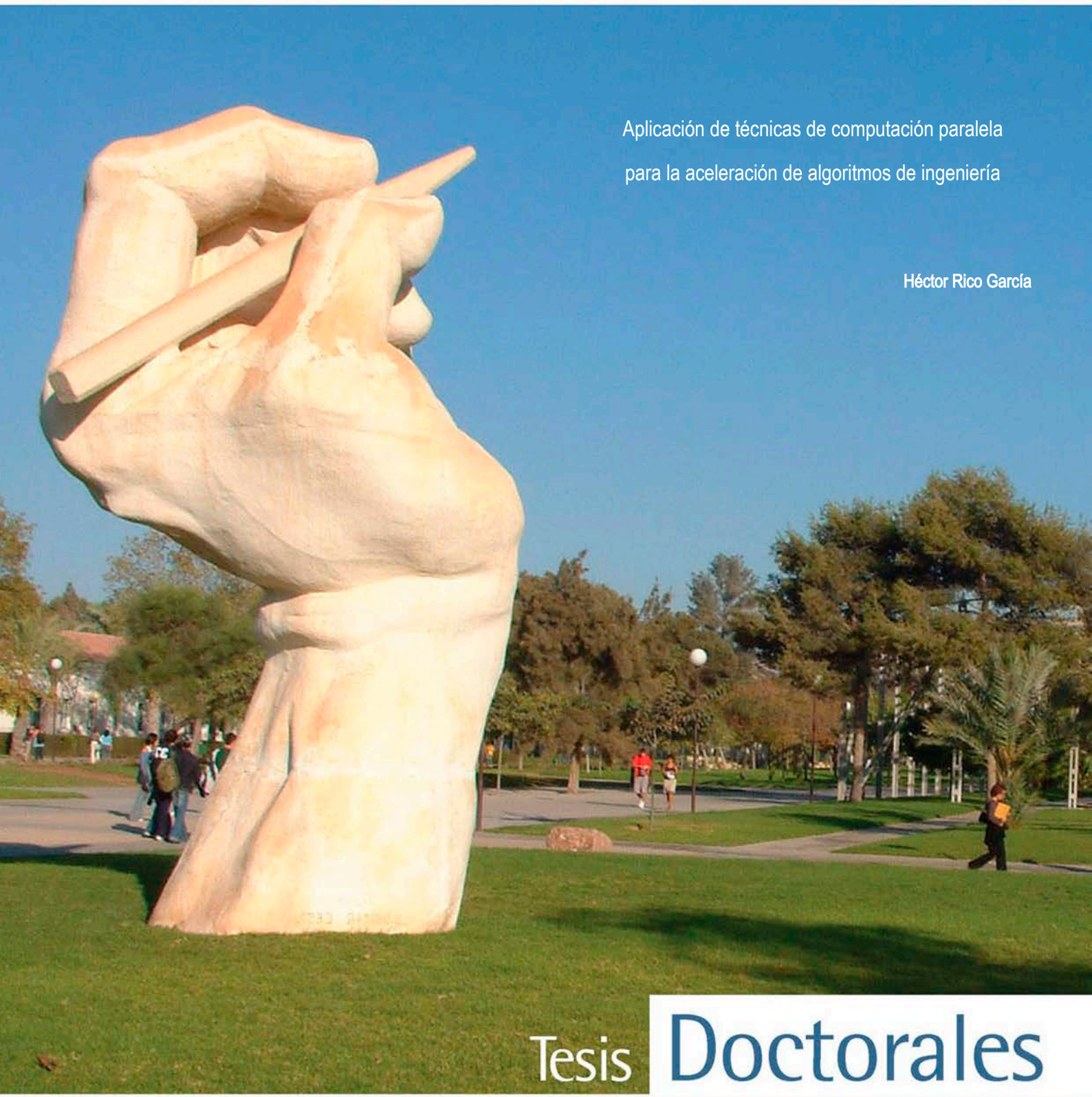
Universitat d'Alacant
Universidad de Alicante

Aplicación de técnicas de computación paralela para la aceleración de algoritmos de ingeniería

Héctor Rico García



Tesis Doctorales

UNIVERSIDAD de ALICANTE

Unitat de Digitalització UA
Unidad de Digitalización UA

Universitat d'Alacant
Universidad de Alicante

Departamento de Tecnología Informática y Computación
Escuela Politécnica Superior

Aplicación de técnicas de computación paralela para la aceleración de algoritmos de ingeniería

Héctor Rico García

Tesis presentada para aspirar al grado de
DOCTOR POR LA UNIVERSIDAD DE ALICANTE

Doctorado en Informática

Dirigida por:

Dr. Antonio Manuel Jimeno Morenilla
Dr. José Luis Sánchez Romero

Índice

Síntesis

1. Resumen

Actualmente nos encontramos dentro de lo que algunos autores catalogan como la Cuarta Revolución Industrial o también llamada Industria 4.0. La tecnología está teniendo un gran impacto sobre la industria y con ello también ha planteado un gran número de nuevos problemas de ingeniería a los ya existentes del pasado.

La optimización de la solución a dichos problemas se puede analizar desde diferentes puntos de vista y diferentes objetivos.

En la actualidad existen diferentes líneas de investigación que intentan dar solución a la mejora en la optimización de la obtención de soluciones a todos estos problemas. En los últimos años se han desarrollado de una forma muy activa varias de estas líneas de investigación como son la Inteligencia Artificial, el Aprendizaje profundo o la aplicación de algoritmos heurísticos. En este trabajo de investigación, se ha estudiado la aplicación de dichos algoritmos heurísticos para la resolución de estos problemas y en la aceleración de dichos algoritmos utilizando técnicas de computación paralela.

Dentro de los algoritmos heurísticos existe una división en dos grandes grupos: los algoritmos evolutivos y los algoritmos basados en la inteligencia de enjambre. En este trabajo de investigación, se ha hecho hincapié en los algoritmos evolutivos, ya que su naturaleza los hace idóneos para la aplicación de técnicas de paralelización. Dentro de los algoritmos evolutivos existe un conjunto de algoritmos basados en poblaciones. En el ámbito de esta investigación, se han elegido dos algoritmos de dicha categoría para su implementación y aplicación en problemas de ingeniería. Se ha trabajado, por un lado, sobre el algoritmo Jaya, que basa su funcionamiento en generar individuos que se acercan el mejor y se alejan del peor de ellos y, por otro lado, sobre el algoritmo TLBO, que imita el proceso de aprendizaje dentro de un aula, donde los alumnos aprenden del profesor en una primera etapa y pueden seguir aprendiendo, compartiendo conocimientos entre ellos en una segunda etapa. Además de las versiones originales, también se ha trabajado durante esta investigación con algunas variantes de los dos algoritmos, como pueden ser *Chaotic Jaya* o DTLBO.

En el marco de la investigación, y tras estudiar dichos algoritmos, se ha procedido a realizar una implementación de los mismos. Se ha realizado una implementación secuencial de los algoritmos siguiendo el pseudocódigo proporcionado por los creadores de los algoritmos para la experimentación junto a una implementación utilizando técnicas de computación paralela. Para la implementación paralela de los algoritmos se ha optado como línea de investigación principal por la implementación paralela utilizando la

tecnología manycore (multinúcleo) GPU (*Graphics Processing Unit*), más concretamente la implementación de Nvidia CUDA (*Compute Unified Device Architecture*).

Para poder realizar una comprobación de la eficacia de los algoritmos que han sido estudiados en esta investigación y sus implementaciones realizadas utilizando técnicas de computación paralela, en este trabajo se han planteado dos escenarios para la experimentación. Por un lado, un primer escenario utilizando como referencia funciones de optimización existentes en la literatura actual y altamente documentadas. En este primer escenario, se ha realizado una comparativa entre la implementación secuencial de cada algoritmo y la correspondiente versión paralelizada propuesta, haciendo especial hincapié en la aceleración obtenida de dichos algoritmos. Además de la aceleración, se ha tenido en cuenta que las soluciones obtenidas estén dentro de unos criterios de calidad. En un segundo escenario, se han aplicado las implementaciones tanto secuencial como paralela de los algoritmos estudiados sobre determinado problema de ingeniería. En concreto, se han aplicado los algoritmos a un conjunto de problemas de optimización orientados al cálculo de rutas; para ello, ha sido necesario modificar los algoritmos implementados para sean capaces de dar soluciones a problemas discretos.

La investigación realizada ha tenido como resultado la publicación de tres artículos en revistas internacionales de alto impacto recogidas en el índice JCR (*Journal Citations Report)*; así mismo, se han publicado tres artículos en otras tantas revistas indizadas igualmente en el JCR.

