VisTHAA, mostrada en la Figura 5.1, se debe entrar al menú *Data*, *Instances* y *Load Instances*, tal y como se muestra en la Figura 5.2; para poder seleccionar el archivo ***logbook***, el cual contiene información de las instancias que serán cargadas a la herramienta, así como el nombre del archivo ***metainstance***, el cual es el que describe a las instancias.

Posteriormente, se debe seleccionar el archivo logbook, la Figura 5.3 muestra la pantalla de VisTHAA que lo realiza.

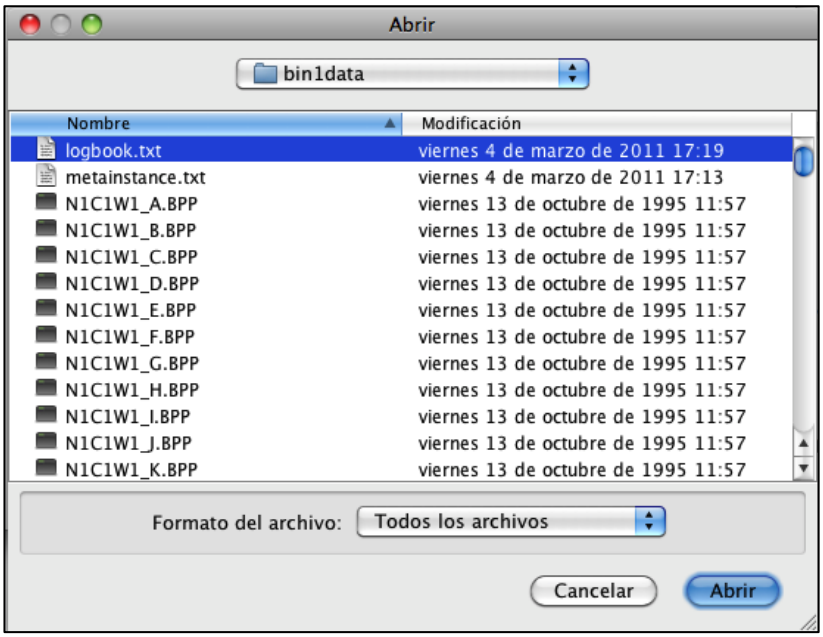


Figura 5.3. Ventana para buscar el archivo ***logbook***.

Una vez localizado el archivo logbook, éste debe de seleccionarse y a continuación hacer clic en el botón “Abrir” para que VisTHAA cargue las instancias correspondientes.

Después de cargar las instancias, VisTHAA manda una ventana que muestra únicamente los nombres de las instancias cargadas, tal como se muestra en la Figura 5.4.

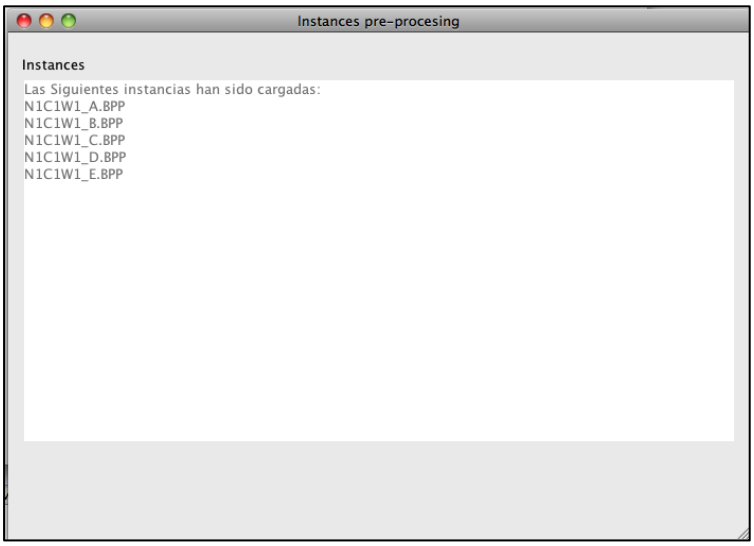


Figura 5.4. Ventana que muestra las instancias que han sido cargadas correctamente.

Finalmente, después de seguir este sencillo proceso, el resultado es que las instancias que el investigador ha seleccionado ahora están dentro de la herramienta, para que el investigador pueda hacer uso de ellas a través de las otras estrategias de mejora.

5.3. Índices Estadísticos, de Caracterización y Propios

En esta sección se muestran las ventanas desarrolladas para VisTHAA referentes a los índices, las cuales se sustentan en las estrategias de mejora descritas en el capítulo 4, específicamente en la sección 4.4.

5.3.1. Introducción de Índices

El módulo de introducción de índices, como su nombre lo sugiere, sirve para que el investigador capture en VisTHAA alguna ecuación que posteriormente utilizará. La ecuación es almacenada en notación polaca inversa. La Figura 5.5 muestra la ventana donde se pueden almacenar los índices propios del investigador.

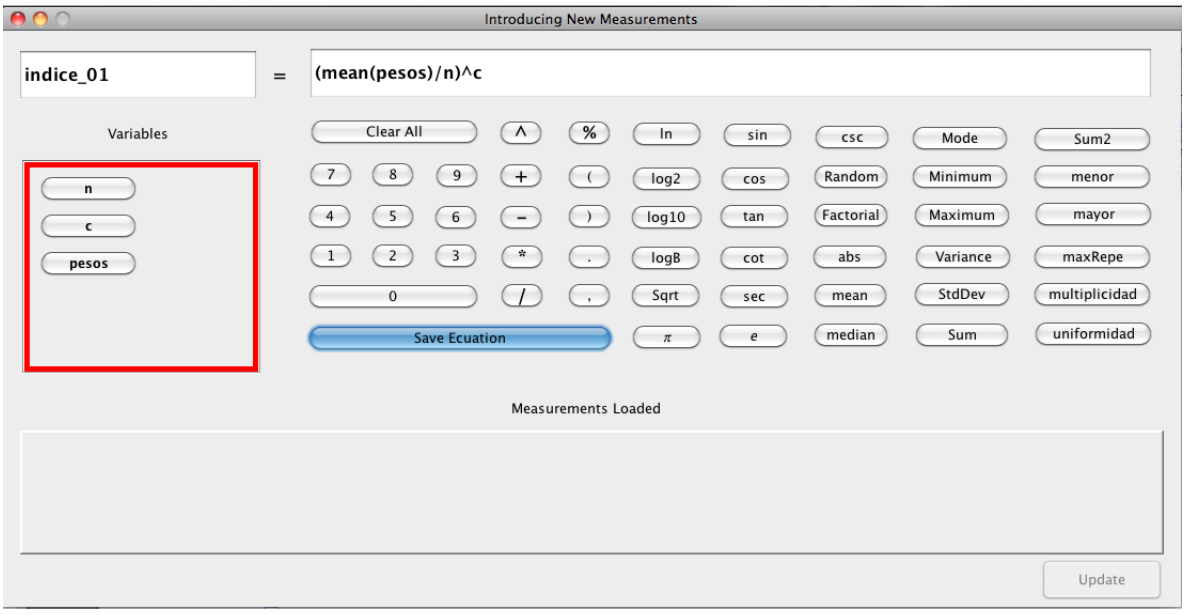


Figura 5.5. Introducción de la Ecuación 5.1, la cual lleva por nombre "indice_01".

La ecuación utilizada para este ejemplo es la Ecuación 5.1, la cual es mostrada a continuación:

$$indice\_01 = \left(\frac{\bar{W}}{n}\right)^{c} \quad \textbf{(5.1)}$$

Lo que se puede notar de la ventana es que en la parte superior de ésta se encuentran dos cajas de texto, de las cuales la que está localizada en la parte izquierda sirve para que el investigador capture el nombre de la variable, mientras que la de la derecha sirve para introducir la ecuación del investigador en notación infija.

Además también podemos notar que hay un cuadro en color rojo, dentro del cuadro vemos unos botones, los cuales se corresponden con las variables de las instancias previamente cargadas y que se han almacenado en la tabla de símbolos, en este caso, las variables son las del problema de BPP.

En la parte central podemos ver que hay botones como los que encontraríamos en cualquier calculadora, botones de los números y de operaciones de aritmética básica, además de los botones “Clear All”, el cual borra el contenido de las cajas de texto y el botón “Save Ecuation”, cuya función es guardar el índice en memoria.

En la parte de la derecha se muestran los botones correspondientes a los estadísticos básicos de uso más frecuente, así como también permite el cálculo de logaritmos en diversas bases, extracción de la raíz cuadrada, la función factorial, una función que genera un número aleatorio entre 0 y 1, todas las funciones trigonométricas, los índices reportados en la literatura [Quiroz 09] y dos de las constantes más importantes: la constante PI y la constante de Euler.

Finalmente, en la parte inferior se muestra un espacio, donde, si hubieran más índices previamente cargados, éstos se mostrarían.

La Figura 5.5 muestra un ejemplo de cómo el investigador puede introducir sus índices a VisTHAA, en este caso el índice consiste en obtener la media del conjunto de pesos de los objetos, posteriormente dividir el resultado por *n* (la cual es una variable de la instancia BPP) y finalmente elevar el resultado a la *c* (otra variable de la instancia). Dicho índice fue capturado bajo el identificador "indice_01".

5.3.2. Calculadora

El módulo denominado calculadora permite al investigador introducir una ecuación y obtener el valor rápidamente, sin que esta ecuación se conserve en VisTHAA. La Figura 5.6 muestra la calculadora.

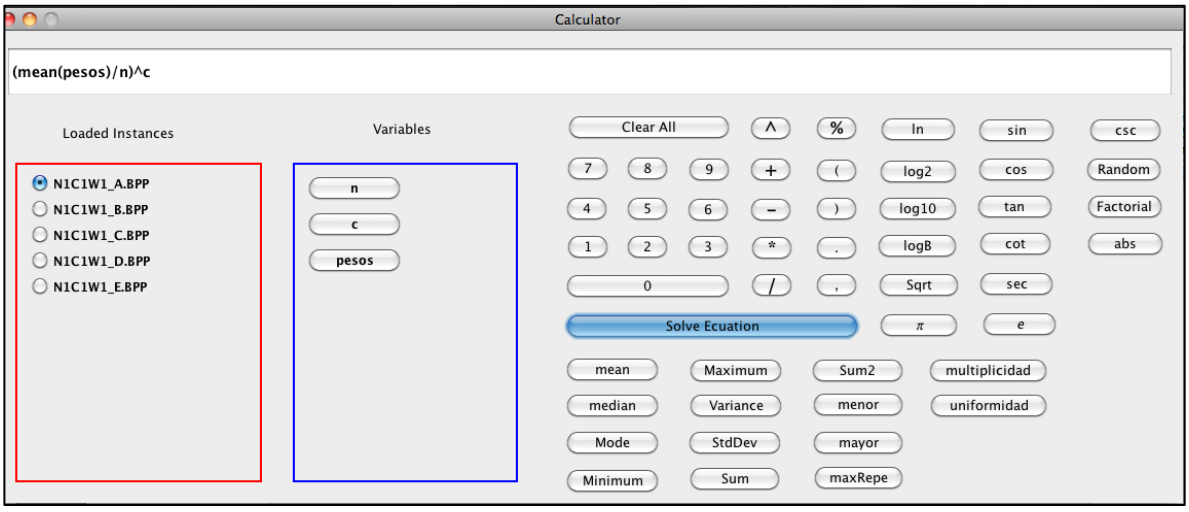


Figura 5.6. Ejemplo de la forma de calcular el valor de un índice propio.

Para obtener el valor resultante de calcular el índice, se debe hacer clic sobre el botón "Solve Ecuation" de la Figura 5.6, el valor resultante es mostrado en la Figura 5.7.

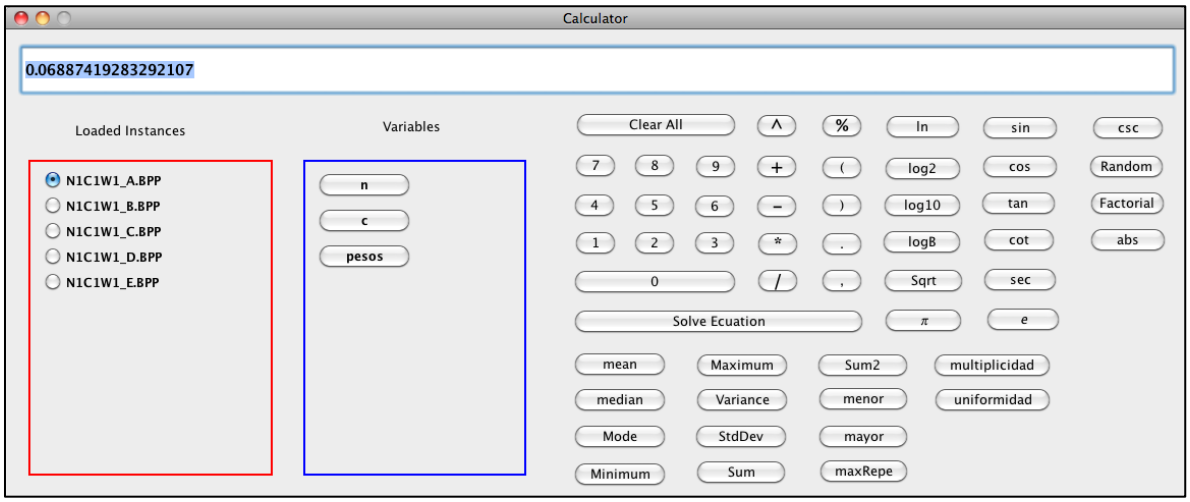


Figura 5.7. Resultado de la evaluación del índice "indice_01".

5.3.3. Matriz de Características

La matriz de características es otra aportación importante de este trabajo, ya que genera una matriz de datos, los cuales posteriormente pueden ser tratados con técnicas de análisis multivariado.

La matriz de características esta conformada en sus filas por las unidades experimentales (instancias) y en sus columnas por las variables / atributos / características (índices que el investigador ingrese a VisTHAA). La Figura 5.8 muestra la ventana correspondiente para realizar los ajustes a la matriz de características.

La ventana para realizar los ajustes tiene en su lado izquierdo, un cuadro rojo, dentro del cual aparecen las instancias previamente cargadas a la herramienta, de las cuales el investigador puede seleccionar aquellas que desee incluir en la matriz.

En la parte central de la ventana, aparece un cuadro en color azul, dentro del cual el investigador puede seleccionar los índices que desee incluir en la matriz.

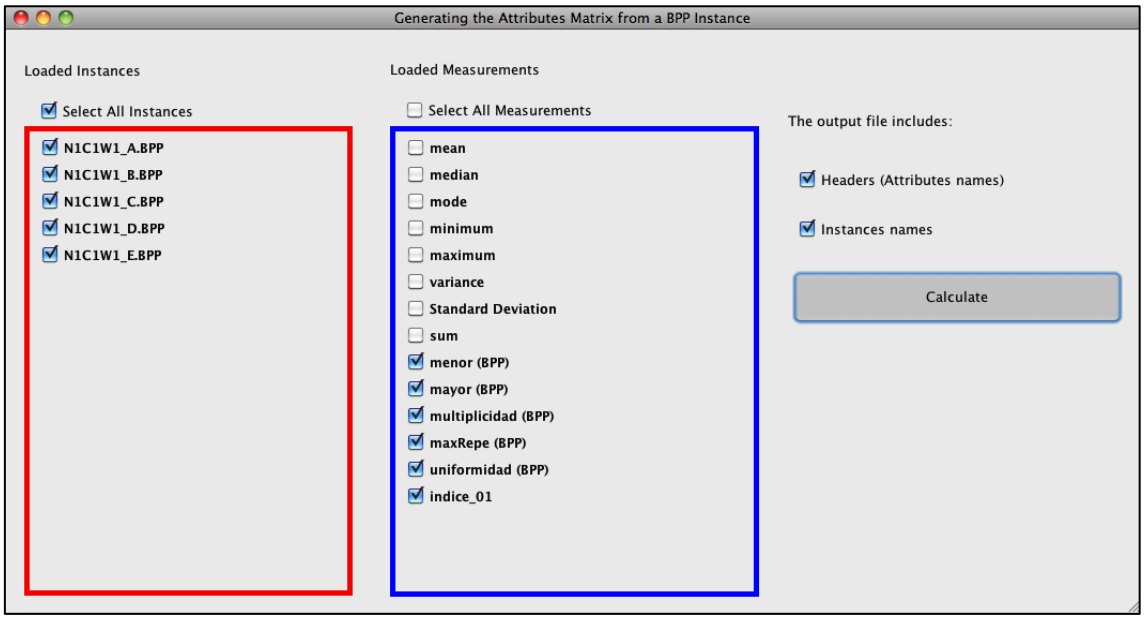
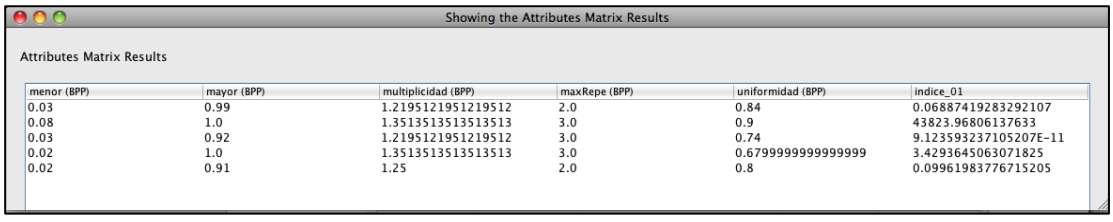


Figura 5.8. Ventana para realizar la configuración de la matriz de características.

Finalmente, en el lado izquierdo de la ventana, aparecen dos opciones, las cuales tienen que ver con el archivo de salida de la matriz de características. Si se selecciona "Headers" el archivo de salida incluirá el nombre de los índices, si se selecciona "Instances names", el archivo de salida incluirá los nombres de las unidades experimentales (instancias). Además se encuentra un botón denominado "Calculate", el cual genera la matriz de características y el archivo de salida en formato de texto plano, así como también muestra los resultados en otra ventana.

Una vez realizados los ajustes, se debe de hacer clic sobre el botón "Calculate", el resultado es mostrado en la Figura 5.9.



menor (BPP)	mayor (BPP)	multiplicidad (BPP)	maxRepe (BPP)	uniformidad (BPP)	indice_01
0.03	0.99	1.2195121951219512	2.0	0.84	0.06887419283292107
0.08	1.0	1.3513513513513513	3.0	0.9	43823.96806137633
0.03	0.92	1.2195121951219512	3.0	0.74	9.1235932371052O7E-11
0.02	1.0	1.3513513513513513	3.0	0.6799999999999999	3.4293645063071825
0.02	0.91	1.25	2.0	0.8	0.09961983776715205

Figura 5.9. Valores resultantes de la matriz de características.

5.4. Visualización

En esta sección se muestran las ventanas desarrolladas para VisTHAA referentes a la visualización, las cuales se sustentan en las estrategias de mejora descritas en el capítulo 4, específicamente en la sección 4.5.

5.4.1. Instancias de BPP

La visualización de la información de las instancias del problema de BPP, se realiza a través de una gráfica de barras bidimensional.

En esta estrategia de mejora, la visualización de la instancia se realiza, como se mencionó anteriormente, por medio de una gráfica de barras, donde el eje de las abscisas representa al i-ésimo objeto, mientras que el eje de las ordenadas representa el peso de los objetos. La Figura 5.10 muestra la ruta para accesar a este módulo.

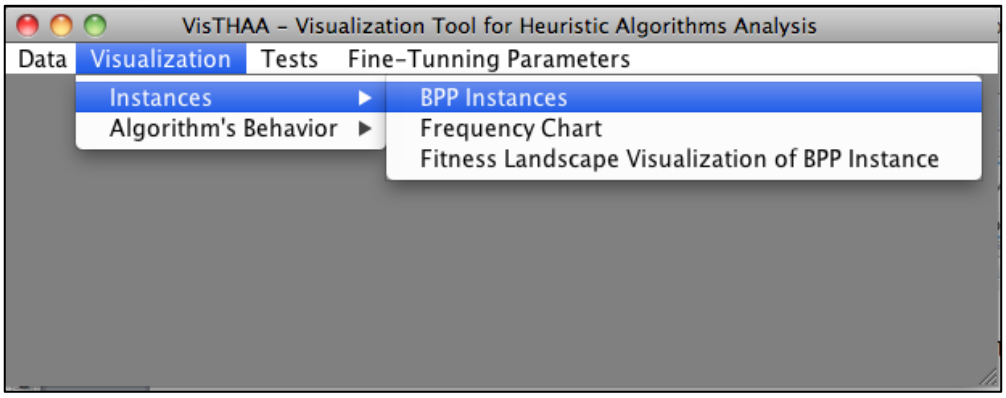


Figura 5.10. Ruta de acceso al módulo para la visualización de una instancia de BPP.

Una vez que se accede al módulo, aparece la ventana mostrada en la Figura 5.11, en la cual se debe seleccionar una de las instancias de BPP, esto con el fin de visualizarla. Posteriormente, la Figura 5.12 muestra la instancia de BPP.

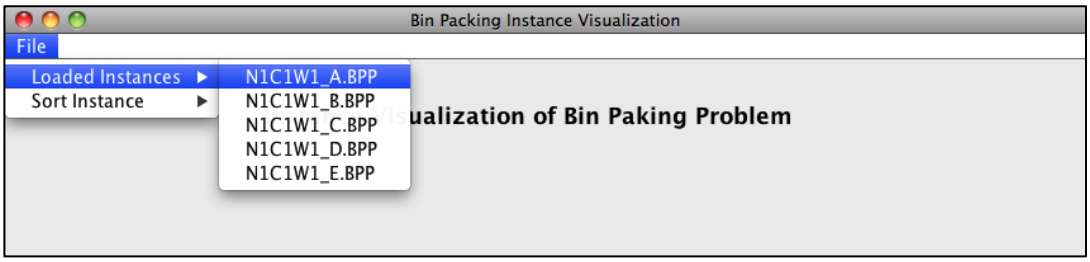


Figura 5.11. Selección de la instancia de BPP que se desea visualizar.

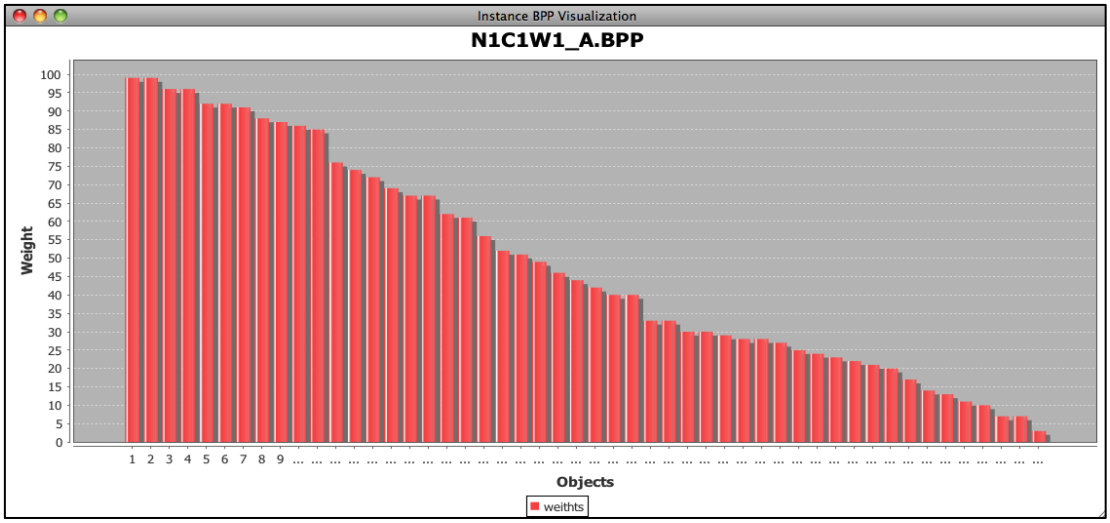


Figura 5.12. Visualización de la instancia de BPP de nombre "N1C1W1_A.BPP".

5.4.2. Instancias de BPP Ordenadas Ascendentemente

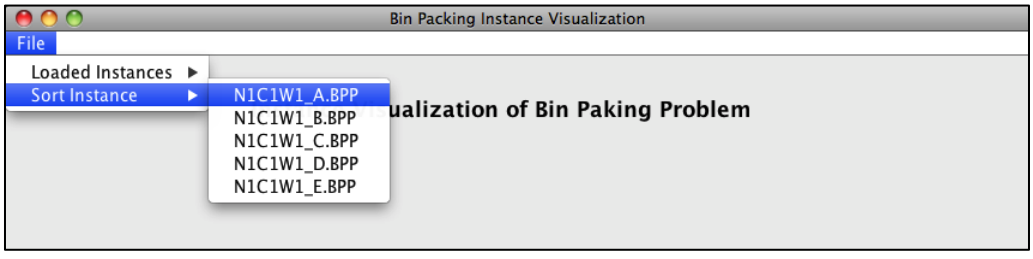


Figura 5.13. Selección de la instancia de BPP que se desea visualizar ascendentemente.

Similar a la visualización de las instancias de BPP, en esta estrategia se hace uso de una gráfica de barras bidimensional, la cual es ordenada de forma ascendente. La Figura 5.10 (mostrada en la sección 5.4.1) indica la ruta para acceder al módulo de visualización

de una instancia de BPP ordenada ascendentemente.

Una vez accedido al módulo de visualización de BPP, lo que continúa es seleccionar la instancia específica a ser visualizada, para lo cual la Figura 5.13 muestra la forma de hacerlo. En este caso se seleccionó la misma instancia de BPP: “N1C1W1_A.BPP”.

Finalmente, después de haber seleccionado la instancia, ésta se muestra en otra ventana. La Figura 5.14 muestra dicha ventana.

Como se puede observar, la Figura 5.14 parece una imagen reflejada en un espejo con respecto a la Figura 5.12, esto se debe a que la instancia seleccionada, N1C1W1_A.BPP, está ordenada por defecto, sin embargo dicha ordenación es de mayor a menor.

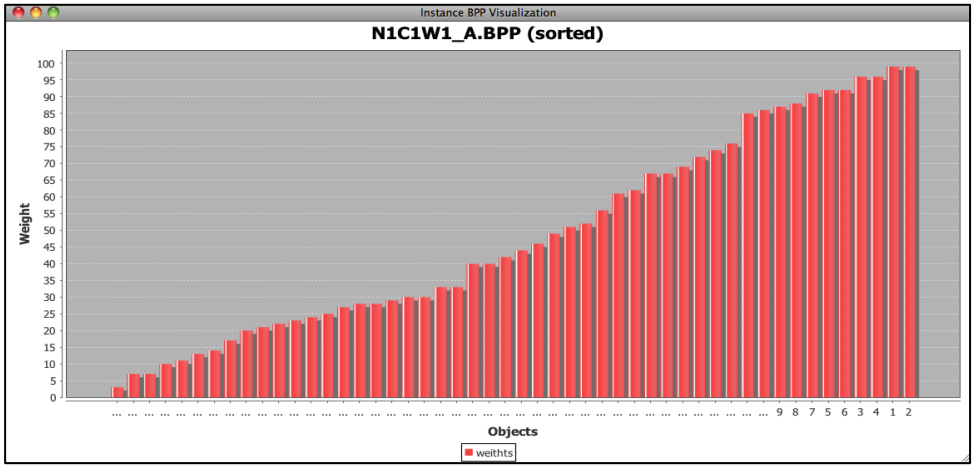


Figura 5.14. Instancia "N1C1W1_A.BPP" ordenada ascendentemente.

5.4.3. Gráfica de Frecuencias

Este tipo de gráficas permiten conocer la cantidad de objetos que se repiten dentro de un determinado porcentaje de la capacidad del contenedor y dentro de que rango se encuentran los objetos. El eje de las abscisas representa el peso de los objetos como porcentaje de la capacidad del contenedor, es decir, $0 < w_i/c \leq 1$. El eje vertical cuantifica el número de objetos que se localizan dentro de un intervalo de porcentaje [Quiroz 09]. El porcentaje de

incremento es del 1%, lo que significa que se contabilizarán cuantos objetos están entre el 0% y el 1%, cuantos objetos están entre el 1% y el 2% y así sucesivamente hasta llegar al 100%. La Figura 5.15 muestra la ruta de acceso al módulo.

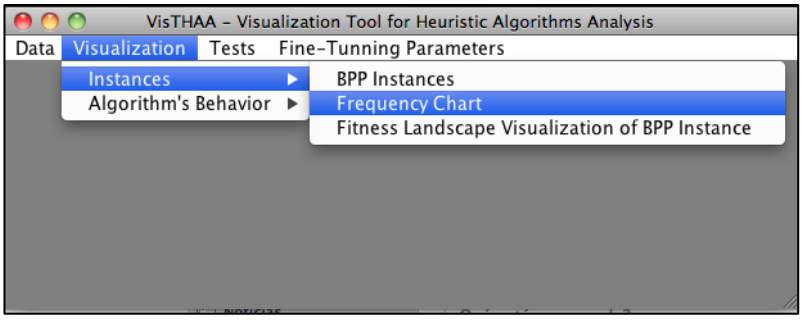


Figura 5.15. Ruta de acceso al módulo para visualizar las gráficas de frecuencias.

Una vez accedido al módulo lo que continúa es seleccionar la instancia específica a ser visualizada, para lo cual la Figura 5.16 muestra la forma de hacerlo. En este caso se seleccionó la misma instancia de BPP: "N1C1W1_A.BPP". Posterior a la selección de la instancia, se muestra la gráfica de frecuencias referente a ella, la Figura 5.17 la muestra.

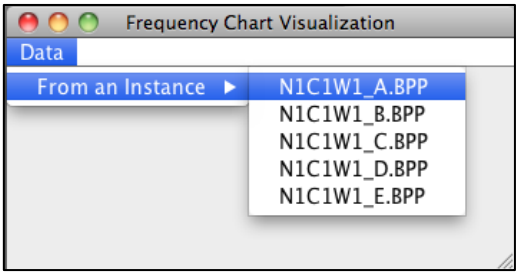


Figura 5.16. Selección de la instancia N1C1W1_A.BPP.

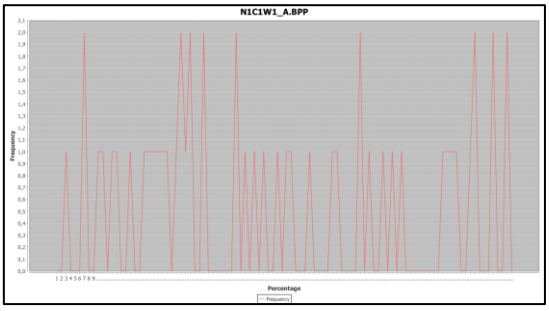


Figura 5.17. Gráfica de frecuencias para la instancia N1C1W1_A.BPP.

