UNIVERSIDAD
DE GRANADA

Trabajo de Fin de Máster

Máster en Ciencia de Datos e Ingeniería de Computadores

Análisis de Consumo Energético en Algoritmos Genéticos Paralelos

Autor
Salvador Moreno Gutiérrez

Tutor y Cotutor
Julio Ortega Lopera
Miguel Damas Hermoso

Granada, 7 de septiembre de 2017

UNIVERSIDAD
DE GRANADA

MÁSTER EN CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

DECLARACIÓN DE ORIGINALIDAD DEL
TRABAJO DE FIN DE MÁSTER

D. Julio Ortega Lopera y D. Miguel Damas Hermoso, profesores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada, como director/es del Trabajo Fin de Máster titulado "Análisis de Consumo Energético en Algoritmos Genéticos Paralelos" y realizado por el alumno D. Salvador Moreno Gutiérrez.

CERTIFICA/N: que el citado Trabajo Fin de Máster, ha sido realizado y redactado por dicho alumno y autorizan su presentación.

Granada,

Fdo. Julio Ortega Lopera

Fdo. Miguel Damas Hermoso

Agradecimientos

A mi familia y amigos por todo su apoyo; a mis tutores Miguel y Julio su inconmesurable ayuda; y a Patri, Manolo, Marce, las chicas del Hogar San Judas Chico, Giorgina, Durso, Tati, Tatiana y Mauricio, que han estado tan presentes durante la redacción de esta memoria en "el Perú".

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad y los fondos FEDER a través del proyecto TIN2015-67020-P.

Análisis de Consumo Energético en Algoritmos Genéticos Paralelos

Energy Consumption Analisys in Parallel Genetic Algorithms

Salvador Moreno Gutiérrez
smoreno94@correo.ugr.es

Resumen. En este Trabajo de Fin de Máster se estudia el consumo energético de algoritmos genéticos aplicado la selección de características multiobjetivo en problemas BCI o *Brain Computer Interface* por sus siglas en inglés. Razones económicas, de impacto medioambiental y de gestión de recursos han puesto en el punto de mira el consumo energético de la ejecución de programas desde máquinas de altas prestaciones hasta dispositivos móviles como *smartphones*. El consumo energético del algoritmo estudiado se ha caracterizado como caja negra, haciendo la metodología que modela el algoritmo, y que es desarrollada a lo largo de toda la memoria, fácilmente portable y exclusivamente dependiente de parámetros de configuración de la evolución de la población. Del análisis de los datos se concluye que la evaluación de los individuos es la parte más importante del algoritmo genético estudiado, y que la repartición de esta carga entre el máximo número de núcleos es beneficiosa para la eficiencia. Además, el modelo lineal construido consigue, tras un análisis de anomalías y clustering con k-medias, ajustarse al test de validación con un 1.1925 % de error.

Palabras calve. Eficiencia energética, modelo, caja negra, clustering, k-medias, detección de anomalías, *local outlier factor*, LOF, MATLAB, Python, Arduino Mega.

Abstract. The energy consumption in parallel genetic algorithms applied to BCI – *Brain Computer Interface* – aplications is studied in this Master's Thesis. Economical, environmental and resource management reasons have risen up the importance of energy consumption when running programs from high performance computers to the cheapest smartphones. The algorithm's energy consumption has been modeled as a black-box, making the methodology deployed along the document easily portable and solely related to settings that configure the population's evolution. Data analysis reveals that evaluation of individuals is the most important part in the entire process and that sharing that workload amongst the maximum number of cores available is good for efficiency. Moreover, the linear regression model built scores a 1.925 % error rate after outlier removal and k-means clustering.

Keywords. Energy efficiency, model, black-box, clustering, k-means, outlier removal, *local outlier factor*, LOF, MATLAB, Python, Arduino Mega.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La eficiencia energética está a la orden del día. Se ha convertido en uno de los objetivos prioritarios a nivel de diseño, no sólo a nivel de hardware, sino también a nivel de software. Esto ha obligado a desarrolladores a maximizar la rentabilidad energética de algoritmos y programas que se ejecutan tanto en supercomputadores de altas prestaciones como en los smartphones que llevamos en nuestros bolsillos. En definitiva, la eficiencia de un programa ya no sólo se mide con lo rápido que es capaz de ejecutarse, sino lo bien que aprovecha sus revursos energéticos, ya sea por razones económicas (coste de la energía), ecológicas (reducción del derroche energético) o pragmáticas (priorización de duración de baterías en dispositivos móviles como *smartphones*).

Es por ello que en este Trabajo de Fin de Máster (TFM) se presenta un estudio del consumo energético de la aplicación del algoritmo genético NSGA-II [2] paralelizado en islas aplicado a la selección de características multiobjetivo en *Brain Computer Interface* (BCI) con el principal objetivo de entender su funcionamiento desde el consumo energético para así poder utilizarlo mejor en un futuro. Para ello se han medido los tiempos de ejecución del experimento en distintas condiciones durante su ejecución en MATLAB, a la vez que se medía el consumo energético directamente en el cable de conexión al nodo a través de un sensor amperímetro conectado a un *Arduino Mega*. Se ha desarrollado un modelo que permite la predicción a priori del consumo energético del algoritmo genético al ejecutarse en el nodo *compute-0-1* del cluster HPMOON de la Universidad de Granada, con un error relativo del 1.1925 %. Para la utilización de este modelo de caja-negra tan sólo se depende de parámetros de ejecución del propio algoritmo, es decir, población inicial, generaciones a evolucionar, hilos de ejecución (núcleos o *cores* para esta aplicación) y número de comunicaciones habilitadas entre islas o *cores* para migración de población. En resumen, depende de parámetros extrínsecos a la máquina donde se ejecuta, lo que lo hace que la metodología para la elaboración del modelo, desarrollada a lo largo de la memoria, sea fácilmente portable y generalizable a otros computadores.

1.1. Objetivos

Los objetivos de este TFM son los siguientes:

1. Realizar una revisión del estado del arte de los distintos métodos de modelado y predicción energética para computadores y ejecución de algoritmos.
2. Estudiar e implementar la medida de tiempos dentro de la ejecución del algoritmo NSGA-II paralelizado en islas en MATLAB.
3. Sincronizar la medida de tiempos del algoritmo con la medida de energía realizada con *Arduino*.
4. Estudiar el comportamiento energético de la ejecución del algoritmo en diferentes condiciones de población inicial, generaciones a evolucionar y número de *cores* o islas donde evoluciona la población.
5. Desarrollar y validar un modelo de predicción de consumo energético.

1.2. Estructura de la memoria

En este Capítulo 1 se ha hecho una introducción al modelado y estudio del consumo energético en el algoritmo genético NSGA-II paralelizado en islas, exponiendo brevemente los resultados obtenidos, los

objetivos perseguidos en el TFM y la estructura de la memoria; en el Capítulo 2 se expone el estudio realizado sobre el estado del arte, recogiendo las distintas propuestas para la caracterización del consumo energético de un sistema, la implementación en detalle del algoritmo NSGA-II para selección de características en aplicaciones BCI y las funciones de la Parallel Computing Toolbox de MATLAB de las que se aprovecha dicha implementación; en el Capítulo 3 se expone el diseño de la arquitectura con la que se han recogido y sincronizado los datos de tiempos y energía en un solo conjunto de datos; en el Capítulo 4 se recoge el análisis de los datos con resultados referidos a la distribución de las gráficas de tiempo y energía, tests de hipótesis, clustering, detección de anomalías y la construcción y validación del modelo desarrollado; y en el Capítulo 5 se recogen las conclusiones y propuestas de trabajo futuro. Además, para una mejor reproducibilidad de este TFM se incluye el Apéndice A con instrucciones y código fuente para poder volver a recoger los datos tomados.

Capítulo 2

Fundamentación: estado del arte

En este capítulo se realiza una revisión bibliográfica del estado del arte dentro de los temas de caracterización de consumo energético en sistemas (Sección 2.1) y sobre la implementación del algoritmo NSGA-II con paralelización en islas para selección de características en *Brain Computer Interface* (Sección 2.2) a modelar energéticamente. Además, en la Sección 2.3 se exponen las partes de la *Parallel Computing Toolbox* con las que el algoritmo estudiado ha sido implementado para una mejor comprensión del mismo.

2.1. Modelos de consumo energético

La importancia de la eficiencia energética está presente tanto en dispositivos móviles como en superordenadores, ya sea para ejecutar una *app*, o como en este TFM se propone, un algoritmo genético para selección de características multiobjetivo dentro de un conjunto de datos. Aun así, es común encontrarnos con que cuando se habla de eficiencia a la hora de programar, tan sólo se tiene en cuenta el tiempo invertido por parte del programa o los algoritmos desarrollados, pero a día de hoy, la energía debe tener la misma relevancia a la hora de hablar de *eficiencia* por razones económicas (coste de la energía), ecológicas (reducción del derroche energético) y pragmáticas (priorización de duración de baterías en dispositivos móviles como *smartphones*).

Por ello, es necesario ser capaces de modelar y entender bien el consumo energético que implica la ejecución de nuestros programas, buscando predecir el gasto esperado a priori o, incluso, en tiempo real. En concreto, en este TFM, lo que se busca es caracterizar la ejecución del algoritmo genético desarrollado en la Sección 2.2 como un modelo de caja negra, únicamente atendiendo a los parámetros de entrada: tamaño de la población (*pop*), número de generaciones a evolucionar (*gen*), número de hilos o, en este caso, procesadores sobre los que se ejecuta (*threads*) y el número de comunicaciones entre los mismos (*comm*). Con esto seremos capaces de estimar el consumo energético de la ejecución antes de lanzarse.

A continuación, en la Subsección 2.1.1 se muestran los objetivos ideales de caracterizar correctamente el consumo energético de nuestro sistema y sus programas; y en la Subsección 2.1.2 se desarrollan y ejemplifican las más comunes y diferentes formas de modelización existentes encontradas en la literatura.

2.1.1. Objetivos del modelo

Un modelo energético ideal sería capaz de valorar el consumo energético antes de ejecutar el programa y de tomar decisiones en tiempo real para ajustar el avance del algoritmo a medida que está en funcionamiento. Dentro de la caracterización, esto requiere no sólo una medida física de los componentes que integran nuestro sistema, sino también cómo son aprovechados los recursos que proveen en su funcionamiento y arquitectura dispuesta. Así, los objetivos a cumplir para cualquier modelo energético serían los siguientes [3]:

1. **Sistema completo:** debe recoger información inherente al sistema en su totalidad, no sólo de componentes individuales.