La presente tesis por compendio de publicaciones se divide en tres apartados, los cuales son descritos a continuación de forma breve:

- Síntesis
- Trabajos publicados
- Conclusiones.
- Anexo

En el apartado de Síntesis, se detalla la fundamentación e investigación realizada junto a los detalles de implementación y experimentación. En dicha sección se incluye un resumen general de la investigación realizada; se describen también las motivaciones sobre las que se ha basado la investigación junto a los objetivos que fueron planteados en el inicio de la investigación. Por otro lado, se incluye así mismo en este apartado una revisión del estado del arte en el que se describen los algoritmos heurísticos, concretando en aquellos que se implementarán durante la investigación y exponiendo su implementación tanto secuencial como paralela. Por último, se plantean dos escenarios de experimentación sobre las implementaciones realizadas: un primer escenario donde se realiza una comparación resultante de la aplicación de los algoritmos a un conjunto de funciones documentadas en la literatura del ámbito, y un segundo escenario donde se realiza la comparación de las implementaciones y sus aceleraciones sobre problemas de ingeniería concretos.

En el siguiente apartado, llamado Trabajos publicados, se plasman de forma íntegra los artículos de mayor repercusión que han sido publicados durante la investigación; en el Anexo se plasma el resto de los artículos que han sido publicados. Dichos artículos aparecen con el formato que fueron publicados para las diferentes publicaciones científicas acompañados de sus referencias. Los respectivos títulos de los artículos publicados son:

- Comparison of High Performance Parallel Implementations of TLBO and Jaya Optimization Methods on Manycore GPU

- Parallel implementation of metaheuristics for optimizing tool path computation on CNC machining

- A Parallel Meta-Heuristic Approach to Reduce Vehicle Travel Time in Smart Cities

En el apartado de Conclusiones, se describen los principales resultados y conclusiones que se han obtenido durante el trabajo de investigación realizado, así como las futuras líneas de investigación que continúan abiertas para ampliar y completar los resultados obtenidos.

Por último, en el anexo se han añadido los artículos restantes publicados durante la investigación. Los respectivos títulos de los artículos publicados son:

- Efficient Subpopulation Based Parallel TLBO Optimization Algorithms

- Settings-Free Hybrid Metaheuristic General Optimization Methods

- Multipopulation-based multi-level parallel enhanced Jaya algorithms

2. Motivación

2.1 Antecedentes.

En el campo de la ingeniería, existen multitud de problemas que deben abordarse en función de dos criterios básicos: la precisión de la solución encontrada y el tiempo necesario para hallar dicha solución. Entre dichos problemas, cabe destacar la optimización de una función determinada. Existen algoritmos de optimización cuyo objetivo es hallar la solución óptima dada una función y una serie de restricciones en los parámetros de ésta. Los algoritmos heurísticos se aplican especialmente a la optimización de funciones no derivables y, dentro de estos algoritmos heurísticos, destacan los algoritmos basados en poblaciones.

Los algoritmos heurísticos basados en poblaciones se dividen en dos grandes grupos: algoritmos evolutivos y algoritmos basados en la inteligencia de enjambre. Ambos tipos de algoritmos son probabilísticos y requieren una serie de parámetros de control, como puede ser el tamaño de la población, el número de generaciones, el tamaño de la élite, etc. Los algoritmos heurísticos tienen ciertas características que los hacen particularmente adecuados para ser sometidos a un proceso de paralelización en sus diversas implementaciones, lo que produce de una forma directa una mejora en su tiempo de ejecución.

Existe una comunidad investigadora muy activa en el estudio de este tipo de algoritmos, y continuamente aparecen no sólo nuevos algoritmos, sino también mejoras de los existentes. Uno de los mayores problemas a la hora de la aplicación de este tipo de algoritmos en problemas reales de ingeniería es el tiempo de ejecución necesario para producir una solución dentro del rango de soluciones aceptables, sobre todo en sistemas de producción o sistemas de tiempo real. Como solución a este problema, la aplicación de técnicas de paralelismo a la implementación de los algoritmos añade una notable mejora en los tiempos de procesamiento, sin menoscabar la precisión de la solución encontrada.

Existen muchos algoritmos de optimización inspirados en la naturaleza que utilizan principios observados en diferentes fenómenos naturales. Cada uno de estos algoritmos se basa en un fenómeno específico observado en la naturaleza. Por ejemplo, el Algoritmo Genético (GA – *Genetic Algorithm*) utiliza la teoría de la

evolución para mejorar la población hacia la solución del problema; la Optimización por Enjambre de Partículas (PSO – *Particle Swarm Optimization*) emula el comportamiento de los enjambres de pájaros en busca de alimento; los algoritmos de la Colonia Artificial de Abejas (ABC – *Artificial Bee Colony*) y la Optimización por Colonia de Hormigas (ACO - *Ant Colony Optimization)* se inspiran en las organizaciones de las colonias de ambos tipos de insectos. Debido a los impresionantes avances que se han producido en los últimos años en el campo de las arquitecturas paralelas, al tiempo que se ha reducido enormemente el coste de las mismas, la mayoría de estos algoritmos de optimización se han paralelizado para obtener un aumento del rendimiento [1-10]. La aplicación de estos algoritmos a la solución de problemas de optimización dentro de la ingeniería se ha testeado en diversos trabajos con resultados satisfactorios [1-3].

Una de las desventajas de estos algoritmos en general es el uso de un gran número de parámetros y restricciones propias de cada uno de los algoritmos. Estos parámetros y restricciones suelen variar en función del problema a tratar, y además influyen directamente en el resultado de la aplicación del algoritmo y en su eficacia. Para hacer frente a estos problemas, se han propuesto nuevos algoritmos de optimización basados en la evolución de poblaciones, pero evitando el uso de parámetros y restricciones. Dos algoritmos de optimización recientes siguen este enfoque: *Teaching-Learning Based Optimization* (TLBO) y Jaya. Estos dos algoritmos no utilizan parámetros o restricciones específicas, sino que sólo utilizan los parámetros intrínsecos a este tipo de algoritmos, como el tamaño de la población, el número de variables de cada individuo en función del problema a optimizar y el número de iteraciones. TLBO y Jaya han demostrado ser más eficientes que otros algoritmos a la hora de optimizar diversos problemas [31-34].

2.2 Contexto de la investigación

La investigación realizada en este trabajo toma como base una investigación realizada dentro del grupo de investigación UniCAD de la Universidad de Alicante. Mediante diversos trabajos del grupo de investigación, se inició una línea de estudio sobre los algoritmos heurísticos y sus implementaciones con técnicas de paralelización, llegando a producir diversas publicaciones en revistas y congresos. Dentro de esta línea abierta por el grupo de investigación, en este trabajo se ahonda en ella estudiando nuevos algoritmos heurísticos y realizando implementaciones secuenciales y paralelas de dichos algoritmos para el estudio comparativo de los mismos y sus posibles mejoras.

Dentro del grupo de investigación se ha reforzado la línea de las implementaciones de algoritmos heurísticos con técnicas *manycore* (multitud de núcleos) GPU y, sobre todo, en la utilización de NVidia CUDA para realizar dichas implementaciones. Esta línea de investigación también se nutre de la experiencia y los conocimientos de otros grupos de investigación de la misma Universidad de Alicante, con los que el grupo UniCAD colabora.

Además, durante la investigación se han realizado colaboraciones y trabajos conjuntos con otros grupos de investigación e investigadores de otras universidades como por ejemplo la Universidad Miguel Hernández de Elche. Junto a los investigadores de la Universidad Miguel Hernández de Elche se ha investigado en la aplicación de otras técnicas de computación paralela a los algoritmos heurísticos, especialmente la utilización de la arquitectura OpenMP para la aceleración de dichos algoritmos. Fruto de esta colaboración y del intercambio de conocimientos y experiencias ha sido la realización de publicaciones con autoría conjunta en varias revistas de impacto.

Es de esperar que esta línea de investigación, reforzada durante la realización de este trabajo de investigación, continúe su desarrollo en un futuro gracias a nuevas colaboraciones, puesto que es un área con muchas posibilidades de estudio, así como de aplicación en una gran variedad de problemas de índole ingenieril con aplicación en la industria.

3. Objetivos

El objetivo principal de esta investigación es demostrar la mejora que proporciona la computación paralela en sus diferentes aproximaciones en el campo de la aceleración de algoritmos para la resolución de problemas de ingeniería. Tras un análisis de diversos algoritmos para la resolución de este tipo de problemas, se han realizado implementaciones utilizando diferentes técnicas de computación paralela y se ha comprobado la mejora en los tiempos de ejecución de dichos algoritmos, lo cual los hace adecuados para aplicaciones en tiempo real o sobre entornos industriales.

Los objetivos específicos de esta investigación se derivan del objetivo general y constituyen una hoja de ruta a seguir para la consecución de este. A continuación, se detallan dichos objetivos específicos:

- Implementar un conjunto de algoritmos de optimización utilizando diferentes técnicas de paralelismo disponibles (CUDA y OpenMP) y su aplicación a problemas de ingeniería. Los algoritmos elegidos para su implementación son Jaya y TLBO junto a sus variantes (DTLBO y *Chaotic Jaya*).

- Realizar un estudio sobre el rendimiento de dichos algoritmos, comparando las implementaciones secuenciales con las implementaciones usando diferentes técnicas de paralelismo. Para realizar este estudio, se procederá a la optimización de un conjunto de funciones disponibles en la literatura actual del área. Se analizará no sólo la mejora obtenida al aplicar las técnicas de paralelismo con respecto al tiempo, sino también los resultados de la optimización y la convergencia hacia el resultado esperado, es decir, la precisión alcanzada.