5.4.4. Superficie de Aptitudes en Dos Dimensiones

La visualización de la superficie de aptitudes en dos dimensiones de una instancia de BPP, se implementó conforme a lo descrito en la sección 4.5. La Figura 5.18 muestra la superficie de aptitudes en dos dimensiones.

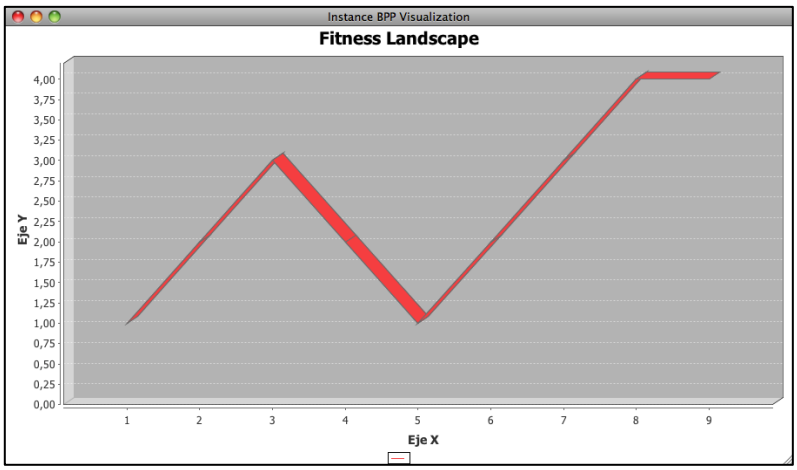


Figura 5.18. Visualización de la superficie de aptitudes en dos dimensiones.

5.4.5. Superficie de Aptitudes en Tres Dimensiones

La visualización de la superficie de aptitudes en tres dimensiones de una instancia de BPP, se implementó conforme a lo descrito en la sección 4.5. La ruta de acceso al módulo se muestra en la Figura 5.19.

Una vez dentro del módulo que permite la visualización de la superficie (véase Figura 5.20), podemos ver que en la parte izquierda de la ventana aparecen todas las instancias que han sido cargadas a la herramienta; en la parte central algunos campos que deben ser configurados para ejecutar el algoritmo de caminata aleatoria y el botón “Execute Random Walk”; finalmente en la parte inferior aparecen las barras de progreso.

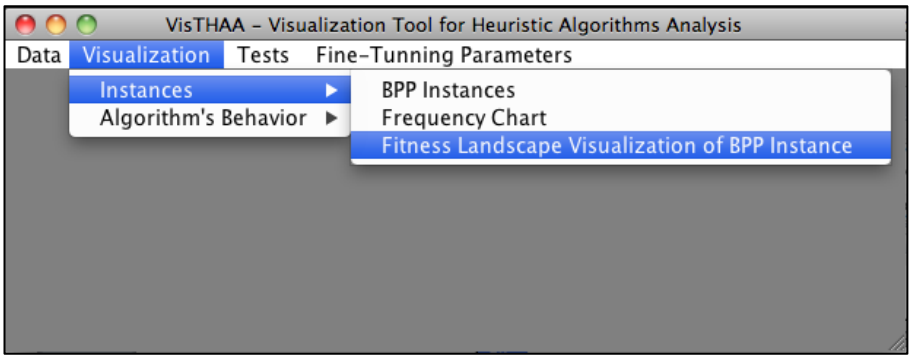


Figura 5.19. Acceso al módulo de visualización de la superficie de aptitudes.

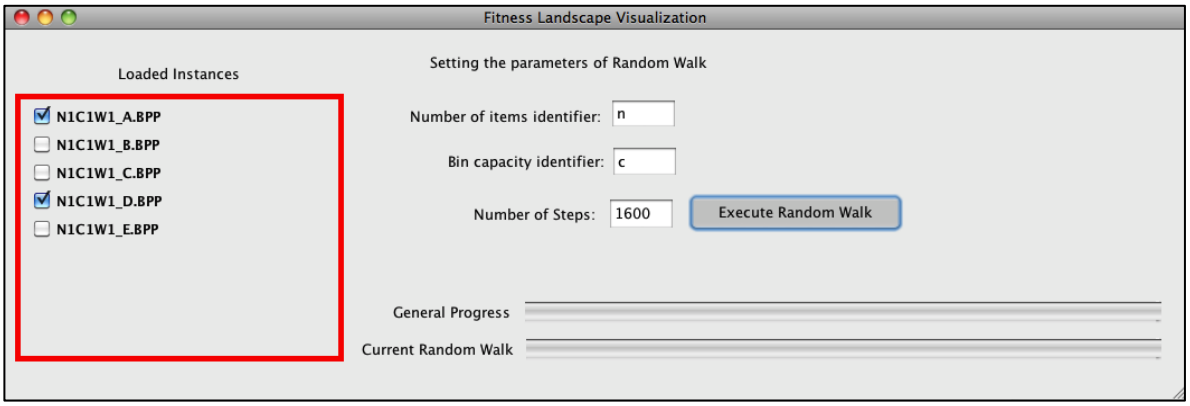


Figura 5.20. Ventana principal para visualizar la superficie de aptitudes.

Se configuran los parámetros necesarios: *Selección de instancias a visualizar*, *identificador del número de objetos*, *identificador de la capacidad del contenedor* y *longitud de la caminata aleatoria,* en este caso el número de pasos de la caminata se fijó en 1600, además se deben seleccionar las instancias que se desean visualizar.

Posteriormente, se debe dar clic en el botón *Execute Random Walk*. Si se ha seleccionado más de una instancia, entonces se ejecutará una caminata aleatoria por cada una de ellas. La barra de progreso de arriba muestra el progreso general, mientras que la de abajo muestra el progreso de la caminata aleatoria en la instancia actual.

Finalmente, cuando la caminata aleatoria llega a su fin, muestra en una pantalla la superficie de aptitudes.

Si se seleccionaron más de una instancia a visualizar, entonces VisTHAA mostrará una ventana por cada una de ellas, en donde el título de la ventana será el nombre de la instancia.

Para este ejemplo, se seleccionaron dos instancias: *N1C1W1_A.BPP* y *N1C1W1_D.BPP*, las Figuras 5.21 y 5.22 muestran las superficies de aptitudes de cada una de ellas, respectivamente.

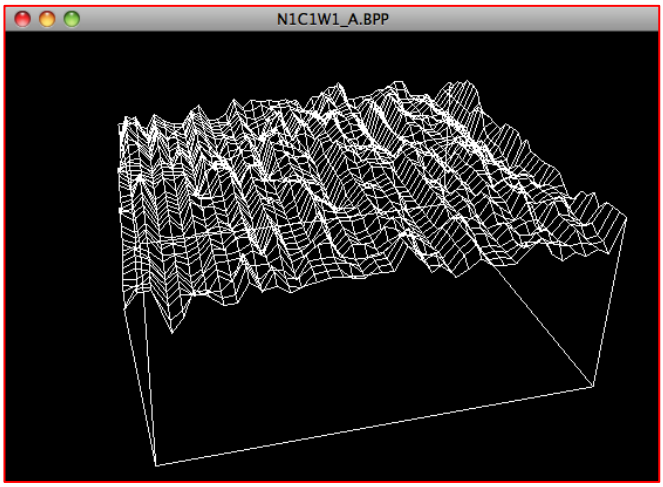


Figura 5.21. Superficie de aptitudes para la instancia N1C1W1_A.BPP.

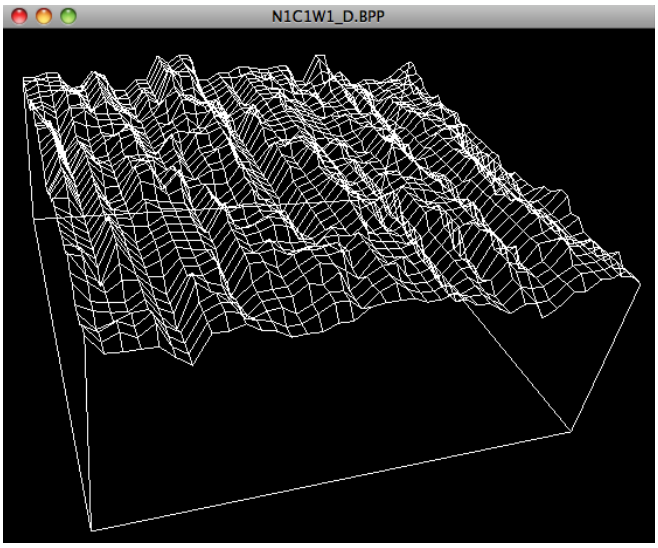


Figura 5.22. Superficie de aptitudes para la instancia N1C1W1_D.BPP

Cabe mencionar que VisTHAA brinda la posibilidad de interactuar con el usuario, es decir, que las superficies de aptitudes pueden responder a acciones del investigador, tales como: rotar la superficie sobre cualquiera de los ejes, acercar o alejar la superficie (hacer zoom) o mover de posición la superficie.

5.5. Prueba No Paramétrica de Wilcoxon

La prueba de Wilcoxon se utiliza para determinar si dos muestras representan dos diferentes poblaciones [García 08].

Específicamente en el ámbito de las ciencias computacionales, la prueba de Wilcoxon se utiliza para determinar si las diferencias en el desempeño de dos algoritmos es estadísticamente significativa, ya que de ser así, se puede afirmar fehacientemente, con un nivel de confianza determinado, que un algoritmo es superior a otro.

Debido a lo anterior, se implementó un módulo en VisTHAA que permitiera determinar si un algoritmo es superior a otro estadísticamente, a través de la aplicación de la prueba de Wilcoxon. La Figura 5.23 muestra la ruta de acceso a dicho módulo.

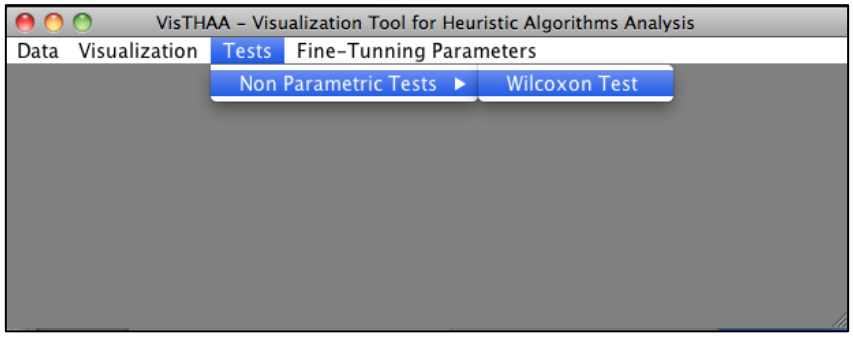


Figura 5.23. Ruta de acceso al módulo de la prueba de Wilcoxon.

Una vez accedido al módulo, se muestra una ventana cuyo principal objetivo es cargar a VisTHAA los datos del desempeño de los algoritmos, para esto accedemos al menú *Load*, y luego al menú ...*from a File*, tal y como se muestra en la Figura 5.24.

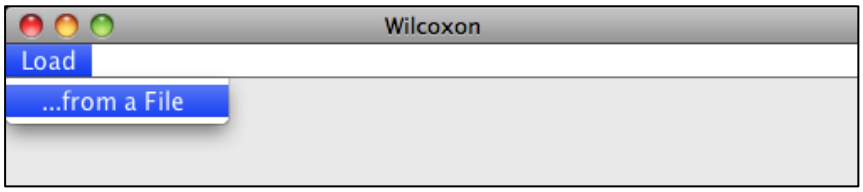


Figura 5.24. Ruta para cargar los datos de la prueba de Wilcoxon a VisTHAA.

En la Figura 5.25 se muestra un navegador de archivos, dentro del cual, al igual que las instancias, debemos buscar el archivo correspondiente, en este caso, el archivo donde se encuentren los datos de desempeño de los algoritmos a ser evaluados.

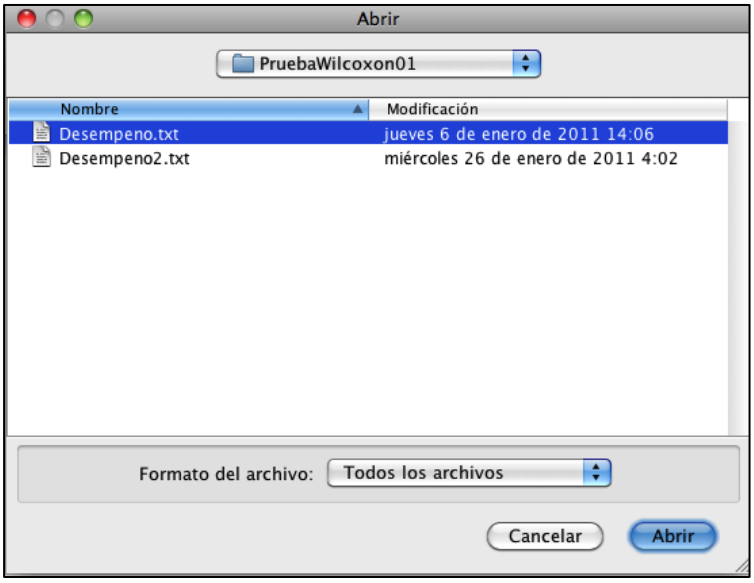


Figura 5.25. Ventana de búsqueda del archivo con los datos de desempeño.

Cabe mencionar que los datos de desempeño se toman de un archivo de texto plano el cual únicamente debe tener dos líneas, donde la primera línea debe contener un conjunto de datos del desempeño del primer algoritmo, mientras que en la segunda línea debe contener los datos del desempeño del segundo algoritmo.

En ambos conjuntos de datos, los elementos deben estar separados por un espacio en blanco o por un tabulador, además ambos conjuntos deben tener exactamente la misma cardinalidad. Además, la cardinalidad mínima de los conjuntos debe ser de 5, mientras que la cardinalidad máxima debe de ser de 40.

Una vez hecho esto, una nueva ventana es mostrada, ya con los datos del archivo seleccionado, ahora únicamente hace falta hacer clic en el botón *Execute Wilcoxon Test* para que VisTHAA automáticamente realice dicha prueba. La Figura 5.26 muestra la ventana principal de la prueba de Wilcoxon.

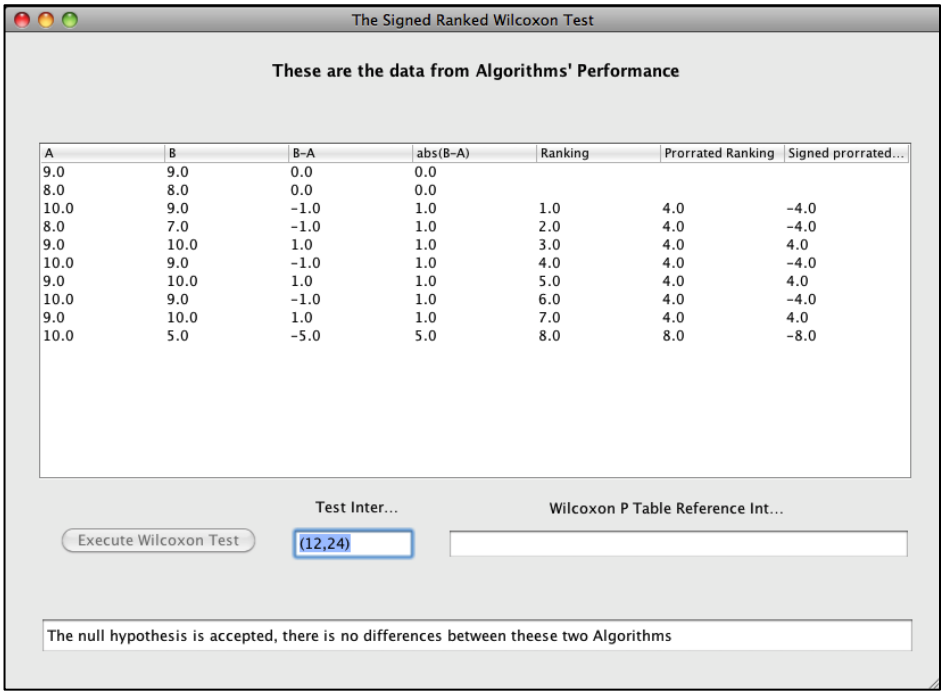


Figura 5.26. Ventana principal para realizar la prueba de Wilcoxon.

Específicamente en este ejemplo, las diferencias entre los desempeños de ambos algoritmos no es estadísticamente significativa, tal como se muestra en la caja de texto de la parte inferior de la ventana.

5.6. Afinación de Parámetros con Algoritmos de Carreras

Los algoritmos de carreras se utilizan para el correcto ajuste de los valores de parámetros de los algoritmos metaheurísticos, es por ello que en VisTHAA se desarrolló un módulo que permite realizar dicho ajuste de parámetros. La Figura 5.27 muestra la ruta de acceso. Posteriormente se muestra la ventana principal para realizar la configuración de parámetros, la Figura 5.28 muestra dicha ventana.

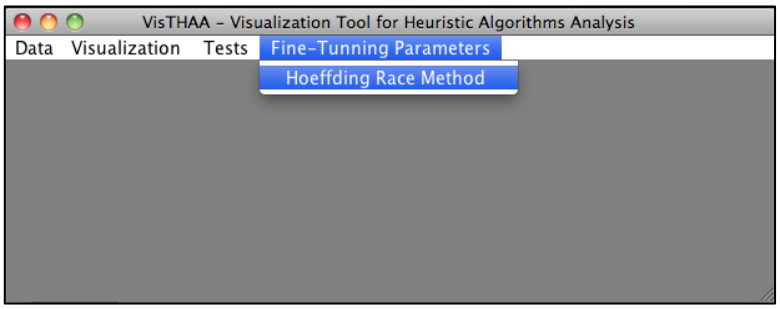


Figura 5.27. Ruta de acceso al módulo para el ajuste de parámetros.

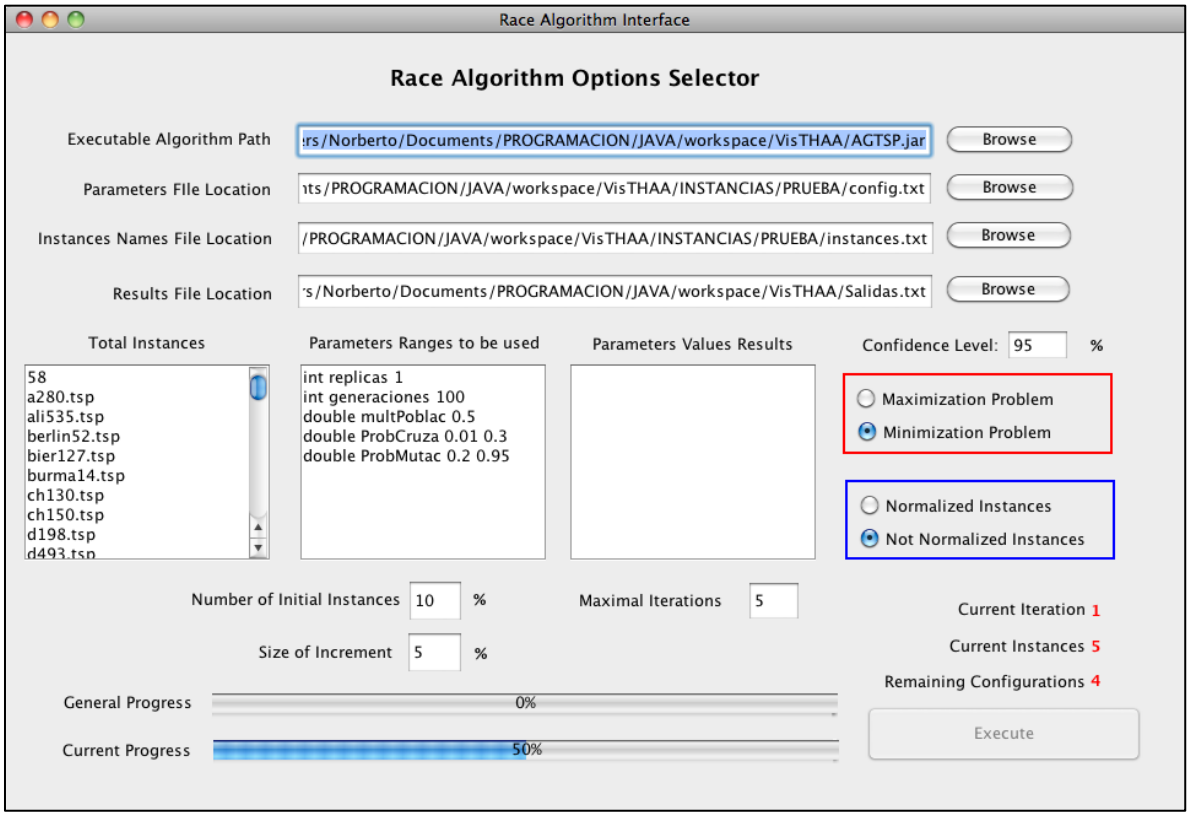


Figura 5.28. Ventana principal del módulo de ajuste de parámetros en VisTHAA.

Como se puede observar en la Figura 5.28, en la parte superior se debe especificar lo siguiente: la ruta donde se ubica el algoritmo metaheurístico ejecutable del investigador, la ruta donde el algoritmo ejecutable leerá los valores de los parámetros, la ruta donde el algoritmo ejecutable leerá las instancias que va a resolver, y la ruta donde el algoritmo ejecutable escribirá los resultados. Además, se debe especificar el total de instancias, las

características de los parámetros (tipo de dato, nombre y valores de rango), el nivel de confiabilidad de la prueba de Hoeffding, si el problema es de maximización o de minimización, si los resultados están normalizados o no, el número inicial de observaciones (instancias) en puntos porcentuales, el tamaño de incremento para las observaciones, y el número máximo permitido de iteraciones.

Finalmente, cuando el algoritmo de carreras ha encontrado la mejor configuración de parámetros, éste conjunto es mostrado en la sección del módulo que dice "*Parameters Values Results*". La Figura 5.29 muestra la ventana con el mejor conjunto de valores para los parámetros posibles, con respecto al conjunto de instancia a resolver.

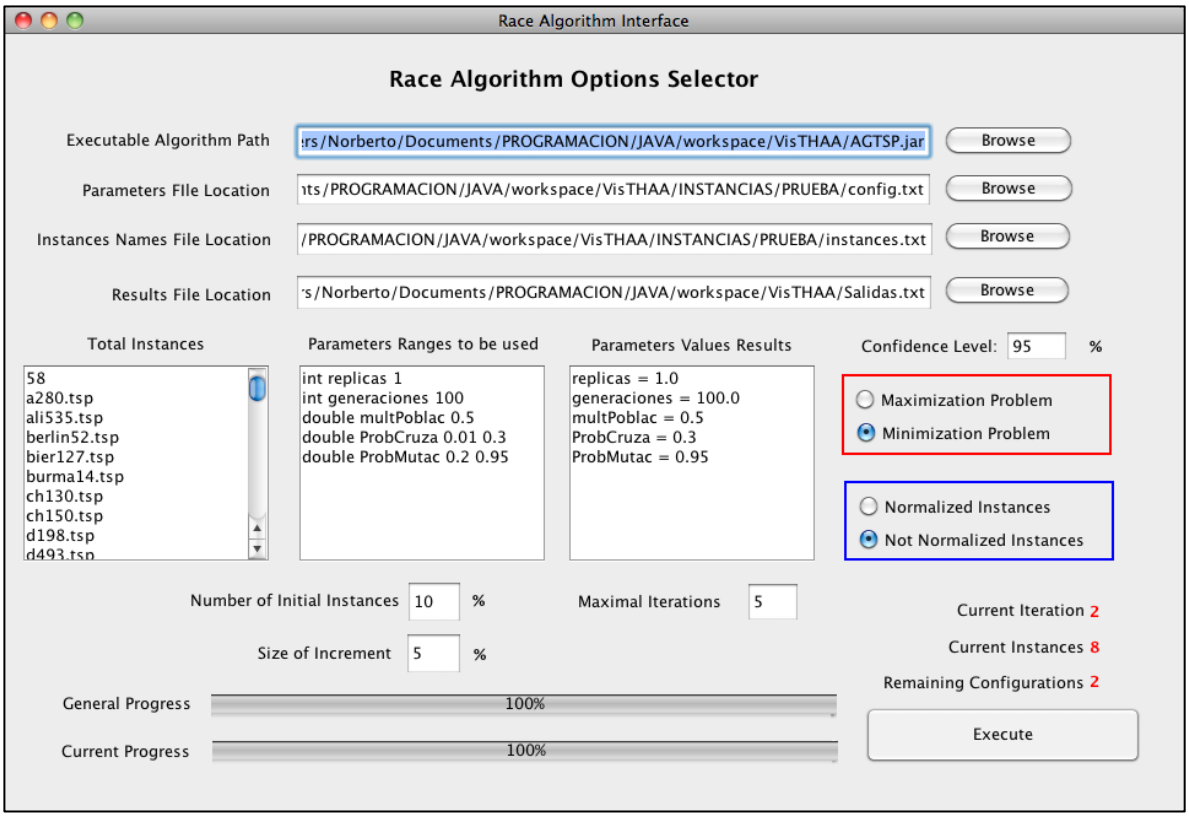


Figura 5.29. Resultados obtenidos por el algoritmo de carreras.

Capítulo 6

Experimentación y Análisis de Resultados

En este capítulo se detallan las experimentaciones realizadas sobre la primera versión de la herramienta VisTHAA. Así como los resultados obtenidos y cómo éstos ayudaron a mejorarla.

6.1. Ambiente Experimental

Hardware

Todos los experimentos fueron realizados bajo las siguientes especificaciones de hardware: Procesador Intel ATOM a 1.67 GHz y 1 Gb en memoria RAM.

Software

El algoritmo heurístico WABP fue implementado en lenguaje C, el sistema operativo donde fue desarrollada la experimentación es Microsoft Windows XP home edition. Además, se utilizaron diferentes clases de instancias, las cuales pueden ser encontradas en sitios de internet bien conocidos. La configuración de parámetros de control utilizada es la recomendada por los autores en [Loh 06]: parámetro de escala, con un valor de 0.05; número de iteraciones máximas con un valor de 50; temperatura inicial con un valor de 1; y reducción de la temperatura con un valor de 0.05.

Descripción de las instancias utilizadas

Uniform [Beasley 90]: Conjunto de 80 instancias identificadas con la letra u debido a que su principal característica es que los pesos de los objetos están uniformemente distribuidos entre 20 y 100. La capacidad del contenedores de 150 y existen cuatro clases de casos cada uno con *n* = 120, 250, 500 y 1000 objetos. Cada clase posee 20 instancias identificadas respectivamente por u_120, u_250, u_500 y u_1000.

Triplets [Beasley90]: Conjunto de 80 instancias difíciles, identificadas con la letra t. Su nombre se debe a que las instancias fueron construidas con una solución óptima conocida de *n*/3 contenedores, de tal forma que cada contenedor de la solución óptima debe almacenar exactamente tres objetos que lo llenan completamente. El tamaño de las instancias es de *n* = 60, 120, 249 y 501, definiéndose así cuatro clases, el tamaño del contenedor *c* es de 100, mientras que los pesos están distribuidos entre 25 y 50. Cada clase posee 20 instancias identificadas respectivamente por t_60, t_120, t_249 y t_501.

Data Set 1 [Klein09]: Construidas de forma similar que algunas instancias propuestas por Martello y Toth [Martello 90a] que resultaron difíciles. Son un conjunto de 720 instancias denotadas con n-c-w por los datos que maneja: *n* = 50, 100, 200 y 500, capacidad del contenedor *c* = 100, 120, 150 y pesos *w* generados uniformemente en intervalos de [1,100], [20,100] y [30,100]. La combinación de los diferentes parámetros resulta en 36 clases, cada clase contiene 20 instancias.

Data Set 2 [Klein09]: Conjunto formado por 480 instancias con pesos generados con una distribución uniforme, denotadas por n-w-b-r. Cada sigla representa la configuración de un parámetro de entrada: número de objetos *n* con valores de 50, 100, 200 y 500, la capacidad de los contenedores *c* es 1000. Con el objetivo de generar instancias cuyo número medio de objetos por contenedor variara entre tres y nueve, se consideraron otros dos parámetros: $\overline{w}$ que representa el peso medio deseado $\overline{w}$ = c/3, c/5, c/7, c/9, y una desviación δ = 20%, 50% y 90% que determina la variación máxima de un peso dado *w* en relación a $\overline{w}$. Por ejemplo cuando $\overline{w} = c/5$ y $\delta = 50\%$ los pesos de los objetos fueron generados de manera aleatoria

con una distribución uniforme en el intervalo discreto [100, 300]. Combinando las características anteriores se cuenta con 48 clases, cada una con 10 instancias.

Data Set 3 [Klein09]: Conjunto formado por una única clase con diez instancias consideradas difíciles, mejor conocidas como *hard*. Cada instancia posee n = 200 objetos, capacidad de contenedor c = 100000 y pesos distribuidos uniformemente entre 20000 y 35000. Los pesos de los objetos se encuentran ampliamente dispersos y el número de elementos por contenedor está entre tres y cinco.

6.2. Experimentación

6.2.1. Experimento 1: Análisis del Algoritmo WABP

Objetivo

Identificar áreas de oportunidad para mejorar el desempeño del algoritmo WABP que resuelve el Bin-Packing.

Procedimiento

El procedimiento utilizado en este estudio consiste en tres fases principales: entrada de datos; caracterización de las instancias; y visualización de instancias y desempeño algorítmico. Después de ellas, está una fase de rediseño del algoritmo, siempre y cuando se observen áreas de mejora. Finalmente, la Prueba de Rangos con Signos de Wilcoxon es utilizada para validar si los resultados obtenidos del rediseño son estadísticamente mejores. La Figura 6.1 muestra el diagrama de la metodología utilizada en este estudio.