2. **Preciso:** debe tener un nivel de precisión suficiente que permita optimización en dicho nivel. Se determina este umbral de precisión en menos de un 10 % de error en el caso medio.

3. **Rápido:** debe generar predicciones con suficiente rapidez como para ser útiles en optimización a tiempo real (*scheduling* dinámico).

4. **Genérico y portable:** debe poder aplicarse a tantos sistemas como sea posible, por ejemplo, dispositivos móviles, PCs, *workstations* o clusters.
5. **No costoso:** la generación y utilización del modelo no debe requerir altos costos ni excesivas herramientas.
6. **No intrusivo:** la recogida de datos no puede alterar significativamente los parámetros del hardware o el rendimiento del programa.
7. **Simple:** la generalización es la prioridad del modelo, dejando particularidades a un lado para, aun así, proporcionar una predicción suficiente que cumpla con los objetivos previamente descritos.

2.1.2. Enfoques de modelado de consumo energético

Pueden describirse tres formas de modelado energético: basados en simulación detallada, analíticos y de caja negra de alto nivel. Por otro lado, cada una de ellas se puede caracteriza por un mejor o peor cumplimiento de los objetivos antes descritos en la Subsección 2.1.1. No nos adentramos en el modelado térmico porque, aunque está relacionado directamente con el consumo energético, su caracterización tiende a ser más complicada ya que la temperatura de los componentes se ve afectada directamente por las condiciones del ambiente a su alrededor.

Modelos basados en simulación detallada

El modelado por simulación de un sistema implica un exhaustivo conocimiento del diseño y arquitectura del sistema sobre el que se quiere realizar la predicción. Estos modelos pueden llegar a ser altamente precisos si la caracterización es buena, y a su vez, no requieren muchos recursos para funcionar correctamente (cumplen los objetivos 2 y 5); sin embargo, su alta complejidad suele llevar a que no se puedan alcanzar el resto de objetivos antes enumerados.

En primer lugar, estos modelos suelen estar orientados a los componentes concretos de la máquina analizada, más allá de una caracterización holística y de todo el sistema (objetivo 1). Además, tienden a ser bastante lentos ya que la simulación de todas las partes de la máquina por separado para su posterior integración de acuerdo a su arquitectura requiere una gran capacidad de cómputo (objetivo 3). Por lo tanto, tampoco son genéricos ni fácilmente portables (objetivo 4). El alto coste de la simulación puede incurrir en una alta intrusividad sobre la ejecución de las rutinas que queremos medir, impidiendo una predicción en tiempo real (objetivo 6). Por último, y por todo la descripción dada hasta ahora, no son para nada simples y requieren una gran cantidad de información de cada uno de los componentes que conforman el sistema y de cómo se agrupan en su arquitectura.

Sin embargo, a pesar de todas estas dificultades y de prácticamente cumplir tan sólo 2 de los 7 objetivos desarrollados, su capacidad de obtener una altísima precisión en la predicción hace que aún se mantenga el interés en este tipo de modelado. Cuenta también con la ventaja de que se puede analizar el comportamiento del sistema componente a componente independientemente del programa o rutina ejecutada.

Uno de los primeros modelos basados en simulación a partir de información de la arquitectura del sistema, no centrándose en circuitería detallada, es Wattch [4]. Wattch se integra dentro del simulador de rendimiento SimpleScalar [5] añadiendo la predicción de consumo energético al mismo. Wattch estructura el procesador en cuatro partes: caché y registros, memorias totalmente asociativas y contenido direccionable como TLBs[1], unidades lógicas como las unidades funcionales y la circuitería de señal de reloj. Sin embargo, a pesar de sus buenos resultados, se trata de una propuesta a muy bajo nivel (nivel de procesador), de alta complejidad y que no interesa de cara a la modelización de algoritmos genéticos estudiados en este TFM.

Una propuesta de más alto nivel es la simulación en Tempo de Shafi *y cols.* [7] del consumo de un PowerPc 405GP SOC. Este modelo predice el consumo de energía a través de eventos de la arquitectura del sistema como fallos de caché y lecturas en TLB. Sin embargo, la generación de este modelo implica la ejecución de más de 300 microbenchmarks a la vez que miden variaciones en el voltaje de cada uno de los componentes, por lo que este tipo de aproximación al problema vuelve a no ser práctica. La precisión que se alcanza en esta propuesta está dentro del 5 % de error, generando un modelo más simple que Wattch, pero requiere un conocimiento muy detallado del sistema para poder desarrollar microbenchmarks óptimos y específicos.

[1] *Translation Lookaside Buffer:* memoria caché que se utiliza para reducir el tiempo de acceso a memoria, concretamente, almacena las últimas relaciones entre direcciones lógicas y físicas [6].

Modelos analíticos

Los modelos de predicción de consumo energético pueden elaborarse también sin necesidad de simulaciones y de tener que caracterizar la máquina a nivel de procesador o arquitectura. Los modelos analíticos se suelen basar en contadores de rendimiento del propio procesador, a nivel de hardware, que registran todo tipo de eventos a nivel de arquitectura; sin embargo, hay que tener en mente que la accesibilidad, número y tipo de contadores con los que cuenta cada procesador varía según el fabricante y familia del mismo. Otro problema radica en que el número de eventos a contar suele ser mayor que el número de registros disponibles, por lo que sólo una pequeña parte de los mismos pueden monitorizarse a la vez.

Esta propuesta es utilizada por Bakkali y *cols.* [8] para predecir el consumo energético de PLCs, siendo capaz de discernir la energía asociada a comunicaciones ethernet del resto del consumo del PLC a través de contadores hardware. Azimi y *cols.* [9] proponen una metodología que es utilizada en la mayoría de los modelos de este tipo: como los contadores hardware que cada procesador es capaz de proveer son mayores al número que pueden ser monitorizados directamente en los registros, se realiza la medida de forma multiplexada para alternar el acceso a los contadores. Así se reduce la exactitud de las medidas pero se obtiene un mayor abanico de posibilidades en la variedad de datos a extraer.

Este tipo de modelado se encuentra en el punto medio entre la simulación y los modelos de caja negra. Al contrario que la simulación, este tipo de caracterización energética sí permite predicción en tiempo real, pero todavía se basa en gran medida en conocimientos específicos de microarquitectura, lo que limita su portabilidad y generalización, además de aplicarse a nivel de componente en lugar de todo el sistema. Es decir, no cumple con los objetivos 1, 4, mientras que el resto sí es capaz de alcanzarlos de forma óptima. No cumplir estos dos objetivos permite una menor tasa de fallos, a nivel de componente, en la que el modelo de caja negra puede incurrir fácilmente por tratarse de una aproximación mucho más generalista.

Modelos de caja negra de alto nivel

El modelo de caja negra es la que más fácilmente cumple los objetivos propuestos en la Subsección 2.1.1 a costa de una mayor tasa de fallos. Es la que más capacidad de portabilidad y generalización tiene, permitiendo así una mayor capacidad de aplicación a gran cantidad de sistemas y ejecución de programas. Se construye únicamente a partir de medidas en tiempo real de benchmarks ejecutados en la máquina.

Dentro de la literatura, existen bastantes propuestas de modelado de caja negra o *black-box* que funcionan muy bien de cara a predecir el consumo energético de algoritmos concretos. Jaiantilal y *cols.* [10] estudian en su propuesta cómo el perfil energético de una tarea es dependiente del tipo de ciclos de ejecución llevados a cabo por el procesador, prediciendo el consumo del benchmark *SPECjvm*, ejecutado en un variable número de procesadores, con modelos de regresión lineales y basados en *Random Forest*. Aliaga y *cols.* [11] estudian en su propuesta los resultados de la búsqueda de eficiencia energética durante la ejecución de ILUPACK – una herramienta iterativa para resolver sistemas lineales dispersos – en la que se aprovechan los estados *idle* de los núcleos introduciendo paralelismo, únicamente utilizando C-estados[2] y P-estados[3] del procesador en dos arquitecturas multicore, llegando a ahorrar un consumo energético entre el 7 % y el 13 %. Barik y *cols.* [14] proponen en su trabajo una técnica de modelado de caja negra basada en planificación y repartición de carga entre CPU y GPU, válido para sistemas de alto (*workstations*) y bajo rendimiento (*tablet*), estudiando el comportamiento de benchmarks ligados a memoria (memory-bound) o al procesador (compute-bound). En concreto, modela cada procesador a partir de los fallos de LLC[4] y el número total de instrucciones.

2.2. Paralelización en islas: selección de características mediante clasificación multiobjetivo en aplicaciones BCI

Los problemas de clasificación que surgen en la aplicación BCI suelen implicar un número relativamente pequeño de ejemplos frente a un alto número de características, situación más conocida como "maldición de la dimensionalidad" [15]. La selección adecuada de características es un requisito importante para construir clasificadores robustos, sin sobreaprendizaje y que permitan generalizar lo suficiente.

[2] *CPU operating states:* estados del procesador definidos de acuerdo a su estado operativo. Los diferentes estados dependen de la velocidad del reloj, conexiones al bus principal y estado de la APIC [12].

[3] *Performance states:* estados del procesador definidos de acuerdo a su rendimiento. Cualquier P-estado requiere un C-estado C0 en el que el procesador esté funcionando [13].

[4] *Last Level Cache:* último nivel de caché en el procesador, normalmente L3.

Para encontrar las mejores soluciones Ortega et cols. [1] han implementado un procedimiento de optimización multiobjetivo evolutivo en un modelo de islas, en el que los individuos se distribuyen entre diferentes subpoblaciones que evolucionan e intercambian de forma independiente a los individuos durante el número de generaciones predispuesto. Los resultados experimentales muestran mejoras tanto en el tiempo de cálculo como en la calidad de la clasificación del EEG con características extraídas por análisis multirresolución (MRA) comparándolas con sus trabajos previos, donde la selección se aborda mediante el uso paralelo multiobjetivo y cooperativo coevolutivo a través de un modelo maestro-trabajador en paralelo [16–18]. Este algoritmo está basado en el algoritmo evolutivo NSGA-II de Deb y *cols.* [2].