- Estudiar el comportamiento de los algoritmos aplicados a diferentes tipos de problemas y el grado de mejora que producen las diferentes técnicas de paralelismo.

- Aplicar los algoritmos a casos particulares dentro del ámbito de la ingeniería.

4. Estado del arte

Los enfoques deterministas para resolver problemas de optimización aprovechan las propiedades de los problemas para generar una secuencia de puntos que converge hacia una solución óptima global. Dichos enfoques deterministas proporcionan herramientas generales para la obtención de estas soluciones óptimas globales. Sin embargo, debido a la naturaleza de determinados problemas, los métodos deterministas pueden no ser capaces de obtener fácilmente una solución global en un tiempo razonable y, además, estos enfoques no garantizan una solución óptima. Una de las causas de esta problemática radica en el hecho de que algunas de estas funciones poseen mínimos locales, de modo que el método de búsqueda de la solución queda *atrapado* en uno de dichos mínimos, por lo que encontrar un óptimo absoluto puede resultar complejo o incluso no llegar a producirse.

4.1. Algoritmos metaheurísticos

Los métodos metaheurísticos son capaces tanto de evitar los mínimos locales como de acelerar la convergencia y cumplir con las restricciones de la función a optimizar. Estos métodos emplean técnicas de búsqueda guiada (aleatoria) para resolver el problema con éxito, aunque su adecuación con respecto al problema objetivo no puede demostrarse formalmente. De hecho, los métodos metaheurísticos son capaces de alcanzar la solución óptima o subóptima sin disponer de toda la información sobre los problemas a resolver. Los algoritmos metaheurísticos no utilizan el gradiente de la función, lo que significa que no es necesario que la función sea diferenciable, como exigen los métodos de optimización determinista [12].

Algunos de los algoritmos metaheurísticos de optimización más conocidos son los algoritmos genéticos (GA) y sus variantes, la evolución diferencial (DE – *Differential Evolution*), la optimización por enjambre de partículas (PSO) y sus variantes, el algoritmo de recocido simulado (SA – *Simulated Annealing*), el algoritmo de búsqueda tabú (TS - *Taboo Search*), la estrategia evolutiva (ES – *Evolutionary Strategy*), la programación evolutiva (EP – Evolutionary Programming), la programación genética (GP – *Genetic Programming*), la Colonia Artificial de Abejas (ABC), el salto de rana barajado (SFL – *Shuffled Frog Leaping*), la optimización por colonia de hormigas (ACO), el algoritmo de la mosca de fuego (FA – *Firefly Algorithm*), y otros. Algunos algoritmos basados en fenómenos de la naturaleza destacables son la búsqueda armónica (HS – *Harmony Search*), la búsqueda del león (LS – *Lion Search*), el algoritmo de búsqueda gravitacional (GSA – *Gravitational Search Algorithm*), la optimización basada en la biogeografía (BBO – *Biogeography-based Optimization*), el método de la explosión de granadas (GEM – *Grenade Explosion Method*) y otros.

El éxito en la obtención del óptimo de la mayoría de estos algoritmos está muy condicionado por sus parámetros específicos. Por citar algunos ejemplos, el GA necesita que se ajusten correctamente la probabilidad de cruce, la probabilidad de mutación y el operador de selección.; el algoritmo SA necesita que se ajusten la temperatura inicial de recocido y el programa de enfriamiento; los parámetros específicos de PSO son el peso de la inercia y los parámetros sociales y cognitivos; HSA necesita que se ajusten correctamente la tasa de consideración de la memoria de armonía y el número de improvisaciones; y la tasa de inmigración y la tasa de emigración, deben ajustarse para BBO.

4.2. Algoritmos basados en poblaciones

Los algoritmos de optimización TLBO y Jaya tienen la ventaja de no necesitar un ajuste de parámetros específicos [11][25]. Sólo utilizan parámetros generales, como el número de iteraciones y la dimensión de la población. Aunque son muy similares, TLBO consta de dos etapas diferenciadas en cada una de iteraciones (etapas de Maestro y de Aprendizaje), mientras que Jaya sólo realiza una etapa en cada una de las iteraciones. El algoritmo Jaya ha generado un creciente interés en muchas áreas científicas e ingenieriles debido a su simplicidad y eficacia. El algoritmo TLBO suele converger a la solución más rápidamente que Jaya, pero TLBO es más complejo que Jaya, no sólo por tener dos etapas por iteración, sino también por las operaciones necesarias en cada etapa, que suponen un mayor coste computacional en comparación con las de Jaya. En este trabajo de investigación se trabajará con la versión original de los algoritmos Jaya y TLBO, aunque se pueden encontrar diversas variantes en la literatura [26], [27].

4.3. Jaya

El algoritmo Jaya es un método de optimización basado en poblaciones propuesto en [7] para resolver problemas de optimización con y sin restricciones. A diferencia de otros algoritmos heurísticos basados en la población, Jaya no tiene ningún parámetro de ajuste específico. Al igual que ocurre con TLBO, sólo se debe configurar el tamaño de la población y las generaciones, es decir, el número de iteraciones. Este algoritmo se basa en el hecho de que la solución óptima para un determinado problema puede obtenerse desplazándose hacia la mejor solución parcial y, al mismo tiempo, alejándose de la peor solución. En comparación con otros métodos de optimización, Jaya ha demostrado obtener mejores resultados en cuanto a los valores mejores, medios y peores de la solución obtenida a la hora de optimizar diferentes funciones sin restricciones [27].

La descripción del algoritmo Jaya se expone a continuación. Sea $f(x)$ la función objetivo a minimizar (o maximizar). En cualquier generación o iteración i, existen p soluciones candidatas o individuos (cada individuo posee un identificador $k = 1, 2, ..., p$). y cada individuo posee n variables de diseño (que quedan identificadas mediante el índice $j = 1, 2, ..., n$). El mejor candidato es el que obtiene el mejor valor al evaluar $f(x)$ (es decir, $f(x)_{best}$) de todas las soluciones candidatas, y el peor candidato es el que obtiene el peor valor al evaluar $f(x)$ (es decir, $f(x)_{worst}$) de todas las soluciones candidatas. Si $X_{j,k,i}$ es el valor de la j-ésima variable para el k-ésimo candidato durante la i-ésima iteración, este valor se modifica mediante la siguiente ecuación:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}\,(X_{j,\text{best},i} - |X_{j,k,i}|) - r_{2,j,i}\,(X_{j,\text{worst},i} - |X_{j,k,i}|) \quad (1)$$

donde $X_{j,\text{best},i}$ es el valor de la variable j para el mejor candidato, y $X_{j,\text{worst},i}$ es el valor de la variable j para el peor candidato.
En la ecuación anterior, $X'_{j,k,i}$ es el valor actualizado de $X_{j,k,i}$, y $r_{1,j,i}$ y $r_{2,j,i}$ son dos números aleatorios en el rango [0, 1], para la j-ésima variable calculada en la i-ésima iteración. El término $r_{1,j,i}\,(X_{j,\text{best},i} - |X_{j,k,i}|)$ indica la tendencia del algoritmo a acercarse a la mejor solución, mientras que el término $r_{2,j,i}\,(X_{j,\text{worst},i} - |X_{j,k,i}|)$ indica la tendencia del algoritmo a evitar la peor solución. Obviamente, el nuevo candidato $X'_{j,k,i}$ se acepta sólo si proporciona una mejor evaluación de la función. Todos los valores de los individuos aceptados al final de cada una de las iteraciones se mantienen, por lo que estos valores se convierten en los individuos de la siguiente generación o iteración.

4.4. TLBO

El TLBO es un método de optimización eficiente que ha sido utilizado para problemas de ingeniería, entre otros [13], [28]-[30]. Este método busca un profesor o *Teacher*, es decir, el mejor individuo, que probablemente influya en los aprendices o *Learners*, es decir, en el resto de los individuos de la población, para mejorar las características de éstos. Como método basado en la población, utiliza una población de individuos (soluciones candidatas) para inferir la solución global. El algoritmo TLBO se divide en dos etapas: la primera es la Etapa del Profesor o *Teaching* y la segunda es la Etapa de Aprendizaje o *Learning*. En la Etapa del Profesor, los individuos aprenden del mejor individuo que proporciona la mejor solución de toda la población, y en la Etapa de Aprendizaje los individuos intentan aprender por medio de la interacción entre individuos de la población.