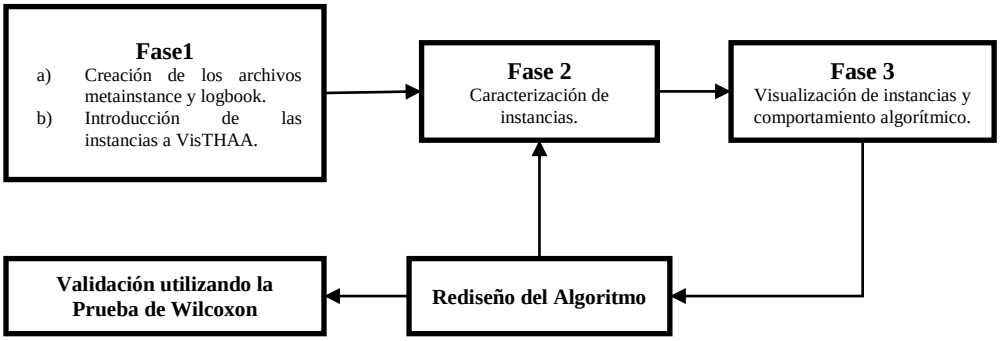


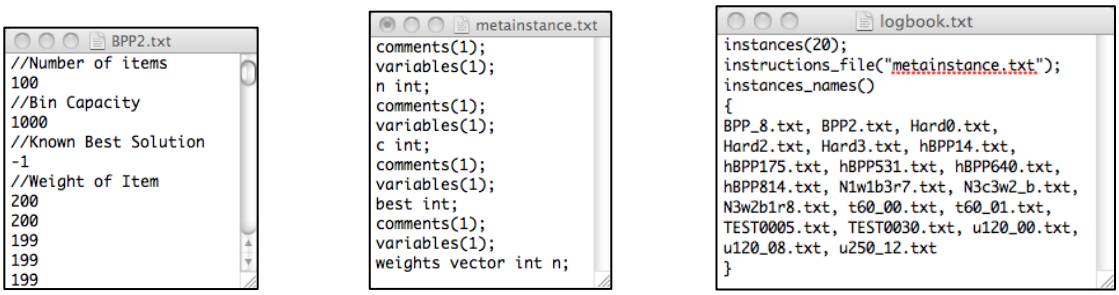
Figura 6.1. Diagrama de la metodología utilizada.

Uno de los problemas de optimización más estudiados es el Bin-Packing (BPP), es clasificado como un problema de optimización combinatoria NP-Duro. Consiste en encontrar una distribución de objetos tal que se minimice el número de contenedores utilizados.

El algoritmo heurístico Recocido de pesos para Bin-Packing (WABP, por sus siglas en inglés), propuesto por Loh, et al., es un algoritmo que empieza con una solución generada por una heurística determinista para BPP, la cual es mejorada siguiendo la estrategia WABP. El algoritmo se ejecuta durante un cierto número de iteraciones donde el llenado de los contenedores y el peso de los objetos son distorsionados, de tal manera que permiten movimientos que son capaces de escapar de soluciones subóptimas, sin permitir soluciones infactibles. El tamaño de la distorsión es controlada a través del parámetro de la temperatura el cual se decrementa hasta que la solución distorsionada coincide con la solución real.

Fase 1: Entrada de datos

El formato de las instancias es descrito en un archivo de texto plano (sin formato), esta descripción es conocida como metainstance. Posteriormente, el número de instancias a ser introducidas es especificado, el nombre del archivo que describe las instancias y el nombre de cada una de ellas, esta especificación recibe el nombre de logbook. La Figura 6.2 muestra una instancia de BPP, el archivo metainstance y el archivo logbook.



a) Instancia de BPP **b)** archivo metainstance **c)** archivo logbook

Figura 6.2. Ilustración de la entrada de datos.

Fase 2: Caracterización de las instancias

Una vez realizada la carga de las instancias, el investigador puede realizar la caracterización las instancias de BPP. La caracterización puede hacerse de dos formas distintas, de manera estadística o de manera visual. Si es de manera estadística, ésta se realiza a través de índices, hasta este momento, la selección entre estadísticos básicos o índices especializados para BPP reportados en la literatura, es posible.

VisTHAA tiene implementados algunos de los índices especializados para BPP, los cuales pueden ser utilizados para obtener conocimiento de las instancias. Sin embargo, VisTHAA ofrece a los investigadores la oportunidad de introducir sus propios índices a partir de las variables definidas en el archivo metainstance.

A manera de ejemplo, el índice que mide la capacidad del contenedor ocupada por un objeto promedio es introducido, el cual a su vez será utilizado en este estudio. La Ecuación 6.1 define el índice y la Figura 6.3 muestra cómo debe de ser introducido.

$$t = \frac{\sum_{i=1}^{n} w_i / n}{c} \qquad \textbf{(6.1)}$$

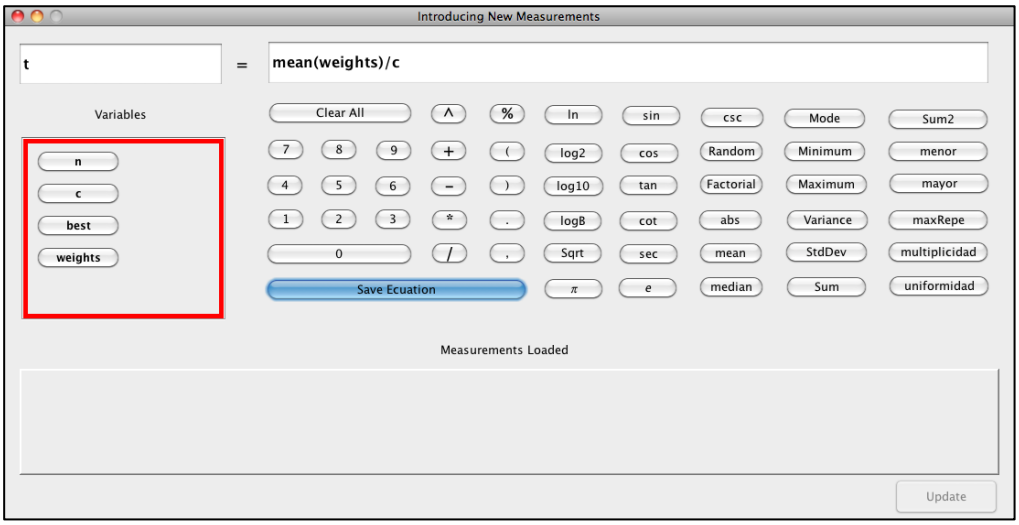


Figura 6.3. Módulo que permite introducir nuevos índices en VisTHAA.

Una vez que un conjunto de índices es introducido, el investigador puede decidir cuáles de ellos considerar para generar la matriz de características sobre el conjunto de instancias seleccionadas, el propósito es obtener información para el análisis de los valores calculados sobre dicho conjunto de instancias. La Figura 6.4 muestra el módulo que permite generar la matriz de características.

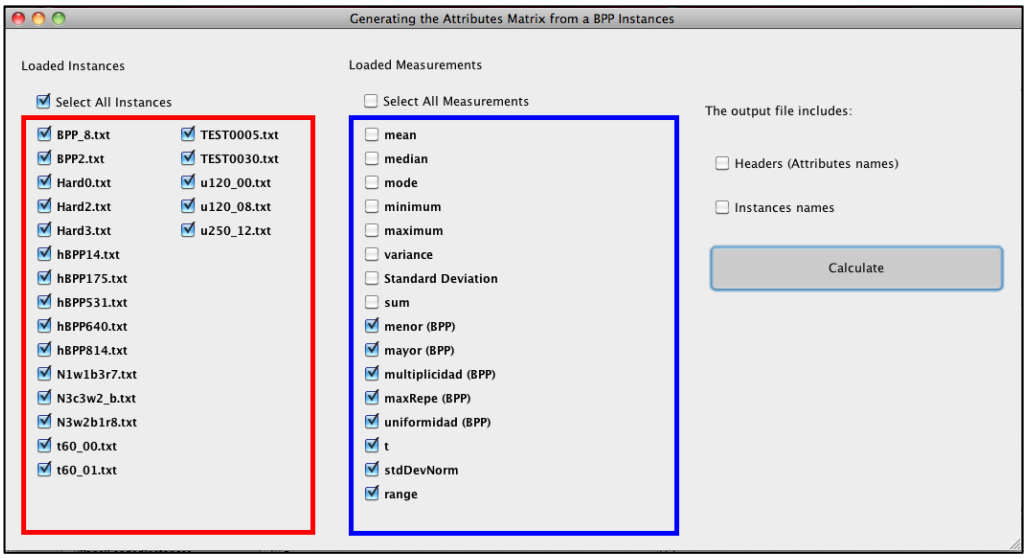


Figura 6.4. Módulo que permite generar la matriz de características.

Como se puede observar en la Figura 6.4, ocho índices son seleccionados, cinco de

los cuales son tomados en la literatura y los tres restantes son índices que VisTHAA no tiene por defecto y han sido introducidos. Los valores de los índices ayudan a diferenciar diversas instancias en un conjunto. Sin embargo, dentro de sus respectivos conjuntos estas diferencias no se aprecian claramente. No obstante, los valores muestran información general acerca de la estructura de la instancia. Por ejemplo, para los índices multiplicidad, t y rango, un patrón que identifica la dificultad de algunas instancias es observado. La Tabla 6.1 muestra la matriz de características sobre todo el conjunto de instancias utilizadas en este estudio.

El simbolismo utilizado en la Tabla 6.1 para los índices calculados son: 1) menor, 2) mayor, 3) multiplicidad, 4) MaxRepe, 5) uniformidad, 6) media estandarizada (t), 7) desviación estándar normalizada, 8) rango estandarizado.

Tabla 6.1. Matriz de características sobre las instancias a ser optimizadas.

Instancia	*ind1*	*ind2*	*ind3*	*ind4*	*ind5*	*ind6*	*ind7*	*ind8*
BPP_8	0.15	0.2	2.727	8.0	0.841	0.172	0.013	0.05
BPP2	0.15	0.2	2.127	5.0	0.85	0.173	0.014	0.05
Hard0	0.201	0.349	1.005	2.0	0.929	0.272	0.042	0.148
Hard2	0.200	0.349	1.005	2.0	0.89	0.277	0.044	0.149
Hard3	0.200	0.347	1.015	2.0	0.94	0.272	0.042	0.147
hBPP14	0.011	0.696	1.176	3.0	0.875	0.380	0.207	0.685
hBPP175	0.005	0.798	1.081	3.0	0.89	0.414	0.223	0.793
hBPP531	0.007	0.798	1.142	3.0	0.91	0.414	0.225	0.791
hBPP640	0.012	0.8	1.090	2.0	0.922	0.410	0.232	0.788
hBPP814	0.001	0.794	1.117	2.0	0.929	0.404	0.237	0.793
N1w1b3r7	0.037	0.603	1.086	2.0	0.78	0.358	0.167	0.566
N3w2b1r8	0.162	0.24	2.739	6.0	0.895	0.199	0.023	0.078
N3c3w2_b	0.14	0.666	2.739	6.0	0.905	0.404	0.149	0.526
t60_00	0.251	0.495	1.2	3.0	0.466	0.333	0.072	0.244
t60_01	0.251	0.475	1.071	2.0	0.55	0.333	0.070	0.223
TEST0005	0.004	0.496	2.0	4.0	0.798	0.245	0.144	0.492

TEST0030	0.003	0.492	2.018	5.0	0.864	0.243	0.151	0.488
u120_00	0.133	0.653	2.068	5.0	0.916	0.393	0.147	0.52
u120_08	0.133	0.666	1.791	4.0	0.891	0.415	0.156	0.533
u250_12	0.133	0.666	3.289	8.0	0.836	0.419	0.155	0.533

Fase 3: Visualización de instancias y comportamiento algorítmico

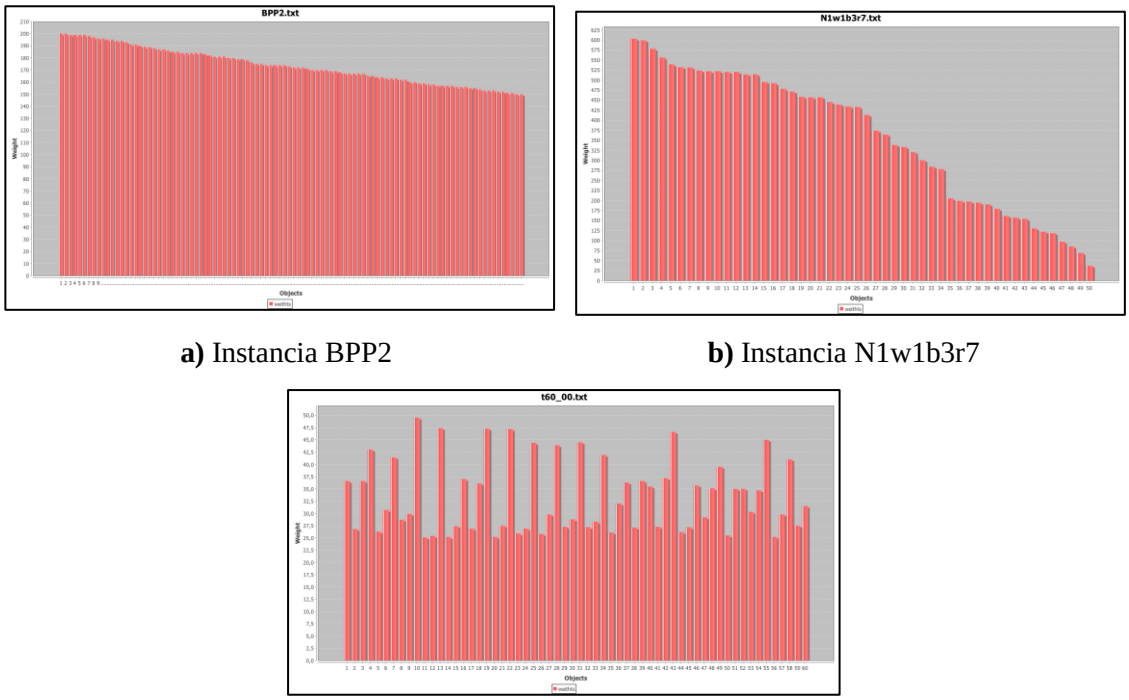
Gráfica de los Pesos de los Objetos

El objetivo de este tipo de gráfica es visualizar los pesos de los objetos de una instancia de BPP. Para ello se utiliza una gráfica de barras, en donde en el eje de las abscisas estará el número de objeto y en el eje de las ordenadas el peso de éste. El objetivo es visualizar las tendencias en la distribución inicial de los pesos de los objetos. La Figura 6.5 muestra 3 gráficas correspondientes con 3 instancias.

La Figura 6.5a muestra la distribución inicial de los pesos de los objetos de la instancia BPP2, de la cual se puede ver que los pesos vienen ordenados en forma decreciente y que su rango es muy pequeño (diferencia entre el mayor de los pesos y el menor); otras instancias que tienen características similares en la distribución de los pesos son: BPP_8, Hard0, Hard2 y Hard3, N3w2b1r8.

La Figura 6.5b muestra la distribución inicial de la instancia N1w1b3r7, de la cual se puede observar que similarmente a la distribución mostrada en la Figura 5a, los pesos vienen dados de manera decreciente, sin embargo en esta distribución el rango es mucho mayor; otras instancias que tienen características parecidas son: N3c3w2_b, TEST0005, TEST0030, hBPP14, hBPP175, hBPP531, hBPP640 y hBPP814.

Finalmente, la Figura 6.5c muestra la distribución de los pesos de la instancia t60_00, de la cual se puede observar que los pesos no vienen dados con un orden; otras instancias que comparten similitudes con ella son: t60_01, u120_00, u120_08 y u250_12.



a) Instancia BPP2 **b)** Instancia N1w1b3r7

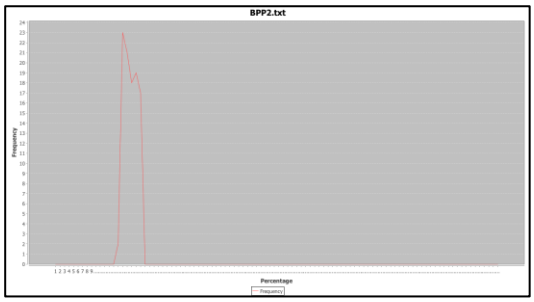
c) Instancia t60_00

Figura 6.5. Visualización de la distribución inicial de los pesos.

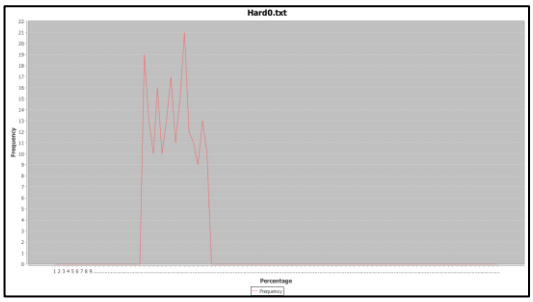
Gráfica de Frecuencias

Las gráficas de frecuencias, como su nombre lo sugiere, permiten conocer la cantidad de objetos que se repiten dentro de un determinado porcentaje de la capacidad del contenedor y dentro de que rango se encuentran los objetos.

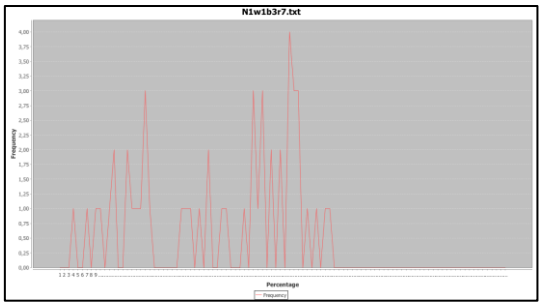
La Figura 6.6 muestra las gráficas de frecuencias obtenidas del conjunto de instancias a resolver, en algunos casos se encontró que una instancia tenía más parecido a instancias de otro conjunto distinto, tal es el caso de la instancia N3w2b1r8, la cual se asemeja más a las instancias del conjunto “Hard” en términos de su gráfica de frecuencias.



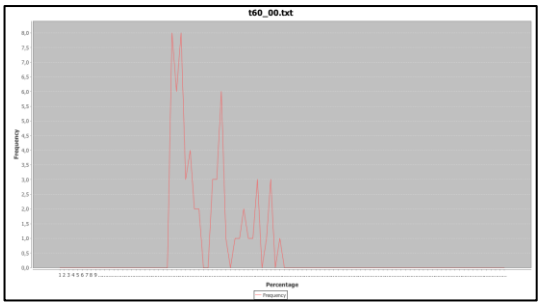
a) Instancia BPP2



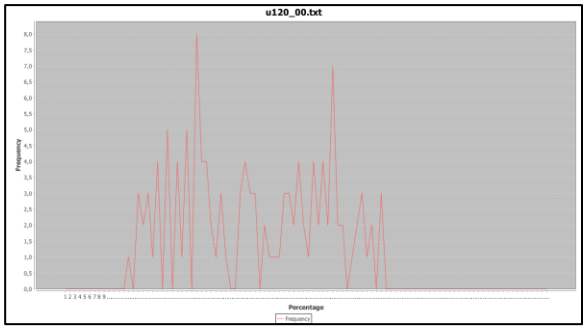
b) Instancia Hard0



c) Instancia N1w1b3r7



d) Instancia t60_00



e) Instancia u120_00

Figura 6.6. Gráficas de frecuencias sobre algunas instancias del caso de estudio.

En la Figura 6.6a se puede observar que los pesos de la instancia BPP2 están distribuidos entre el 14% y el 20% de la capacidad del contenedor. Además, entre el 15% y el 16% están 23 objetos (el pico más alto). La única instancia que comparte características parecidas es la BPP_8.

La Figura 6.6b muestra la distribución de los pesos de la instancia Hard0, en la gráfica se puede observar que los pesos de los objetos están distribuidos entre el 20% y el 35% de la capacidad del contenedor. Además en ese intervalo (que es un 9% más grande que el de la instancia BPP2) existen muchos altibajos (muchos picos). Las instancias con

gráficas similares son las siguientes: Hard2, Hard3 y N3w2b1r8.

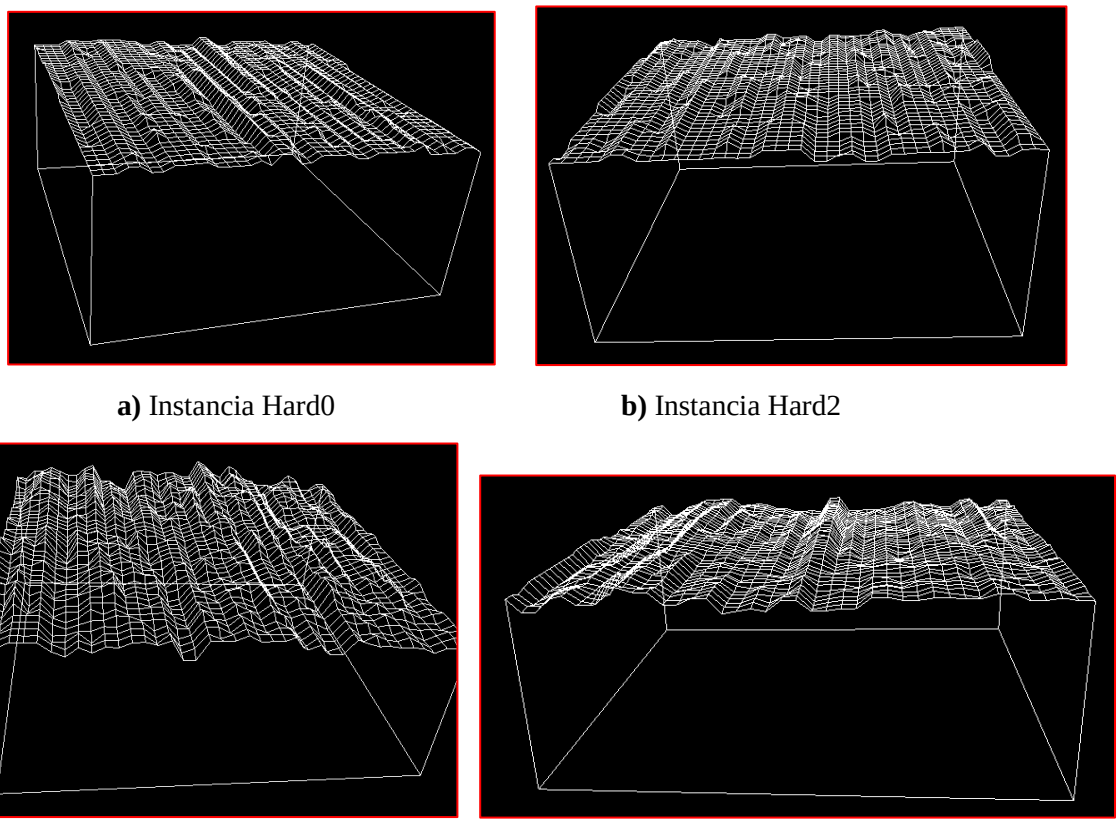
En la Figura 6.6c se puede ver que la distribución de los pesos de la instancia N1w1b3r7 empieza a partir del 3% y termina en el 61%, por lo que la amplitud de ese intervalo es del 58% de la capacidad del contenedor. Además dentro de ese intervalo existen muchos picos y valles, algunos de estos últimos llegan a cero, i.e., en esta gráfica, entre el 22% y 26% no hay ningún objeto que esté contenido en ese intervalo. Las instancias que se asemejan a ésta son: N3c3w2_b, TEST0005, TEST0030, hBPP14, hBPP175, hBPP531, hBPP640 y hBPP814.

En la Figura 6.6d se muestra la gráfica de frecuencias de la instancia t60_00, en la cual la distribución de los pesos se encuentra entre el 26% y el 50% de la capacidad del contenedor, lo cual implica un rango del 24%. Similarmente a la Figura 6c existen varios picos y valles, algunos de los cuales llegan a tocar el eje de las abscisas. Sin embargo, a diferencia de la Figura 6.6c, la Figura 6.6d no empieza su distribución tan cargado a la izquierda, sino más bien en el segundo cuarto de la gráfica, además la amplitud del rango de la Figura 6.6c es mucho menor que la de la Figura 6.6d. La única instancia parecida es la t60_01.

Por último, la Figura 6e muestra la gráfica de la instancia u120_00, en la cual el intervalo de la distribución de pesos se encuentra entre el 14% y el 65% de la capacidad del contenedor, es decir, su amplitud es de 51%. Las instancias que asemejan a esta son: u120_08 y u250_12.

Visualización de la Superficie de Aptitudes.

Otra forma de caracterizar la estructura del problema es a través de la visualización del espacio de búsqueda. La literatura establece que mientras más rugosa es la superficie, es más difícil de resolver la instancia [Pérez 07, Fraire 10]. Sin embargo, en este estudio existen excepciones que pudieran ser explicadas por características aún no identificadas.



a) Instancia Hard0 **b)** Instancia Hard2

c) Instancia u120_00 **d)** instancia u250_12

Figura 6.7. Superficies de aptitudes de cuatro instancias en VisTHAA.

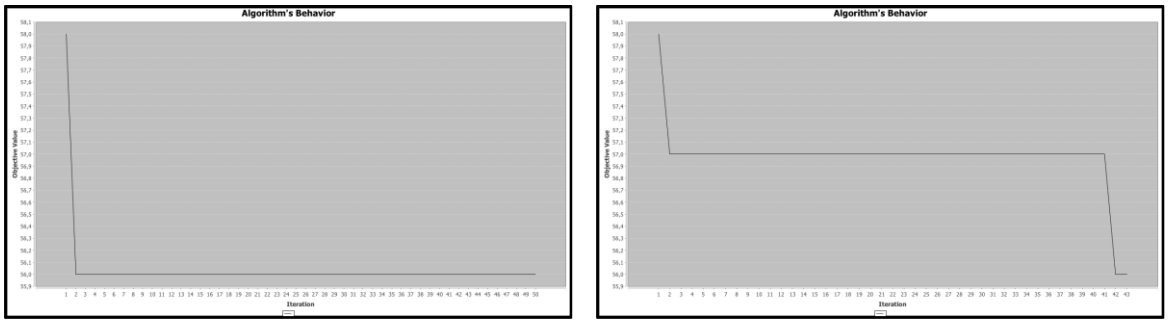
En la Figura 6.7 se muestran cuatro gráficas de las superficies de aptitudes que corresponden a cuatro instancias. Las Figuras 6.7a y 6.7b muestran las superficies de las instancias Hard0 y Hard2, respectivamente. Además de estas gráficas, la Tabla 6.1 muestra la matriz de características, la cual contiene información descriptiva acerca de las instancias del conjunto, el cual es denominado "Hard". Estas instancias son muy similares entre sí, sin embargo, en las gráficas de superficie de aptitudes se puede observar que la instancia Hard2 es menos rugosa que la Hard0, los resultados experimentales muestran que la instancia Hard2 es más difícil de resolver que la instancia Hard0 para el algoritmo WABP.

Las Figuras 6.7c y 6.7d muestran las instancias u120_00 y u120_08, respectivamente. Dichas instancias pertenecen al conjunto denominado "U". Similarmente al conjunto "Hard", los valores de la matriz de características son muy parecidos entre sí, excepto en el

tercer y cuarto atributos, multiplicidad y maxRepe. Las gráficas de superficie de aptitudes muestran que la instancia u120_00 es más rugosa que la instancia u250_12, y los resultados experimentales muestran que u250_12 es más difícil de resolver que u120_00 para el algoritmo WABP.

Visualización del Comportamiento Algorítmico

Además de los resultados previamente descritos del análisis de las instancias, en esta sección se analizará el comportamiento del algoritmo WABP durante su ejecución. Al analizar el comportamiento del algoritmo a través del valor objetivo con respecto al tiempo, se podría identificar si el algoritmo se ha quedado estancado en soluciones subóptimas, es decir, la gráfica no mostrará cambios en el valor objetivo durante un número de iteraciones continuas.



a) Instancia Hard0 **b)** Instancia Hard2

Figura 6.8. Comportamiento algorítmico con dos instancias.

La Figura 6.8 visualiza el comportamiento del algoritmo durante su ejecución sobre dos instancias, Hard0 y Hard2. Al analizar las gráficas, en la Figura 6.8a se puede observar que el algoritmo no mejora la mejor solución encontrada durante la mayor parte del tiempo de ejecución, lo cual representa una pérdida de tiempo de cómputo. Similarmente, la Figura 6.8b muestra que el algoritmo converge en 42 iteraciones. Sin embargo, desde la segunda, hasta la cuadragésima segunda iteración, el algoritmo no mejora la calidad de la solución, lo cual también representa una pérdida de tiempo computacional.

Fase de Rediseño del Algoritmo.

Basado en el análisis gráfico del número de iteraciones sin mejora para el algoritmo bajo estudio, se observó que se podía reducir el tiempo de cómputo reconfigurando el parámetro de máximas iteraciones. Dicho valor fue obtenido calculando la iteración promedio donde el algoritmo lograba su última mejora sobre cada instancia, el cual fue 4.7, en consecuencia un máximo número de 5 iteraciones fue utilizado.

Resultados

Después de haber calculado el nuevo número de iteraciones máximas, se volvió a realizar la experimentación sobre el mismo conjunto de instancias, los resultados se muestran en la Tabla 6.2.

Tabla 6.2. Resultados experimentales con ambas configuraciones del algoritmo WABP.

		Iteraciones máximas = 50		*Iteraciones máximas= 5*	
Instancia	*Óptimo*	*Contenedores*	*Iteraciones*	*Contenedores*	*Iteraciones*
BPP_8	21	21	2	21	2
BPP2	18	18	1	18	1
Hard0	56	56	50	56	5
Hard2	56	56	41	57	5
Hard3	55	55	21	56	5
hBPP14	62	62	50	62	5
hBPP175	84	84	50	84	5
hBPP531	83	84	50	84	5
hBPP640	74	75	50	75	5
hBPP814	81	82	50	82	5
N1w1b3r7	18	19	50	19	5
N3w2b1r8	40	41	50	41	5
N3c3w2_b	81	82	50	82	5
t60_00	20	21	50	21	5
t60_01	20	21	50	21	5
TEST0005	28	29	50	29	5
TEST0030	27	28	50	28	5
u120_00	48	48	3	48	3
u120_08	50	51	50	51	5

u250_12	105	106	50	106	5
TOTAL	1027	1039	818	1041	91

La Tabla 6.2 muestra que WABP logra un ahorro considerable en tiempo de 727 iteraciones. Lo cual refleja una mejora del 88% en tiempo de cómputo, y una ligera pérdida de la calidad de las soluciones encontradas de 0.19%. Los resultados obtenidos pueden ser considerados buenos ya que existen aplicaciones reales en donde es más importante el tiempo de cómputo que la insignificante pérdida en la calidad de las soluciones.