2.2.1. Procedimiento de Paralelización en Islas para Selección de Características Multiobjetivo

Entre los tres enfoques diferentes para la selección de características – filtro, envoltorio (*wrapper*) y métodos incrustrados (*embedded*) – [19], se utiliza la propuesta como envoltorio en un algoritmo evolutivo multiobjetivo con paralelización en islas. Los individuos de una población representan diferentes selecciones de características, y la aptitud de un determinado individuo se determina mediante la evaluación del rendimiento del clasificador después de haber sido entrenado de manera supervisada.

El rendimiento del clasificador se evalúa a partir de la precisión obtenida después de su entrenamiento. Sin embargo, otras medidas que cuantifican propiedades como la generalización de la información, han de tenerse en cuenta a fin de mejorar el comportamiento del clasificador en un entorno real. Para abordar esta cuestión, este método propone un procedimiento evolutivo multiobjetivo en el que se optimiza la selección de características tanto para la precisión como para la capacidad de generalización.

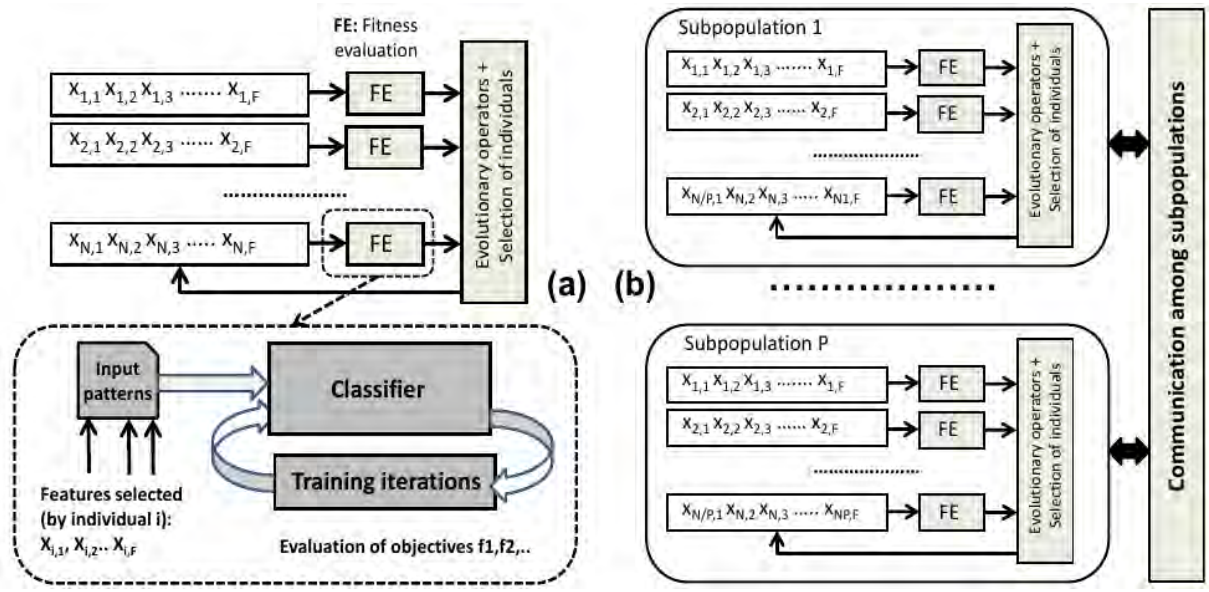


Figura 2.1: Selección de características mediante método envoltorio: (a) procedimiento secuencial; (b) procedimiento paralelizado en islas. Fuente: [1].

La Figura 2.1 muestra un esquema del método de envoltura propuesto para la selección de características. Éste se basa en un procedimiento de optimización multiobjetivo que busca un vector de decisión Variables $x = [x1, x2, ..., xn] \in Rn$ para optimizar un vector de función f(x), cuyos valores escalares $F1(x), f2(x), ..., fm(x)$ representan los m objetivos a optimizar.

Estos objetivos están, por lo general, en conflicto, y por lo tanto la optimización multiobjetivo debe obtener un conjunto de soluciones pareto-óptimas que definen el frente de Pareto, de los cuales es posible elegir la solución más conveniente en función de las circunstancias [20]. Para resolver el problema de optimización multiobjetivo se ha implementado un algoritmo evolutivo basado en el algoritmo NSGA-II [2], con codificación individual específica y operadores genéticos.

La principal diferencia de esta última propuesta frente a sus trabajos previos es la implementación paralela del procedimiento resumido en la Figura 2.1a basada en modelo de isla. Este enfoque paralelo en Figura 2.1b distribuye los N individuos de la población entre los P hilos disponibles, definiendo P subpoblaciones, cada una con N / P individuos. El pseudocódigo del procedimiento paralelo se proporciona

Algorithm 1 Selección de Características mediante Paralelización en Islas

```
1: procedimiento PARALLEL_NSGAII_FEATURE_SELECTION(N, P)
2:     Initialization P(i, N/P, SP(i)) hilos /i = 1, ..., P;
3:     wait(i);                                   ▷ Barrera para sincronizar los P hilos.
4:     Island_evolution(i, N/P, SP(i), commprof, comm, genpar) en hilos /i = 1, ..., P;
5:     wait(i);                                   ▷ Barrera para sincronizar los P hilos.
6:     NSGAII_nondomination_sort(SP(1), ..., SP(P));
7:     guardar resultados;
8:     end

9: function INITIALIZATION(P(i, N/P, SP(i)))
10:    (SP(i)) = Initialize_population(N, P)
11:    f(SP(i)) = Evaluation(SP(i), DS)
12:    SP*(i) = NSGAII_nondomination_sort(SP(i))

13: function ISLAND_EVOLUTION(i,N/P, SP(i), commprof, comm, genpar)
14:    for j = 1 to comm do
15:        for k = 1 to genpar do
16:            (SP'(i), f(SP'(i))) = NSGAII_tournament_selection(SP(i), f(SP(i));
17:            SP"(i) = Genetic_operators(SP'(i));
18:            f(SP"(i)) = Evaluation(SP"(i), DS);
19:            (SP*(i)) = NSGAII_nondomination_sort(SP(i), SP"(i));
20:            (SP, f(SP)) = NSGAII_replace_chromosome (SP*(i));
21:        communication(SP(1), ..., SP(P), commprof);
22:    end
```

en el Algoritmo 1.

El procedimiento descrito en la línea 1, primero crea P hilos con la función *Initialization* y distribuye los N individuos en P subpoblaciones SP en la línea 2. Estos P hilos son sincronizados a través de una barrera en la línea 3 para realizar la evolución de las P subpoblaciones con la función *Island_evolution* de la línea 4. Cada hilo requiere conocer el número de comunicaciones *comm*, el número de generaciones que la subpoblación correspondiente tiene que completar entre las comunicaciones *genpar*, y las parejas de hilos $commprof$ seleccionados al azar que se tienen que comunicar después de cada *genpar* número de generaciones. Este procedimiento paralelo evolutivo y multiobjetivo, cuyo comportamiento es diferente del secuencial, permite mejorar la calidad de las soluciones encontradas mediante el uso de y / o reducción en el tiempo de computación.

La función *Initialization* descrita a partir de la línea 9 inicializa la población con N individuos. En la línea 10 a cada individuo se le asignan 30 características aleatorias de las 3600 disponibles, y además, se distribuyen en P subpoblaciones SP de N/P individuos. A continuación, sólo queda evaluar cada uno de los individuos en SP entrenando el modelo con el *dataset* de entrenamiento DS (línea 11), y construir el frente de Pareto de soluciones no dominantes ordenadas con la función *NSGAII_nondomination_sort*, que nos devuelve las mejores soluciones de acuerdo a dos parámetros de optimización. En este caso, y para el resto de la memoria, el clasificador será un *Linear Disciminant Analisys* o LDA, y las funciones de optimización multiobjetivo $f_1 = accuracy$ y $f_2 = 1 - Kappa$, llegando así a un compromiso entre soluciones con alto acierto pero alto rendimiento de clasificación interclase.

A partir de la línea 13 se describe la función *Island_evolution* en la que, para el número de comunicaciones *comm* establecido (línea 14), se desarrolla la subpoblación un número de generaciones *genpar* (línea 15). Primero se comparan a todos los individuos de la subpoblación SP con *NSGAII_tournament_selection* (línea 16), se les aplican los operadores genéticos de cruce y mutación (línea 17) y se evalúan (línea 18). Realizados estos pasos, puede volver a construirse el frente de Pareto de soluciones no dominadas con *NSGAII_nondomination_sort* (línea 19) y, por último, reemplazar los peores individuos de cada isla de acuerdo a su evaluación previa (línea 20). Para cada una del número de comunicaciones *comm* definidas se realiza este bucle *genpar* veces para finalmente intercambiar individuos entre las islas con *communication* de acuerdo a las parejas de islas aleatorias definidas en *commprof*.

2.2.2. Selección de características en BCI con MRA

El problema de clasificación de alta dimensionalidad que se examina es la Interfaz Cerebro-Ordenador (BCI) basado en la clasificación de señales de EEG correspondientes a las tareas de imágenes motoras (MI). Este paradigma BCI utiliza una serie de amplificaciones y atenuaciones de corta duración ocasionadas por el movimiento de extremidades imaginadas, la *desincronización relacionada con eventos* (ERD) y la *sincronización relacionada con eventos* (ERS). Un sistema de análisis por multiresolución MRA [21] aplica una secuencia de sucesivos espacios de aproximación que describen la señal objetivo, siendo así útiles siempre que la señal objetivo presente diferentes características a través de los distintos espacios de aproximación. Los patrones utilizados en este trabajo se construyen, a partir de ensayos EEG, por un procedimiento de extracción de características basado en el MRA descrito en [22].

Cada señal obtenida de cada electrodo contiene varios segmentos a los que un conjunto de ondas se les asignan coeficientes de aproximación. De esta manera, considerando S segmentos, E electrodos y L niveles de wavelets, cada patrón EEG se caracteriza por $2 \cdot S \cdot E \cdot L$ conjuntos de coeficientes. En el conjunto de datos considerado, registrado en el Laboratorio BCI de la Universidad de Essex, S = 20 segmentos, E = 15 electrodos, y L = 6 niveles. Por lo tanto, 3600 conjuntos de coeficientes wavelet en total en cada patrón, con 4 a 128 Coeficientes en cada conjunto, caracterizan cada patrón: un total de 151200 coeficientes.