El comportamiento del algoritmo TLBO se explica a continuación. Se crea la población y se realiza la inicialización de los individuos (valores iniciales de las variables de diseño) mediante la asignación de datos aleatorios. Tras crear la primera generación de individuos, el algoritmo itera actualizando la población. Al inicio de cada una de las iteraciones, se determina el *Profesor* de la población (mejor solución) y se calcula así mismo el valor medio para cada variable de diseño. Estos datos se utilizan en las dos etapas principales del algoritmo: la Etapa del Profesor y la Etapa de Aprendizaje. En la Etapa de Aprendizaje, la mejor solución obtenida (X_{best}) de la generación actual se utiliza para crear una nueva versión de cada individuo (X_{new}), utilizando la siguiente ecuación:

$$Xnew(i,j) = X(i,j) + rand(0,1) \cdot (Xbest(j), -TFactor \cdot Xm(j)) \quad (2)$$

$X(i, j)$ corresponde a la variable de diseño j del individuo i, y se modifica utilizando el valor del Profesor $X_{best}(j)$, la media de la variable $X_m(j)$ y el parámetro *TFactor*. Este parámetro adopta el valor entero 1 ó 2 y se calcula mediante la siguiente expresión:

$$TFactor = round(1 + rand\,(0,1)) \quad (3)$$

Tras generar un nuevo individuo, se somete a evaluación. Si el resultado de la evaluación del nuevo individuo es mejor que el del individuo original, este individuo obsoleto es sustituido por el nuevo. En la Etapa de

Aprendizaje, a cada individuo se le asigna un rival aleatorio en la población. Ambos individuos se someten para su evaluación. El individuo con una mejor evaluación es etiquetado como el Profesor Parcial o *Partial Teacher*, y el otro es etiquetado como el Aprendiz o *Learner*, de modo que se genera un nuevo individuo utilizando la siguiente expresión:

$$Xnew\,(i,j) \;=\; X\,(i,j) \;+\; rand\,(0,1) \;\cdot\; (PartialTeacher\,(j) \;-\; Learner(j)) \qquad (4)$$

Al generar cada $X_{new}(i)$, este nuevo individuo es evaluado y comparado con el individuo original. Si la evaluación del nuevo individuo mejora la del antiguo, el nuevo individuo sustituye al original.

4.5. TLBO discreto - DTLBO

Durante la investigación, se estudiaron diversos problemas de ingeniería en los que, por la naturaleza discreta y combinatoria que poseen, no es posible aplicar directamente el algoritmo TLBO original. Por ello se planteó realizar una implementación de una versión discreta del propio TLBO que pudiese ser capaz de dar solución a problemas discretos. Para realizar estas modificaciones sobre el algoritmo TLBO, se partió del estudio propuesto en [35], realizando modificaciones para adaptarlo a las necesidades de los problemas estudiados. Dichos problemas son el Problema del Viajante de Comercio (TSP – *Travelling Salesman Problem*) [28-30] y el problema del taladrado CNC o CNC *drilling* (CNC – *Computer Numerical Control*). El TSP se orienta a encontrar el camino más corto que recorre todos y cada uno de los nodos de una malla una y sólo una vez. Por su parte, el CNC *drilling* posee características similares al TSP, ya que se trata del procedimiento de corte de materiales con una máquina de control numérico por computadora. El proceso de mecanizado CNC interpreta el modelo CAD 3D y traduce los datos a la máquina de CNC [23-25]. Tomando como partida este problema se realizó una adaptación del TLBO Discreto (DTLBO) propuesto para poder aplicarlo a dicho problema.

La representación de los individuos en el DTLBO cambia sustancialmente con respecto al TLBO del dominio continuo. En la versión discreta del algoritmo, los individuos representan rutas a través de un conjunto de nodos, de modo que cada variable de diseño dentro de un individuo corresponde a un nodo o lugar. Como se muestra en la Figura 1, si hay que recorrer un conjunto de ocho nodos (v0, v1... v7), un individuo queda representado como ocho valores discretos que determinan el orden de recorrido de los nodos: v7 → v0 → v1 → v6 → v4 → v2 → v5 → v3

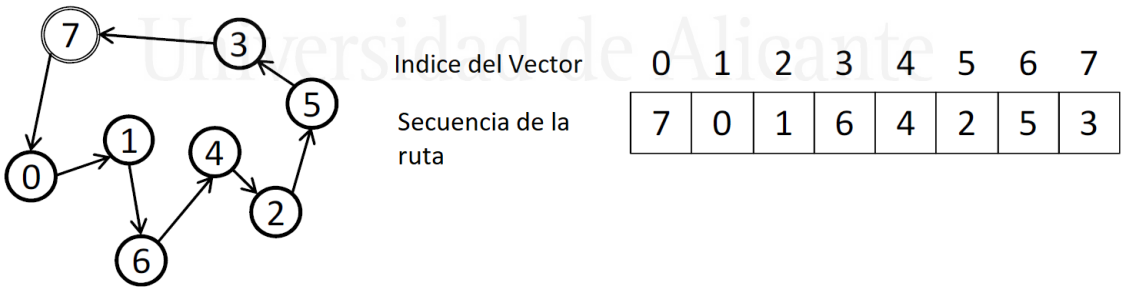


Figura 1. Representación de un individuo en DTLBO

A diferencia del algoritmo propuesto en [35], que utiliza cuatro subpoblaciones completamente aleatorias, en este trabajo se ha modificado el DTLBO de forma que uno de los individuos no se genera aleatoriamente, sino que se le asignan valores iniciales basados en una estrategia codiciosa (*greedy*). Con ello se pretende acelerar la convergencia del algoritmo y preservar su capacidad de evitar los mínimos locales manteniendo la aleatoriedad en las subpoblaciones.

4.5.1. DTLBO. Etapa de Profesor

En esta etapa, se selecciona un Profesor Parcial (*PartialTeacher*) dentro de cada subpoblación. Este individuo se utiliza para mejorar a los alumnos de cada subpoblación teniendo en cuenta también el valor medio (*Mean*) de la misma. Además, para no perder la visión global de la ruta, se selecciona un Profesor Global (*Teacher*) como el mejor individuo de toda la población.

Hay que tener en cuenta que el cálculo del individuo medio podría resultar en un individuo que no sea viable respecto de las condiciones intrínsecas del problema TSP (nodos repetidos y ausencia de algunos de los nodos en la ruta), por lo que hay que realizar una serie de comprobaciones para verificar su viabilidad y, en caso de que sea necesario, realizar las modificaciones pertinentes para lograr dicha viabilidad.

La operación de cruce (*crossover*) se utiliza en el DTLBO para generar nuevos individuos a partir de los existentes. Esta operación se indica con el símbolo □. En esta versión del algoritmo, se introdujeron cuatro formas diferentes de crear nuevos individuos a partir de cruces, que se muestran en la ecuación (5). A diferencia del algoritmo propuesto en [35], en el que se asigna un cruce fijo entre individuos, se ha modificado el algoritmo en el presente trabajo de modo que se realiza una selección aleatoria para cada operación de cruce. Esta aleatoriedad evita caer en mínimos locales y aumenta la variabilidad de las poblaciones.

$$\begin{aligned} Xnew(i) &= \mathrm{X(i)} \otimes Teacher \\ Xnew(i) &= X(i) \otimes PartialTeacher\ (i) \\ Xnew(i) &= X(i) \otimes Mean(i) \\ Xnew(i) &= PartialTeacher(i) \otimes Mean(i) \end{aligned} \qquad (5)$$

El funcionamiento de la operación de cruce se muestra en la figura 2, en la que dos individuos *A* y *B* deben generar uno nuevo llamado A_c. Las posiciones inicial y final del nuevo individuo se seleccionan al azar y así el nuevo individuo A_c sustituye las posiciones originales de *A* por las del individuo *B*. Después de realizar este cruce, la comprobación de viabilidad debe aplicarse sobre el nuevo individuo.

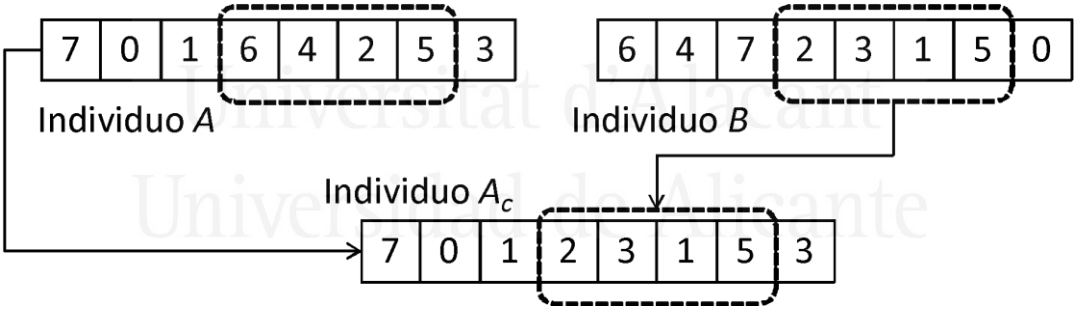


Figura 2. Ejemplo de la operación de cruce.