Análisis de Resultados

Las instancias que encontraron el óptimo con 50 iteraciones, pero no lo encontraron con 5 son caracterizadas por tener en promedio pocos objetos repetidos (baja multiplicidad); peso promedio de un objeto utiliza un 27% de la capacidad del contenedor (valor de *t*), coincidiendo con lo reportado en la literatura [Quiroz 09]; y la diferencia promedio de los pesos de los objetos utiliza un 15% de la capacidad del contenedor (valor de *rango*).

Fase de Validación estadística utilizando la Prueba de Wilcoxon

Al final del experimento, para soportar los resultados del caso de estudio, fue realizada una prueba no paramétrica llamada la Prueba de Rangos con Signos de Wilcoxon. El objetivo de esta prueba es determinar si las diferencias entre la eficiencia del algoritmo con diferentes configuraciones de parámetros son estadísticamente significativas con un determinado nivel de confiabilidad, o no lo son.

Aplicando la prueba a los conjuntos generados (iteraciones de convergencia), ésta reveló que existen diferencias significativas con un 99% de confiabilidad. Por lo cual se puede concluir que el ahorro en tiempo de cómputo es estadísticamente significativo.

Es evidente que la Prueba de Wilcoxon sobre los conjuntos de efectividad de ambas configuraciones no se puede llevar a cabo, esto se debe a que para realizar esta prueba, debe haber al menos cinco diferencias en las unidades experimentales (instancias) no nulas. Sin

embargo, únicamente hay dos de éstas que cumplen la condición.

Además, la Prueba de Wilcoxon es utilizada en aquellos casos donde determinar si las diferencias son significativas no es tan obvio. En el caso de los datos de la efectividad, sólo en dos instancias no se logró encontrar el mismo número de contenedores, en las demás sí se logró. Así que resulta muy evidente que las diferencias en estos datos no son significativas.

Para realizar la prueba de Wilcoxon en VisTHAA, se debe de crear un archivo de texto plano, el cual contendrá en su primer renglón los valores del primer conjunto de datos, en este caso puede ser los resultados con la configuración original; en el segundo renglón los valores del segundo conjunto, es decir, los resultados utilizando la nueva configuración propuesta en la fase de rediseño. La Figura 6.9 muestra el archivo de texto plano utilizado para introducir los datos a VisTHAA.

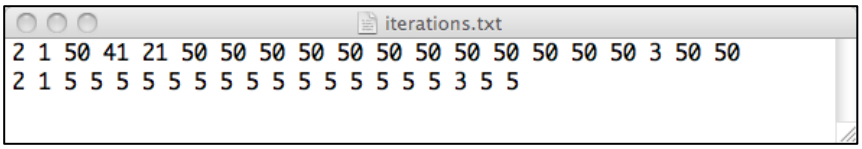


Figura 6.9. Archivo con los datos necesarios para realizar la prueba de Wilcoxon.

Una vez introducidos los datos a VisTHAA, aparece una ventana donde se muestran dichos datos y en ella un botón que dice “Execute Wilcoxon Test”, se debe hacer clic sobre éste de manera que la prueba pueda llevarse a cabo. La Figura 6.10 muestra el módulo en VisTHAA para realizar la Prueba de Wilcoxon.

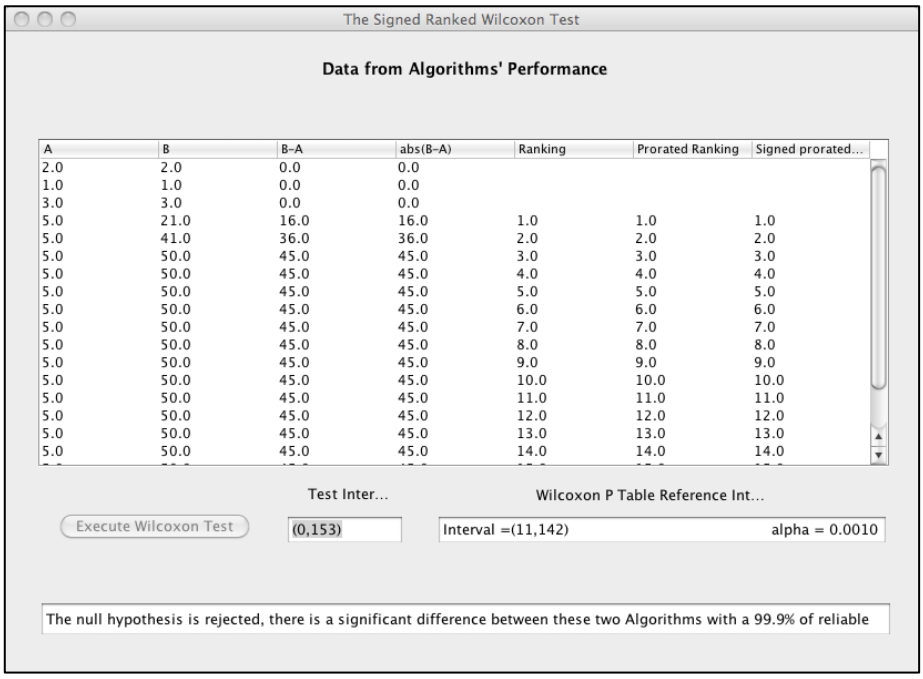


Figura 6.10. Módulo en VisTHAA que realiza la prueba de Wilcxon.

Conclusiones del Experimento 1

Se encontró que después de caracterizar las instancias en diferentes formas así como el comportamiento algorítmico, el diagnóstico visual fue el más importante para identificar que tipo de estrategia se debió seguir para mejorar el desempeño del algoritmo. Las gráficas proveyeron información extra que no está en la matriz de atributos.

Además se concluye que el análisis visual y estadístico hacen posible mejorar los algoritmos heurísticos. Particularmente, se realizó el ajuste del parámetro de iteraciones máximas del algoritmo WABP, esto permitió el logro de una notable mejora en la eficiencia. Este estudio fue capaz de reducir el tiempo de cómputo en un 88% con una pérdida no significativa del 0.19% en la efectividad, es decir, el número de contenedores utilizados. De acuerdo con lo anterior, se puede concluir que se cumplió el objetivo propuesto.

6.2.2. Experimento 2: Análisis de la Familia de Instancias "Hard"

Objetivo

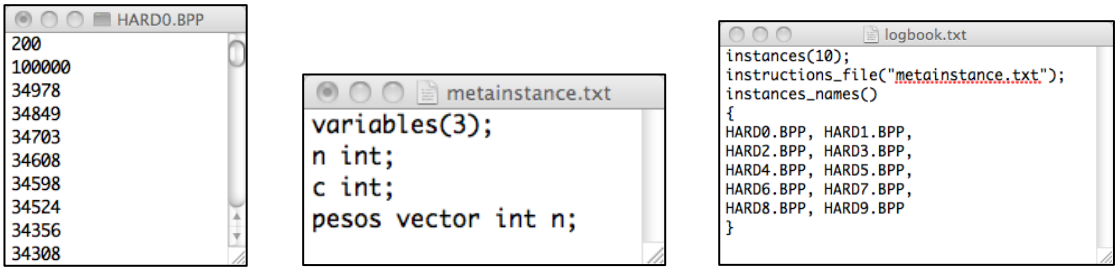
Analizar la estructura de las instancias para identificar características descriptivas dentro del conjunto "Hard".

Procedimiento

El procedimiento utilizado, el problema de optimización y el algoritmo solucionador, son los mismos que se utilizaron en el experimento 2. El conjunto de instancias a resolver es el denominado "Hard", del cual tres de las diez instancias del conjunto (Hard0, Hard2 y Hard3) fueron utilizadas en el experimento 2.

Fase 1: Entrada de datos

En esta fase se realiza la descripción de las instancias, la especificación de la cantidad y ubicación de éstas. La Figura 6.11 muestra la forma de hacerlo.



a) Instancia de BPP **b)** archivo metainstance **c)** archivo logbook

Figura 6.11. Introducción de datos de instancias a VisTHAA.

Fase 2: Caracterización de las instancias

En esta fase se realiza la caracterización estadística de las instancias. El conjunto de índices introducidos a la herramienta fue el mismo que el utilizado en el experimento 2: media

normalizada, desviación estándar normalizada y rango normalizado.

La Ecuación 6.2 muestra el índice rango normalizado, y en la Figura 6.12 puede verse cómo introducirlo a VisTHAA.

$$range = \frac{\max(W_i) - \min(w_i)}{c} \qquad \textbf{(6.2)}$$

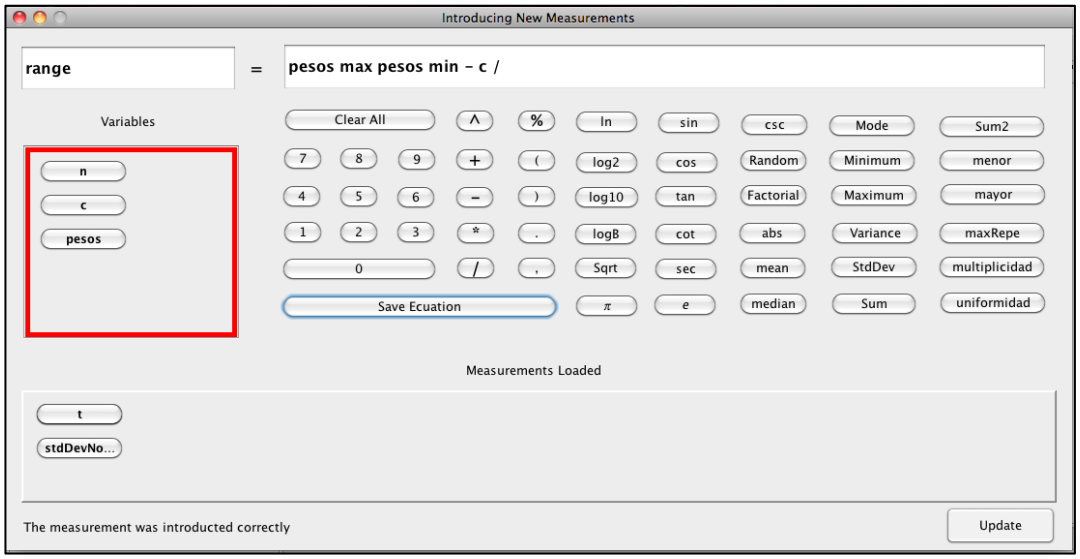


Figura 6.12. Módulo que permite la introducción de nuevos índices en VisTHAA.

Una vez que un conjunto de índices es introducido, el investigador puede decidir cuales de ellos considerar para generar la matriz de características sobre el conjunto de instancias seleccionadas. La Figura 6.13 muestra el módulo que permite generar la matriz de características.

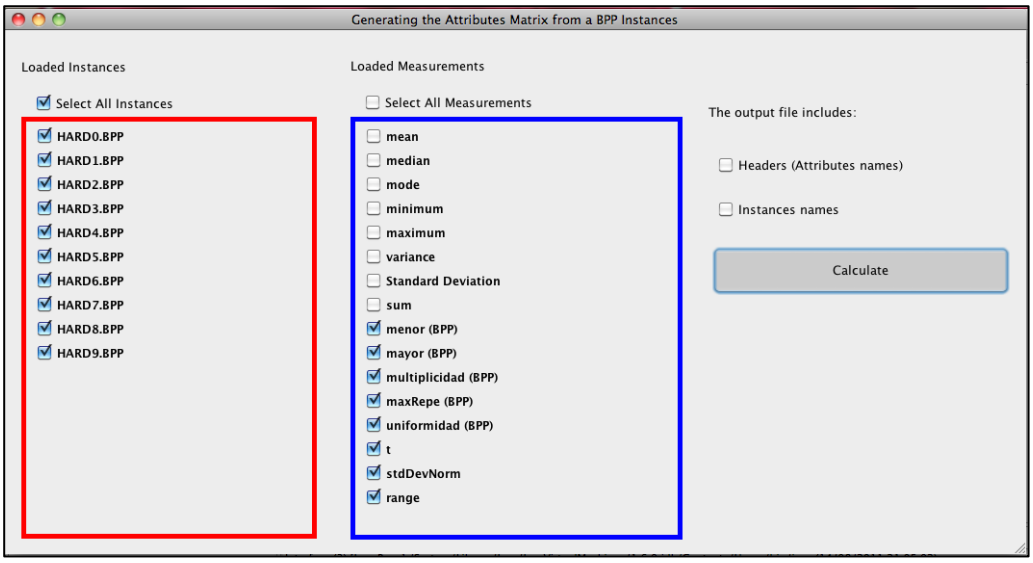


Figura 6.13. Módulo que permite generar la matriz de características.

La Tabla 6.3 muestra la matriz de características sobre el conjunto de instancias utilizadas en este estudio.

La simbología utilizada en la Tabla 6.3 es la siguiente: 1) menor; 2) mayor; 3) multiplicidad; 4) maxRepe; 5) Uniformidad; 6) media normalizada; 7) Desviación estándar normalizada y 8) rango normalizado.

Tabla 6.3. Matriz de características sobre el conjunto "Hard".

Instancia	***Ind1***	***Ind2***	***Ind3***	***Ind4***	***Ind5***	***Ind6***	***Ind7***	***Ind8***
Hard0	0.201	0.349	1.005	2	0.929	0.272	0.042	0.148
Hard1	0.200	0.349	1.000	1	0.95	0.276	0.043	0.149
Hard2	0.200	0.349	1.005	2	0.89	0.277	0.044	0.149
Hard3	0.200	0.347	1.015	2	0.94	0.272	0.042	0.147
Hard4	0.201	0.350	1.010	3	0.92	0.277	0.041	0.148
Hard5	0.200	0.349	1.005	2	0.87	0.274	0.041	0.149
Hard6	0.200	0.349	1.005	2	0.92	0.277	0.042	0.149
Hard7	0.200	0.348	1.000	1	0.84	0.269	0.044	0.147
Hard8	0.200	0.349	1.005	2	0.85	0.277	0.043	0.149
Hard9	0.200	0.349	1.000	1	0.96	0.275	0.043	0.149

Posterior al análisis de la matriz de características, podemos notar que, para los

atributos medidos, no existe una diferencia notable entre las instancias del conjunto.

Fase 3: Visualización de instancias y comportamiento algorítmico

Gráfica de los Pesos de los Objetos

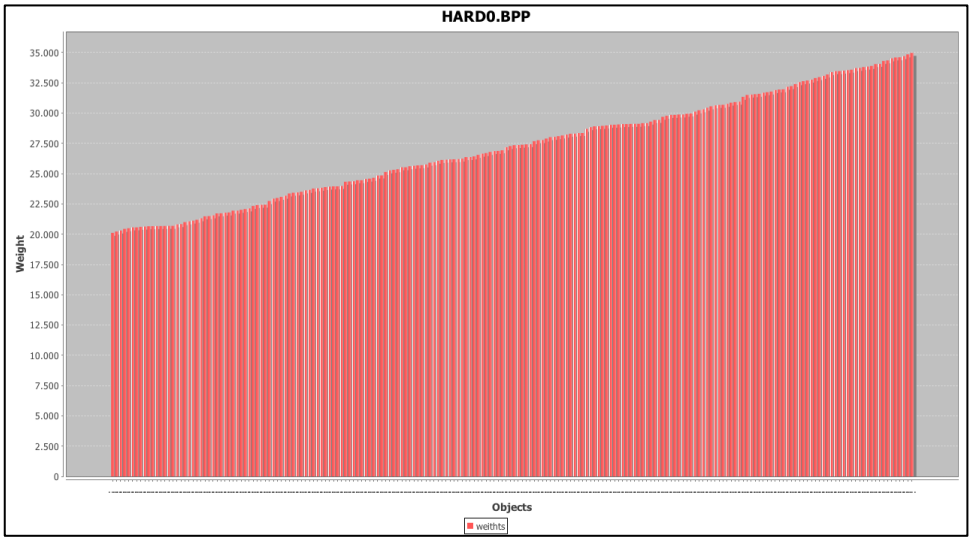
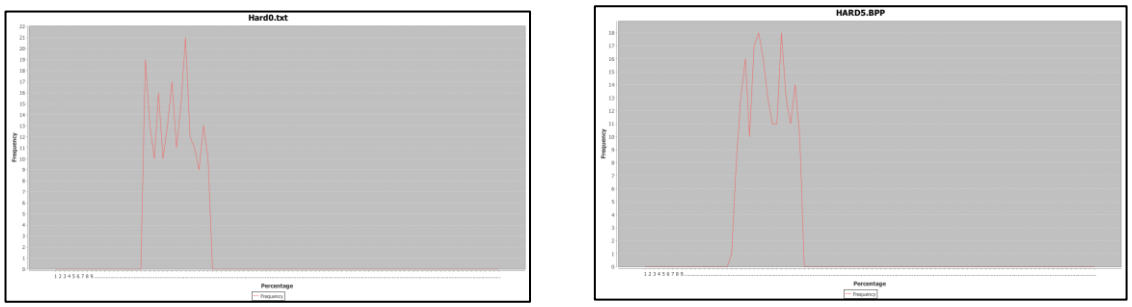


Figura 6.14. Visualización de la distribución inicial de los pesos de la instancia Hard0.

La primera característica visual a analizar es cómo vienen dados los objetos. La Figura 6.14 muestra la gráfica de la instancia Hard0. Al realizar el análisis resultó evidente que todas las instancias del conjunto tenían la misma distribución inicial de pesos.

Gráfica de Frecuencias

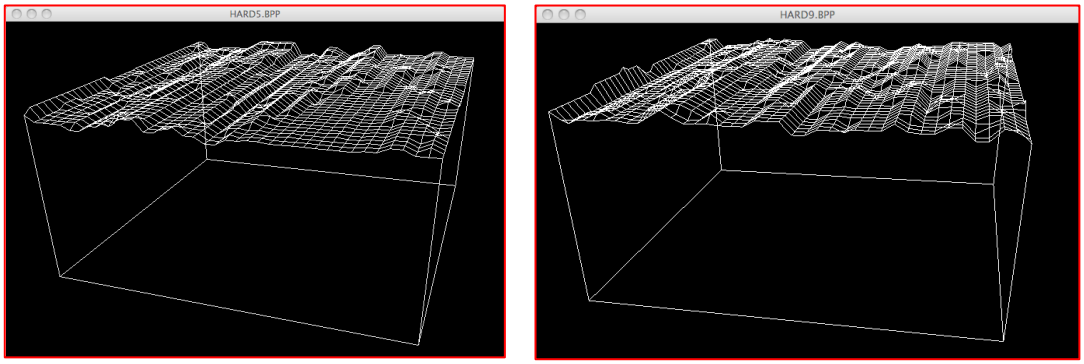
La Figura 6.15 muestra las gráficas de frecuencias obtenidas de dos instancias del conjunto, en general todas ellas presentan características similares en la frecuencia. A excepción de Hard5 que tiene un objeto que ocupa entre el 19% y 20% de la capacidad del contenedor, las frecuencias de todas las instancias oscilan entre el 20% y el 35%. Otra característica importante en esta gráfica, es que se puede ver existen muchos altibajos en la frecuencia, esto es consistente en todas las instancias del conjunto.



a) Instancia Hard0 **b)** Instancia Hard5

Figura 6.15. Gráficas de frecuencias sobre algunas instancias del caso de estudio.

Visualización de la Superficie de Aptitudes.



a) Instancia Hard5. **b)** Instancia Hard9.

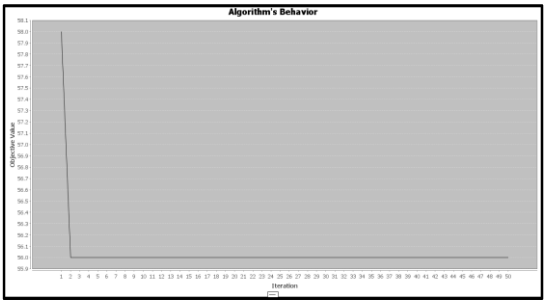
Figura 6.16. Superficies de aptitudes de dos instancias en VisTHAA.

En la Figura 6.16 se muestran dos gráficas de superficie de aptitudes, de igual manera que con las gráficas de frecuencias, no se observaron diferencias muy marcadas entre las instancias del conjunto.

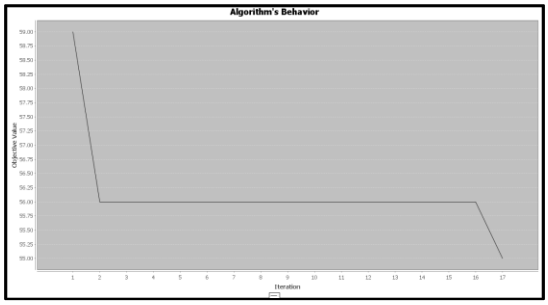
Visualización del Comportamiento Algorítmico

La Figura 6.17 visualiza el comportamiento del algoritmo durante su ejecución sobre tres instancias. Al analizar la gráfica en la Figura 6.17a se puede observar que el algoritmo no mejora la mejor solución encontrada durante la mayor parte del tiempo de ejecución, es decir 49 iteraciones, lo cual representa una pérdida de tiempo, la mayoría de las instancias analizadas presentan el mismo comportamiento, a excepción de las instancias Hard3 y

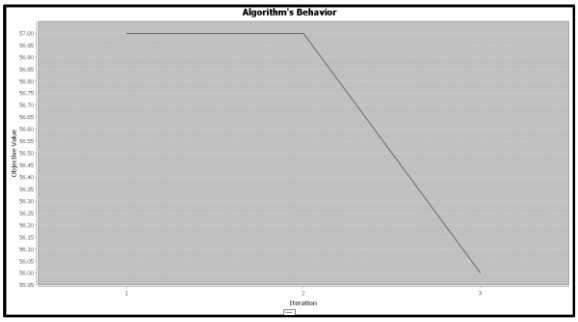
Hard9. La Figura 6.17b muestra que su última mejora en la calidad de la solución encontrada fue en la iteración 17 y que después de ella el algoritmo se detiene. Finalmente, en la Figura 6.17c se puede ver que el algoritmo converge en la tercera iteración y posteriormente se detiene su ejecución.



a) Instancia Hard0.



b) Instancia Hard3.



c) Instancia Hard9.

Figura 6.17. Comportamiento algorítmico con tres instancias.

Fase de Rediseño del Algoritmo.

Al igual que en el experimento 2, el análisis gráfico del número de iteraciones sin mejora fue la pieza clave para mejorar el desempeño el algoritmo. Se observó que se podía reducir el tiempo de cómputo reconfigurando el parámetro de máximas iteraciones. A diferencia del experimento 2, se consideraron 2 posibles valores para el parámetro de iteraciones máximas, el primer valor se obtuvo reconociendo el siguiente patrón: a excepción de las instancias Hard3 y Hard9, todas las demás convergen en iteraciones muy tempranas, es decir la segunda o la tercera (Hard5, Hard7 y Hard9). Además la instancia Hard3 es la que tarda más en convergir (las otras instancias terminan en 50 iteraciones pero no mejoran la

calidad de las solución después de la segunda o tercer iteración). Derivado de lo anterior, se utilizó el valor de la iteración de convergencia de la instancia Hard3 para obtener el valor de máxima iteraciones, es decir, se estableció *nloop* = 17.

Similarmente al experimento 2, el segundo valor de máximas iteraciones fue obtenido calculando la iteración promedio donde el algoritmo lograba su última mejora sobre cada instancia, el cual fue 3.8, en consecuencia un máximo número de 4 iteraciones fue utilizado para la segunda reconfiguración del algoritmo.

Resultados

Después de haber calculado los nuevos valores de iteraciones máximas, se volvió a realizar la experimentación, los resultados se muestran en la Tabla 6.4.

Tabla 6.4. Resultados experimentales con ambas configuraciones del algoritmo WABP.

		nloop=50		*nloop*=17		*nloop*=4	
Instancias	Óptimo	Z_{enc}	Iteraciones	Z_{enc}	Iteraciones	Z_{enc}	Iteraciones
Hard0	56	56	50	56	17	56	4
Hard1	57	57	50	57	17	57	4
Hard2	56	57	50	57	17	57	4
Hard3	55	55	17	55	17	56	4
Hard4	57	57	50	57	17	57	4
Hard5	56	56	50	56	17	56	4
Hard6	57	57	50	57	17	57	4
Hard7	55	55	50	55	17	55	4
Hard8	57	57	50	57	17	57	4
Hard9	56	56	3	56	3	56	3
Totales	562	563	420	563	156	564	39

La Tabla 6.4 muestra que con la configuración inicial de parámetros, el algoritmo WABP encuentra nueve de diez valores óptimos, los cuales suman 562 contenedores utilizados.

Con la primera reconfiguración del algoritmo se puede observar que éste logra un ahorro considerable en tiempo de 264 iteraciones, lo cual refleja una mejora del 62% en tiempo de cómputo, sin pérdida en la calidad de las soluciones, es decir, de todas las instancias se encontraron los mismos valores objetivo que con 50 iteraciones máximas.

Al analizar el desempeño del algoritmo con la segunda reconfiguración, se puede notar que se logra un ahorro mayor en tiempo de 381 iteraciones, es decir, el 90%, a cambio de una ligera pérdida en la calidad de las soluciones encontradas de 0.17%, esto es, únicamente con la instancia Hard3 el algoritmo no pudo encontrar el mismo valor que con las configuraciones anteriores, se tuvo un error de un contenedor.

Análisis de Resultados

Para analizar los resultados obtenidos y validar estadísticamente la contundencia en la mejora del desempeño algorítmico, se utilizó la prueba no paramétrica de Wilcoxon.

Fase de Validación estadística utilizando la Prueba de Wilcoxon

El objetivo de esta prueba es determinar si las diferencias entre la eficiencia del algoritmo con diferentes configuraciones de parámetros son estadísticamente significativas con un determinado nivel de confiabilidad, o no lo son.

Debido a que se realizaron tres experimentaciones con el algoritmo (*nloop*=50, *nloop*=17 y *nloop*=4), y que la prueba de Wilcoxon sólo puede hacerse con dos conjuntos de datos, es necesario realizar la prueba dos veces. La primera con los resultados de la configuración inicial de parámetros (*nloop*=50) contra la primera reconfiguración de parámetros (*nloop*=17) y la segunda con los resultados de la primera y la última reconfiguración de parámetros.

Realizando la primera prueba a los conjuntos generados (datos de eficiencia), ésta

reveló que existen diferencias significativas con un 99% de confiabilidad. Por lo cual se puede concluir que el ahorro en tiempo de cómputo es estadísticamente significativo entre la configuración inicial y la primera reconfiguración.

Al realizar la segunda prueba, ésta determinó que también existen diferencias significativas entre ambas reconfiguraciones del algoritmo, con un 99% de confiabilidad. Razón por la cual se concluye que existe una diferencia significativa.

De lo anterior se puede concluir que si existen diferencias entre la configuración inicial y la primera reconfiguración, y además existen diferencias entre la primera y la segunda reconfiguración, entonces por transitividad, también existen diferencias significativas entre la última reconfiguración y la configuración inicial con un 99% de confiabilidad.

Es evidente que la Prueba de Wilcoxon sobre los conjuntos de efectividad de la configuración inicia y la primera reconfiguración no se puede llevar a cabo, esto se debe a que para realizar esta prueba, debe haber al menos cinco diferencias en las unidades experimentales (instancias) no nulas. Sin embargo, no hay ninguna instancia que cumpla con esta condición.

De igual manera, la Prueba de Wilcoxon sobre los conjuntos de efectividad de la primera y la segunda reconfiguración tampoco se puede llevar a cabo ya que únicamente una instancia (Hard3) cumple con dicha condición.

Por lo que se puede concluir que no existen diferencias significativa entre ninguno de los tres conjuntos de efectividad.

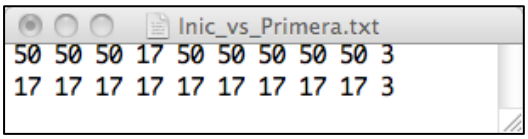


Figura 6.18. Archivo con los datos de eficiencia para realizar la prueba de Wilcoxon.

La Figura 6.18 muestra el archivo de texto plano utilizado para introducir los datos de eficiencia entre la configuración inicial y la primera reconfiguración.

Una vez introducidos los datos a VisTHAA, aparece una ventana donde se muestran dichos datos y en ella un botón que dice “Execute Wilcoxon Test”, se debe hacer clic sobre éste de manera que la prueba pueda llevarse a cabo. La Figura 6.19 muestra el módulo en VisTHAA para realizar la Prueba de Wilcoxon.

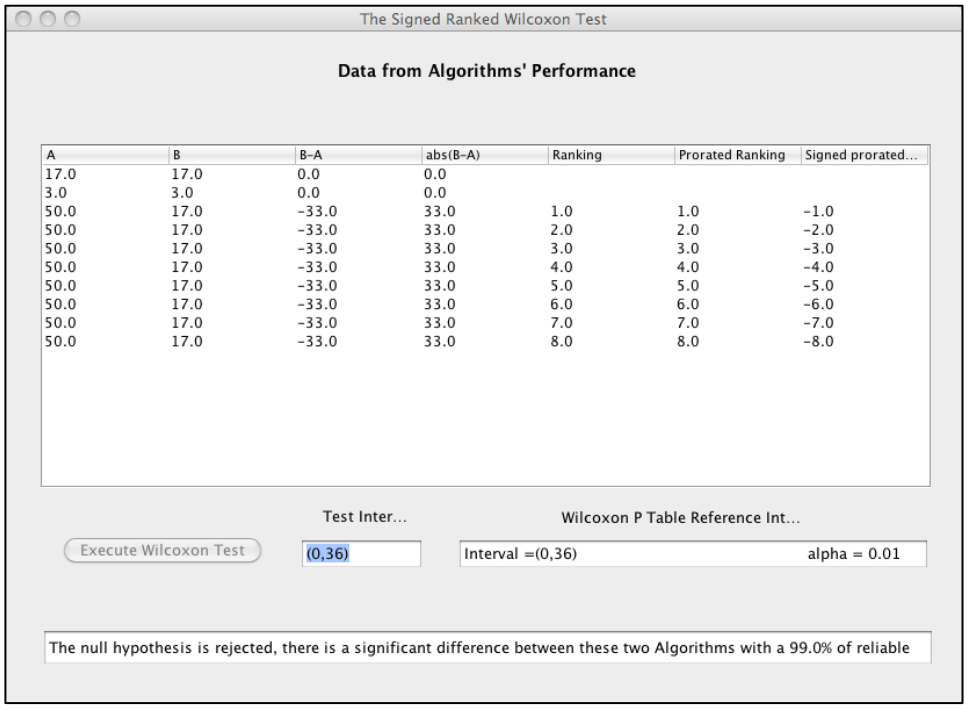


Figura 6.19. Módulo en VisTHAA que realiza la prueba de Wilcoxon.