Sin embargo, en [22] sólo una característica es asignada a cada electrodo y cada nivel de aproximación y detalle, y se obtiene calculando la varianza de la distribución de los coeficientes y normalizando los valores obtenidos entre 0 y 1. De este modo, $2 \cdot S \cdot E \cdot L = 3600$ características que constituyen cada patrón. Como el número de patrones para cada sujeto es aproximadamente 180, es obvio que es necesario un método para la extracción de características más importantes.

Para caracterizar el desempeño del clasificador (LDA en este caso) mientras se ha entrenado o ajustado para un conjunto dado de características (es decir, un individuo de la población), es importante no sólo tener en cuenta la precisión obtenida para el conjunto de entrenamiento, sino también su capacidad de generalización, es decir, su precisión para instancias no vistas durante el entrenamiento. Por lo tanto, la primera función de coste está relacionada con el índice *Kappa* en el conjunto de datos de formación, que tiene en cuenta la distribución del error por clase que se calcula como $(p_0 - p_c)/(1 - p_c)$, siendo p_0 la proporción de coincidencias entre las salidas de clasificación y las etiquetas de los patrones y p_c la proporción de patrones en los que se espera la coincidencia por azar. La segunda función de coste es la función de pérdida promedio en un análisis de validación cruzada de 10 veces a los patrones de entrenamiento.

2.3. Parallel Computing Toolbox de MATLAB

Para la implementación de este algoritmo en MATLAB se ha aprovechado la *Parallel Computing Toolbox*, concretamente, el conjunto de órdenes orientado a programación *Single Program Multiple Data* o SPMD dentro de los *Communication Jobs*, necesario para la migración de soluciones entre islas propuesta [23]. La forma de lanzar estos programas es la siguiente:

1. Definir perfil de trabajo, en nuestro caso `'SPMD'`.
2. Seleccionar los hilos o *cores* donde ejecutar, es decir, definir el número de núcleos sobre los que se va a evolucionar la población a estudiar.
3. Crear un *communicatingJob* con `createCommunicatingJob` donde las tareas puedan comunicarse entre sí.
4. Crear una tarea o *task* con `createTask` a reproducir en cada uno de los cores indicados.
5. Enviar las tareas a los hilos indicados con `submit`.
6. Esperar y recoger los resultados con `wait`.
7. Eliminar las tareas y *communicatingJobs* lanzadas con `delete`.

Una vez se han lanzado las tareas, una para cada trabajador, se pueden establecer comunicaciones entre ellas. Se utiliza la variable interna `labindex` para controlar en todo momento qué hilo de ejecución se está manipulando. La orden `labBroadcast(labindexSender, data)` manda información al resto de trabajadores, que deberán estar a la escucha con la orden `labBroadcast(labindexSender)`.

Así es como están sincronizadas las distintas islas del programa sobre las que evoluciona la población. Están emparejadas dos a dos, lo que hace que cuando se emite un `labBroadcast`, ambas están preparadas y sincronizadas para poder enviar y recibir sus mejores individuos de la población.

Capítulo 3

Recogida de datos

En este capítulo se recoge la metodología seguida para la recogida de los datos.

3.1. Metodología

La recogida de datos se ha efectuado en el cluster HPMOON de la Universidad de Granada. Concretamente, se ha utilizado el nodo *compute-0-1* con un procesador Intel Xeon E5-2620 v4 (2.1 GHz) 8 cores x 2 threads/core, 32 GBytes de memoria RAM, NVS 315 y Tesla K40m. Utiliza Ubuntu con kernel 3.10.0-327.36.3.el7.x86_64 con los programas MATLAB R2014a (8.3.0.532) 64-bit (glnxa64) y Python 2.7.5. Se han ejecutado 10 repeticiones de cada uno de los experimentos mostrados en la Tabla 3.1, correspondiendo su nombre al esquema *threads-population-generations-communications*. Se ha estudiado en todo momento el poblema con una única comunicación para tratar de modelar los aspectos más básicos del algoritmo: su distribución en 4 u 8 hilos, una mayor o menor población (160-320), y la evolución durante más o menos generaciones (50-100).

Tabla 3.1: Experimentos realizados.

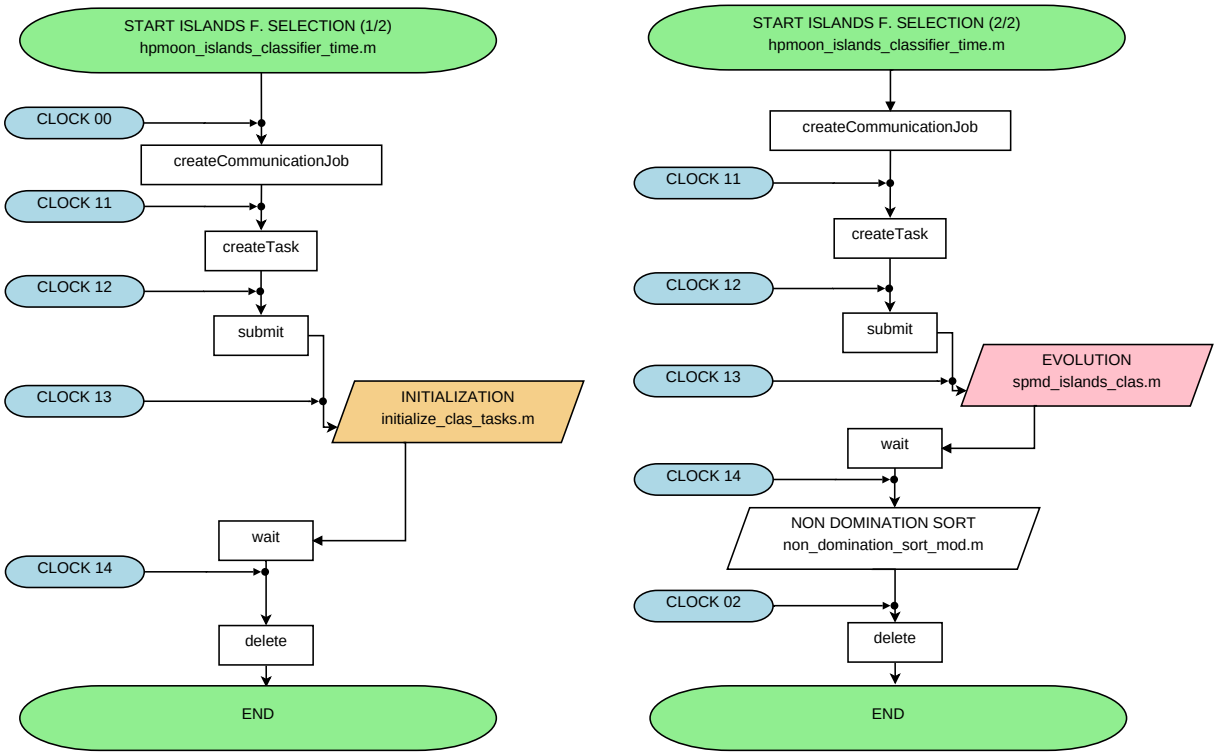
#	**Experimento**	#	**Experimento**
#1	4-160-50-1	#5	4-320-50-1
#2	8-160-50-1	#6	8-320-50-1
#3	4-160-100-1	#7	8-320-100-1
#4	8-160-100-1	#8	8-320-100-1

3.2. Medidas de tiempo en runtime

Para medir el tiempo en MATLAB se ha utilizado la orden *clock* para ir determinando los tiempos, midiendo el tiempo inicial t_i y comparándolo con *etime* a cada medida posterior realizada. Las medidas de tiempo se han realizado de manera simultánea e independiente para cada hilo de ejecución. En la Tabla 3.2 se muestran las etiquetas para cada uno de los tiempos medidos para las fases del algoritmo NSGAII adaptado a paralelización en hebras y para las órdenes de la *Parallel Computing Toolbox* de MATLAB [23]. En la Figura 3.1 se muestra un esquema general del algoritmo; en las Figuras 3.2 y 3.3 se desarrollan los bloques *Initialization* y *Evolution* de las Figuras 3.1a y 3.1b, respectivamente.

Tabla 3.2: Etiquetas para cada medida de tiempo.

Etiqueta	**Fase NSGAII-Islands**	**Etiqueta**	**Orden MATLAB**
00	Inicialización		
-1	Evaluación	11	*createCommunicationJob*
02	Ordenación no dominada	12	*createTask*
03	Selección por torneo	13	*submit*
04	Operaciones genéticas	14	*wait*
05	Reemplazo de soluciones	16	*labSend* + *labReceive*



(a) Parte 1: Inicialización de la población. (b) Parte 2: Evolución de la población.

Figura 3.1: Diagrama de flujo: extracción de características mediante paralelización en Islas.

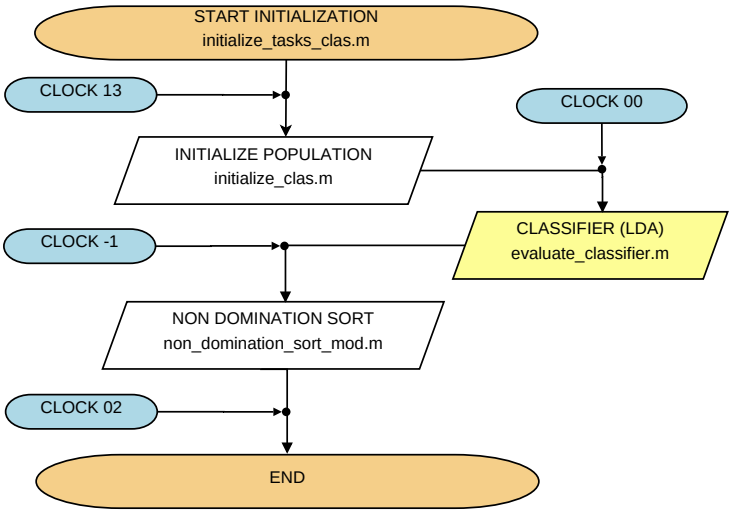


Figura 3.2: Subrutina: inicialización de la población.