Para aumentar la variabilidad de las poblaciones, se incluye un operador de mutación. La figura 3 muestra el funcionamiento de este operador. Dado un individuo A_c, las posiciones inicial y final se seleccionan al azar; en el ejemplo actual, son las posiciones 3 y 6. Los elementos incluidos en esta secuencia (elementos de las posiciones 3, 4, 5 y 6) se invierten para crear el individuo mutado A_{cm}

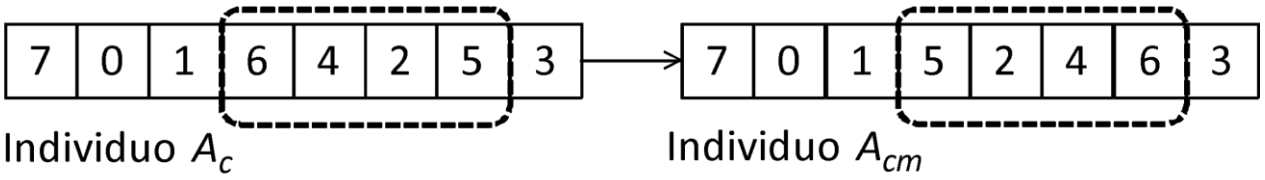


Figura 3. Ejemplo de la operación de mutación.

4.5.2. DTLBO. Etapa de Aprendizaje

En la etapa de aprendizaje se elige un individuo *k* al azar para cada individuo *i* de la subpoblación. El nuevo individuo se creado a partir de la ecuación (6), donde ⊗ representa la operación de cruce, que funciona del mismo modo que la operación de cruce en la etapa de Profesor. Será necesario aplicar a continuación la comprobación de viabilidad en el nuevo individuo *Xnew*(*i*), así como aplicar después el operador de mutación tal y como se describió para la etapa de Profesor.

$$Xnew(i) = X(i) \otimes X(k) \tag{6}$$

En [35], una de las conclusiones expuestas es que el rendimiento del DTLBO decrece en gran medida cuando las ciudades tienen un gran número de nodos a recorrer. Por este motivo, esta investigación se centró en una revisión de este algoritmo basada en la mejora de su convergencia y en una implementación paralela que permita aprovechar el rendimiento del multiprocesamiento basado en las GPU.

4.6. Paralelización de algoritmos

Una vez estudiados los algoritmos y realizadas sus implementaciones secuenciales, se plantea su implementación paralela. Para poder realizar una implementación paralela, en el caso del presente estudio una implementación mediante CUDA, se debe analizar el algoritmo para poder extraer toda la información necesaria que permita realizar una implementación adecuada. Para la implementación paralela de algoritmos, existe una metodología de trabajo específica ya que no todos los algoritmos pueden implementarse de una forma eficiente mediante CUDA, puesto que éste no es un entorno multiproceso de propósito general.

Antes de comenzar con la implementación, existen algunos conceptos relativos a la arquitectura CUDA que se deben conocer y gestionar correctamente. Se explicará para ello a continuación la organización de los hilos (*threads*) y de la memoria en la arquitectura CUDA; ambos serán componentes esenciales a la hora de realizar la implementación de los algoritmos.

Al abordar la implementación de un algoritmo mediante el uso de técnicas de paralelización en GPU, y en este caso en la arquitectura CUDA, existen varios factores que se deben tener en cuenta para una correcta implementación. Este tipo de arquitecturas no son de propósito general, por lo que si el algoritmo a implementar no cumple con los requisitos y restricciones (algunos de ellos a nivel de lenguaje de programación) de estas arquitecturas, puede haber algunos casos en los que el algoritmo no se pueda implementar correctamente o, si se logra su implementación, incluso empeorar el coste computacional del algoritmo en lugar de mejorar el rendimiento. Otro factor crítico a tener en cuenta es la memoria, tanto en la transferencia de datos desde la CPU a la GPU como en su organización para mejorar los tiempos de acceso a la memoria. En cuanto a la organización de la memoria realizada en este trabajo, se ha abordado la mejora de los accesos a la memoria por los diferentes hilos como uno de los puntos clave para mejorar la eficiencia de la implementación de CUDA.

Para la implementación paralela de TLBO y Jaya, el conjunto completo de núcleos de GPU se dividió en bloques. Cada bloque es independiente de los demás y realiza una ejecución independiente del algoritmo. Al mismo tiempo, los núcleos de un bloque se configuraron en una matriz 2D, de modo que cada hilo se ejecuta en un solo núcleo. Los subprocesos dentro del mismo bloque comparten datos a través de un banco de memoria compartida, por lo que cada fila dentro de un bloque corresponde a un individuo (solución candidata), mientras que cada columna dentro de una fila corresponde a una variable de diseño. De esta forma, cada subproceso realiza una evaluación parcial del individuo, por lo que se requiere una operación de reducción para obtener la evaluación general (generalmente la suma de las evaluaciones parciales de cada subproceso), que se lleva a cabo por el primer subproceso dentro de la fila de un individuo.

Inicialmente en la investigación se utilizó CUDA 8 como entorno de desarrollo, pero durante el trabajo apareció CUDA 9 con algunos cambios que tenían un impacto directo en la implementación. Hubo dos nuevas

funcionalidades que fueron utilizadas en los últimos trabajos de investigación y tuvieron un impacto significativo en los resultados obtenidos a nivel de aceleración. En esta nueva revisión de CUDA se hace una gestión de la memoria de una forma más inteligente y transparente para el usuario, siendo el propio entorno el que puede gestionar la memoria en su nivel óptimo. Además, otra nueva funcionalidad que se utilizó en las implementaciones es el uso de cuRAND, una función propia de CUDA a nivel de GPU para obtener números aleatorios. En la implementación inicial todo el cálculo de números aleatorios se realizaba a nivel de *Host Device*, lo cual tiene impacto negativo en el tiempo de ejecución del algoritmo, y cabe destacar que en cada iteración de ambos algoritmos se utiliza de forma intensiva el cálculo de números aleatorios. Al disponer de esta nueva función a nivel de GPU, realizando simplemente el cambio de la llamada a la función de cálculo de números aleatorios se obtuvo una mejora en la aceleración de los algoritmos sin un impacto significativo en el cálculo de las soluciones. CUDA ya dispone de una versión 11 estable que sigue añadiendo mejoras y nuevas características que podrían incluirse en posteriores implementaciones para mejorar los resultados obtenidos.

4.6.1. Paralelización de Jaya

En la propuesta de implementación paralela del presente trabajo, se tienen en cuenta en el diseño de la organización de los bloques tanto el número de variables como el tamaño de la población. Una de las limitaciones actuales de CUDA es el número máximo de hilos (1024 hilos por bloque). Durante la etapa inicial de la implementación se creará una matriz 2D donde se dispone de tantas columnas como variables y se utilizará una fila por cada individuo de la población. Además, se dispondrá de un vector con las evaluaciones de cada individuo para poder ser consultado por el algoritmo. Esta disposición inicial cambió en los últimos trabajos añadiendo la evaluación de la función de cada individuo dentro de la matriz inicial. Como se mencionó anteriormente, la memoria compartida de la GPU es clave para el comportamiento eficiente debido a que se utiliza para almacenar toda la población, los valores parciales de las evaluaciones de las funciones, los nuevos candidatos y otros datos relacionados con la implementación del algoritmo. Por ejemplo, los índices de las mejores y peores soluciones actuales.

En el Algoritmo 1 se muestra el esqueleto de la implementación GPU del algoritmo Jaya.

ALGORITMO 1
ESQUELETO DE LA IMPLEMENTACIÓN GPU DE JAYA

```
1:desde Run = 1 a Runs en un bucle paralelo //GPU block B
2: for all threads(i,j)∈(Pop,VARS) bucle paralelo
3:  CreatePopulation(i,j)   // Crear una nueva población
4:  EvaluatePopulation(i,j) // Evaluar el resultado de la función F
5:  SyncThreads
6:  for iter = 1 to Iterations do
7:    best = GetBest(Pop) // Recuperar la mejor solución
8:    worst = GetWorst(Pop) // Recuperar la peor solución
9:    SyncThreads()
10:   (r1,r2) = ObtainRandomNumbers()// Obtener numeros aleatorios para ser utilizados
11:   varnew(i,j) =
         GenerateNewIndividual(i,best,r1,worst,r2) // Generar un nuevo invividuo
12:   SyncThreads()
13:   Fnew(i) = EvaluateNewIndividual(varnew)// Evaluación del nuevo individuo
14:   SyncThreads()
15:   if Fnew(i)<F(i) then var(i,j) = varnew(i,j)
16: end for
17: Store(GetBest(Pop))
18: end for
19:end for // final del bucle
```

4.6.2. Paralelización de TLBO

En el Algoritmo 2, se muestra el esqueleto de la implementación principal GPU del TLBO. Así mismo, en el Algoritmo 3 se muestra la implementación de la etapa de Profesor, y por último en el Algoritmo 4 la etapa de Aprendizaje. Al ser ambos algoritmos de una naturaleza parecida, ya que son algoritmos basados en poblaciones, la estructura inicial de ambos es muy similar y comparten algunas funciones como son la de crear poblaciones, evaluar poblaciones y evaluar individuos. Cabe destacar que el uso de la memoria de TLBO es

mayor que la de Jaya, ya que necesita guardar más valores en las iteraciones además de compartir información entre individuos de la población. Este uso de la memoria hizo más limitada la aplicación del algoritmo a diferentes problemas y, sobre todo, limitó el tamaño de los problemas. Por ello, durante el trabajo de investigación la implementación de TLBO se fue mejorando para un uso más eficiente de la memoria al igual que el uso de algunas técnicas para la minimización de transferencias de memoria entre la GPU y el host.