Conclusiones del Experimento 2

Después de realizar la caracterización estadística y visual de todas las instancias del

conjunto, se puede concluir que no existen diferencias evidentes entre éstas, es decir, el conjunto "Hard" es un grupo muy homogéneo.

Sin embargo, al analizar el comportamiento del algoritmo WABP sobre todo el conjunto, sí se encontraron diferencias notables entre las instancias, lo cual permitió realizar dos reconfiguraciones del algoritmo.

Estas reconfiguraciones consistieron en encontrar un nuevo valor para el máximo número de iteraciones de algoritmo (parámetro *nloop*). Dichas reconfiguraciones superaron en eficiencia a su predecesor, es decir, el primer ajuste permitió la mejora significativa de la configuración inicial, sin pérdida en su efectividad; el segundo ajuste permitió la mejora significativa del primer ajuste, con una insignificante pérdida de efectividad de 0.17%. De acuerdo con lo anterior, se puede concluir que se cumplió el objetivo propuesto.

6.2.3. Experimento 3: Análisis de Tres Familias de Instancias Diferentes

Objetivo

Identificar diferencias entre los atributos de tres conjuntos de instancias de procedencia distinta. Específicamente, se pretende saber cuáles características distinguen a cada una de ellas, incluyendo el comportamiento del algoritmo HGGA-BP sobre éstas.

Procedimiento

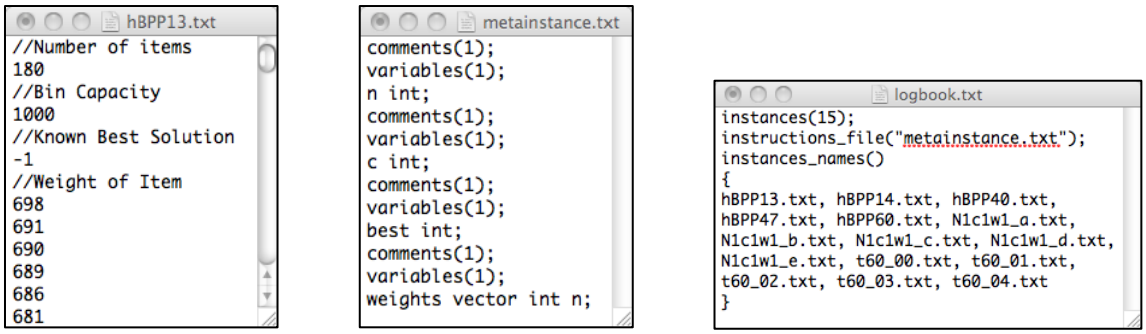
El procedimiento utilizado en este estudio es muy parecido al de los experimentos anteriores, con la diferencia en que no es de interés identificar áreas de mejora para el algoritmo estudiado, por lo que las fases de rediseño y validación de resultados no serán utilizadas. Este procedimiento consiste en tres fases: entrada de datos; caracterización de las instancias; y visualización de instancias y desempeño algorítmico.

Las instancias utilizadas en este estudio, fueron seleccionadas por la dificultad que

presenta el algoritmo HGGA-BP al resolverlas. Se clasificaron tres grupos: instancias fáciles, instancias de dificultad mediana e instancias difíciles; dentro de los cuales están los conjuntos: “DataSet1”, “T” y “Hard28”, respectivamente clasificados.

Fase 1: entrada de datos

En esta fase se realiza la descripción de las instancias, la especificación de la cantidad y ubicación de éstas. La Figura 6.20 muestra la forma de hacerlo.



a) Instancia de BPP. **b)** archivo metainstance. **c)** archivo logbook.

Figura 6.20. Introducción de los datos de las instancias a VisTHAA.

Fase 2: Caracterización de las Instancias

En esta fase se realiza la caracterización estadística de las instancias. El conjunto de índices introducidos a la herramienta fue el mismo que los utilizados en los experimentos anteriores: media normalizada, desviación estándar normalizada y rango normalizado. Evidentemente, la forma de introducir dichos índices es también igual que la descrita en anteriormente.

Una vez que un conjunto de índices es introducido, el investigador puede decidir cuales de ellos considerar para generar la matriz de características sobre el conjunto de instancias seleccionadas. La Tabla 6.5 muestra la matriz de características sobre el conjunto de instancias seleccionadas.

Tabla 6.5. Matriz de características sobre los tres grupos de instancias.

Instancia	*Ind1*	*Ind2*	*Ind3*	*Ind4*	*Ind5*	*Ind6*	*Ind7*	*Ind8*
N1c1w1_a	0.03	0.99	1.219	2	0.84	0.486	0.294	0.96
N1c1w1_b	0.08	1	1.351	3	0.9	0.556	0.291	0.92
N1c1w1_c	0.03	0.92	1.219	3	0.74	0.396	0.274	0.89
N1c1w1_d	0.02	1	1.351	3	0.68	0.506	0.322	0.98
N1c1w1_e	0.02	0.91	1.25	2	0.8	0.488	0.290	0.89
t60_00	0.251	0.495	1.2	3	0.466	0.333	0.072	0.244
t60_01	0.251	0.475	1.071	2	0.55	0.333	0.070	0.224
t60_02	0.25	0.498	1.25	3	0.466	0.333	0.081	0.248
t60_03	0.25	0.495	1.224	2	0.45	0.333	0.079	0.245
t60_04	0.25	0.498	1.25	3	0.433	0.333	0.078	0.248
hBPP13	0.001	0.698	1.118	3	0.811	0.372	0.211	0.697
hBPP14	0.011	0.696	1.176	3	0.875	0.380	0.207	0.685
hBPP40	0.019	0.789	1.111	2	0.837	0.368	0.240	0.77
hBPP47	0.003	0.783	1.139	3	0.855	0.393	0.234	0.78
hBPP60	0.053	0.696	1.111	2	0.862	0.393	0.193	0.643

La simbología utilizada en la Tabla 6.5 es la misma que en los experimentos anteriores: 1) *menor*; 2) *mayor*; 3) *multiplicidad*; 4) *maxRepe*; 5) *Uniformidad*; 6) *media normalizada*; 7) *Desviación estándar normalizada*; y 8) *rango normalizado*.

De la Tabla 6.5 podemos observar que en algunos atributos existen diferencias muy marcadas entre los grupos de las instancias. Para el primer atributo, *menor,* se puede ver que las instancias fáciles (las primeras cinco instancias en la Tabla 6.5) el objeto con peso más pequeño no sobrepasa el 10% de la capacidad del contenedor, característica que comparte con las instancias difíciles (últimas cinco instancias de la Tabla 6.5), mientras que en las instancias de mediana dificultad (las instancias del conjunto "T") el objeto con menor peso ocupa el 25% de la capacidad del contenedor.

En el segundo atributo, *mayor,* hay aún más diferencias entre grupos, por ejemplo, para el grupo de las instancias fáciles, el objeto con mayor peso ocupa entre el 92% y el 100% de la capacidad del contenedor, para el grupo de las instancias "T", este índice muestra que el mayor de los objetos ocupa entre el 47% y 50%, finalmente, para el grupo

de las instancias difíciles, los mayores objetos ocupan entre el 69% y el 79% del contenedor.

En el tercer atributo, *multiplicidad*, no se encontraron diferencias entre los valores de cada grupo, los valores en todos los conjuntos oscilan entre el 1.071 y el 1.351.

De igual manera que para el atributo *multiplicidad*, para *maxRepe* no se observaron muchas diferencias en los valores de los grupos, los cuales van desde 2 hasta 3. Cada grupo tiene tres instancias con *maxRepe* = 3 y dos instancias con *maxRepe* = 2.

Para el atributo *uniformidad*, podemos ver que los grupos de instancias más fáciles y más difíciles son más uniformes en la distribución de sus pesos, mientras que las instancias de dificultad mediana son menos uniformes. En el conjunto de instancias fáciles los valores de uniformidad van desde el 68% y el 90%, en el conjunto de instancias de dificultad mediana desde el 43% hasta el 55% y para el conjunto de instancias difíciles, desde el 81% hasta el 87%.

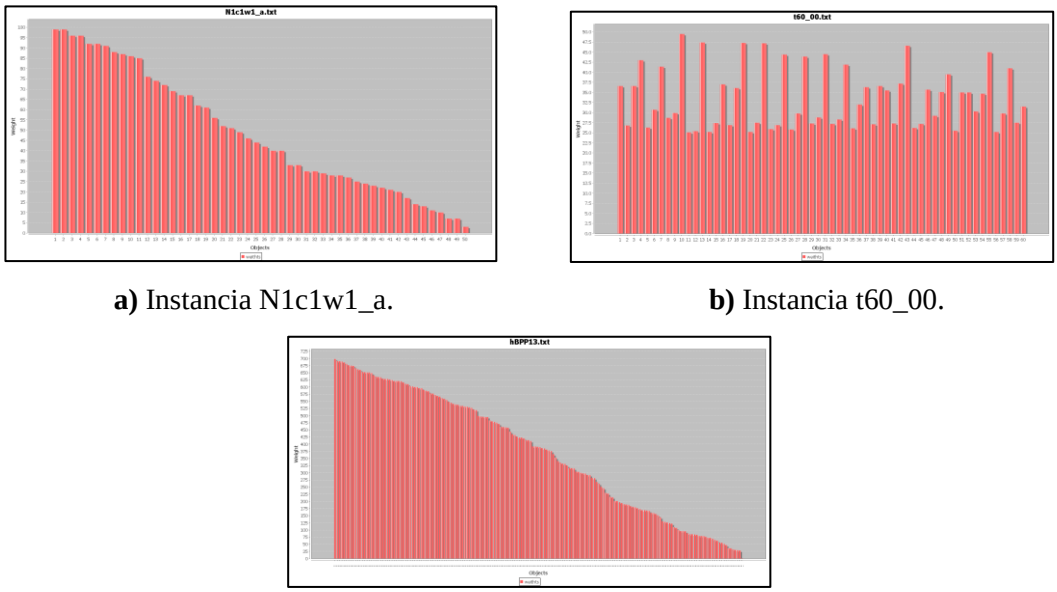
Para el atributo *media normalizada*, sí se aprecian diferencias entre grupos. Para el primer conjunto (instancias fáciles) el objeto promedio ocupa entre el 39% y 56% de la capacidad del contenedor, en el conjunto de instancias de dificultad mediana, el objeto promedio ocupa en todos los casos la tercera parte de la capacidad del contenedor, mientras que para el grupo de instancias difíciles, el objeto promedio desde el 36% hasta el 40%.

Otro índice en donde también hay diferencias entre los tres grupos de instancias, es el de la *desviación estándar normalizada*. Este índice mide que tan alejados están los objetos con respecto al objeto promedio en función del tamaño del contenedor. Para el primer grupo de instancias, los valores obtenidos van desde el 27% al 33%, para el segundo desde el 7% al 8% y finalmente, para el tercer grupo desde el 19% al 24%. Podemos ver que las instancias más fáciles tienen una mayor desviación estándar normalizada que los grupos restantes.

El último atributo a ser analizado, es el *rango normalizado*, el cual mide la diferencia entre el mayor de los objetos y el menor. Este es otro atributo donde los tres grupos de instancias presentan diferencias evidentes. Para el grupo de instancias fáciles, este índice va desde el 89% al 98% de la capacidad del contenedor. Para el segundo grupo, este índice va del 22% al 25%. Por último, para el tercer grupo, los valores del rango van del 64% al 78%.

Fase 3: Visualización de las Instancias y Desempeño Algorítmico

Gráfica de los Pesos de los Objetos



a) Instancia N1c1w1_a. **b)** Instancia t60_00.

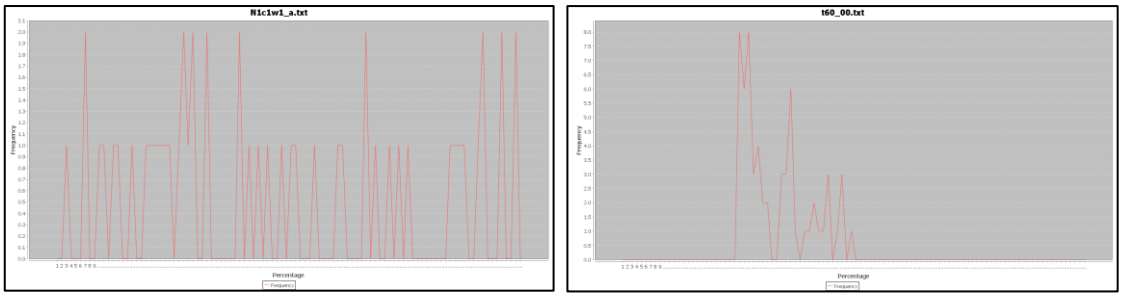
c) Instancia hBPP13.

Figura 6.21. Distribución inicial de pesos de instancias representativas.

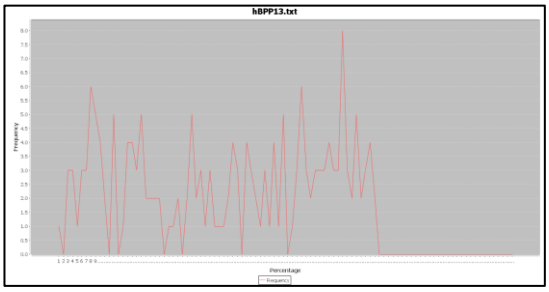
La primera característica visual a analizar es cómo vienen dados los objetos. La Figura 6.21 muestra las tres instancias representativas de cada conjunto de datos. Como se puede apreciar, las gráficas de los conjuntos de instancias fáciles y difíciles son muy parecidas entre sí (Figura 6.21a y Figura 6.21c).

Gráfica de Frecuencias

La Figura 6.22 muestra las gráficas de frecuencias de las instancias representativas de cada conjunto, como se puede ver, las instancias fáciles y difíciles se parecen mucho entre sí. El primer grupo de instancias, posee la característica que la amplitud de su frecuencia esta entre el 2% y el 100% de la capacidad del contenedor. En el segundo grupo las frecuencias oscilan entre el 23% y el 50%, y finalmente en el tercer grupo, las frecuencias aparecen desde el 1% y el 80%.



a) Instancia N1c1w1_a. **b)** Instancia t60_00.



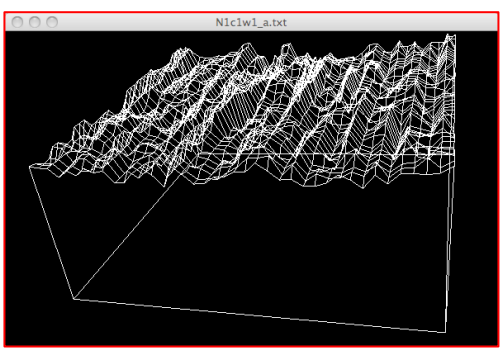
c) Instancia hBPP13.

Figura 6.22. Gráficas de frecuencias sobre las instancias representativas.

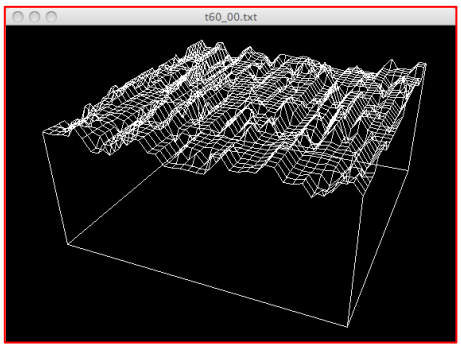
Visualización de la Superficie de Aptitudes.

En la Figura 6.23 se muestran tres gráficas de superficie de aptitudes; en estas gráficas se pueden apreciar más claramente las diferencias entre las superficies de cada uno de los conjuntos. El grupo de instancias fáciles, se caracteriza por tener una superficie de aptitudes muy rugosa, el grupo de las instancias de dificultad mediana tiende a una superficie rugosa,

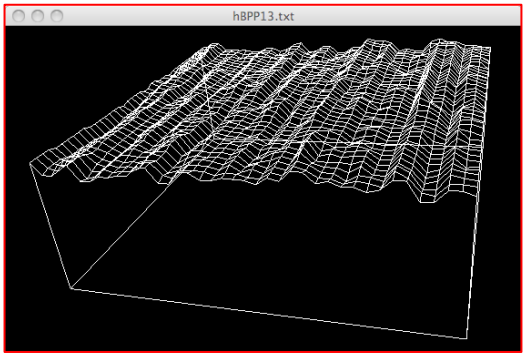
pero no tanto como las del primer grupo, mientras que en el grupo de instancias difíciles se observa una superficie más plana.



a) Instancia N1c1w1_a.



b) Instancia t60_00.



c) Instancia hBPP13.

Figura 6.23. Superficies de aptitudes de las instancias representativas.

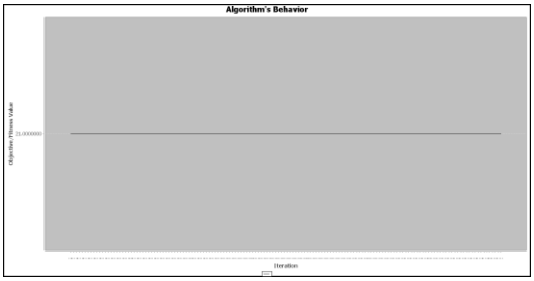
Visualización del Comportamiento Algorítmico

La Figura 6.24 visualiza el comportamiento del algoritmo durante su ejecución sobre dos instancias representativas de los conjuntos de dificultad mediana y alta dificultad, la razón es que el algoritmo HGGA-BP resuelve con heurísticas especializadas las instancias seleccionadas del conjunto de instancias fáciles, por lo que no tiene la necesidad de ejecutar el algoritmo genético, mientras que para las instancias de dificultad mediana y alta no logra resolverlas con las heurísticas y sí tiene que ejecutar el algoritmo genético.

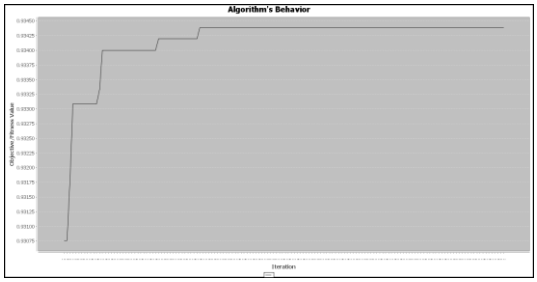
Las Figuras 6.24a y 6.24b muestran la gráfica del desempeño de HGGA-BP sobre la

instancia t60_00, la primera representa el mejor valor objetivo en cada generación, la segunda el mejor valor de aptitud en cada generación. Similarmente, las Figuras 6.24c y 6.24d muestran el desempeño obtenido de ejecutar el algoritmo HGGA-BP sobre la instancia hBPP13.

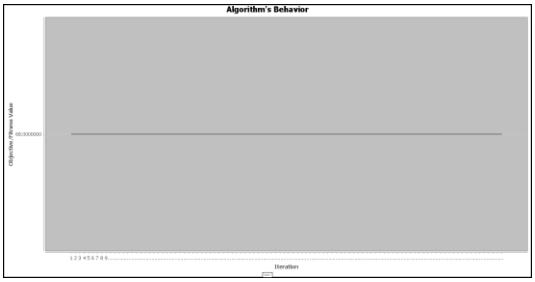
Al analizar las gráficas de ambas instancias podemos ver que el algoritmo mejora la calidad del valor de aptitud, sin embargo esta mejora no es suficiente para decrementar el número de contenedores utilizados en ningún caso.



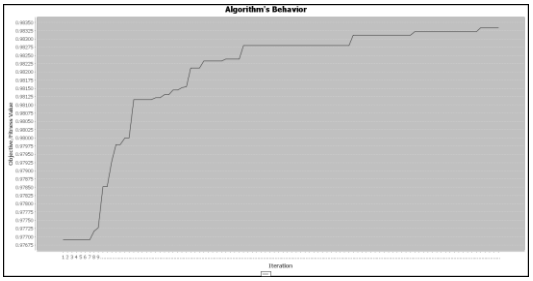
a) Valor Objetivo de la Instancia t60_00.



b) Valor de Aptitud de la Instancia t60_00.



c) Valor Objetivo de la Instancia hBPP13.



d) Valor de Aptitud de la Instancia hBPP13.

Figura 6.24. Comportamiento algorítmico con dos instancias.

Conclusiones del Experimento 3

Tras realizar el análisis estadístico y visual de las instancias de los tres conjuntos, se pudo demostrar que sí existen diferencias entre éstos, por lo cual se cumplió el objetivo del experimento.

Algunas de las características de las instancias del conjunto “DataSet1” (fáciles) son:

los valores del índice *mayor* siempre son muy grandes, en promedio del 96% de la capacidad del contenedor, su *media normalizada promedio* se localiza en el 48% de la capacidad del contenedor, su *desviación estándar normalizada promedio* es la más grande con un 29% de la capacidad del contenedor y finalmente cuenta con el valor *promedio del rango normalizado* más grande, con un 92.8% de la capacidad del contenedor.

Las características propias del conjunto "T" (dificultad mediana) son: el valor promedio del índice *menor* es el mayor de los tres grupos, mientras que su valor promedio del índice *mayor* es el menor. La *uniformidad* promedio es la menor de los tres grupos, con un 47% de la capacidad del contenedor. La *media normalizada* es muy característica de este conjunto, con un 33% de la capacidad del contenedor, es decir, la tercera parte exacta. Además este conjunto posee la menor *desviación estándar normalizada promedio*, con únicamente un 7% con respecto a la capacidad del contenedor. Finalmente, el valor promedio del índice *rango* es el más bajo en este conjunto, únicamente con un 24% de la capacidad del contenedor.

El último conjunto de instancias a analizar es el más difícil de resolver para el algoritmo HGGA-BP, sin embargo se encontraron pocos patrones en sus características, los cuales se mencionan a continuación: este grupo de instancias posee el valor promedio de *uniformidad* más alto de los tres grupos con un valor de 84%, por lo que la distribución en los pesos es muy similar, en promedio. Además las instancias de este grupo poseen las superficies de aptitudes *menos rugosas*.

6.2.4. Experimento 4: Visualización de Instancias para un Problema de Grafos

Objetivo

Visualizar un grupo de instancias de un problema de coloreo de grafos, con el fin de confirmar la hipótesis de que la estructura de las instancias es un factor importante en la resolución de éstas.

Procedimiento

Debido a que el único objetivo que se persigue en este experimento es el de confirmar una hipótesis, lo único que hay que hacer es introducir los datos a la herramienta VisTHAA para posteriormente visualizar la superficie de aptitudes en tres dimensiones.

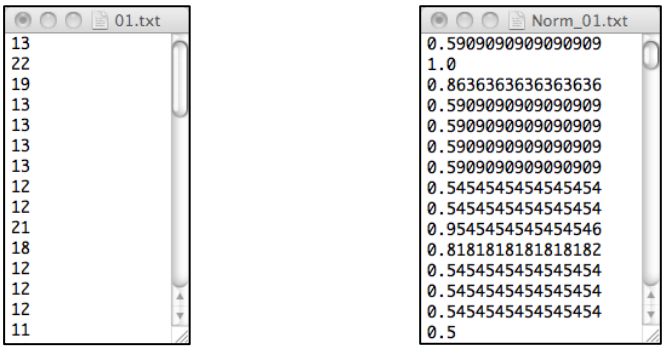
Entrada de Datos

En esta parte este experimento difiere de los anteriores, ya que no se trata del problema BPP, por esta razón no se puede utilizar la caminata aleatoria para obtener el conjunto de datos de valores de aptitud, sino que éstos deben ser proporcionados por el investigador.

Debido a que se desea visualizar un grupo de instancias, cada una de ellas debe ser introducida de forma independiente, es decir, cada conjunto de valores de aptitud que correspondan a una instancia debe estar en su propio archivo de texto.

Además, es preferible que los valores se encuentren normalizados entre cero y uno, esto es, expresados en puntos porcentuales. Por esta razón se sugirió al investigador una forma de normalización de datos rápida, buscar el mayor valor de su conjunto de datos y hacerlo el cien por ciento, los valores restantes se expresarían como una fracción de él. De esta manera se conservan las respectivas proporciones, simplemente se “reduce la escala”.

La Figura 6.25 muestra un ejemplo de una instancia con los valores objetivo originales (Figura 6.25a) y sus valores proporcionales en puntos porcentuales (Figura 6.25b).



a) Valores objetivo originales. **b)** Valores objetivo normalizados.

Figura 6.25. Instancia con sus valores objetivo originales y los normalizados.

Visualización de la Estructura de la Instancia

Para visualizar la estructura de la instancia se utilizó el método que visualiza la superficie de aptitudes en tres dimensiones. La figura 6.26 muestra la forma de introducir los datos desde un archivo, haciendo uso de un explorador de archivos.

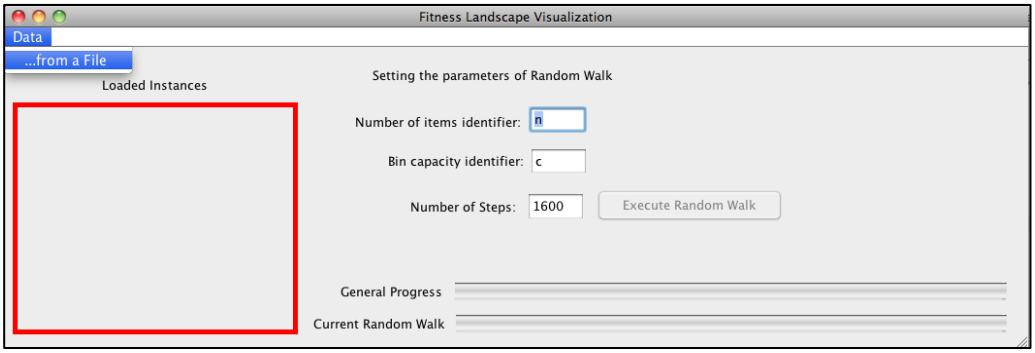
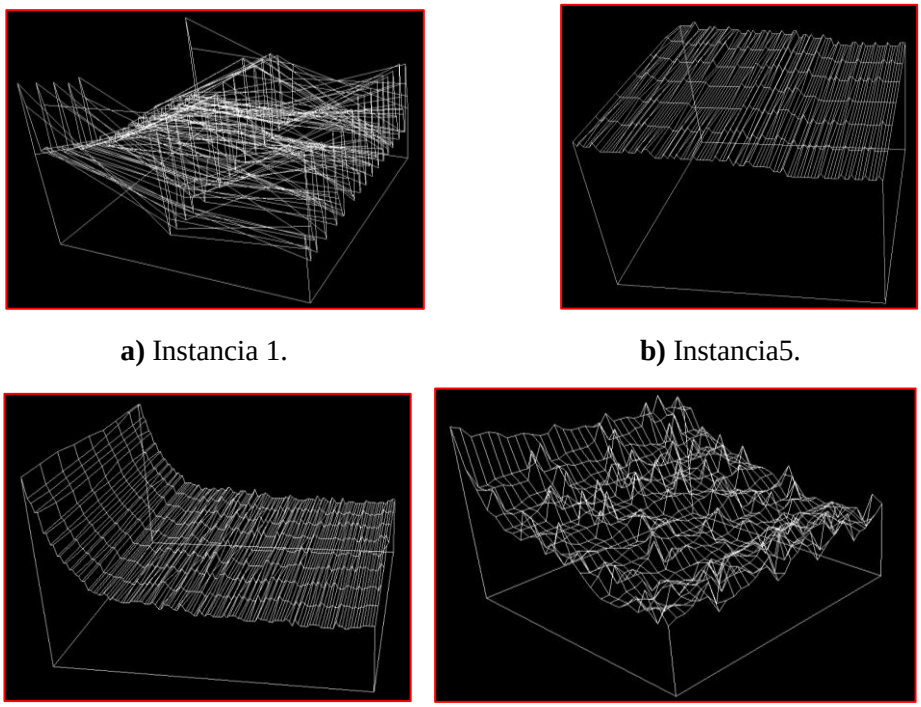


Figura 6.26. Introducción de los datos de aptitud a VisTHAA.

Una vez buscado el archivo de texto que representa la instancia a ser visualizada, VisTHAA automáticamente genera la superficie de aptitudes. La Figura 27 muestra la estructura de las instancias 1, 5 y 16.

Como se puede observar en la Figura 6.27, el experimento arrojó diversas formas en las superficies. La Figura 6.27a es muy particular, ya que ese tipo de superficie no se asemeja a otra de los experimentos anteriores. La Figura 6.27b revela una estructura muy

plana, lo cual implica que las soluciones son muy similares entre sí, este tipo de superficie ya se había observado en instancias de BPP en experimentos anteriores. La Figura 6.27c también es muy particular, ya que su forma se parece a la de una "cama", como se puede apreciar, los primeros valores de aptitud (aproximadamente el 30%) tienden a decrecer en forma muy suave, sin embargo, el resto de la superficie se mantiene plana, es decir ya no hay diferencias notables entre las soluciones. Finalmente la Figura 6.27b muestra una superficie muy rugosa, este tipo de estructura ya se había visto en experimentos anteriores.



a) Instancia 1. **b)** Instancia5.

c) Instancia 13. **d)** Instancia 16.

Figura 6.27. Visualización de las superficies de aptitudes de diferentes instancias.

Conclusiones del Experimento 4

Al analizar las gráficas proporcionadas por VisTHAA, el investigador pudo concluir que en efecto, la hipótesis que el investigador había planteado era correcta, ahora tenía un argumento visual que lo respaldaba, por lo tanto se cumplió el objetivo del experimento. VisTHAA ayudó a un investigador a confirmar una hipótesis con un argumento visual.

Cabe mencionar que el investigador había intentado dar esta misma explicación con anterioridad, haciendo uso de una herramienta del estado del arte llamada VIZ [Halim 06, Halim 07], la cual es simplemente una extensión de [Lau 05]. Sin embargo, la visualización de esta herramienta con respecto a la superficie de aptitud, resultó confusa, por lo cual el investigador no estaba seguro si la superficie que visualizaba era rugosa o no.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del presente proyecto de investigación, así como también el trabajo futuro que se pueda derivar.

7.1. Conclusiones

En este trabajo se presenta un enfoque innovador para auxiliar la mejora de estrategias de solución de problemas NP-Duros, haciendo uso principalmente del análisis estadístico y visual. Tras desarrollar el proyecto y realizar las experimentaciones, se puede concluir que se ***cumplió satisfactoriamente*** con el objetivo general, razón por la cual se considera que fue ***factible*** el desarrollo de este proyecto. Los objetivos particulares de este proyecto que fueron realizados se enlistan a continuación.