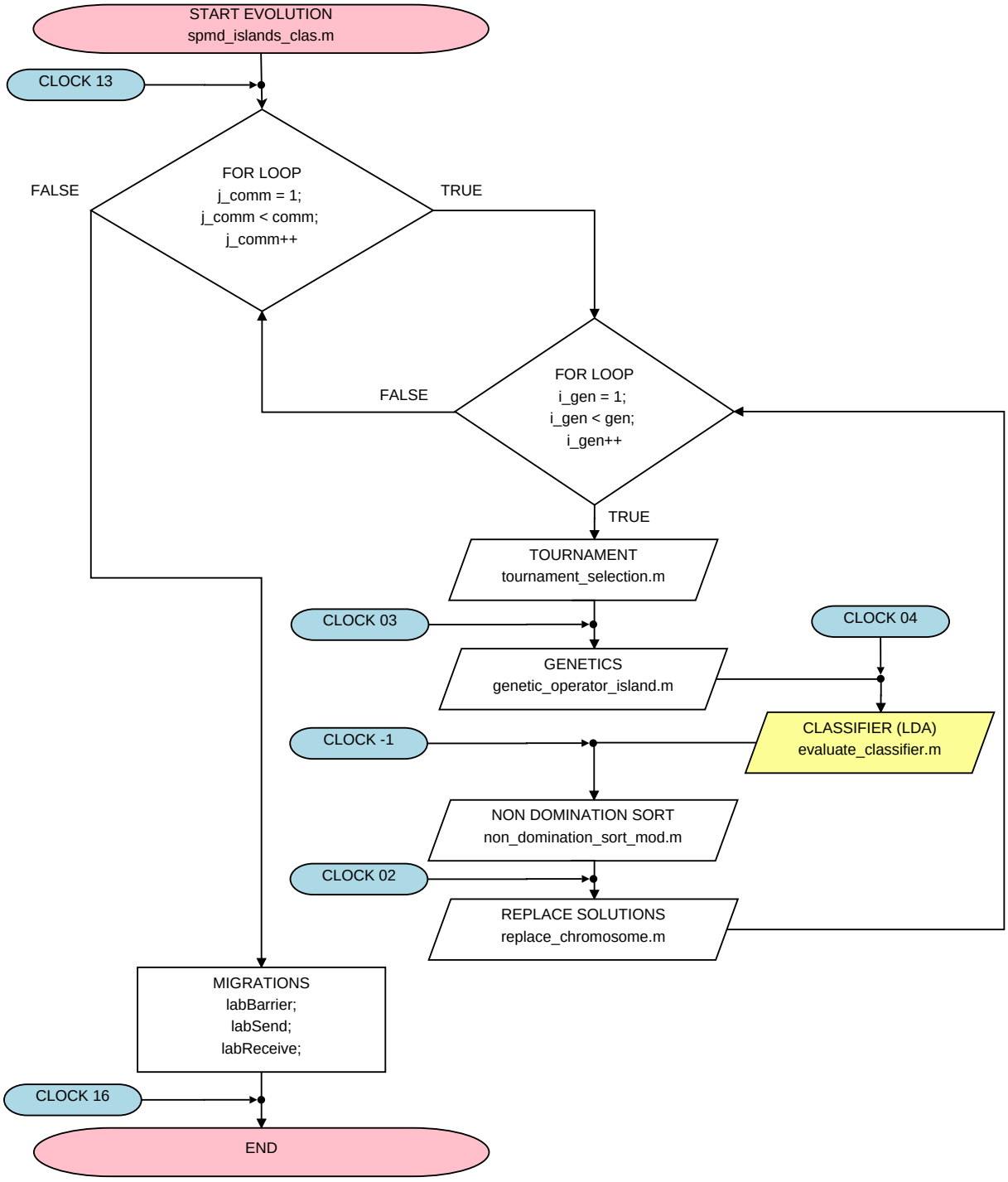


Figura 3.3: Subrutina: evolución de la población.

3.3. Medida de energía

La medida de energía se ha hecho a través de un dispositivo *Arduino Mega* directamente en los nodos del cluster HPMOON, instalado por el departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada (Figura 3.5). Esta medida se realiza a través de sensores que indican la intensidad de corriente que circula en el cable de corriente que conecta el equipo a la red eléctrica, es decir, se está midiendo el consumo real de todo el nodo, incluyendo tanto componentes activos como el de pérdidas en el transformador de corriente alterna a corriente continua.

Figura 3.4: Arduino Mega y sensores de corriente

El sistema de medida está compuesto de una placa de *Arduino Mega* y cuatro sensores (uno para cada nodo del cluster HPMOON) para medir su potencia instantánea en W y su consumo energético acumulado en $W \cdot h$. El sensor utilizado es el YHDC-SCTD010T-5A y puede medir hasta 5A con una salida proporcional entre 0 y 5V con una precisión del $\pm 2\,\%$. *Arduino* cuenta con un conversor A/D interno de 10 bits y, para aprovechar mejor su rango dinámico, se utiliza su referencia interna a 2.56V tan sólo disponible en *Arduino Mega*. La frecuencia de muestreo es teóricamente de 1Hz, aunque en la práctica ha resultado ser algo menor, detalle que afectará a la hora de unir las medidas de tiempo con las de energía. Se explica más adelante dentro de este mismo capítulo. *Arduino Mega* se conecta a uno de los nodos del cluster por USB para suministrarle corriente y así poder recoger los datos mediante conexión TCP local que se distribuyen desde el puerto 5214. El programa de recogida de datos está escrito en *Python* y su código fuente está en la página 59.

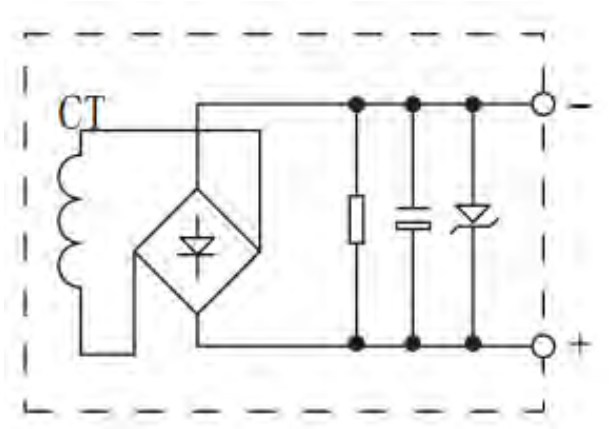


Figura 3.5: Diagrama interno de sensor de corriente YHDC-SCTD010T-5A.

Para medir correctamente el impacto de la ejecución del algoritmo genético en el consumo energético del cluster, en primer lugar, se mide el consumo medio por segundo del nodo *compute-0-1*, exclusivamente ejecutando el sistema operativo, para poder sustraerlo posteriormente a las medidas tomadas durante la ejecución del programa. El resultado de la que llamaremos $\Delta\bar{E}_{SO}$ que se obtiene después de recoger 4764

muestras (más de 1h de tiempo), corresponde al consumo de energía en algo más de 1s de tiempo, y es de :

$$\Delta\bar{E}_{SO} = (65{,}38 \pm 0{,}08)\ W \cdot h \tag{3.1}$$

Suponiendo que el consumo energético total E_T se puede expresar como la suma del consumo del sistema operativo más el consumo del algoritmo entonces podemos calcular la energía consumida exclusivamente por el mismo:

$$E_T = E_{SO} + E_{Alg} \tag{3.2}$$
$$E_{Alg} = E_T - E_{SO} \tag{3.3}$$

Para el caso, nos interesa modelar las medidas de tiempo y energía como incrementales para poder así tener una nube de puntos a partir de la cual extraer conocimiento. Reformulamos la Ecuación 3.3 en forma de incrementos:

$$\Delta E_{Alg} = \Delta E_T - \Delta\bar{E}_{SO} \tag{3.4}$$

Ahora es necesario sincronizar las medidas de tiempo con las de energía y, para ello, hay que tener en cuenta que la frecuencia de muestreo original es algo inferior a 1Hz. Primero en 3.5 se define un desfase (*delay*) entre el tiempo total medido por MATLAB (*timeElapsed*) y el tiempo medido por *Arduino Mega* (*energyTimeMeasured*). A continuación, en 3.6 se define *realtime* como el tiempo medido por MATLAB (*runtime*) menos la parte proporcional a ese desfase introducido en la muestra.

$$delay = timeElapsed - energyTimeMeasured \tag{3.5}$$
$$realtime(i) = runtime(i+1) - delay * \frac{runtime(i+1)}{timeElapsed} \tag{3.6}$$

Este vector de tiempos *realtime* sincroniza ambas medidas, de tiempos de ejecución del algoritmo y de energía consumida, así que sirve también para poder realizar la estimación energética utilizando una aproximación de integral por trapecios. En la Figura 3.6 se desarrolla el diagrama de flujo para la sincronización de medidas y estimación energética mencionadas para cada núcleo de ejecución. Por último, en el Apéndice A se expone, acompañado de código y una mención más detallada de los scripts utilizados, cómo se he realizado la sincronización entre las medidas de energía con *Arduino* y las medidas de tiempos en MATLAB.

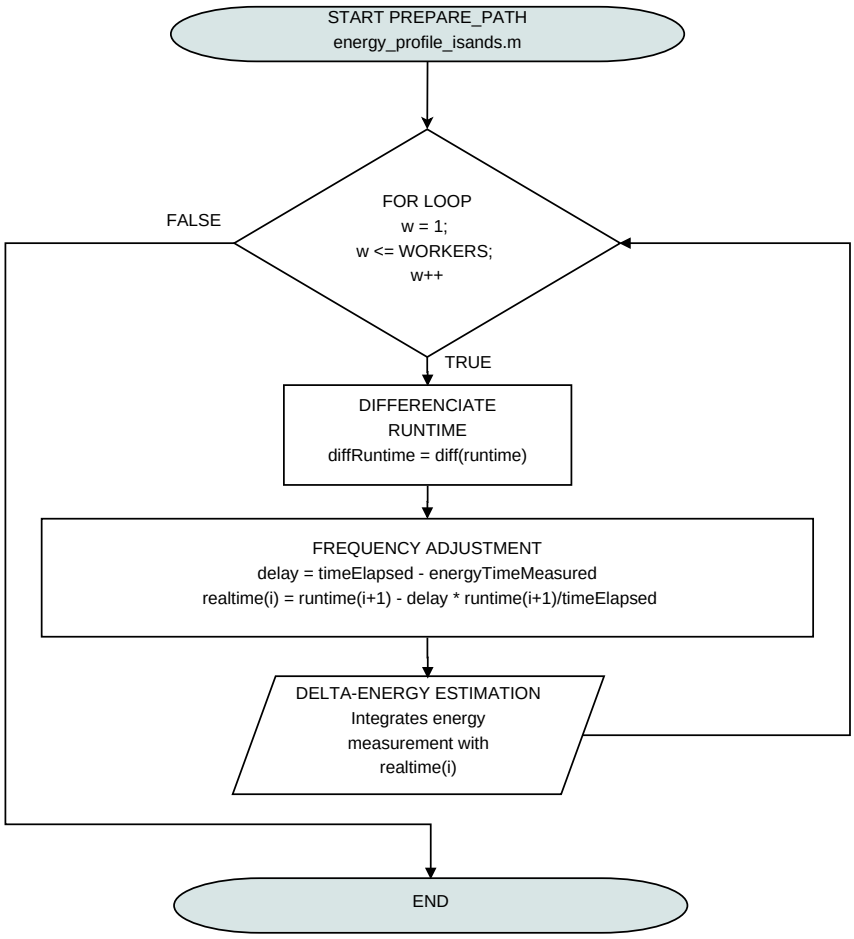


Figura 3.6: Preprocesamiento de datos de tiempo y energía.