ALGORITMO 2
ESQUELETO DE LA IMPLEMENTACIÓN GPU DE TLBO

```
1:desde Run = 1 a Runs en un bucle paralelo //GPU block B
2: for all threads(i,j)∈(Pop,VARS) bucle paralelo
3:   CreatePopulation(i,j) //Crear una nueva población
4:   EvaluatePopulation(i,j) //Evaluar el resultado de la función F
5:   SyncThreads
6:   for iter = 1 to Iterations do
7:     teacher = GetBest(Pop) //Obtener mejor solución
8:     SyncThreads()
9:     CalculateMean(j) //Compute mean of
                          each j ∈ VARS
10:    SyncThreads()
11:    TeacherStage(i,j,teacher)
12:    SyncThreads()
13:    LearnerStage(i,j)
14:    SyncThreads()
15:  end for
16:  Store(GetBest(Pop))
17: end for
19:end for
20:Obtener y almacenar mejor solución
```

ALGORITMO 3
FUNCIÓN DE LA ETAPA DE PROFESOR THREAD(I, J)

```
1: TeacherStage(i,j,teacher) ∈ (Pop,VARS,Pop)
2: {
3:  r = ObtainRandomNumber()
4:  t_factor = ObtainTFactor()
5:  varnew (i,j)=
        GenerateNewIndividual(r,t_factor,teacher)
6:  Fnew(i) = EvaluateNewIndividual(varnew)
7:  SyncThreads
8:  if Fnew(i)<F(i) then var(i,j) = varnew(i,j)
9: }
```

ALGORITMO 4
FUNCIÓN DE LA ETAPA DE APRENDIZAJE PARA THREAD(I,J)

```
1: LearnerStage(i,j) ∈ (Pop,VARS)
2: {
3:  learner = ObtainRandomLearnerFromPopulation()
4:  (r1,r2) = ObtainRandomNumbers()
5:  (best,worst) = CompareLearnerWithIndividual()
6:  varnew(i,j) =
          GenerateNewIndividual(best,r1,worst,r2)
7:  Fnew(i) = EvaluateNewIndividual(varnew)
8:  SyncThreads
9:  if Fnew(i)<F(i) then var(i,j) = varnew(i,j)
10:}
```

4.6.3. Paralelización de DLTBO

Como se explicó anteriormente, la aplicación de una serie de modificaciones al algoritmo TLBO puede lograr resultados significativos en problemas combinatorios discretos, en este caso, en problemas del tipo TSP o CNC. Se obtienen resultados satisfactorios en comparación con otros algoritmos de optimización, como se demostró en [35].

Los cambios implementados en el TLBO original hacen que las diferentes etapas sean más complejas y agregan un mayor coste computacional al algoritmo, aunque su estructura fundamental es la reflejada en el apartado anterior. Como consecuencia, las iteraciones del DTLBO son significativamente más lentas que las del TLBO

original y, por lo tanto, se penaliza el tiempo de cálculo. Con el objetivo de minimizar el impacto de estas modificaciones y el coste extra de la computación en el rendimiento del algoritmo, se desarrolló una implementación paralela del algoritmo utilizando una arquitectura CUDA en un entorno GPU. Aunque la paralelización de un algoritmo que utiliza CUDA no siempre es la mejor solución cuando se aplican técnicas de paralelización, las características específicas de DTLBO pueden explotarse para proporcionar una mejora notable en el rendimiento en comparación con las implementaciones secuenciales.

4.7. Experimentación

Dentro de la investigación, la experimentación se planteó en dos fases. En la primera fase se realizó la experimentación utilizando funciones de optimización extraídas de la literatura actual para comprobar la eficiencia de los algoritmos y realizar comparativas entre ellos y también con otros algoritmos. En la segunda fase, una vez comprobado que las implementaciones realizadas cumplían las expectativas, se realizó la experimentación utilizando problemas reales, objetivo final de la investigación. A continuación, se describen ambas fases de experimentación.

4.7.1. Funciones de optimización

En una etapa inicial de la experimentación, se seleccionó un conjunto de funciones de optimización extraídas de la literatura actual, las cuales son utilizadas para la comprobación de los algoritmos y para la comparación de los resultados obtenidos tanto a nivel de tiempo como de precisión de las soluciones. Estas funciones están bien documentadas y se dispone de las soluciones junto a los dominios de sus variables o argumentos. Dentro de estas funciones de optimización, existen diferentes grupos según la naturaleza de la función; para este trabajo, se eligieron funciones con varios mínimos locales ya que Jaya y TLBO son algoritmos que gestionan adecuadamente la evitación de bloqueo en la convergencia al caer en mínimos locales. Durante la experimentación, se utilizaron funciones de diversa complejidad, tanto en lo que respecta a la cantidad de variables que intervienen en ellas como en el coste de los cálculos necesarios para evaluar las funciones. Para la aplicación de los algoritmos a estas funciones, se tuvo en cuenta las limitaciones hardware (sobre todo a nivel de memoria) del entorno de experimentación, por lo cual se tuvo que ajustar los parámetros de tamaño de población y tamaño de individuo de los algoritmos a dichas funciones.

4.7.2. Problemas reales

Una vez comprobado que los algoritmos cumplían las expectativas con las funciones de optimización, se planteó la aplicación de los algoritmos a problemas reales como objetivo final de la investigación. Para poder aplicar los algoritmos a estos problemas reales, fue necesario realizar diversas modificaciones a los algoritmos como se describió previamente. Los problemas reales elegidos durante la investigación para realizar la investigación fueron del tipo TSP. El problema del CNC *drilling*, aunque posee diversas peculiaridades, puede englobarse perfectamente dentro de la categoría de problemas de tipo TSP.

Los problemas del tipo TSP son muy frecuentes en la actualidad. La optimización de rutas es utilizada por ejemplo en un GPS para gestionar de forma óptima un viaje o un reparto de mercancías; el corte de un patrón de una pieza de calzado tiene una repercusión directa, no solo en el tiempo, sino también en los recursos implicados en el proceso. Además, si es un proceso que se va a repetir continuamente, esta optimización puede ser una mejora clave a nivel de competitividad industrial, tanto por los tiempos de producción como por la minimización del uso de recursos claves (como puede ser el uso de una máquina o de una parte de la cadena de producción). Para poder evaluar la implementación realizada se utilizó una batería de problemas de los disponibles en TSPLIB los cuales son utilizados para validación de algoritmos de optimización de TSP [37].

Por último, se realizó un experimento de CNC *drilling* utilizando como datos de entrada ficheros con los puntos de soldadura de una placa PCB real. Es muy común el uso de robots o mesas de corte adaptadas para la realización de la mayoría de las soldaduras en los procesos de producción actuales de placas PCB. En los ficheros, se halla información a nivel de diseño de las placas PCB junto con las coordenadas de los puntos donde están las conexiones a soldar. Para la experimentación y la obtención de resultados a nivel de tiempo

no se tuvo en cuenta el tiempo necesario para el movimiento de la punta de soldado, si fuera necesario, al igual que el tiempo de soldado ya que ambos serían constantes en la solución iterativa y la manycore; es decir, este tiempo no es acelerado en la aplicación del algoritmo.

Resultados

Los resultados obtenidos en los diferentes experimentos fueron satisfactorios, ya que en la gran mayoría de ellos se consiguieron mejoras en los tiempos de obtención de las soluciones y, por tanto, todos ellos fueron acelerados al aplicar las técnicas descritas en el trabajo. El factor de aceleración conseguido en los diferentes experimentos tuvo una relación directa en muchos de ellos con la naturaleza del problema y las variables utilizadas, como puede ser el tamaño de la población o el número de variables. Algunos de los valores de estas variables fueron descartados ya que hay que tener en cuenta que estos algoritmos, sobre todo para la implementación manycore, están concebidos para poblaciones con un mínimo de tamaño, puesto que en poblaciones muy pequeñas el tiempo necesario para algunas operaciones propias de las arquitecturas manycore, como la inicialización, transferencia de memoria y otras, pueden tener un impacto significativo en los resultados obteniendo, incluso produciendo aceleraciones (*speedup*) negativas. Por tanto, estos algoritmos necesitan unos valores mínimos en cuanto a los tamaños tanto de las poblaciones como del número de variables para obtener mejoras, ya que para problemas de tamaño reducido la paralelización no suele ser una opción viable.

En la experimentación no sólo se buscó que los resultados de aceleración fueran satisfactorios, sino que también las soluciones obtenidas tuvieran calidad con respecto a la precisión de los resultados. En la realización de los experimentos con solución conocida, se definieron en cada uno de ellos unos criterios de tolerancia para considerar una solución como válida o de calidad cumpliendo con dichos criterios de precisión. También hay que destacar que en varios de los experimentos se obtuvieron exactamente las soluciones óptimas documentadas.