- **Caracterización el proceso del modelado del desempeño.** Este objetivo se cumplió de dos maneras: estadística y visual. La primera forma fue con la implementación de índices de la literatura especializados [Pérez 07, Quiroz 09] en todo el proceso de optimización. La segunda fue a través del diseño de estrategias visuales para cada paso del proceso (instancias del problema y desempeño algorítmico).

- **Visualización del desempeño del algoritmo.** Al desarrollar un módulo que permite al investigador visualizar el valor de aptitud con respecto al tiempo, se cumplió este objetivo.

- **Caracterización del espacio de soluciones a través del análisis de la superficie de aptitudes.** Para lograr este objetivo, se diseñó e implementó un método que permite visualizar la superficie de aptitudes en tres dimensiones, además de incorporar a VisTHAA índices especializados en dicho tema [Pérez 07].

- **Comparación del desempeño de dos algoritmos mediante la aplicación de la prueba de Wilcoxon.** Este objetivo se logró al desarrollar un módulo que permite realizar el análisis comparativo a través de la prueba de hipótesis estadística de Wilcoxon.

- **Diseño e implementación de una metodología para incorporar nuevos índices a partir de los existentes que permitan la evaluación visual de la información del algoritmo.** Este objetivo se logró gracias al algoritmo *shunting yard* [Dijkstra 61] y al desarrollo de tres módulos en VisTHAA. El primero es un módulo que permite ingresar el índice en notación infija y almacenarlo en VisTHAA. El segundo es un módulo llamado "*calculadora*", el cual permite obtener valores rápidos acerca de una característica de una instancia en particular. Finalmente, se desarrolló un módulo llamado "*matriz de características*", el cual permite generar una matriz de datos, cuyas unidades experimentales son instancias de BPP y cuyos atributos pueden ser índices de propósito general, de propósito específico o índices propios previamente introducidos.

- **Diseñar e implementar una método para el pre-procesamiento de los datos provenientes de las instancias.** Este objetivo fue alcanzado con éxito al crear un metalenguaje que permite el ingreso de instancias de cualquier problema y con cualquier formato, simplemente realizando una descripción de éstas.

7.2. Aportaciones de la Investigación

Las principales contribuciones de la investigación, se dan proponiendo las siguientes estrategias de mejora.

- El ***método de pre-procesamiento de datos de entrada***, que permite la introducción de

instancias con cualquier formato proporciona a VisTHAA, proporciona la facilidad de que el investigador pueda introducir los datos de sus instancias tal y como los utiliza, característica que no está presente en ninguna de las herramientas analizadas en el estado del arte. Este método ***minimiza el riesgo*** de cometer errores al momento de realizar la adecuación de la instancia. Además, proporciona a VisTHAA un metalenguaje universal, independiente del problema a resolver, de sus características, de su complejidad y del algoritmo solucionador.

- El ***método para el ingreso de nuevos índices*** permite al investigador introducir sus propios índices a VisTHAA, lo cual es innovador, ya que de todas las herramientas revisadas ninguna posee esta característica. Además, la matriz de características es una de las aportaciones más importantes del presente proyecto de investigación, ya que le da la posibilidad al investigador de utilizar sus propios índices sobre un conjunto de instancias para posteriormente utilizar esos resultados y obtener un mayor conocimiento de su problema de optimización.

- El ***método para visualizar la superficie de aptitudes en tres dimensiones*** es una de las mayores aportaciones de este trabajo, ya que es extremadamente versátil y fácil de implementar, sin embargo el conocimiento que se puede extraer de este método puede llegar a ser determinante para la mayor comprensión de la estructura de algunas instancias y así desarrollar estrategias que puedan resolver esas instancia de una manera más eficiente y/o eficaz.

- El ***método para la afinación de parámetros*** utilizando las competencias de Hoeffding y algoritmos de carreras es otra aportación importante, ya que se desarrolló un método para obtener la mejor configuración de ***n*** cantidad de parámetros, cada uno de ellos con sus propios niveles.

7.3. Trabajo Futuro

Como trabajo futuro, se puede considerar lo siguiente:

- En el caso del método de introducción de datos, es deseable extender el metalenguaje para permitir el ingreso de los datos del desempeño parcial y del desempeño final del algoritmo.
- Después de la creación de la matriz de características, es deseable que VisTHAA pueda realizar el tratamiento de los datos con técnicas de análisis multivariado, y no que el investigador tenga que emplear otra herramienta para ello, i.e., SAS, minitab, R, entre otras.
- Implementar más pruebas de hipótesis estadísticas tanto paramétricas como no paramétricas, con el fin de hacer más robusta a la herramienta VisTHAA.
- Conectar a VisTHAA con herramientas de libre distribución y aceptadas por la comunidad científica, i.e., weka, R, TETRAD, entre otros.
- Desarrollar métodos para la visualización de un tipo de instancia específica, i.e., instancias que involucran grafos.
- Hacer que VisTHAA funcione de manera *on-line*, ya que esta primera versión de la herramienta funciona en modo *off-line*.

Anexo A

Estadísticos Implementados

En este anexo se detalla la manera en que deben ser calculados los estadísticos que fueron implementados en la herramienta VisTHAA, así como su definición formal.

A.1. Media

La *media aritmética* de un conjunto de *n* mediciones: $x_1, x_2, \dots, x_n$ es el promedio de las mediciones [MendenHall 97]. Es una medida de la tendencia central de los datos, para obtener su valor, es necesario seguir la Ecuación A.1.

$$\overline{X} = \frac{1}{N}\sum_{j=1}^{N} x_j \qquad \text{(A.1)}$$

donde:

X es el conjunto de datos.

x_i es el i-ésimo elemento del conjunto *X*.

N es la cardinalidad (número de elementos) del conjunto.

Dado el siguiente conjunto de datos ***A***, obtener la media de dicho conjunto.

A = {100, 98, 95, 86, 88, 99, 79, 82, 84, 100}

$$\overline{A} = \frac{1}{10}(100 + 98 + 95 + 86 + 88 + 99 + 79 + 82 + 84 + 60)$$

$$\overline{A} = \frac{1}{10}(911)$$

$$\backslash \quad \overline{A} = 91.1$$

A.2. Mediana

La mediana de un conjunto de *n* determinaciones $x_1, x_2, \ldots, x_n$ es el número de en medio cuando las determinaciones se acomodan en orden ascendente (o descendente); es decir, el valor de *x* en una posición tal que la mitad del área bajo el histograma de frecuencia relativa queda a su izquierda y la mitad del área queda a la derecha [MendenHall 97]. La Ecuación A.2 se debe seguir para obtener el valor de este estadístico.

$$m = \begin{cases} x_{[(n+1)/2]} & si\ n\ es\ par \\ \dfrac{x_{(n/2)} + x_{[(n/2)+1]}}{2} & si\ n\ es\ non \end{cases} \qquad \text{(A.2)}$$

Dado el mismo conjunto **A** del ejemplo anterior, obtener la mediana.

A = {100, 98, 95, 86, 88, 99, 79, 82, 84, 100}

Primero se debe ordenar el conjunto de datos, quedando de la siguiente manera:

A' = {79, 82, 84, 86, 88, 95, 98, 99, 100, 100}

Luego, como el número total de elementos es par (diez), se obtiene la media de los dos elementos centrales del conjunto ordenado ***A'***, es decir, se obtiene el promedio entre la quinta y la sexta posición, el resultado obtenido es la mediana del conjunto.

$$M_e(A) = \frac{1}{2}(88 + 95)$$

$$M_e(A) = \frac{1}{2}(183)$$

$$\therefore \quad M_e(A) = 91.5$$

A.3. Moda

La moda de un conjunto de *n* determinaciones $x_1, x_2, \ldots, x_n$ es el valor de *x* que ocurre con mayor frecuencia [MendenHall 97].

Dado el mismo conjunto ***A*** del ejemplo anterior, obtener la moda.

A = {100, 98, 95, 86, 88, 99, 79, 82, 84, 100}

Lo primero que se debe de hacer es obtener las frecuencias de cada elemento del conjunto, la Tabla A.1 muestra las frecuencias de aparición de cada uno de los elementos del conjunto.

Tabla A.1. Frecuencias de los elementos del conjunto **A**.

Elemento del conjunto	Frecuencia de aparición
100	2
98	1
95	1
86	1
88	1
99	1
79	1
82	1
84	1

Posteriormente, se debe seleccionar como la moda el elemento cuya ***frecuencia*** sea la

mayor del conjunto.

\ **moda(*A*)** = 100

A.4. Varianza

Es una medida de variabilidad de los datos, la varianza de una ***muestra*** de *n* determinaciones se calcula a través de la Ecuación A.3 [MendenHall 97].

$$s_X^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2 \qquad \text{(A.3)}$$

La varianza de la ***población*** se define en la Ecuación A.4 [MendenHall 97].

$$s_X^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2 \qquad \text{(A.4)}$$

Para una población finita con *n* determinaciones.
donde:
X es el conjunto de datos.
x_i es el i-ésimo elemento del conjunto *X*.
μ es la media del conjunto *X*.
N es la cardinalidad del conjunto *X*.

Dado el conjunto ***A*** del ejemplo anterior, obtener la varianza.

A = {100, 98, 95, 86, 88, 99, 79, 82, 84, 100}

Primero debemos obtener la media del conjunto, en este caso la media es *91.1*.

Luego aplicamos la fórmula para la obtención de la varianza (Ecuación A.4).

$$s_A^2 = \frac{1}{10 - 1}\left[(100 - 91.1)^2 + (98 - 91.1)^2 + (95 - 91.1)^2 + (86 - 91.1)^2 + (88 - 91.1)^2 + (99 - 91.1)^2 + (79 - 91.1)^2 + (82 - 91.1)^2 + (84 - 91.1)^2 + (100 - 91.1)^2\right]$$

$$s_A^2 = \frac{1}{9}\left[(8.9)^2 + (6.9)^2 + (3.9)^2 + (-5.1)^2 + (-3.1)^2 + (7.9)^2 + (-12.1)^2 + (-9.1)^2 + (-7.1)^2 + (8.9)^2\right]$$

$$s_A^2 = \frac{1}{9}(79.21 + 47.61 + 15.21 + 26.01 + 9.61 + 62.41 + 146.41 + 82.81 + 50.41 + 79.21)$$

$$s_A^2 = \frac{1}{9}(598.9)$$

$$\therefore s_A^2 = 66.54$$

A.5. Desviación estándar

Es una medida de dispersión, La desviación estándar de una muestra de n determinaciones es igual a la raíz cuadrada de la varianza [MendenHall 97]. La fórmula para obtenerla se detalla en la Ecuación A.5.

$$s_X = \sqrt{\frac{1}{N - 1}\sum_{i=1}^{N}(x_i - \mu)^2} \qquad \text{(A.5)}$$

donde:

X es el conjunto de datos.

x_i es el i-ésimo elemento del conjunto X.

μ es la media del conjunto X.

N es la cardinalidad del conjunto X.

Dado el conjunto ***A*** del ejemplo anterior, obtener su desviación estándar

A = {100, 98, 95, 86, 88, 99, 79, 82, 84, 100}

Primero debemos obtener la media del conjunto, en este caso la media es 91.1. Luego calculamos la varianza, para este conjunto, la varianza es 66.54. Finalmente, extraemos la raíz cuadrada de la varianza, el resultado es la desviación estándar.

$s_A = \sqrt{66.54}$

$\therefore \; s_A = 8.16$

A.6. Búsqueda del Máximo

Dado un conjunto de datos, se debe elegir el mayor valor de sus elementos, i.e., del conjunto *A* del ejemplo anterior, calcular cual es el máximo del conjunto.

$\boldsymbol{A} = \{100, 98, 95, 86, 88, 99, 79, 82, 84, 100\}$

$\therefore$ **max(*A*)** = 100

A.7. Búsqueda del Mínimo

Dado un conjunto de datos, se debe elegir el menor valor entre ellos, i.e., Dado el conjunto A del ejemplo anterior, calcular cual es el mínimo del conjunto.

$\boldsymbol{A} = \{100, 98, 95, 86, 88, 99, 79, 82, 84, 100\}$

$\therefore$ **min(*A*)** = 79

Anexo B

Índices de Caracterización

En este anexo se detalla la manera en que deben ser calculados los índices para lograr caracterizar correctamente tanto el problema BPP como el desempeño final de un algoritmo.

B.1. Índices para la Caracterización del Problema

En esta sección se ejemplificará la forma en cómo deben de ser calculados los índices que propone Quiroz en [Quiroz 09] para caracterizar una instancia de BPP.

Menor

Representa el peso del objeto más pequeño, en relación al tamaño del contenedor y permite ubicar el inicio de la distribución de pesos. El índice es definido por medio de la Ecuación B.1.

$$Menor = \frac{\min(W)}{c} \tag{B.1}$$

donde:
W es el arreglo de pesos de objetos.
c es la capacidad del contenedor.
$\min(W)$ es una función que devuelve el menor de los pesos.

Supóngase que se desea calcular el índice *menor* con los siguientes datos:
$\boldsymbol{W}$ = {10, 12, 20, 15, 17, 21, 11, 22, 19, 13} y $\boldsymbol{c}$ = 50

$$Menor = \frac{\min(W)}{c}$$
$$Menor = \frac{10)}{50}$$
$$\therefore Menor = 0.2$$

El resultado obtenido, *Menor* = 0.2, indica que el objeto más pequeño del conjunto de pesos, ocupa el 20% de la capacidad del contenedor.

Mayor

Cuantifica el peso del objeto más grande, en relación al tamaño del contenedor y ubica el final de la distribución de pesos. La medida es calculada con la Ecuación B.2.

$$Mayor = \frac{\max(W)}{c} \quad \text{(B.2)}$$

donde:
W es el arreglo de pesos de objetos.
c es la capacidad del contenedor.
max (W) es una función que devuelve el mayor de los pesos.

Supóngase que se desea calcular el índice *mayor* con los siguientes datos:
$\boldsymbol{W}$ = {10, 12, 20, 15, 17, 21, 11, 22, 19, 13} y $\boldsymbol{c}$ = 50

$$Mayor = \frac{\max(W)}{c}$$

$$Mayor = \frac{22}{50}$$

$$\therefore Mayor = 0.44$$

El resultado obtenido, *Mayor* = 0.44, indica que el objeto más grande del conjunto de pesos *W*, ocupa el 44% de la capacidad del contenedor.

Multiplicidad

Caracteriza el número promedio de repeticiones de cada peso de objeto y ayuda a identificar tendencias o picos en la distribución de frecuencias de pesos. La Ecuación B.3 define este índice,

$$Multiplicidad = \frac{\sum_{i=1}^{\hat{n}} m_i}{n} \tag{B.3}$$

donde:
m_i es el número de objetos con peso s_i, donde *S* es un subconjunto de *W* y guarda los $\hat{n}$ diferentes pesos sin repeticiones $(1 \leq \hat{n} \leq n)$.
$\hat{n}$ es la cardinalidad del conjunto *S*.

Supóngase que el investigador quiere conocer la *multiplicidad* de una instancia de BPP, los siguientes datos representan la instancia antes mencionada.

W = {10, 9, 10, 8, 15, 13, 20, 9, 7}

Lo primero que se debe de hacer es obtener el subconjunto ***S***, el cual estará formado por los elementos de ***W*** sin que se repita alguno de ellos.

En este caso vemos que el objeto de peso 10 se repite en la primera y tercera posición

y que el objeto de peso 9 se repite en la segunda y octava posición. En el subconjunto ***S***, sólo se incluirá un objeto de peso 10 y un objeto de peso 9, quedando ***S*** de la siguiente manera:

S = {10, 9, 8, 15,13,20,7}

Después de obtener *S*, es necesario obtener el conjunto ***M***, el cual indica la frecuencia de aparición de los elementos de *S* en el conjunto original *W*.

En este ejemplo, el ***primer elemento*** de *S* es 10, el cual se repite dos veces en *W* (primera y tercera posición), por lo tanto el conjunto *M* en su ***primer elemento*** contendrá el valor de dos (que son las veces que se repite el 10 en *W*). Este proceso se realiza sucesivamente quedando ***M*** de la siguiente manera:

M = {2, 2, 1, 1, 1, 1, 1}

Posteriormente, es necesario calcular $\hat{n}$, cuyo valor viene dado por la cardinalidad del conjunto *M*.

$$\hat{n} = |M|$$
$$\hat{n} = 7$$

Finalmente, lo único que falta por realizar es la aplicación de la Ecuación B.3 para obtener el valor de la *multiplicidad*.

$$Multiplicidad = \frac{2 + 2 + 1 + 1 + 1 + 1 + 1}{7}$$
$$Multiplicidad = \frac{9}{7}$$
$$\therefore Multiplicidad \approx 1.2857$$

MaxRepe

Representa la frecuencia máxima con la que un peso se repite en el conjunto de objetos. El índice es definido por la Ecuación B.4.

$$MaxRepe = \max(M) \tag{B.4}$$

donde:
M Es el conjunto que guarda la frecuencia de aparición de cada diferente peso.
max(*M*) es una función que devuelve el valor máximo del conjunto *M*.

Dado el mismo conjunto de datos del ejemplo anterior (cálculo de la multiplicidad), obtener el índice *MaxRepe*.
W = {10, 9, 10, 8, 15, 13, 20, 9, 7}
S = {10, 9, 8, 15,13,20,7}
M = {2, 2, 1, 1, 1, 1, 1}

MaxRepe = max(*M*)
\ ***MaxRepe*** = 2

Uniformidad

Mide el grado de uniformidad de la distribución de los pesos de los objetos, a partir de la división del rango de datos en cuatro segmentos de igual magnitud y las diferencias entre el número esperado y el número real de objetos en cada subrango.

Sean:
n = número de objetos.
$W = \{w_j | w_j < w_{j+1}\}, \forall 1 \leq i \leq n$, arreglo de pesos de objetos en orden creciente.

$R = \max(W) - \min(W)$, amplitud del rango de distribución de pesos de los objetos.

$r_i = \min(W) + \frac{iR}{4}, \forall 0 \leq i \leq 4$, marcador i-ésimo de división en subrangos.

El conjunto de objetos contenidos en cada subrango *j* es:

$B_j = \{w \in W | r_{j-1} < w \leq r_j\}, \forall 1 \leq j \leq 4$

La Ecuación B.5 define formalmente este índice.

$$Uniformidad = 1 - \frac{\sum_{j=1}^{4} \left| \left(\frac{n}{4} - |B_j| \right) \right|}{n} \quad \textbf{(B.5)}$$

Un valor de uniformidad cercano a uno, representa una distribución uniforme, y mientras más alejado se encuentre este valor de la unidad, se observarán mayores diferencias en la frecuencia de aparición de cada uno de los pesos de los objetos.

Dado el siguiente conjunto de datos, obtener la ***uniformidad*** de la instancia.

W = {15, 10, 7, 9, 20, 10, 13, 8, 9}

n = 9

Primero es necesario ordenar ascendentemente el conjunto ***W***.

W = {7, 8, 9, 9, 10, 10, 13, 15, 20}

Posteriormente, debemos calcular la amplitud del rango de la distribución de los pesos.

$R = \max(W) - \min(W)$

$R = 20 - 7$

\ $R = 13$

Una vez obtenido el rango, se calculan los cinco marcadores de división en

subrangos.

$$\hat{R} = \{r_0, r_1, r_2, r_3, r_4\}$$

Para calcular el conjunto de marcadores, se debe calcular cada marcador de manera individual:

Calculando $\boldsymbol{r_0}$:

$$r_0 = \min(W) + \frac{(0)(R)}{4}$$
$$r_0 = 7 + \frac{0(13)}{4}$$
$$\therefore \quad r_0 = 7$$

Calculando $\boldsymbol{r_1}$:

$$r_1 = \min(W) + \frac{(1)(R)}{4}$$
$$r_1 = 7 + \frac{13}{4}$$
$$\therefore \quad r_1 = 10.25$$

Calculando $\boldsymbol{r_2}$:

$$r_2 = \min(W) + \frac{(2)(R)}{4}$$
$$r_2 = 7 + \frac{2(13)}{4}$$
$$r_2 = 7 + \frac{13}{2}$$
$$\therefore \quad r_2 = 13.5$$

Calculando $\boldsymbol{r_3}$:

$$r_3 = \min(W) + \frac{(3)(R)}{4}$$
$$r_3 = 7 + \frac{3(13)}{4}$$
$$r_3 = 7 + \frac{39}{4}$$
$$\therefore \; r_3 = 16.75$$

Calculando $\boldsymbol{r_4}$:

$$r_4 = \min(W) + \frac{(4)(R)}{4}$$
$$r_4 = 7 + 13$$
$$\therefore \; r_4 = 20$$

Una vez calculados los marcadores, el conjunto $\hat{R}$ queda de la siguiente manera:

$$\hat{R} = \{7, 10.25, 13.5, 16.75, 20\}$$

Luego, es necesario calcular el conjunto de objetos contenidos en cada *j-ésimo* subrango, utilizando la siguiente expresión:

$$B_j = \{w \in W | r_{j-1} < w \leq r_j\}, \forall 1 \leq j \leq 4$$

El primer elemento del conjunto $\boldsymbol{W}$ se introduce directamente en el primer subrango, es decir:

$$7 \in B_1$$

Después, para conocer cual objeto pertenece a cual subrango, se compara objeto por objeto, con cada marcador de subrango, para poder conocer en cual intervalo está contenido y poder saber a cual subrango pertenece, es decir:

$7 < 8 \leq 10.25 \Rightarrow 8 \in B_1$
$7 < 9 \leq 10.25 \Rightarrow 9 \in B_1$
$7 < 10 \leq 10.25 \Rightarrow 10 \in B_1$
$10.25 < 13 \leq 13.5 \Rightarrow 13 \in B_2$
$13.5 < 15 \leq 16.75 \Rightarrow 15 \in B_3$
$16.75 < 20 \leq 20 \Rightarrow 20 \in B_4$

$\therefore$
$B_1 = \{7, 8, 9, 9, 10, 10\}$
$B_2 = \{13\}$
$B_3 = \{15\}$
$B_4 = \{20\}$

Hasta este momento, únicamente se han realizado los preparativos para obtener el índice de uniformidad, lo que continúa es precisamente calcular el valor del índice a través de la Ecuación B.5.

$$Uniformidad = 1 - \frac{\left|\left(\frac{9}{4} - 6\right)\right| + \left|\left(\frac{9}{4} - 1\right)\right| + \left|\left(\frac{9}{4} - 1\right)\right| + \left|\left(\frac{9}{4} - 1\right)\right|}{9}$$

$$Uniformidad = 1 - \frac{|2.25 - 6| + |2.25 - 1| + |2.25 - 1| + |2.25 - 1|}{9}$$

$$Uniformidad = 1 - \frac{|-3.75| + 1.25 + 1.25 + 1.25}{9}$$

$$Uniformidad = 1 - \frac{3.75 + 3.75}{9}$$

$$Uniformidad = 1 - \frac{7.5}{9}$$

$$Uniformidad = 1 - 0.8333$$

$$\therefore Uniformidad \approx 0.1666$$

El resultado del índice es un ***valor muy cercano al cero***, por lo que representa a una ***distribución poco uniforme***.

B.2. Índices para la Caracterización del Desempeño Final

En esta sección se ejemplificará la forma en cómo deben de ser calculados los índices que propone Quiroz en [Quiroz 09] para caracterizar el desempeño final del algoritmo.

Max_MejorF_{BPP}

Registra la aptitud de la mejor solución obtenida, para una instancia de BPP, en r ejecuciones del algoritmo. La Ecuación B.6 define formalmente este índice.

$$Max\_MejorF_{BPP} = \max(M) \tag{B.6}$$

donde:
M es un conjunto que incluye las aptitudes de las r soluciones obtenidas en las r ejecuciones del algoritmo.
$\max(M)$ es una función que devuelve el mayor valor del conjunto M.

El valor de aptitud difiere con el valor objetivo en el problema BPP, ya que mientras el valor objetivo mide la cantidad de contenedores utilizados por una solución al problema, el valor de aptitud se utiliza para discriminar entre soluciones aparentemente iguales, i.e., dos soluciones de BPP pueden utilizar el mismo número de contenedores, sin embargo el porcentaje de llenado de cada uno de ellos puede variar entre las soluciones.

El *valor de aptitud* es el ***porcentaje promedio de llenado*** de los contenedores de una solución, y no se debe de confundir con el valor objetivo, ya que mientras el valor objetivo se trata de ***minimizar***, el valor de aptitud, por otra parte, se trata de ***maximizar***.

Dado el siguiente conjunto de valores de aptitud, encontrados tras ejecutar r=10 veces el algoritmo sobre la misma instancia de BPP, obtener Max_MejorF_{BPP}.

M = {0.98876, 0.95667, 0.94556, 0.87887, 0.90123, 0.95666, 0.99881, 0.92345, 0.93112,

0.89999}

$Max\_MejorF_{BPP} = \max(M)$
$\therefore Max\_MejorF_{BPP} = 0.99881$

Min_MejorF$_{BPP}$

Representa la aptitud de la peor solución obtenida, para una misma instancia de BPP, en r ejecuciones del algoritmo. Para calcular su valor, se debe emplear la Ecuación B.7.

$$Min\_MejorF_{BPP} = \min(M) \quad \text{(B.7)}$$

donde:
M es un conjunto que incluye las aptitudes de las r soluciones obtenidas en las r ejecuciones del algoritmo.
min(M) es una función que devuelve el menor valor del conjunto M.

Dado el siguiente conjunto de valores de aptitud, encontrados tras ejecutar *r*=10 veces el algoritmo sobre la misma instancia de BPP, obtener *Min_MejorF*$_{BPP}$.

M = {0.98876, 0.95667, 0.94556, 0.87887, 0.90123, 0.95666, 0.99881, 0.92345, 0.93112, 0.89999}

$Min\_MejorF_{BPP} = \min(M)$
$\therefore Min\_MejorF_{BPP} = 0.87887$

Prom_MejorF$_{BPP}$

Calcula el promedio de las aptitudes de las soluciones obtenidas, para el mismo caso de prueba, en r ejecuciones del algoritmo. La Ecuación B.8 define este índice.

$$Prom\_MejorF_{BPP} = \frac{\sum_{i=1}^{r} ma_i}{r} \quad \text{(B.8)}$$

donde:
ma_i es la aptitud de la solución alcanzada en la i-ésima ejecución del algoritmo.
r es el número de veces que se replicó la ejecución del algoritmo sobre una instancia en particular.

Dado el siguiente conjunto de valores de aptitud, encontrados tras ejecutar *r*=10 veces el algoritmo sobre la misma instancia de BPP, obtener $Prom\_MejorF_{BPP}$.

M = {0.98876, 0.95667, 0.94556, 0.87887, 0.90123, 0.95666, 0.99881, 0.92345, 0.93112, 0.89999}
r = 10

$$Prom\_MejorF_{BPP} = \frac{(0.98876+0.95667+0.94556+0.87887+0.90123+0.95666+0.99881+0.92345+0.93112+0.89999)}{10}$$

$$Prom\_MejorF_{BPP} = \frac{9.38112}{10}$$

$$\therefore \; Prom\_MejorF_{BPP} = 0.938112$$

Radio_teórico

Es otra medida de calidad, representa la razón de la mejor solución encontrada por el algoritmo (Z_{enc}) y la solución óptima (Z_{opt}). Cuando la solución óptima no es conocida, Z_{opt} toma el valor del mejor límite inferior del número de contenedores reportado en la literatura [Alvim 04]. La Ecuación B.9 define este índice formalmente

$$Radio\_teorico = \frac{Z_{enc}}{Z_{opt}} \quad \text{(B.9)}$$

Supongamos que se ejecutó el algoritmo que resuelve el BPP, encontrando un determinado valor objetivo, además, el valor óptimo de esa instancia es conocido, obtener el radio teórico. Los datos son los siguientes:

$Z_{enc} = 29$
$Z_{opt} = 25$

$$Radio_{teorico} = \frac{Z_{enc}}{Z_{opt}}$$
$$Radio\_teorico = \frac{29}{25}$$
$$\therefore Radio\_teorico = 1.16$$

Desv_Z_{enc}

Cuantifica la variabilidad de las soluciones encontradas en diferentes ejecuciones del algoritmo, para acomodar la misma instancia de BPP. La Ecuación B.10 define este índice.

$$Desv\_Z_{enc} = \sqrt{\frac{\sum_{i=1}^{r}(\overline{z} - z_i)^2}{r}} \quad \text{(B.10)}$$

donde:
z_i es el número de contenedores que conforman la solución alcanzada en la i-ésima ejecución del algoritmo.
$\overline{z}$ es el promedio de las soluciones encontradas en las r ejecuciones.
r es el número de veces que se replicó la ejecución del algoritmo sobre una instancia en particular.

Tiempo

Cuantifica el promedio del tiempo, expresado en segundos, que el algoritmo tarda en encontrar una solución, para una instancia de BPP. La Ecuación B.11 define este índice.

$$Tiempo = \frac{\sum_{i=1}^{r} t_i}{r} \tag{B.11}$$

donde:

t_i cuantifica el tiempo utilizado en la i-ésima ejecución del algoritmo.

r es el número de veces que se replicó la ejecución del algoritmo sobre una instancia en particular.

Supongamos que una instancia se resolvió *r*=5 veces, el conjunto ***T*** representa los tiempos en segundos que cada ejecución del algoritmo se tardó en encontrar la solución. Obtener el *Tiempo*.

T = {10.40, 11.22, 13.22, 9.87, 13.33}

r = 5

$$Tiempo = \frac{\sum_{i=1}^{r} t_i}{r}$$

$$Tiempo = \frac{(10.40 + 11.22 + 13.22 + 9.87 + 13.33)}{5}$$

$$Tiempo = \frac{58.04}{5}$$

$$\therefore Tiempo = 11.608$$

Generación

Representa la generación promedio en la cual el algoritmo encuentra la mejor solución de una instancia. Este índice es definido mediante la Ecuación B.12.

$$Generacion = \frac{\sum_{i=1}^{r} gen_i}{r} \quad \text{(B.12)}$$

donde:

gen_i es la generación en la que se encontró la mejor solución de la i-ésima ejecución del algoritmo.

r es el número de veces que se replicó la ejecución del algoritmo sobre una instancia en particular.

Supongamos que una instancia se resolvió *r*=5 veces, el conjunto ***G*** representa la generación en la que cada ejecución del algoritmo encontró la solución. Obtener el índice *Generación*.