Capítulo 4

Análisis de los datos

En este capítulo se detalla el análisis realizado a los datos. Primero se exponen todos los datos recogidos de forma detallada según la fase del algoritmo, y se les aplican tests de hipótesis entre las distribuciones generadas. A continuación, se muestran los resultados de aplicar clustering y detección de anomalías y, por último, se realiza la construcción y validación del modelo de predicción de consumo energético.

4.1. Datos de tiempo y consumo energético

En primer lugar, vamos a fijarnos en la distribución de tiempos y consumo energético para cada uno de los experimentos. A partir de ahora, los distintos experimentos van a ir refiriéndose con la nomenclatura *threads-pop-gen-comm*. Así, por ejemplo, el experimento 4-160-50-1 corresponde al ejecutado en 4 hilos (procesadores en este caso), 160 individuos, 50 generaciones y una comunicación entre procesos.

Se presenta en las Tablas 4.1 - 4.8 un resumen de los datos con las medias y desviaciones estándar de los tiempos y energía de cada uno de los experimentos; además, en las Figuras 4.1 - 4.16 puede verse la distribución de tiempo y energía en cada uno de ellos. En dichas Figuras puede verse la gran importancia de la evaluación de los individuos a nivel de consumo energético y de tiempo, en colores rojo y azul, respectivamente. Los histogramas están en escala logarítmica para favorecer la visualización de tan distintas proporciones. Calculando su proporción media respecto a las distintas fases, en todos los experimentos, se puede concluir lo siguiente:

- La evaluación de los individuos representa, frente al total, el 96.65 % del tiempo de ejecución, y el 97.07 % del consumo energético total.
- Teniendo en cuenta ese gran peso de la evaluación, es mucho mejor distribuir la carga de la misma entre el máximo número de núcleos disponibles. Ejecutar un algoritmo en 8 núcleos consume de media el 57.40 % del tiempo, y el 71.26 % de la energía de lo que supone lanzarlo en 4 .
- La aplicación de operadores genéticos (*GenOp*) y del reemplazo generación tras generación (*Replace*) es mucho más robusta en 4 que en 8 procesadores. En las Tablas 4.3, 4.4, 4.7 y 4.8 puede verse cómo las desviaciones estándar son tan pequeñas que, computacionalmente, valen cero para el tiempo, y prácticamente cero para la energía.
- La desviación estándar de las funciones internas de MATLAB *CreateCommJob* y *CreateTask* es prácticamente nula a nivel energético y de tiempo.

Dada la importancia de la fase de evaluación de la población, a partir de ahora los esfuerzos de análisis de datos se van a centrar en los datos referentes a esta fase del algoritmo.

Tabla 4.1: Media de tiempo total en segundos por experimento y fase (1/2)

Experiment	*E*val	Init	Non*D*omSort	Tournament	G*e*nOp
4-160-50-1	$2.19 \cdot 10^{+03}$	$1.12 \cdot 10^{-03}$	$6.71 \cdot 10^{-01}$	$6.68 \cdot 10^{+00}$	$6.19 \cdot 10^{-02}$
8-160-50-1	$1.26 \cdot 10^{+03}$	$1.31 \cdot 10^{-03}$	$5.50 \cdot 10^{-01}$	$6.86 \cdot 10^{+00}$	$1.84 \cdot 10^{-02}$
4-160-100-1	$4.33 \cdot 10^{+03}$	$1.11 \cdot 10^{-03}$	$1.07 \cdot 10^{+00}$	$6.53 \cdot 10^{+00}$	$1.06 \cdot 10^{-01}$
8-160-100-1	$2.48 \cdot 10^{+03}$	$1.28 \cdot 10^{-03}$	$7.30 \cdot 10^{-01}$	$6.93 \cdot 10^{+00}$	$3.36 \cdot 10^{-02}$
4-320-50-1	$4.39 \cdot 10^{+03}$	$1.11 \cdot 10^{-03}$	$1.73 \cdot 10^{+00}$	$6.58 \cdot 10^{+00}$	$5.81 \cdot 10^{-02}$
8-320-50-1	$2.52 \cdot 10^{+03}$	$1.30 \cdot 10^{-03}$	$9.06 \cdot 10^{-01}$	$6.87 \cdot 10^{+00}$	$2.44 \cdot 10^{-02}$
4-320-100-1	$8.59 \cdot 10^{+03}$	$1.10 \cdot 10^{-03}$	$3.11 \cdot 10^{+00}$	$6.72 \cdot 10^{+00}$	$5.82 \cdot 10^{-02}$
8-320-100-1	$4.93 \cdot 10^{+03}$	$1.29 \cdot 10^{-03}$	$1.43 \cdot 10^{+00}$	$7.00 \cdot 10^{+00}$	$4.57 \cdot 10^{-02}$

Tabla 4.2: Media de tiempo total en segundos por experimento y fase (2/2)

Experiment	R*e*place	Cr*e*ateComJob	Cr*e*ateTask	Submit	Wait
4-160-50-1	$2.46 \cdot 10^{-02}$	$1.48 \cdot 10^{+00}$	$7.21 \cdot 10^{-01}$	$1.59 \cdot 10^{+01}$	$5.12 \cdot 10^{+01}$
8-160-50-1	$7.69 \cdot 10^{-02}$	$1.58 \cdot 10^{+00}$	$7.59 \cdot 10^{-01}$	$1.81 \cdot 10^{+01}$	$4.64 \cdot 10^{+01}$
4-160-100-1	$4.30 \cdot 10^{-02}$	$1.16 \cdot 10^{+00}$	$7.15 \cdot 10^{-01}$	$1.56 \cdot 10^{+01}$	$8.48 \cdot 10^{+01}$
8-160-100-1	$7.95 \cdot 10^{-02}$	$1.63 \cdot 10^{+00}$	$7.92 \cdot 10^{-01}$	$1.80 \cdot 10^{+01}$	$8.06 \cdot 10^{+01}$
4-320-50-1	$3.53 \cdot 10^{-02}$	$1.20 \cdot 10^{+00}$	$7.32 \cdot 10^{-01}$	$1.55 \cdot 10^{+01}$	$8.84 \cdot 10^{+01}$
8-320-50-1	$7.41 \cdot 10^{-02}$	$1.30 \cdot 10^{+00}$	$7.12 \cdot 10^{-01}$	$1.81 \cdot 10^{+01}$	$7.12 \cdot 10^{+01}$
4-320-100-1	$6.59 \cdot 10^{-02}$	$1.16 \cdot 10^{+00}$	$7.28 \cdot 10^{-01}$	$1.56 \cdot 10^{+01}$	$1.23 \cdot 10^{+02}$
8-320-100-1	$7.42 \cdot 10^{-02}$	$1.28 \cdot 10^{+00}$	$7.26 \cdot 10^{-01}$	$1.77 \cdot 10^{+01}$	$1.30 \cdot 10^{+02}$

Tabla 4.3: Desviación estándar de tiempo total en segundos por experimento y fase (1/2)

Experiment	*E*val	Init	Non*D*omSort	Tournament	G*e*nOp
4-160-50-1	$3.73 \cdot 10^{+01}$	$5.22 \cdot 10^{-05}$	$3.56 \cdot 10^{-02}$	$5.26 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$
8-160-50-1	$2.80 \cdot 10^{+01}$	$1.10 \cdot 10^{-04}$	$1.59 \cdot 10^{-02}$	$3.45 \cdot 10^{-03}$	$5.10 \cdot 10^{-04}$
4-160-100-1	$7.40 \cdot 10^{+01}$	$5.69 \cdot 10^{-05}$	$7.08 \cdot 10^{-02}$	$5.32 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$
8-160-100-1	$4.64 \cdot 10^{+01}$	$7.56 \cdot 10^{-05}$	$2.43 \cdot 10^{-02}$	$3.77 \cdot 10^{-03}$	$1.15 \cdot 10^{-03}$
4-320-50-1	$6.74 \cdot 10^{+01}$	$4.63 \cdot 10^{-05}$	$1.10 \cdot 10^{-01}$	$4.68 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$
8-320-50-1	$4.63 \cdot 10^{+01}$	$1.04 \cdot 10^{-04}$	$3.86 \cdot 10^{-02}$	$1.97 \cdot 10^{-03}$	$9.57 \cdot 10^{-04}$
4-320-100-1	$1.18 \cdot 10^{+02}$	$3.22 \cdot 10^{-05}$	$2.69 \cdot 10^{-01}$	$3.34 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$
8-320-100-1	$8.22 \cdot 10^{+01}$	$6.86 \cdot 10^{-05}$	$8.14 \cdot 10^{-02}$	$1.11 \cdot 10^{-03}$	$2.17 \cdot 10^{-03}$

Tabla 4.4: Desviación estándar de tiempo total en segundos por experimento y fase (2/2)