Con todo ello, los resultados obtenidos permiten corroborar que se ha cumplido con los objetivos planteados inicialmente en este trabajo de investigación, y dejan abiertas líneas de investigación para, por un lado, aplicar estos algoritmos a nuevos problemas de ingeniería y, por otro lado, para continuar con la mejora de la implementación de los algoritmos implementados mediante la aplicación de todas las mejoras sobre estas arquitecturas. Por último, se ha comprobado que este tipo de implementaciones tienen aplicación en la industria, de modo que la aceleración de diversos algoritmos de optimización en diferentes segmentos de la cadena de producción puede constituir un elemento esencial para la mejora de la productividad.

4.8. Bibliografía

[1] A. J. Umbarkar, M. S. Joshi, and P. D. Sheth, “OpenMP dual population genetic algorithm for solving constrained optimization problems,” Int. J. Inf. Eng. Electron. Bus., vol. 7, no. 1, pp. 59-65, 2015.

[2] R. Baños, J. Ortega, and C. Gil, “Comparing multicore implementations of evolutionary meta-heuristics for transportation problems,” Ann. Multicore GPU Program., vol. 1, no. 1, pp. 9-17, 2014.

[3] R. Baños, J. Ortega, and C. Gil, “Hybrid MPI/OpenMP parallel evolutionary algorithms for vehicle routing problems,” in Proc. Eur. Conf. Appl. Evol. Comput., Granada, Spain, 2014, pp. 653-664.

[4] P. Delisle, M. Krajecki, M. Gravel, and C. Gagné, ``Parallel implementation of an ant colony optimization metaheuristic with OPENMP," in Proc. 3rd Eur. Workshop OpenMP (EWOMP), Barcelona, Spain, 2001, pp. 8-12.

[5] Y. Tan and K. Ding, “A survey on GPU-based implementation of swarm intelligence algorithms," IEEE Trans. Cybern., vol. 46, no. 9, pp. 2028-2041, Sep. 2016.

[6] G.-H. Luo, S.-K. Huang, Y.-S. Chang, and S.-M. Yuan, “A parallel Bees Algorithm implementation on GPU," J. Syst. Archit., vol. 60, pp. 271-279, Mar. 2014.

[7] A. Delévacq, P. Delisle, M. Gravel, and M. Krajecki, “Parallel ant colony optimization on graphics

processing units," J. Parallel Distrib. Comput.,vol. 73, pp. 5261, Jan. 2013.

[8] L. Mussi, F. Daolio, and S. Cagnoni, “Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture,” Inf. Sci., vol. 181, pp. 46424657, Oct. 2011.

[9] L. de P. Veronese and R. A. Krohling, “Differential evolution algorithm on the GPU with C-CUDA,” in Proc. IEEE Congr. Evol. Comput., Jul. 2010, pp. 1-7.

[10] Y. Zhou and Y. Tan, “GPU-based parallel particle swarm optimization” in Proc. IEEE Congr. Evol. Comput., May 2009, pp. 14931500.

[12] Askarzadeh, A. “A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm.” Comput. Struct. 2016, 169, 1–12.

[13] R. V. Rao, “Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems,” Int. J. Ind. Eng. Comput., vol. 7, no. 1, pp. 19-34, 2016.

[14] M. Ebraheem and T. R. Jyothsna, “Comparative performance evaluation of teaching learning based optimization against genetic algorithm on benchmark functions,” in Proc. IEEE Power, Commun. Inf. Technol. Conf. (PCITC), Oct. 2015, pp. 327-331.

[15] S. R. Shah and S. B. Takmare, “A review of methodologies of TLBO algorithm to test the performance of benchmark functions,” Program. Device Circuits Syst., vol. 9, no. 7, pp. 141-145, 2017.

[16] R. Azizipanah-Abarghooee, M. Malekpour, M. Zare, and V. Terzija, "A new inertia emulator and fuzzy-based LFC to support inertial and governor responses using Jaya algorithm," in Proc. IEEE Power and Energy Soc. Gen. Meeting (PESGM), Jul. 2016, pp. 1-5.

[17] K. Abhishek, V. R. Kumar, S. Datta, and S. S. Mahapatra, “Application of JAYA algorithm for the optimization of machining performance characteristics during the turning of CFRP (epoxy) composites: Comparison with TLBO, GA, and ICA,” Eng. Comput., vol. 33, pp. 457-475, Jul. 2017.

[18] M. Bhoye, M. H. Pandya, S. Valvi, I. N. Trivedi, P. Jangir, and S. A. Parmar, “An emission constraint economic load dispatch problem solution with microgrid using JAYA algorithm,” in Proc. Int. Conf. Energy Eficient Technol. Sustainability (ICEETS), Apr. 2016, pp. 497-502.

[19] I. N. Trivedi, S. N. Purohit, P. Jangir, and M. T. Bhoye, “Environment dispatch of distributed energy resources in a microgrid using JAYA algorithm,” in Proc. 2nd Int. Conf. Adv. Elect. Electron. Inf. Commun. Bio-Inform. (AEEICB), Feb. 2016, pp. 224-228.

[20] S. Mishra and P. K. Ray, “Power quality improvement using photovoltaic fed DSTATCOM based on JAYA optimization,” IEEE Trans. Sustain. Energy, vol. 7, no. 4, pp. 1672-1680, Oct. 2016.

[21] A. J. Umbarkar, N. M. Rothe, and A. S. Sathe, “OpenMP teaching-learning based optimization algorithm over multi-core system,” Int. J. Intell. Syst. Appl., vol. 7, no. 7, pp. 57-65, 2015.

[22] R. V. Rao and G. G.Waghmare, “A new optimization algorithm for solving complex constrained design optimization problems,” Eng. Optim., vol. 49, no. 1, pp. 60-83, 2016.

[23] Abidin, N.W.Z., Rashid, Ab, Mohamed, M.F.F., N, N.M.Z., 2019]. “A review of multi-holes drilling path optimization using soft computing approaches”. Arch. Comput.Methods Eng. 26 (1), 107–118.

[24] Bai, J., et al., 2013]. “A model induced max-min ant colony optimization for asym-metric traveling salesman problem.” Appl. Soft Comput. 13 (3), 1365–1375.

[25] Castelino, K., D’Souza, R., Wright, P.K., 2003]. “Toolpath optimization for minimiz-ing airtime during machining.” J. Manuf. Syst. 22, 173–180.

[26] X. He, J. Huang, Y. Rao, and L. Gao, “Chaotic teaching-learning-based optimization with Lévy ight for global numerical optimization,” Comput. Intell. Neurosci., vol. 2016, p. 43, Jan. 2016, Art. no. 8341275.

[27] J. Yu, C. H. Kim, A. Wadood, T. Khurshiad, and S.-B. Rhee, “A novel multi-population based chaotic JAYA algorithm with application in solving economic load dispatch problems,” Energies, vol. 11, no. 8, p. 1946, 2018.

[28] Benevolo, C.; Dameri, R.P.; D'Auria, B. Smart Mobility in Smart City. In Empowering Organizations, Lecture Notes in Information Systems and Organisation; Torre, T., Braccini, A., Spinelli, R., Eds.; Springer: Cham, Switzerland, 2016; Volume 11.

[29] Rizwan, P.; Suresh, K.; Babu, M.R. Real-Time Smart Traffic Management System for Smart Cities by using Internet of Things and Big Data. In Proceedings of the 2016 International Conference on Emerging Technological Trends (ICETT), New York, NY, USA, 1–6 August 2016.

[30] Nikitas, A.; Michalakopoulou, K.; Njoya, E.T.; Karampatzakis, D. Artificial Intelligence, Transport and the Smart City: Definitions and Dimensions of a New Mobility Era. Sustainability 2020, 12, 2789.

[31] Rao, R.V.; Patel, V. "An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems." Int. J. Ind. Eng. Comput. 2012, 3, 535–560.

[32] Rao, R.V.; Patel, V. "Comparative Performance of an elitist Teaching-Learning-Based Optimization algorithm for solving unconstrained optimization problems." Int. J. Ind. Eng. Comput. 2013, 4, 29–50.

[33] Ebraheem, M.; Jyothsna, T.R. "Comparative performance evaluation of Teaching Learning Based Optimization against genetic algorithm on benchmark functions." In Proceedings of the 2015 Power, Communication and Information Technology Conference

(PCITC), Bhubaneswar, India, 15–17 October 2015; pp. 327–331.

[34] Shah, S.R.; Takmare, S.B. "A Review of Methodologies of TLBO Algorithm to Test the Performance of Benchmark Functions." Program. Device Circuits Syst. 2017, 9, 141–145.

[35] Wu, L.; Zoua, F.; Chen, D. "Discrete Teaching-Learning-Based Optimization Algorithm for Traveling Salesman Problems." In Proceedings of the MATECWeb of Conferences 128, 02022 EDP Sciences (2017), Zhuhai, China, 23–24 September 2017.

[36] Arnautovic, M.; Curic, M.; Dolamic, E.; Nosovic, N. Parallelization of the ant colony optimization for the shortest path problem using OpenMP and CUDA. In Proceedings of the 2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2013.