G = {89, 86, 93, 91, 88}

r = 5

$$Generacion = \frac{\sum_{i=1}^{r} gen_i}{r}$$

$$Generacion = \frac{89 + 86 + 93 + 91 + 88}{5}$$

$$Generacion = \frac{447}{5}$$

$$Generacion = 89.4$$

Objetos_Contenedor

Calcula el número promedio de objetos que fueron almacenados en cada contenedor de la solución alcanzada por el algoritmo. La medida puede ser calculada a partir de la Ecuación B.13.

$$Objetos\_Contenedor = \frac{n}{Z_{enc}} \quad \text{(B.13)}$$

donde:

Z_{enc} representa el número de contenedores de la solución final obtenida.

n es el número de objetos de la instancia de BPP.

Dados los siguientes datos, obtener el valor del índice *Objetos_Contenedor*.

n = 100

Z_{enc} = 50

$$Objetos\_Contenedor = \frac{n}{Z_{enc}}$$

$$Objetos\_Contenedor = \frac{100}{50}$$

$$\therefore Objetos\_Contenedor = 2$$

Anexo C

Índices de Rugosidad

En este anexo se detalla la forma en cómo deben de ser correctamente calculados los índices de rugosidad del trabajo de Pérez [Pérez 07]

C.1. Coeficiente de Autocorrelación

Weinberg en [Weinberg 90] propuso este índice para medir la rugosidad de una superficie de aptitudes. Se considera que existe un fuerte enlace de este concepto y la dureza de un problema de optimización para un algoritmo basado en búsqueda local, es decir, que intuitivamente el número de óptimos locales depende del enlace (correlación) entre el costo de una solución y el costo de sus vecinos [Pérez 07].

Para su cálculo, se debe utilizar la Ecuación C.1. De donde si $|r_k| \gg 1$, entonces existe mucha correlación entre los k puntos medidos, mientras que si $|r_k| \gg 0$ existe poca correlación entre los mismos. El valor de k más común es uno, ya que permite encontrar las correlaciones entre cada par de aptitudes consecutivas [Pérez 07].

$$r_k = \frac{\sum_{i=1}^{N-K} \left(f_i - \bar{f}\right)\left(f_{i+k} - \bar{f}\right)}{\sum_{i=1}^{N} \left(f_i - \bar{f}\right)^2}, n > k \qquad \text{(C.1)}$$

donde:

N es el número de total de individuos (soluciones) en la superficie de aptitudes.
K es un valor de la separación entre individuos.
f_i es el i-ésimo individuo en la superficie.
f_{i+k} es el individuo cuya distancia de separación del i-ésimo individuo es k.
$\bar{f}$ es el valor de aptitud promedio de los individuos en la superficie de aptitudes.

Un valor alto de la función de autocorrelación indica que las diferencias en las aptitudes es en promedio muy similar, implicando una superficie menos rugosa. En caso contrario, un valor cercano a cero indica que los valores de aptitud son casi independientes, y por tanto la superficie es muy rugosa y tentativamente más difícil.

Supongamos que un algoritmo Random Walk realizó una caminata de únicamente 5 pasos (para una ejemplificación sencilla) sobre una instancia de BPP, dando como resultado el siguiente conjunto ***F*** con los valores de aptitud de la caminata.

$\boldsymbol{F} = \{0.95, 0.94, 0.96, 0.89, 1\}$

Lo primero que debemos hacer es obtener la media del conjunto ***F***.

$$\bar{f} = \frac{0.95+0.94+0.96+0.89+1}{5}$$
$$\bar{f} = \frac{4.74}{5}$$
$$\bar{f} = 0.948$$

Una vez obtenida la media, se aplica la Ecuación C.1 con el valor de *k*=1, para obtener el valor del coeficiente de autocorrelación entre soluciones vecinas.

$$r_k = \frac{(0.95 - 0.948)(0.94 - 0.948) + (0.94 - 0.948)(0.96 - 0.948) + (0.96 - 0.948)(0.89 - 0.948) + (0.89 - 0.948)(1 - 0.948)}{(0.95 - 0.948)^2 + (0.94 - 0.948)^2 + (0.96 - 0.948)^2 + (0.89 - 0.948)^2 + (1 - 0.948)^2}$$

$$r_k = \frac{(0.002)(-0.008) + (-0.008)(0.012) + (0.012)(-0.058) + (-0.058)(0.052)}{(0.002)^2 + (-0.008)^2 + (0.012)^2 + (-0.058)^2 + (0.052)^2}$$

$$r_k = \frac{(-0.000016) + (-0.000096) + (-0.000696) + (-0.003016)}{(0.000004) + (0.000064) + (0.000144) + (0.003364) + (0.002704)}$$

$$r_k = \frac{-0.004097}{0.00628}$$

$$\therefore r_k \approx -0.6524$$

Una vez obtenido el valor, notamos que es un valor negativo y más cercano a la unidad que al cero, lo que significa que la superficie es mayormente plana (menos rugosa).

C.2. Longitud de Autocorrelación

Según Pérez [Pérez 07], Weinberg también propone este índice, el cual nos indica cuál es el mayor valor de distancia a la que el conjunto de soluciones se vuelve no correlacionado. La longitud de autocorrelación $\|r_k\|$ entre un conjunto de soluciones, separadas a una distancia k, se calcula mediante la Ecuación C.2.

$$\|r_k\| = \frac{1}{1 - r_k} \qquad \text{(C.2)}$$

donde:

r_k es el coeficiente de autocorrelación (obtenido con la Ecuación C.1).

Un valor alto de la longitud de autocorrelación nos indica que la superficie es muy plana, mientras que un valor pequeño de ésta sugiere una superficie más rugosa.

Visualización de la Rugosidad del Espacio de Búsqueda

Los índices de información son un enfoque que se basa en la teoría clásica de la información, y tiene como objetivo cuantificar el grado de rugosidad o llanura de una

superficie de aptitudes [Pérez 07]. Los índices que Pérez propone evalúan la entropía generada en la trayectoria que siguen las soluciones encontradas utilizando un operador de búsqueda.

Basándose en este enfoque Pérez propone los índices: contenido de información, contenido de la información parcial e información de la densidad de valles. Para los fines de este trabajo, únicamente se utilizó el primero.

Retomando el ejemplo anterior y sus resultados, obtener la longitud de autocorrelación.

$$\|r_k\| = \frac{1}{1 - r_k}$$

$$\|r_k\| = \frac{1}{1 - (-0.6524)}$$

$$\therefore \|r_k\| \approx 0.6052$$

C.3. Contenido de Información

El contenido de información caracteriza el grado de rugosidad con respecto a las áreas planas de la superficie de aptitudes. El grado de llanura detectado depende de un parámetro de sensibilidad ε, que es un valor real en el rango [0, L], donde L representa la máxima diferencia en la secuencia $\{ f_i \}_{i=1}^{n}$.

Para calcular este índice, la secuencia de los valores de aptitud $f_1, f_2, ..., f_n$ se codifica en una cadena $S(\varepsilon) \in \{\bar{1}, 0, 1\}$. Para un valor particular de ε la cadena se genera utilizando la función $s_i = \gamma_{ft}(i, \varepsilon)$, descrita en la Ecuación C.3.

$$y_{ft}(i,\varepsilon) = \begin{cases} \bar{1} & si \quad f_i - f_{i-1} < -\varepsilon \\ 0 & si \quad |f_i - f_{i-1}| \leq \varepsilon \\ 1 & si \quad f_i - f_{i-1} > \varepsilon \end{cases} \tag{C.3}$$

donde:

f_i es el i-ésimo valor de aptitud en la superficie de aptitudes.

f_{i-1} es un valor de aptitud anterior al *i-ésimo* valor en la superficie de aptitudes

Una vez codificada la cadena ternaria S, la Ecuación C.4 presenta la expresión para calcular el contenido de información

$$H(\varepsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]} \tag{C.4}$$

donde:

$P_{[pq]}$ son las probabilidades de los sub-bloques de aptitudes pq encontrados, donde $p \neq q$

$H(\varepsilon)$ es una medida de la entropía del sistema, por lo que el análisis será más sensible cuando el parámetro $\varepsilon = 0$, ya que produce la mayor cadena de 1's y -1's. De manera inversa, el análisis será menos sensible cuando $\varepsilon = L$, ya que generará una cadena de 0's al enumerar S y por consecuencia proveerá una descripción menos detallada de la superficie de aptitudes. Un valor cercano a la unidad indica una superficie muy rugosa [Pérez 07].

Supongamos que se tienen los siguientes valores de aptitud en el conjunto ***F*** obtenidos de una caminata aleatoria de 10 pasos de longitud. Obtener el contenido de información.

F = {0.96668, 0.96968, 0.98841, 0.98844, 0.9766, 0.96999, 0.97194, 0.98174, 0.98921,

0.98921}

Lo primero que se debe de hacer es obtener la cadena ***S***, para lo cual debemos calcular el valor de *L*.

$L = \max(F) - \min(F)$
$L = 0.98921 - 0.96668$
$\therefore L = 0.02253$

Una vez obtenido el valor de *L*, debemos seleccionar un valor dentro del intervalo cerrado [0,*L*] aleatoriamente, para obtener el valor de e, aunque lo más recomendado es trabajar con un valor de $e = 0$.

$e = Random(0, L)$
$\therefore e = 0$

Ya con el valor del parámetro épsilon, podemos empezar la construcción de la cadena *S*, tal como se muestra en los siguientes pasos:

1. $f_i - f_{i-1} = 0.96968 - 0.96668$
$f_i - f_{i-1} = 0.003$
¿$0.003 < -0$? Þ *Falso*
¿$|0.003| \leq 0$? Þ *Falso*
¿$0.003 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$
$\therefore S = \{1$

2. $f_i - f_{i-1} = 0.98841 - 0.96968$
$f_i - f_{i-1} = 0.01873$
¿$0.01873 < -0$? Þ *Falso*
¿$|0.01873| \leq 0$? Þ *Falso*
¿$0.01873 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$

$\therefore S = \{1,1$

3. $f_i - f_{i-1} = 0.98844 - 0.98841$

$f_i - f_{i-1} = 0.00003$

¿$0.00003 < -0$? Þ *Falso*

¿$|0.00003| \pounds 0$? Þ *Falso*

¿$0.00003 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$

$\therefore S = \{1,1,1$

4. $f_i - f_{i-1} = 0.9766 - 0.98844$

$f_i - f_{i-1} = -0.01184$

¿$-0.01184 < -0$? Þ *Verdadero*

¿$|-0.01184| \pounds 0$? Þ *Falso*

¿$-0.01184 > 0$? Þ *Falso*

$\Rightarrow S \bigcup \bar{1}$

$\therefore S = \{1,1,1,\bar{1}$

5. $f_i - f_{i-1} = 0.96999 - 0.9766$

$f_i - f_{i-1} = -0.00661$

¿$-0.00661 < -0$? Þ *Verdadero*

¿$|-0.00661| \pounds 0$? Þ *Falso*

¿$-0.00661 > 0$? Þ *Falso*

$\Rightarrow S \bigcup \bar{1}$

$\therefore S = \{1,1,1,\bar{1},\bar{1}$

6. $f_i - f_{i-1} = 0.97194 - 0.96999$

$f_i - f_{i-1} = 0.00195$

¿$0.00195 < 0$? Þ *Falso*

¿$|0.00195| \pounds 0$? Þ *Falso*

¿$0.00195 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$

$\therefore S = \{1,1,1,\bar{1},\bar{1},1$

7. $f_i - f_{i-1} = 0.98174 - 0.97194$

$f_i - f_{i-1} = 0.0098$

¿$0.0098 < 0$? Þ *Falso*

¿$|0.0098| \pounds 0$? Þ *Falso*

¿$0.0098 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$

$\therefore S = \{1,1,1,\bar{1},\bar{1},1,1$

8. $f_i - f_{i-1} = 0.98921 - 0.98174$

$f_i - f_{i-1} = 0.00747$

¿$0.00747 < 0$? Þ *Falso*

¿$|0.00747| \pounds 0$? Þ *Falso*

¿$0.00747 > 0$? Þ *Verdadero*

$\Rightarrow S \bigcup 1$

$\therefore S = \{1,1,1,\bar{1},\bar{1},1,1,1$

9. $f_i - f_{i-1} = 0.98921 - 0.98921$

$f_i - f_{i-1} = 0$
¿$0 < -0$? Þ *Falso*
¿$|0| \pounds 0$? Þ *Verdadero*
¿$0 > 0$? Þ *Falso*

$\Rightarrow S \bigcup 0$
$\therefore S = \{1,1,1,\bar{1},\bar{1},1,1,1,0\}$

Una vez obtenida la cadena *S*, es posible visualizar el espacio de soluciones en una gráfica poligonal de dos dimensiones, siguiendo las siguientes reglas:

1. Si el elemento de la cadena *S* es 1, entonces se debe dibujar una línea recta a 45º a partir del último punto de la serie.
2. Si el elemento de la cadena *S* es 0, entonces se debe dibujar una línea recta a 0º (horizontal) a partir del último punto de la serie.
3. Si el elemento de la cadena *S* es $\bar{1}$, entonces se debe dibujar una línea recta a -45º a partir del último punto de la serie.

Para la cadena $S = \{1,1,1,\overline{1},\overline{1},1,1,1,0\}$ la gráfica queda de la siguiente manera:

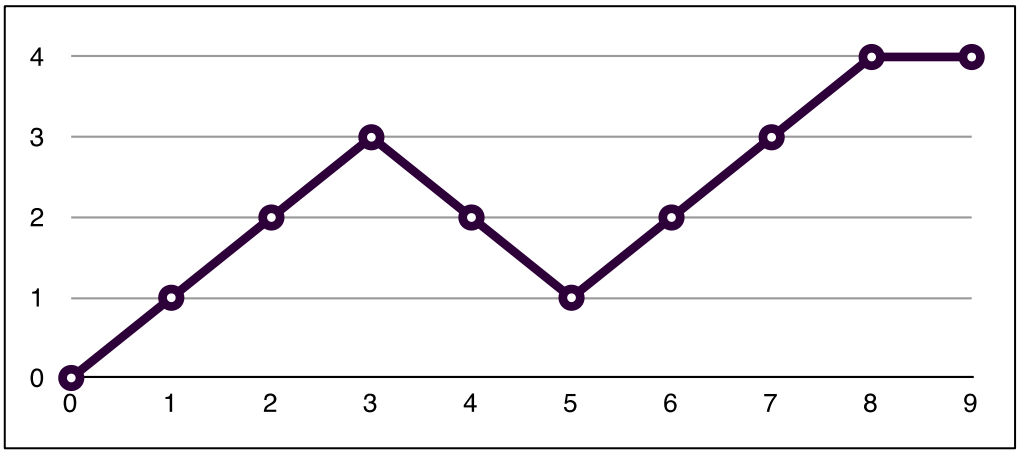


Figura C.1. Superficie de aptitudes del conjunto ***F***.

Para calcular el contenido de información que proporciona la cadena S se utiliza la Ecuación C.4, para lo cual es necesario calcular las probabilidades de ocurrencia de las ecuencias pq donde $p \neq q$. La Tabla C.1 muestra todas las probabilidades de ocurrencia.

Tabla C.1. Probabilidades de ocurrencia de las secuencias pq en *S*.

$[pq], p, q \in S$	$Ocurrencia([pq])$	Pr $([pq])$
11	4	0.5
$1\bar{1}$	1	0.125
$\bar{1}\bar{1}$	1	0.125
$\bar{1}1$	1	0.125
10	1	0.125

Como podemos darnos cuenta, las secuencias que debemos utilizar para el cálculo del índice son: $\{[1\bar{1}], [\bar{1}1], [10]\}$.

Aplicando la Ecuación C.4 tenemos:

$$H(e) = -\left((0.125)\left(\log_6(0.125)\right) + (0.125)\left(\log_6(0.125)\right) + (0.125)\left(\log_6(0.125)\right)\right)$$
$$H(e) = -\left(3(0.125)(-1.1606)\right)$$
$$H(e) = -\left(3(-0.1451)\right)$$
$$\therefore \; H(e) = 0.4352$$

La interpretación del valor obtenido es que la superficie de aptitudes no es muy rugosa [Pérez 07].

Anexo D

Introducción de Nuevos Índices

En este anexo se detalla la forma en cómo VisTHAA permite al investigador introducir sus propios índices.

D.1. Funcionamiento del Algoritmo Shunting Yard

Con motivo de ilustrar la forma de trabajar el algoritmo *shunting yard*, se plantea el siguiente ejemplo. Suponga que se quiere convertir la Ecuación D.1 de infija a postfija.

$$3 + \frac{4(2)}{(1 - 5)^{2^3}} \qquad \textbf{(D.1)}$$

Lo primero que se debe de hacer es escribir la expresión infija (Ecuación D.1) en forma lineal:

infija lineal = 3 + 4 * 2 / (1 - 5) ^ 2 ^ 3

El algoritmo lee todos los tokens (caracteres) de entrada, uno por uno y en cada paso se pregunta de que tipo de token se trata, si es un número, una variable o un operador, entre otros.

1. Token: **3**

El token actual es un número, por lo tanto se agrega a la cola de salida (ya en notación postfija).

postfija = { 3,

pila_operadores = { ø }

2. Token: **+**

Es un operador de adición, por lo tanto se agrega a la pila de operadores.

postfija = { 3,

pila_operadores = { +,

3. Token: **4**

Es un número, se agrega a la cola de salida.

postfija = { 3, 4,

pila_operadores = { +,

4. Token: *****

Es un operador de multiplicación, sin embargo como la pila de operadores no está vacía, tiene que ver cual operador está en el tope. Si el operador leído (‘*’) tiene mayor precedencia (prioridad) que el operador que está en el tope (como en este caso), entonces simplemente se añade a la pila.

postfija = { 3, 4,

pila_operadores = { +, *,

5. Token: **2**

Es un número, se agrega a la salida.

postfija = { 3, 4, 2,

pila_operadores = { +, *,

6. Token: **/**

Es un operador de división, pero en el tope de la pila de operadores está un operador que tiene la misma precedencia, por lo que el operador del tope de la pila (‘*’) se retira de ésta y se introduce a la salida, mientras que el operador de división (‘/’) se inserta como nuevo tope de la pila.

postfija = { 3, 4, 2, *

pila_operadores = { +, /,

Nota importante: En este caso únicamente se retiró de la pila al operador de multiplicación porque el otro operador (el operador suma) que se encontraba en el tope de la pila tenía una precedencia ***menor*** que el operador leído (operador de división). En el caso de que se encontraran en la pila una serie de operadores de ***mayor o igual*** precedencia que el leído, entonces **todos ellos** se retirarían de la pila y se introducirían a la expresión de salida en el mismo orden en que se van extrayendo.

7. Token: **(**

Es un paréntesis de apertura, simplemente se agrega a la pila de operadores

postfija = { 3, 4, 2, *

pila_operadores = { +, /, (,

8. Token: **1**

Es un número, se agrega a la expresión de salida.

postfija = { 3, 4, 2, *, 1,

pila_operadores = { +, /, (,

9. Token: **-**

Es el operador de sustracción, su precedencia es menor que el operador en el tope de la pila (‘**(**’), por lo que simplemente se añade a la pila de operadores

postfija = { 3, 4, 2, *, 1,

pila_operadores = { +, /, (, -,

10. Token: **5**

Es un número, se agrega a la salida.

postfija = { 3, 4, 2, *, 1, 5,

pila_operadores = { +, /, (, -,

11. Token: **)**

Es un paréntesis de cerradura, se deben retirar todos los operadores y/o funciones de la pila y agregarlos a la expresión de salida, hasta encontrar el paréntesis de apertura en la pila. Posteriormente, se debe de eliminar de la pila el paréntesis de apertura sin agregarlo a la salida. Finalmente, si no hay más elementos en la pila y no se encontró el paréntesis de apertura, entonces se produce un error, hay un paréntesis sin pareja (no en este caso).

postfija = { 3, 4, 2, *, 1, 5, -,

pila_operadores = { +, /,

12. Token: ^

Es el operador de exponenciación, como tiene mayor prioridad que el operador del tope de la pila, entonces simplemente se añade a la pila de operadores.

postfija = { 3, 4, 2, *, 1, 5, -,

pila_operadores = { +, /, ^,

13. Token: **2**

Es un número, se agrega a la salida.

postfija = { 3, 4, 2, *, 1, 5, -, 2,

pila_operadores = { +, /, ^,

14. Token: ^

Es el operador de exponenciación, sin embargo aunque tiene igual precedencia que el operador que está en el tope de la pila (otro operador de exponenciación), simplemente se agrega a la pila de operadores, ya que es un operador *asociativo derecho*.

postfija = { 3, 4, 2, *, 1, 5, -, 2,

pila_operadores = { +, /, ^, ^,

15. Token: **3**

Es un número, se agrega a la salida.

postfija = { 3, 4, 2, *, 1, 5, -, 2, 3 }

pila_operadores = { +, /, ^, ^ }

Después de haber leído el token “3”, la entrada ya no tiene más tokens por leer, por lo que se procede a vaciar la pila de operadores en la salida, conforme vayan saliendo los operadores del tope se irán agregando a la salida, finalmente la expresión postfija queda de la siguiente manera:

postfija = { **3, 4, 2, *, 1, 5, -, 2, 3,** ^, ^, /, + }

D.2. Evaluación en Notación Postfija

Supóngase que se quiere evaluar la siguiente expresión en notación postfija (calculada en la sección anterior, D.1):

postfija = {3, 4, 2, *, 1 ,5, -, 2, 3, ^, ^, /, +}

Al igual que en la conversión de infija a postfija, al momento de evaluar también se debe leer caracter por caracter.

1. Caracter leído: **3**

Es un número, entonces se debe añadir a la pila de números.
pila_números = { 3,

2. Caracter leído: **4**

Es un número, se añade a la pila de números.
pila_números = { 3, 4,

3. Caracter leído: **2**

Es un número, se añade a la pila de números.
pila_números = { 3, 4, 2

4. Caracter leído: *****

Es un operador que utiliza dos operandos, por lo que se sacan los dos últimos números de la

pila y se realiza la operación de multiplicación. El resultado se guarda en el tope de la pila.
operando2 = 2
operando1 = 4
resultado = 4 * 2 = 8
pila_números = { 3, 8,

5. Caracter leído: **1**

Es un número, se guarda en la pila.
pila_números = { 3, 8, 1,

6. Caracter leído: **5**

Es un número, simplemente se agrega a la pila.
pila_números = { 3, 8, 1, 5,

7. Caracter leído: **-**

Es el operador de sustracción, el cual utiliza dos operandos. Se sacan los dos últimos números de la pila, se efectúa la operación y se almacena el resultado en el tope de la pila.
operando2 = 5
operando1 = 1
resultado = 1 - 5 = -4
pila_números = { 3, 8, -4,

8. Caracter leído: **2**

Es un número, solamente se añade a la pila.
pila_números = { 3, 8, -4, 2,

9. Caracter leído: **3**

Es un número, por lo que se añade al tope de la pila.
pila_números = { 3, 8, -4, 2, 3,

10. Caracter leído: ^

Es el operador de exponenciación, el cual requiere de dos operandos, por lo cual se sacan los dos últimos números de la pila, se realiza la operación y el resultado se guarda en el tope de la pila.

operando2 = 3

operando1 = 2

resultado = $(2)^3$ = 8

pila_números = { 3, 8, -4, 8,

11. Caracter leído: ^

Al igual que en el paso anterior (10) es el operador de exponenciación, se hace lo mismo que en (10).

operando2 = 8

operando1 = -4

resultado = $(-4)^8$ = 65536

pila_números = { 3, 8, 65536,

12. Caracter leído: /

Es el operador de división, involucra dos operandos, por lo que se extraen los dos últimos números de la pila, se realiza la división y se guarda el resultado en el tope de la pila.

operando2 = 65536

operando1 = 8

resultado = 8 / 65536 = 0.0001220703125

pila_números = { 3, 0.0001220703125,

13. Caracter leído: +

Finalmente, el último caracter leído es el operador de adición, el cual involucra dos operados, se sacan los dos últimos números de la pila, se realiza la operación y se guarda el resultado en la pila.

operador2 = 0.0001220703125

operador1 = 3

resultado = 3 + 0.0001220703125 = 3.0001220703125

pila_números = { 3.0001220703125 }

Debido a que ya no hay más caracteres por leer, el proceso termina y el ***único número*** en la pila es el ***resultado*** de la expresión. Por otra parte, si en la pila hubiera más de un elemento, entonces hay un error en la expresión, la ecuación fue mal ingresada.

Anexo E

Análisis de la Superficie de Aptitudes

En este anexo se ejemplifica la manera en cómo se realizan los cálculos y se detallan los métodos referentes a la representación del espacio de búsqueda a través de la visualización de la superficie de aptitudes en tres dimensiones.

E.1. Cálculo de la Función Objetivo de BPP

Dada la siguiente instancia de BPP, una solución candidata es generada aleatoriamente, el procedimiento para calcular su valor objetivo se detalla a continuación

Instancia de BPP:

c=50

n=8

pesos = {10, 15, 22, 30, 11, 16, 21, 10}

solución_candidata:

30	22	16	10	21	11	15	10

1. Elemento actual: **30**

Contenedor_1 = {30,

llenado_contenedor = 30

2. Elemento actual: **22**

$30 + 22 > c$

La suma del llenado actual del contenedor (30) y el elemento actual (22) sobrepasan la capacidad del contenedor (50), por lo que el objeto de peso 22 ya no cabe y se tiene que introducir en otro contenedor nuevo.

Contenedor_1 = {30}

Contenedor_2 = {22,

llenado_contenedor = 22

3. Elemento actual: **16**

$22 + 16 £ c$

El objeto de peso 16 puede ser almacenado en el contenedor actual, ya que al incorporarse dicho objeto, éste no hace que se sobrepase la capacidad del contenedor.

Contenedor_1 = {30}

Contenedor_2 = {22, 16

llenado_contenedor = 38 (22 + 16)

4. Elemento actual: **10**

$38 + 10 £ c$

El elemento actual cabe en el contenedor actual.

Contenedor_1 = {30}

Contenedor_2 = {22, 16, 10

llenado_contenedor = 48

5. Elemento actual: **21**

$48 + 21 > c$

El objeto con peso 21 ya no cabe en el segundo contenedor, por lo que se tiene que introducir en otro.

Contenedor_1 = {30}

Contenedor_2 = {22, 16, 10}

Contenedor_3 = { 21,

llenado_contenedor = 21

6. Elemento actual: **11**

$21+11 \pounds c$

El objeto con peso 11 todavía cabe en el contenedor actual.

Contenedor_1 = {30}

Contenedor_2 = {22, 16, 10}

Contenedor_3 = { 21, 11

llenado_contenedor = 32

7. Elemento actual: **15**

$32+15 \pounds c$

El objeto con peso 15 todavía cabe en el contenedor actual.

Contenedor_1 = {30}

Contenedor_2 = {22, 16, 10}

Contenedor_3 = { 21, 11, 15

llenado_contenedor = 47

8. Elemento actual: **10**

$47+10>c$

El objeto cuyo peso es 10 ya no cabe en el contenedor actual, por lo que se introduce en otro.

Contenedor_1 = {30}

Contenedor_2 = {22, 16, 10}

Contenedor_3 = { 21, 11, 15}

Contenedor_4 = {10}

llenado_contenedor = 10

Al no haber más objetos a evaluar, el proceso termina tras ocho iteraciones, el número de contenedores utilizados para esta solución candidata es 4.

$\therefore m = 4$

E.2. Cálculo de la Función de Aptitud de BPP

En el siguiente ejemplo se ilustra la forma de calcular la función de aptitud de una solución candidata.

Tomando la solución del ejemplo anterior, obtener la función de aptitud con el índice que propone Falkenauer [Quiroz 09].

$$F_{BPP} = \frac{\left(\frac{30}{50}\right)^2 + \left(\frac{48}{50}\right)^2 + \left(\frac{47}{50}\right)^2 + \left(\frac{10}{50}\right)^2}{4}$$

$$F_{BPP} = \frac{(0.6)^2 + (0.96)^2 + (0.94)^2 + (0.2)^2}{4}$$

$$F_{BPP} = \frac{0.36 + 0.9216 + 0.8836 + 0.04}{4}$$

$$F_{BPP} = \frac{2.2052}{4}$$

$$\therefore \quad F_{BPP} = 0.5513$$

E.3. Generación de una Solución Vecina de BPP

Supongamos que se tiene la misma solución candidata del ejemplo de la sección E.1, obtener una solución vecina a través de la técnica de mutación por intercambio recíproco.

solución_candidata:

30	22	16	10	21	11	15	10

Lo primero que se debe de hacer es seleccionar aleatoriamente dos elementos de la solución, supongamos que se eligen la segunda y octava posición.

Posteriormente, lo único que se debe realizar es el intercambio de posiciones,

quedando la nueva solución vecina de la siguiente manera:

solución_candidata_vecina:

30	**10**	16	10	21	11	15	**22**

Con la nueva solución vecina generada, ahora se puede obtener su valor objetivo y su valor de aptitud, los cuales fueron calculados como en los ejemplos anteriores y sus resultados son:
Valor_Objetivo (*m*) = 3
Valor_Aptitud »0.8121

Lo que se puede concluir es que la solución vecina recientemente generada es ***mejor*** tanto en número de contenedores utilizados como el valor de aptitud (porcentaje de llenado).

E.4. Visualización de la Superficie en Tres Dimensiones

Supóngase que un investigador ha ejecutado una caminata aleatoria sobre una instancia de BPP, dicha caminata tuvo una longitud de *m*=16 pasos y generó el siguiente conjunto ***F*** de valores de aptitud:

F = {0.98, 0.97, 0.98, 0.99, 0.94, 0.96, 0.93, 0.99, 0.89, 0.90, 0.91, 0.93, 0.94, 0.94, 0.95, 0.93}

Debido a que el primer paso de la metodología ya se ha realizado (obtener los valores de aptitud), lo que continúa como segundo paso es precisamente realizar la partición del conjunto unidimensional *F*.

Para lograrlo, primero es necesario saber cuántas particiones son necesarias, para ello

se procede conforme a los criterios descritos en la sección 2.2.1.d.

Para este ejemplo el número 16 tiene raíz cuadrada exacta, por lo que la descomposición factorial queda de la siguiente manera:

$\sqrt{16} = 4$

Luego, los valores de *k* y *l* serán de 4, es decir:

$k = l = 4$

$\therefore k = 4, l = 4$

Ahora que ya tenemos los valores de *k* y *l*, podemos realizar la partición del conjunto unidimensional.