Experiment	R*e*place	Cr*e*ateComJob	Cr*e*ateTask	Submit	Wait
4-160-50-1	$1.13 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$7.16 \cdot 10^{-04}$	$3.73 \cdot 10^{+01}$
8-160-50-1	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$5.33 \cdot 10^{-04}$	$2.80 \cdot 10^{+01}$
4-160-100-1	$1.91 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$7.46 \cdot 10^{-04}$	$7.39 \cdot 10^{+01}$
8-160-100-1	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$5.37 \cdot 10^{-04}$	$4.64 \cdot 10^{+01}$
4-320-50-1	$1.80 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$1.01 \cdot 10^{-03}$	$6.74 \cdot 10^{+01}$
8-320-50-1	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$5.11 \cdot 10^{-04}$	$4.63 \cdot 10^{+01}$
4-320-100-1	$4.75 \cdot 10^{-03}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$8.14 \cdot 10^{-04}$	$1.18 \cdot 10^{+02}$
8-320-100-1	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$5.30 \cdot 10^{-04}$	$8.22 \cdot 10^{+01}$

Tabla 4.5: Media de energía total en Wh por experimento y fase (1/2)

Experiment	*E*val	Init	Non*D*omSort	Tournament	G*e*nOp
4-160-50-1	$1.94 \cdot 10^{+01}$	$6.15 \cdot 10^{-06}$	$4.37 \cdot 10^{-03}$	$4.52 \cdot 10^{-02}$	$1.72 \cdot 10^{-04}$
8-160-50-1	$1.45 \cdot 10^{+01}$	$1.02 \cdot 10^{-05}$	$3.10 \cdot 10^{-03}$	$7.19 \cdot 10^{-02}$	$2.11 \cdot 10^{-04}$
4-160-100-1	$3.93 \cdot 10^{+01}$	$6.70 \cdot 10^{-06}$	$8.11 \cdot 10^{-03}$	$4.34 \cdot 10^{-02}$	$1.91 \cdot 10^{-04}$
8-160-100-1	$2.74 \cdot 10^{+01}$	$1.01 \cdot 10^{-05}$	$5.55 \cdot 10^{-03}$	$6.83 \cdot 10^{-02}$	$3.70 \cdot 10^{-04}$
4-320-50-1	$3.98 \cdot 10^{+01}$	$6.76 \cdot 10^{-06}$	$1.30 \cdot 10^{-02}$	$4.23 \cdot 10^{-02}$	$1.63 \cdot 10^{-04}$
8-320-50-1	$2.79 \cdot 10^{+01}$	$1.21 \cdot 10^{-05}$	$6.35 \cdot 10^{-03}$	$6.99 \cdot 10^{-02}$	$2.71 \cdot 10^{-04}$
4-320-100-1	$7.76 \cdot 10^{+01}$	$8.12 \cdot 10^{-06}$	$2.46 \cdot 10^{-02}$	$4.51 \cdot 10^{-02}$	$1.05 \cdot 10^{-04}$
8-320-100-1	$5.47 \cdot 10^{+01}$	$1.14 \cdot 10^{-05}$	$1.24 \cdot 10^{-02}$	$7.14 \cdot 10^{-02}$	$5.07 \cdot 10^{-04}$

Tabla 4.6: Media de energia total en Wh por experimento y fase (2/2)

Experiment	R*e*place	Cr*e*ateComJob	Cr*e*ateTask	Submit	Wait
4-160-50-1	$1.98 \cdot 10^{-04}$	$3.82 \cdot 10^{-03}$	$2.21 \cdot 10^{-03}$	$6.78 \cdot 10^{-02}$	$4.39 \cdot 10^{-01}$
8-160-50-1	$1.37 \cdot 10^{-04}$	$3.15 \cdot 10^{-03}$	$1.95 \cdot 10^{-03}$	$1.06 \cdot 10^{-01}$	$5.23 \cdot 10^{-01}$
4-160-100-1	$3.76 \cdot 10^{-04}$	$3.25 \cdot 10^{-03}$	$2.71 \cdot 10^{-03}$	$6.79 \cdot 10^{-02}$	$7.52 \cdot 10^{-01}$
8-160-100-1	$2.22 \cdot 10^{-04}$	$4.00 \cdot 10^{-03}$	$2.18 \cdot 10^{-03}$	$9.91 \cdot 10^{-02}$	$8.82 \cdot 10^{-01}$
4-320-50-1	$3.05 \cdot 10^{-04}$	$3.46 \cdot 10^{-03}$	$2.29 \cdot 10^{-03}$	$6.72 \cdot 10^{-02}$	$7.84 \cdot 10^{-01}$
8-320-50-1	$1.33 \cdot 10^{-04}$	$3.00 \cdot 10^{-03}$	$1.21 \cdot 10^{-03}$	$9.97 \cdot 10^{-02}$	$7.69 \cdot 10^{-01}$
4-320-100-1	$5.65 \cdot 10^{-04}$	$1.63 \cdot 10^{-03}$	$1.57 \cdot 10^{-03}$	$7.08 \cdot 10^{-02}$	$1.10 \cdot 10^{+00}$
8-320-100-1	$2.11 \cdot 10^{-04}$	$3.68 \cdot 10^{-03}$	$2.20 \cdot 10^{-03}$	$1.04 \cdot 10^{-01}$	$1.43 \cdot 10^{+00}$

Tabla 4.7: Desviación estándar de energía total en Wh por experimento y fase (1/2)

Experiment	*E*val	Init	Non*D*omSort	Tournament	G*e*nOp
4-160-50-1	$3.38 \cdot 10^{-01}$	$5.94 \cdot 10^{-06}$	$4.80 \cdot 10^{-04}$	$6.71 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$
8-160-50-1	$3.32 \cdot 10^{-01}$	$7.15 \cdot 10^{-06}$	$2.27 \cdot 10^{-04}$	$4.17 \cdot 10^{-05}$	$9.39 \cdot 10^{-06}$
4-160-100-1	$6.75 \cdot 10^{-01}$	$4.82 \cdot 10^{-06}$	$5.11 \cdot 10^{-04}$	$4.66 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$
8-160-100-1	$5.22 \cdot 10^{-01}$	$4.86 \cdot 10^{-06}$	$3.19 \cdot 10^{-04}$	$4.07 \cdot 10^{-05}$	$1.95 \cdot 10^{-05}$
4-320-50-1	$6.17 \cdot 10^{-01}$	$5.16 \cdot 10^{-06}$	$1.52 \cdot 10^{-03}$	$5.06 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$
8-320-50-1	$5.18 \cdot 10^{-01}$	$6.07 \cdot 10^{-06}$	$4.74 \cdot 10^{-04}$	$2.59 \cdot 10^{-05}$	$1.57 \cdot 10^{-05}$
4-320-100-1	$1.07 \cdot 10^{+00}$	$5.79 \cdot 10^{-06}$	$2.75 \cdot 10^{-03}$	$3.99 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$
8-320-100-1	$9.19 \cdot 10^{-01}$	$6.02 \cdot 10^{-06}$	$9.87 \cdot 10^{-04}$	$1.64 \cdot 10^{-05}$	$2.98 \cdot 10^{-05}$

Tabla 4.8: Desviación estándar de energía total en Wh por experimento y fase (2/2)

Experiment	R*e*place	Cr*e*ateComJob	Cr*e*ateTask	Submit	Wait
4-160-50-1	$4.43 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$6.06 \cdot 10^{-06}$	$3.38 \cdot 10^{-01}$
8-160-50-1	$0.00 \cdot 10^{+00}$	$1.39 \cdot 10^{-19}$	$1.62 \cdot 10^{-19}$	$5.58 \cdot 10^{-06}$	$3.32 \cdot 10^{-01}$
4-160-100-1	$3.97 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$7.49 \cdot 10^{-06}$	$6.75 \cdot 10^{-01}$
8-160-100-1	$2.32 \cdot 10^{-20}$	$2.78 \cdot 10^{-19}$	$1.16 \cdot 10^{-19}$	$6.32 \cdot 10^{-06}$	$5.22 \cdot 10^{-01}$
4-320-50-1	$4.55 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$8.86 \cdot 10^{-06}$	$6.16 \cdot 10^{-01}$
8-320-50-1	$0.00 \cdot 10^{+00}$	$9.27 \cdot 10^{-20}$	$9.27 \cdot 10^{-20}$	$5.09 \cdot 10^{-06}$	$5.18 \cdot 10^{-01}$
4-320-100-1	$5.86 \cdot 10^{-05}$	$0.00 \cdot 10^{+00}$	$0.00 \cdot 10^{+00}$	$7.99 \cdot 10^{-06}$	$1.07 \cdot 10^{+00}$
8-320-100-1	$1.16 \cdot 10^{-20}$	$1.85 \cdot 10^{-19}$	$6.95 \cdot 10^{-20}$	$5.72 \cdot 10^{-06}$	$9.19 \cdot 10^{-01}$