[37] TSPLIB - Discrete and Combinatorial Optimization, Ruprecht-Karls-Universität Heidelberg. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Artículos publicados

A continuación, se muestran íntegramente los artículos publicados más destacados resultantes de la investigación realizada y que constituyen la columna vertebral de la tesis por compendio de publicaciones. En el Anexo se incluyen otros artículos publicados. Los artículos más relevantes de esta investigación son los siguientes:

- H. Rico-García, J.-L. Sánchez-Romero, A. Jimeno-Morenilla, H. Migallón-Gomis, H. Mora-Mora, R. V. Rao, "Comparison of High Performance Parallel Implementations of TLBO and Jaya Optimization Methods on Manycore GPU", *IEEE Access*, vol. 7, pp. 133822-133831, 2019, https://doi.org/10.1109/ACCESS.2019.2941086
- H. Rico-García, J.-L. Sánchez-Romero, H. Migallón-Gomis, R.V. Rao, "Parallel implementation of metaheuristics for optimizing tool path computation on CNC machining", *Computers in Industry*, vol. 123, pp. 103322, 2020 https://doi.org/10.1016/j.compind.2020.103322

- H. Rico-García, J.-L. Sánchez-Romero, A. Jimeno-Morenilla, H. Migallón-Gomis, "A Parallel Meta-Heuristic Approach to Reduce Vehicle Travel Time in Smart Cities", *Applied Sciences* 11, no. 2: 818. 2021 https://doi.org/10.3390/app11020818

Conclusiones

1. Conclusiones.

Durante este trabajo de investigación se ha realizado un estudio sobre la aplicación de algoritmos heurísticos a problemas de ingeniería y su posterior aceleración mediante técnicas de computación paralela con NVidia CUDA.

Con respecto a la aplicación a problemas reales de ingeniería, se ha demostrado, por un lado, que los algoritmos utilizados durante la investigación son totalmente validos en la obtención de soluciones y, por otro, que la aceleración de dichos algoritmos mediante técnicas de computación paralela puede hacer que dichos algoritmos sean más relevantes para la resolución de diversos problemas. A nivel de aplicación, también se ha comprobado que pueden realizarse modificaciones sobre los algoritmos originales para poder ser aplicados a otro tipo de problemas de diferente naturaleza. En el caso de esta investigación, se han realizado modificaciones a DTLBO, la versión discreta de TLBO, para poder ser utilizado en la resolución del problema de TSP como ejemplo de problema discreto aplicado al cálculo de rutas entre ciudades y para optimizar los desplazamientos durante la soldadura de una placa PCB.

Durante la investigación se han encontrado así mismos puntos a mejorar e incluso limitaciones. Hay que tener siempre en mente que las técnicas de computación paralela no pueden aplicarse a todos los algoritmos ya que no son arquitecturas de propósito general. Existen técnicas para analizar los algoritmos a implementar que permitirán decidir si dichos algoritmos son factibles o no de ser implementados utilizando computación paralela. En otras ocasiones, aunque sea factible realizar implementaciones paralelas de los algoritmos, puede darse el hecho de que los resultados no sean los esperados, incluso empeorando el rendimiento obtenido con implementaciones secuenciales. Es importante tener un buen conocimiento en las técnicas de computación paralela y todas las implicaciones que conlleva el uso de estas arquitecturas para ser capaces de realizar implementaciones que exploten todas las posibilidades de aceleración de que disponen.

Además, los entornos GPU tienen algunos inconvenientes relacionados con el hecho de que no son entornos de propósito común. Uno de estos grandes inconvenientes es la gestión de la memoria, tanto su limitación dentro de la GPU en sus diferentes niveles (general, bloque, hilo) como el coste de la transferencia entre sus diferentes niveles e incluso la transferencia con el host. Aunque es un aspecto

en el que se sigue evolucionando, es quizás el aspecto que puede hacer que un algoritmo acelerado mediante GPU llegue a tener peor rendimiento que una solución secuencial. Por tanto, es siempre un punto clave analizar, organizar y planificar correctamente la memoria necesaria en la solución al problema y saber de antemano cómo se han de gestionar correctamente las necesidades de memoria. El otro gran inconveniente son los bloqueos de sincronización: la dependencia de ciertas partes del algoritmo con otras partes puede llegar a neutralizar e incluso a empeorar la aceleración conseguida con el paralelismo. Este otro inconveniente sigue alineado con la idea de que estos entornos no son entornos de propósito general sino especifico, y no todos los problemas son solucionables con estas técnicas de computación paralela.

Por tanto, si el algoritmo es factible de ser implementado con técnicas de computación paralela y se realiza un correcto diseño de su implementación paralela mediante un adecuado diseño de memoria, de estructura y de ejecución, es de esperar que se obtenga una aceleración significativa.

Como aportaciones, este trabajo de investigación ha puesto de manifiesto que los algoritmos heurísticos aportan soluciones adecuadas a diferentes problemas de ingeniería y también se ha demostrado que la aplicación de técnicas de computación paralela acelera estos algoritmos para conseguir resultados en mejores tiempos, dando con ello una mejora competitiva sobre otras soluciones. Este trabajo ha quedado reforzado mediante una serie de publicaciones, tanto en revistas de gran impacto como en congresos, dejando así mismo abiertas muchas líneas de investigación y mejoras.

Finalmente, se concluye por tanto que la aplicación de técnicas de computación paralela, y concretamente NVidia CUDA, a problemas reales de ingeniería es un factor a tener en cuenta en la optimización de las soluciones a dichos problemas. Esta optimización puede aplicarse a diferentes factores como se ha demostrado durante el trabajo, ya sea disminuyendo el tiempo para realizar un recorrido entre nodos o ciudades, lo cual puede mejorar diferentes factores tanto económicos como medio ambientales, o mejorando la producción de fabricación de placas PCB. Existen infinidad de problemas donde se puede aplicar todo lo expuesto en esta investigación, lo que a su vez abre una gran cantidad de nuevas líneas de investigación.

2. Líneas abiertas de investigación

Durante este trabajo de investigación se han seguido dos líneas principales: por un lado, la investigación y elección en los algoritmos a implementar y, por otro lado, las diferentes técnicas de implementación paralela para acelerar dichos algoritmos. Ambos aspectos dejan abiertas nuevas líneas de investigación. tanto para la mejora de la aceleración como para la aplicación de dichas técnicas a nuevos problemas de ingeniería. A continuación, se enumeran algunas líneas de investigación que han quedado abiertas durante la investigación:

- Mejora en la implementación de los algoritmos utilizando las nuevas funcionalidades añadidas. La computación paralela es un campo que está creciendo año tras año añadiendo nuevas funcionalidades y mejorando algunas de las existentes. Aplicar estas mejoras a los algoritmos implementados y comprobar las mejoras sobre los resultados obtenidos, tanto en funciones de *benchmarking* como en problemas reales, es una línea interesante a explorar.

- Implementación de nuevos algoritmos. Durante la investigación, han ido apareciendo nuevos algoritmos o incluso nuevas revisiones de algoritmos ya implementados que mejoran en algunos aspectos los resultados. Sería interesante analizar dichos algoritmos, mejorar las implementaciones propuestas y comparar los resultados con los obtenidos previamente.

- En las implementaciones que se realizaron durante la investigación, se ha hallado un obstáculo relevante relacionado con el tamaño de los problemas y la forma de gestionar la memoria necesaria para poder dar solución a estos problemas. En la fase final de la investigación se han realizado grandes mejoras en la gestión de la memoria de la GPU, sobre todo en CUDA, lo que puede propiciar una mejora significativa en la implementación de los algoritmos realizada durante la investigación. Este hecho puede dar cabida a la aplicación de los algoritmos en otros problemas de mayor envergadura.

Apéndice

En este apéndice, se muestran otros artículos relevantes producidos durante este trabajo de investigación. Dichos artículos son:

- A. García-Monzó, H. Migallón-Gomis, A. Jimeno-Morenilla, J.-L. Sánchez-Romero, H. Rico-García, R.V. Rao, "Efficient Subpopulation Based Parallel TLBO Optimization Algorithms", *Electronics*, *8*, 19, 2019
  https://doi.org/10.3390/electronics8010019

- H. Migallón-Gomis, A. Belazi, J.-L. Sánchez-Romero, H. Rico-García, A. Jimeno-Morenilla, "Settings-Free Hybrid Metaheuristic General Optimization Methods", *Mathematics*, 2020; 8(7):1092, 2020
  https://doi.org/10.3390/math8071092

- H. Migallón-Gomis, A. Jimeno-Morenilla, J.-L. Sánchez-Romero, H. Rico-García, R.V. Rao, "Multipopulation-based multi-level parallel enhanced Jaya algorithms", Journal of Supercomputation, vol. 75, pp. 1697–1716, 2019
  https://doi.org/10.1007/s11227-019-02759-z

- Migallón, H., Jimeno-Morenilla, A., Rico, H. et al. Multi-level parallel chaotic Jaya optimization algorithms for solving constrained engineering design problems. J Supercomput (2021).
  https://doi.org/10.1007/s11227-021-03737-0