Se realizarán ***l*** particiones de longitud ***k*** en el conjunto ***F***, es decir, se realizarán 4 particiones de longitud 4 cada una, quedando un conjunto bidimensional $\boldsymbol{F'}_{4 \times 4}$, de tamaño 4 x 4. El conjunto ***F´*** del ejemplo queda seccionado de la siguiente manera:

0.98	0.97	0.98	0.99
0.99	0.93	0.96	0.94
0.89	0.90	0.91	0.93
0.93	0.95	0.94	0.94

El tercer paso de la metodología es considerar al conjunto bidimensional ***F´*** como una cuadrícula sobre un plano tridimensional, la Figura E.1 muestra la cuadrícula referente al ejemplo.

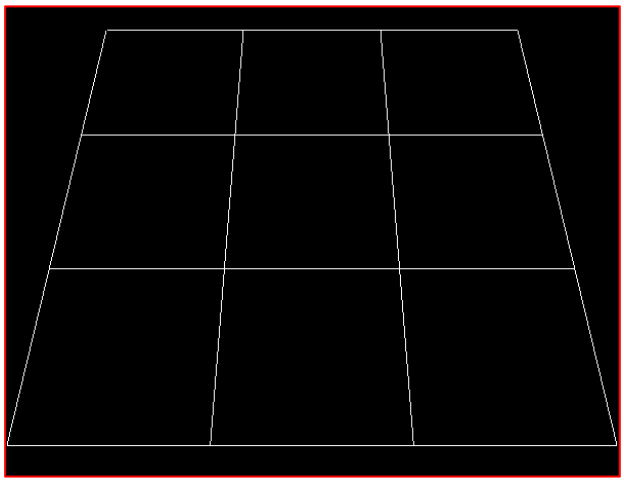


Figura E.1. Cuadrícula representativa del ejemplo.

Para poder graficar un punto en el espacio y darle volumen a la superficie se necesitan al menos tres valores (uno por cada dimensión). Los valores para el eje de las abscisas serán las filas, los valores para el eje de las ordenadas serán las columnas y finalmente los valores para el eje de las cotas serán los valores de aptitud de la *i-ésima* fila y la *j-ésima* columna, de tal manera que los puntos del conjunto ***F´*** del ejemplo son:

$$
\begin{array}{llll}
p_1(1,1,0.98) & p_5(2,1,0.99) & P_9(3,1,0.89) & P_{13}(4,1,0.93) \\
p_2(1,2,0.97) & p_6(2,2,0.93) & P_{10}(3,2,0.90) & P_{14}(4,2,0.95) \\
p_3(1,3,0.98) & p_7(2,3,0.96) & P_{11}(3,3,0.91) & P_{15}(4,3,0.94) \\
p_4(1,4,0.99) & p_8(2,4,0.94) & P_{12}(3,4,0.93) & P_{16}(4,4,0.94)
\end{array}
$$

Si consideramos a la cuadrícula como la superficie, hasta este punto solamente nos hace falta darle la altura a la cuadrícula, por lo que la Figura E.2 muestra la cuadrícula con volumen y desde una mejor perspectiva.

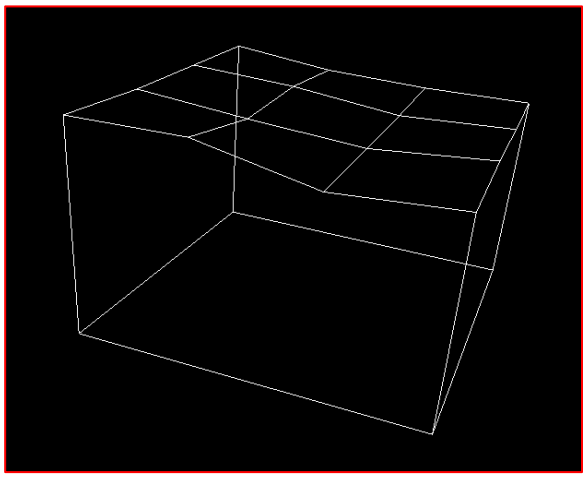


Figura E.2. Superficie de aptitudes tridimensional para el ejemplo.

Como puede verse en la Figura E.2, la superficie de aptitudes no está tan definida como se esperaría, esto se debe a que la caminata aleatoria únicamente fue de 16 pasos (para este ejemplo), sin embargo, si el investigador deja al algoritmo tomar una caminata considerablemente más larga, la superficie será mucho más definida y se podrá apreciar mejor de qué tipo de superficie se trata, si es una superficie plana o rugosa.

Anexo F

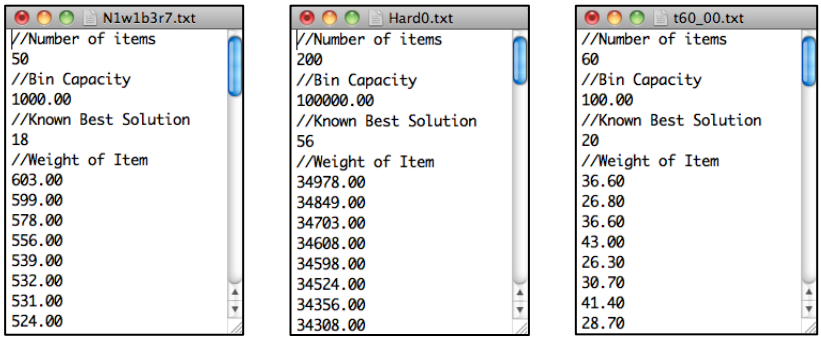
Pre-procesamiento de Instancias

En este anexo se muestra un ejemplo detallado de cómo introducir varias instancias de optimización a VisTHAA y cómo es que la herramienta las almacena en memoria.

F.1. Describiendo las Instancias

Lo primero que se debe de hacer para introducir cualquier conjunto de instancias a VisTHAA es realizar su descripción, esto se debe hacer a través de las palabras reservadas y reglas de sintaxis definidas en la sección 4.3.

Es importante mencionar que todas las instancias a introducir deben tener el mismo formato, esto se debe a que solamente se crea un archivo metainstance para todas las instancias. Supongamos que se desean introducir a VisTHAA el conjunto de 3 instancias de BPP mostrado en la Figura F.1.



a) Instancia N1w1b3r7 **b)** Instancia Hard0 **c)** Instancia t60_00

Figura F.1. Conjunto de 3 instancias a ser introducidas a VisTHAA.

Como se puede observar en la Figura F.1 las tres instancias a ser introducidas tienen el mismo formato, es decir, las tres instancias en la primera línea tienen un comentario, en la segunda tienen el número de objetos, en la tercera tienen otro comentario, en la cuarta tienen la capacidad del contenedor, en la quinta tienen otro comentario, en la sexta tienen la mejor solución conocida, en la séptima tienen otro comentario y finalmente de la fila ocho a la fila 8+*n*-1 tienen los pesos de los objetos, donde *n* representa al número de objetos. Es precisamente este formato de las instancias el que se debe describir utilizando las palabras reservadas y su respectiva sintaxis, de tal manera que el archivo *metainstance* queda como se muestra en la Figura F.2.

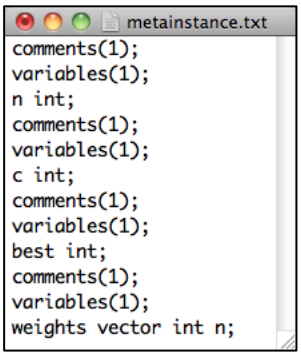


Figura F.2. Archivo *metainstance* para las tres instancias.

Posteriormente, es necesario crear otro archivo de texto denominado *logbook*, en el cual se le indicará a VisTHAA la cantidad de instancias a introducir (en este caso 3), el nombre del archivo metainstance y el nombre de los archivos de las instancias a ser introducidas, al igual que el metainstance, se deben utilizar las palabras reservadas de metalenguaje y su respectiva sintaxis .La Figura F.3 muestra el archivo *logbook* para este ejemplo.

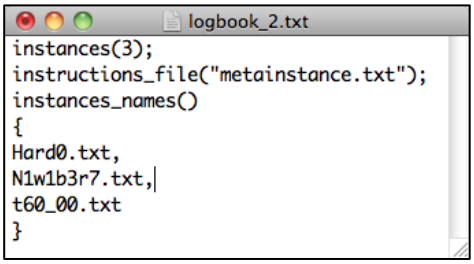


Figura F.3. Archivo *logbook* para las tres instancias.

F.2. Introduciendo las Instancias a VisTHAA

Una vez que se han creado los archivos metainstance y *logbook*, es necesario buscar este último y abrirlo desde VisTHAA. La Figura F.4 muestra la ruta de acceso al módulo en VisTHAA que permite realizar la carga de las instancias, mientras que la Figura F.5 muestra el explorador de archivos que permite realizar la búsqueda del archivo *logbook*.

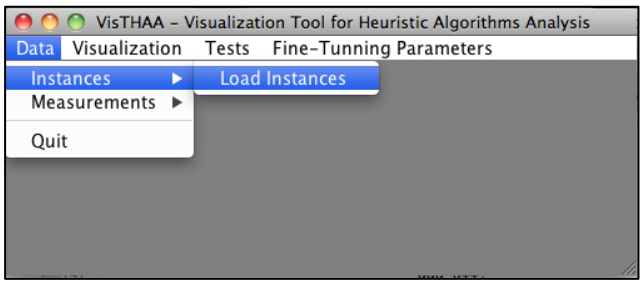


Figura F.4. Ruta de acceso para cargar instancias en VisTHAA.

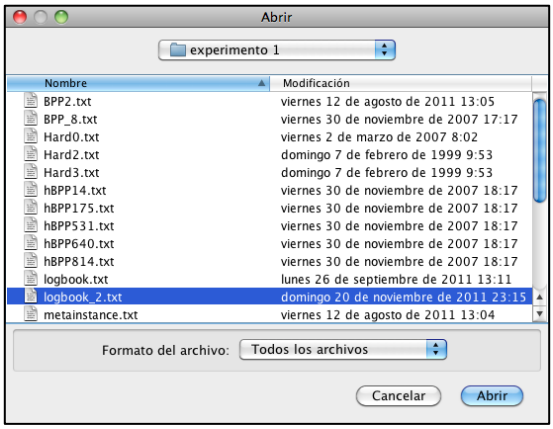


Figura F.5. Explorador de archivos para buscar el *logbook*.

Una vez que se ha seleccionado el archivo *logbook* en el explorador, lo que hace VisTHAA es leer dicho archivo, esto con el fin de obtener la cantidad de instancias que serán ingresadas, el nombre del archivo que almacena la descripción de éstas (*metainstance*) y el nombre de las instancias.

Posteriormente, VisTHAA lee el archivo *metainstance* y almacena en memoria temporal las instrucciones de cómo leer cada instancia (descripción del formato). Una vez hecho esto, VisTHAA calcula el número de variables que serán ingresadas a la herramienta para reservar el espacio de memoria suficiente para todas las estructuras de almacenamiento. La forma en como VisTHAA calcula la cantidad de variables es simple, en todo el archivo *metainstance* (almacenado en memoria temporal) busca la palabra reservada "*variables*" y realiza la sumatoria de sus argumentos. Para este ejemplo, hay cuatro palabras reservadas "*variables*" (líneas dos, cinco, ocho y once del *metainstance*, véase Figura F.2) y en cada una de ellas su argumento es uno, por lo tanto, ahora VisTHAA sabe que se introducirán 4 variables.

Después de calcular el número de variables y conociendo la cantidad de instancias que se introducirán (la cual se obtiene directamente del logbook del argumento de la palabra reservada "instances", véase Figura F.3) VisTHAA puede ahora reservar únicamente el espacio suficiente para sus estructuras de almacenamiento. Específicamente para este ejemplo, VisTHAA dimensiona las estructuras: tabla de símbolos, tipo de variable y tipo de dato de tamaño tres por cuatro (ya que son arreglos de dos dimensiones). La estructura longitud la dimensiona de tamaño tres por cuatro por dos, ya que es un arreglo de tres dimensiones y la última siempre será de tamaño dos; y finalmente, la estructura principal, datos, es dimensionada de tamaño tres por cuatro por cero por cero, ya que es un arreglo de cuatro dimensiones. La razón por la cual VisTHAA asigna las dos últimas dimensiones de datos con cero, es porque posteriormente serán redimensionadas dependiendo del tipo de variable que se trate, simple, vector o matriz y del tamaño de éstas en cada una de las instancias. Una vez hecho esto, lo que sigue es leer cada una de las instancias y obtener sus respectivos valores de las variables.

Para la primera instancia, N1w1b3r7.txt, VisTHAA procede de la siguiente manera leyendo línea por línea las instrucciones del metainstance:

1. "**comments(1);**"

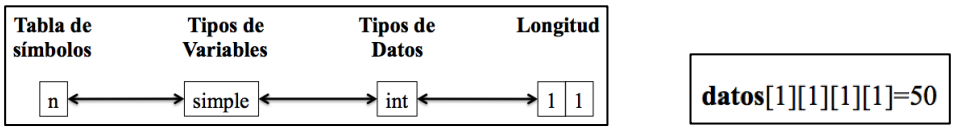
   Le indica a VisTHAA que la primera línea de la instancia es un comentario, éste es ignorado.

2. "**variables(1);**"

   Le indica a VisTHAA que la segunda línea de la instancia es una variable, sin embargo, todavía no sabe de que tipo de variable se trata, por lo cual tiene que leer la siguiente línea del *metainstance*.

3. "**n int;**"

   Le indica a VisTHAA que la variable a ser leída de la instancia es simple, de tipo entero y que tiene como identificador *n*. VisTHAA lee la segunda línea de la instancia y realiza las asignaciones a las estructuras correspondientes, tal como se muestra en la Figura F.6.



a) Estructuras auxiliares. **b)** Estructura principal.

Figura F.6. Asignación de la primera variable a las estructuras.

4. "**comments(1);**"

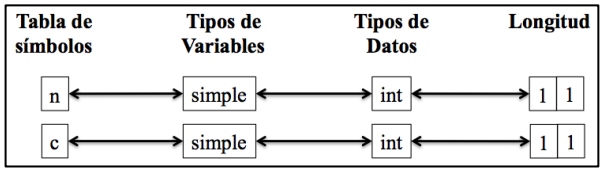
   Le indica a VisTHAA que la tercera línea de la instancia es un comentario, éste es ignorado.

5. "**variables(1);**"

   Le indica a VisTHAA que la cuarta línea de la instancia es una variable, sin embargo, todavía no sabe de que tipo de variable se trata, por lo cual tiene que leer la siguiente línea del *metainstance*.

6. "**c int;**"

Le indica a VisTHAA que la variable a ser leída de la instancia es simple, de tipo entero y que tiene como identificador *c*. VisTHAA lee la cuarta línea de la instancia y realiza las asignaciones a las estructuras correspondientes, tal como se muestra en la Figura F.7.



datos[1][1][1][1]=50

datos[1][2][1][1]=1000

a) Estructuras auxiliares. **b)** Estructura principal.

Figura F.7. Asignación de la segunda variable a las estructuras.

7. "**comments(1);**"

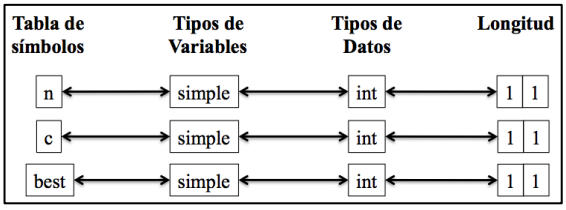
Le indica a VisTHAA que la quinta línea de la instancia es un comentario, éste es ignorado.

8. "**variables(1);**"

Le indica a VisTHAA que la sexta línea de la instancia es una variable, sin embargo, todavía no sabe de que tipo de variable se trata, por lo cual tiene que leer la siguiente línea del *metainstance*.

9. "**best int;**"

Le indica a VisTHAA que la variable a ser leída de la instancia es simple, de tipo entero y que tiene como identificador *best*. VisTHAA lee la sexta línea de la instancia y realiza las asignaciones a las estructuras correspondientes, tal como se muestra en la Figura F.8.



datos[1][1][1][1]=50

datos[1][2][1][1]=1000

datos[1][3][1][1]=18

a) Estructuras auxiliares. **b)** Estructura principal.

Figura F.8. Asignación de la tercera variable a las estructuras.

10. "**comments(1);**"

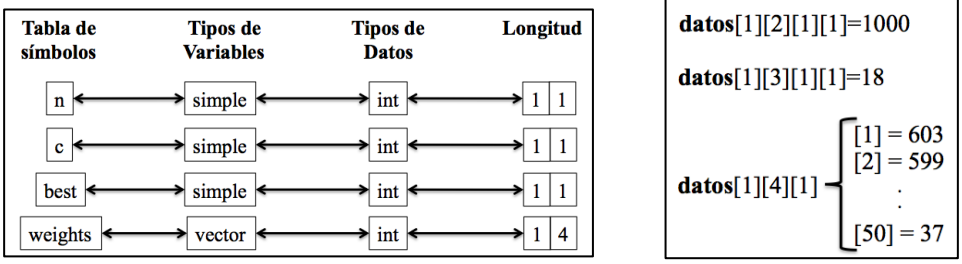
    Le indica a VisTHAA que la séptima línea de la instancia es un comentario, éste es ignorado.

11. "**variables(1);**"

    Le indica a VisTHAA que lo que sigue en el archivo de la instancia es una variable, sin embargo, todavía no sabe de que tipo de variable se trata, por lo cual tiene que leer la siguiente línea del *metainstance*.

12. "**weights vector int n;**"

    Le indica a VisTHAA que la variable a ser leída de la instancia es un vector, de tipo entero de tamaño *n* y que tiene como identificador *weights*. VisTHAA lee las siguientes *n* líneas a partir de la octava y realiza las asignaciones a las estructuras correspondientes, tal como se muestra en la Figura F.9.



a) Estructuras auxiliares. **b)** Estructura principal.

Figura F.8. Asignación de la última variable a las estructuras.

Finalmente, después de haber leído la última línea de instrucciones del metainstance, el proceso termina para la primera instancia, N1w1b3r7.txt. Este proceso se repite hasta terminar de cargar todas las instancias, para este ejemplo se repite otras dos veces.

Anexo G

Diseño de Interfaces

La interfaz de usuario juega un papel importante en el desarrollo y puesta en marcha de todo software de base o aplicación. El objetivo del diseño de la interfaz es resolver problemas de comunicación de manera efectiva tanto funcional como estética, siendo el principal medio de comunicación con el investigador, es por ello que las reglas propuestas por Shneiderman se tomaron en consideración al desarrollar VisTHAA.

G.1. Las Ocho Reglas de Oro del Diseño de Interfaces

Ben Shneiderman en su libro, *diseño de interfaces de usuario*, propone una colección de principios que fueron derivados heurísticamente de la experiencia y son aplicables en la mayoría de los sistemas interactivos después de ser propiamente refinados, extendidos e interpretados [Shneiderman 05].

Para mejorar la usabilidad de una aplicación es importante tener una interfaz bien diseñada. Las ocho reglas de oro de Shneiderman son una guía para un buen diseño, a continuación se muestran dichas reglas:

1. **Buscar la coherencia.** Secuencias coherentes de acciones deberían ser requeridas en situaciones similares; terminología idéntica debería ser utilizada en avisos, menús y pantallas de ayuda; y comandos consistentes deberían ser empleados en todo momento.
2. **Permitir a los usuarios frecuentes utilizar accesos directos.** A medida que la frecuencia de uso incrementa, también lo hacen los deseos del usuario por reducir el número de interacciones e incrementar la velocidad de ésta. El uso de abreviaturas, teclas de función, comandos ocultos, macros resultan muy útiles para un usuario experto.

3. **Ofrecer retroalimentación informativa.** Por cada acción del usuario debería haber alguna retroalimentación del sistema. Para acciones frecuentes y de menor importancia, la respuesta puede ser modesta, mientras que para acciones poco frecuentes y de importancia mayor, la respuesta debería ser más sustancial.
4. **Diseñar un diálogo para producir el cierre.** Las secuencias de acciones deberían estar organizadas en grupos con un principio, medio y final. La retroalimentación informativa en la realización de un conjunto de acciones da a los usuarios la satisfacción del logro, una sensación de alivio, la señal de abandonar los planes de contingencia y las opciones de su mente, y una indicación de que el camino esta libre para prepararse para el siguiente grupo de acciones.
5. **Ofrecer un manejo simple de errores.** En la medida de lo posible, diseñar el sistema para que el usuario no pueda cometer un grave error. Si comete un error, el sistema debe ser capaz de detectarlo y ofrecer mecanismos sencillos y comprensibles para el correcto manejo del error.
6. **Permitir la fácil reversión de acciones.** Esta característica alivia la ansiedad, ya que el usuario sabe que los errores se puede deshacer. De esta manera se promueve la exploración de opciones desconocidas. Las unidades de reversibilidad puede ser una sola acción, una entrada de datos o un grupo completo de acciones.
7. **Apoyar al locus interno de control.** Los usuarios experimentados desean fuertemente la sensación de que ellos tienen el control del sistema y que éste responde a sus acciones. Se debe diseñar el sistema para hacer que los usuarios sean los que inicien las acciones, en lugar de que sean los que responden.
8. **Reducir la carga de la memoria a corto plazo.** La limitación del procesamiento humano de información en la memoria a corto plazo requiere que las pantallas se muestren simples, se consoliden pantallas de varias páginas, la frecuencia de movimiento de la ventana se reduzca y tiempo de entrenamiento suficiente debe ser asignado para códigos, nemotécnicos y secuencias de acciones.

Literatura Citada

[Alvim 04] Alvim A., Glover F., Ribeiro C., Aloise D. "*A hybrid improvement heuristic for the one-dimensional bin packing problem*". Journal of Heuristics, 10: 205-229, 2004.

[Barr 95] Barr Richard S., Golden Bruce L., Kelly James P., Resendez M., Stewart William R. "*Designing and Reporting on Computational Experiments with Heurístic Methods*". Journal of Heuristics, pp. 19-32. 1995.

[Blum 03] C. Blum and A. Roli. "*Metaheuristics in combinatorial optimization: Overview and conceptual comparison*". ACM Computing Surveys, 35(3):268–308, 2003.

[Borbón 06] Borbón A. Alexander.: "*¿Cómo evaluar expresiones matemáticas en el computador?*". Cidse-Revista Virtual Matemática, Educación e Internet. Escuela de Matemática. Instituto Tecnológico de Costa Rica. (2006).

[Brglez 07] Brglez F., Osborne J. A. "*Performance testing of combinatorial solvers with isomorph class instances*". Workshop On Experimental Computer Science, Article No. 13. (2007).

[Canavos 88] George C. Canavos.: "*Probabilidad y Estadística Aplicaciones y Métodos*". Primera edición en español. McGraw-Hill / Interamericana de México, S.A. de C.V. ISBN: 968-451-856-0. (1988).

[Coello 08] Carlos A. Coello Coello.: "*Introducción a la Computación Evolutiva (notas del curso)*". Departamento de Computación, CINVESTAV-IPN. México. 2008.

[Cohen 95] Cohen P. R. "*Empirical Methods for Artificial Intelligence*". The MIT Press, Cambridge, Massachusetts, 1995.

[Cruz 04] Laura Cruz Reyes.: "*Clasificación de Algoritmos Heurísticos para la Solución de Problemas de Bin Packing*". Tesis de Doctorado. Centro Nacional de Investigación y Desarrollo Tecnológico. México. (2004).

[Defanti 91] Thomas A. Defanti and Maxine D. Brown.: "*Visualization in Scientific Computing*". Electronic Visualization Laboratory. University of Illinois at Chicago. Chicago, Illinois. (1991).

[Dijkstra 61] Dijkstra E. W.: "*ALGOL-60 Translation*". Stichting Mathematisch Centrum. 2e Boerhaavestraat 49. Amsterdam. Rakenafdeling. 1961.

[Dooms 07] Grégoire Dooms, Pascal Van Hentenryck y Laurent Michel.: "*Model-Driven Visualizations of Constraint-Based Local Search*". University of Connecticut. 2007.

[Duarte 07] Abraham Duarte Muñoz, Juan José Pantrigo Fernández y Micael Gallego Carrillo.: "*Metaheurísticas*". Universidad Rey Juan Carlos. España. (2007).

[Fraire 10] Héctor Joaquín Fraire Huacuja, José Luis González-Velarde, Guadalupe Castilla Valdez and Blanca Nelly Santos Sntoy.: "*Analyzing the Instances Structure and the Search Landscape for the Robust Capacitated International Sourcing Problem (RoCIS)*". Proceedings of the Congress. 16th International Congress on Computer Science Research CIICC'10.

[García 08] Salvador García, Daniel Molina, Manuel Lozano y Francisco Herrera.: "*A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization*". Springer Science+Business Media, LLC 2008.

[Gendreau 10] Gendreau Michel, Potvin Jean-Yves.: "*Handbook of Metaheuristics*". Second Edition. International Series in Operations Research & Management Science. Volume 146. Springer. ISBN: 978-1-4419-1663-1, e-ISBN: 978-1-4419-1665-5. (2010).

[Gent 99] Gent I. P., Walsh T. Paul R. Book Review: Cohen's Empirical Methods for Artificial Intelligence. Artificial Intelligence 113: 285-290, Elsevier Science B.V. (1999).

[Glover 86] Glover, F.: "*Future Paths for Integer Programming and Links to Artificial Intelligence*". Comput. Oper. Res. Vol 13. 533-549. (1986).

[Gómez 09] Claudia Guadalupe Gómez Santillán.: "*Afinación Estática Global de Redes Complejas y Control Dinámico Local de la Función Tiempo de Vida en el Problema de Direccionamiento de Consultas Semánticas*". Tesis de doctorado. Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada. Altamira, Tamaulipas, México. (2009).

[Halim 06] Steven Halim., Roland H. C. Yap.: "*Viz: A Visual Analysis Suite for Explaining Local Search Behavior*". UIST'06. Copyright 2006. ACM 1-59593-313-1/06/0010. (2006).

[Halim 07] Steven Halim., Roland H. C. Yap., Hoong Chuin Lau.: "*An Integrated White+Black Approach for Designing and Tuning Stochastic Local Search*". (2007).

[Hedar 04] Abdel-Rahman Hedar A. Ahmed.: "*Studies on Metaheuristics for Continuous Global Optimization Problems*". PhD Thesis. Kyoto University, Japan. 2004.

[Hooker 94] Hooker J. N. Needed: "*An empirical science of algorithms*". Operations Research, 42(2): 201-212. (1994).

[Hooker 95] Hooker J. N. "*Testing Heuristics: We have it all wrong*". Journal of Heuristics, 1: 33-42. (1995).

[Lau 05] Hoong Chuin Lau, Wee Chong Wan, Steven Halim.: "*Tuning Tabu Search*

Strategies via Visual Diagnosis". School of Information Systems. Singapore Management University. (2005).

[Loh 06] Kok-Hua Loh, Bruce Golden, Edward Wasil.: "*Solving the one-dimensional bin packing problem with a weight annealing heuristic*". Computer & Operations Research 35 (2008) 2283-2291. (2006).

[Maarten 04] Maarten Vankerkom Jin Yu.: "*Visualizing Swarm algorithms*". Tesis de Maestría. Faculteit Toegepaste Wetenschappen. (2004).

[McGeoch 92] McGeoch C. C.: "*Analysis of algorithms by simulation: variance reduction techniques and simulation speedups*". ACM Computing Surveys 24(5): 195-212. (1992).

[McGeoch 00] McGeoch, C.C.: "*Experimental Analysis of Algorithms*". In: Pardalos, P.M., Romeijn, H.E.: Handbook of Global Optimization, Vol. 2, pp. 489-513. (2000).

[Mendenhall 97] William Mendenhall, Terry Sincich.: "*Probabilidad y Estadística para Ingeniería y Ciencias*". Cuarta edición. Prentice-Hall Hispanoamérica, S.A. ISBN (traducción): 968-880-960-8. ISBN (idioma original): 0-02-380581-1. (1997).

[Merz 98] Merz Peter Freisleben Bernd.: "*Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning*". Scientific literature Digital Library. (1998).

[O'Brien 08] O'Brien R. (2008). "*Ant Algorithm Hyperheuristic Approaches for Scheduling Problems*". The University of Nottingham.

[Padma 96] Padma Reddy, M. Balaram, Chenjerai Bones, and Y.B. Reddy.: "*Visualization in Scientific Computing*". Grambling State University, Grambling LA 71245. Department of Mathematics and Computer Science. (1996).

[Pérez 07] Verónica Pérez Rosas.: "*Modelado Causal del Desempeño de Algoritmos Metaheurísticos en Problemas de Distribución de Objetos*". Tesis de Maestría. Instituto Tecnológico de Ciudad Madero. (2007).

[Scott 92] David W. Scott.: "*Multivariate Density Estimation Theory, Practice and Visualization*". John Wiley & Sons, Inc. ISBN: 0-471-54770-0. (1992).

[Spence 07] Robert Spence.: "*Information Visualization Design for Interaction*". Second Edition. Pearson Prentice Hall. (2007).

[Quiroz 09] Marcela Quiroz Castellanos.: "*Caracterización de Factores de Desempeño de Algoritmos de Solución de BPP*". Tesis de Maestría. Instituto Tecnológico de Ciudad Madero. México. (2009).

[Shneiderman 05] Shneiderman Ben.: "*Las ocho reglas de oro*". http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html. Última visita 30 de Septiembre de 2011. (2005).

[Ugur 10a] Aybars U., Dogan A.: "*An interactive simulation and analysis software for solving TSP using Ant Colony Optimization algorithms*". Advances in Engineering Software, 40 (5). (2008).

[Ugur 10b] Ugur A., TSPAntSim, software disponible en línea, http://yzgrafik.ege.edu.tr/projects/TSPAntSim/, último acceso: 19/Septiembre/2011.

[Ugur 10c] Ugur A.: "*Path Planning On A Cuboid Using Genetic Algorithms*". Information Sciences 178, pp. 3275-3287. (2008).

[Ugur 10d] Ugur A., CuboidTSP Versión 2.0, software disponible en línea, http://yzgrafik.ege.edu.tr/~ugur/CuboidTSP/, último acceso: 19/Septiembre/2011.

[Vassilev 99] Vassilev Vesselin K.,Julian F. Miller.: "*Digital Circuit Evolution: The Ruggedness and Neutrality of Two-Bit Multiplier Landscapes*". IEE Half-day Colloquium on Evolutionary Hardware Systems. (1999).

[Weinberg 90] Weinberg, E. D.: "*Correlated and uncorrelated fitness landscapes and how to tell the difference*". Biological Cybernetics, 63, 325-336. (1990).

[Wright 32] Wright, S. "*The roles of mutation, inbreeding, crossbreeding and selection in evolution*". Proceedings of the sixth international genetics, vol. 1 pp. 356-366. (1932).