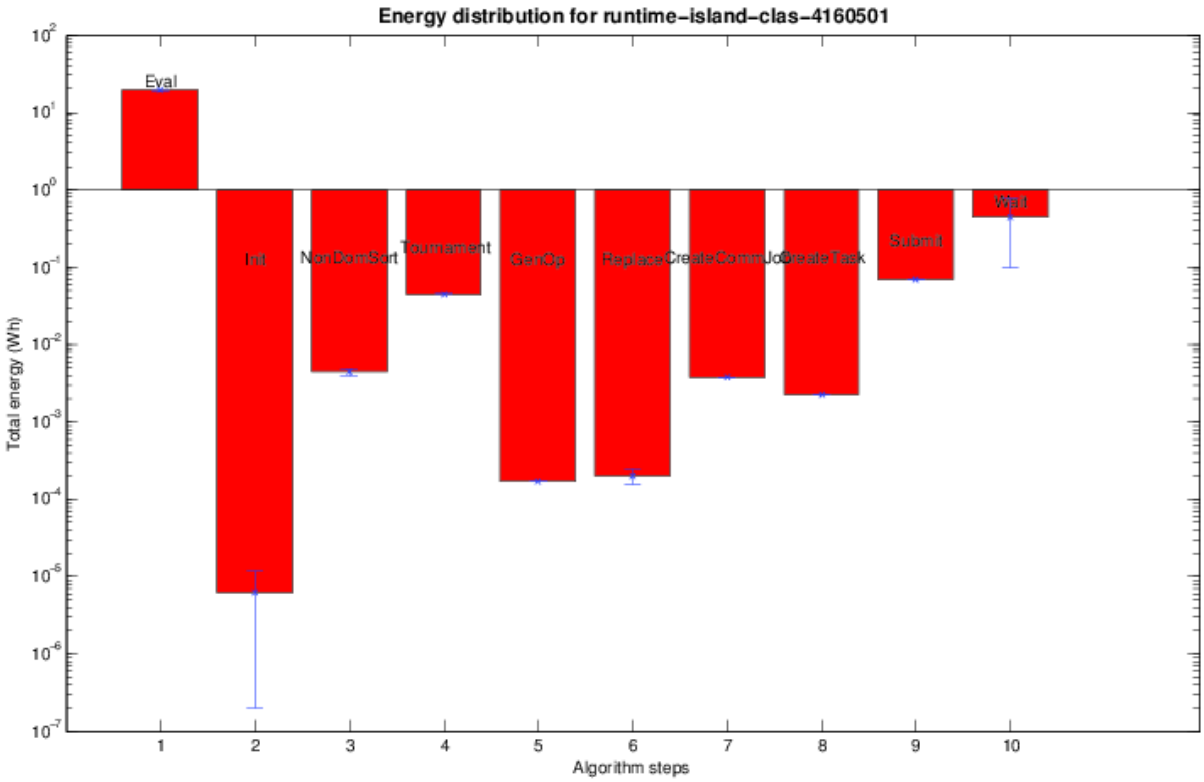


Figura 4.1: Distribución energética para experimento 4-160-50-1

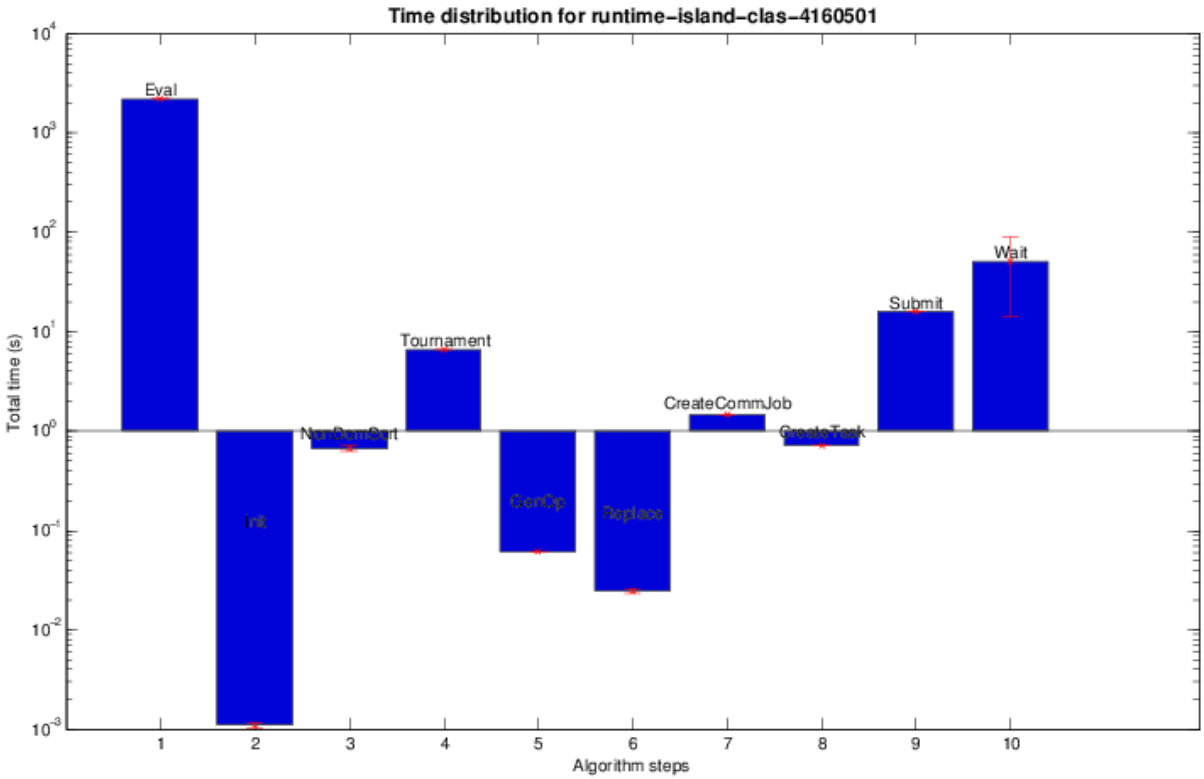


Figura 4.2: Distribución de tiempo para experimento 4-160-50-1

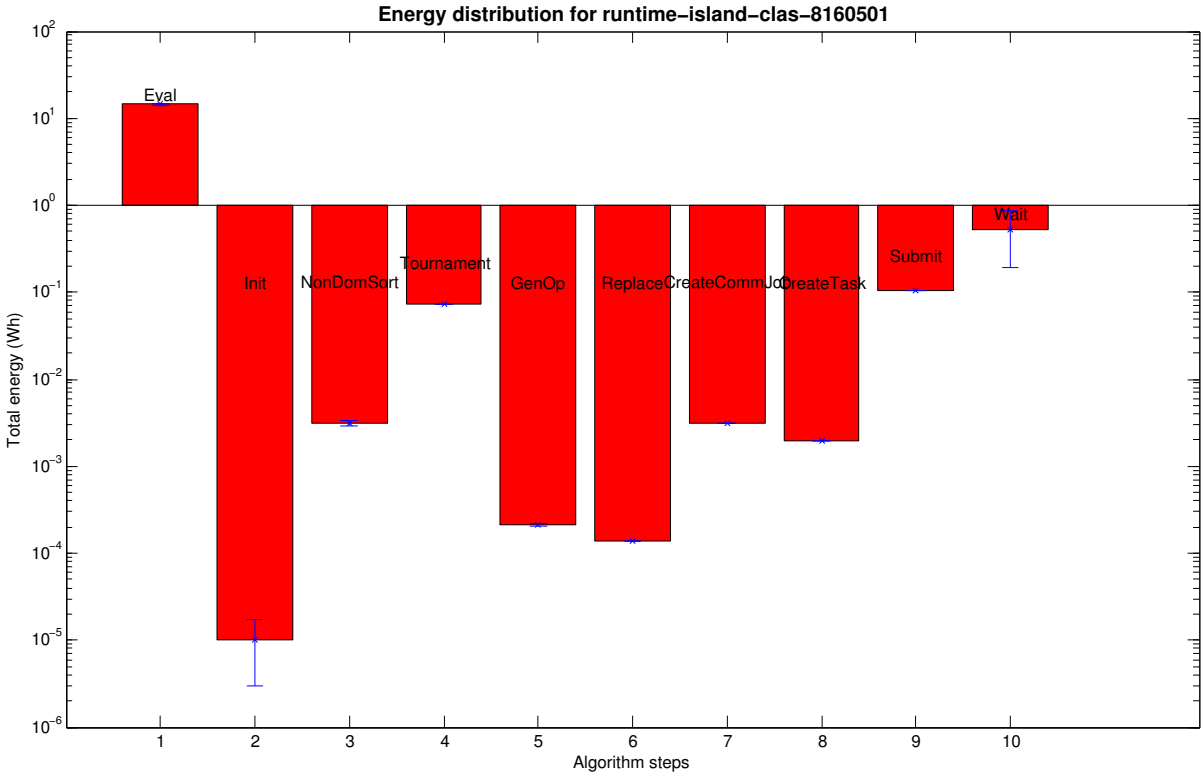


Figura 4.3: Distribución energética para experimento 8-160-50-1

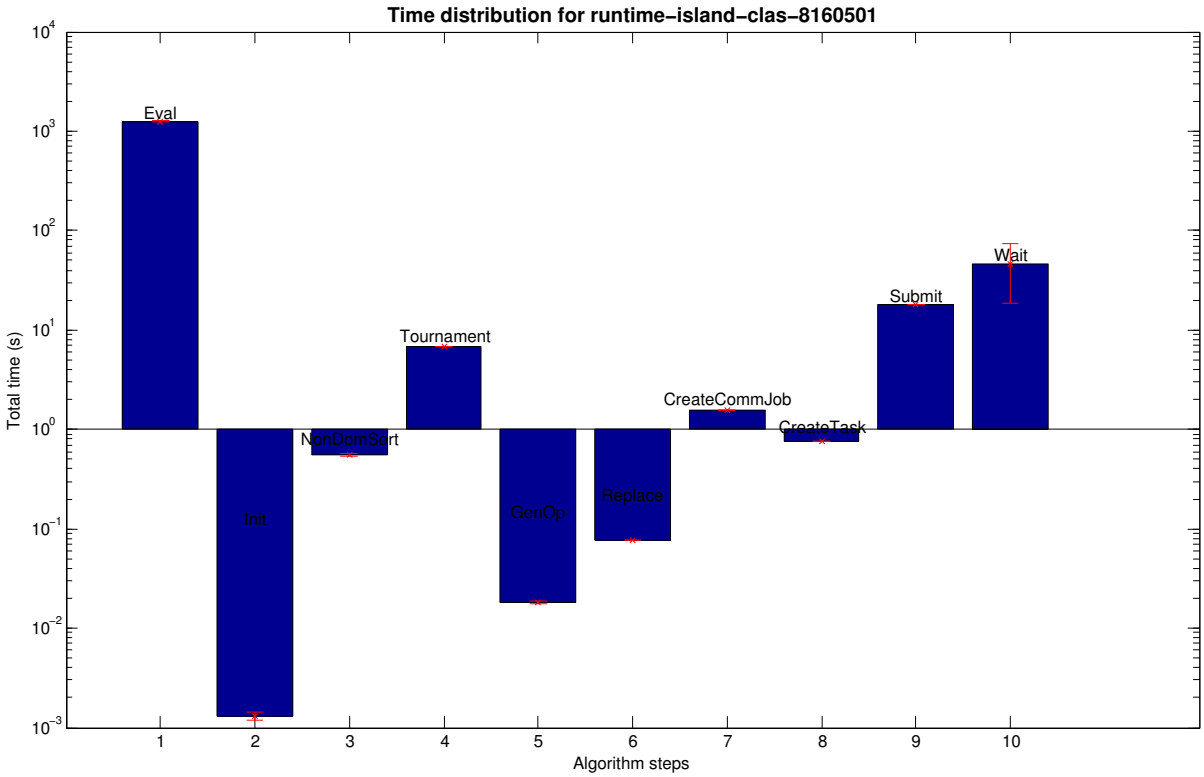


Figura 4.4: Distribución de tiempo para experimento 8-160-50-1

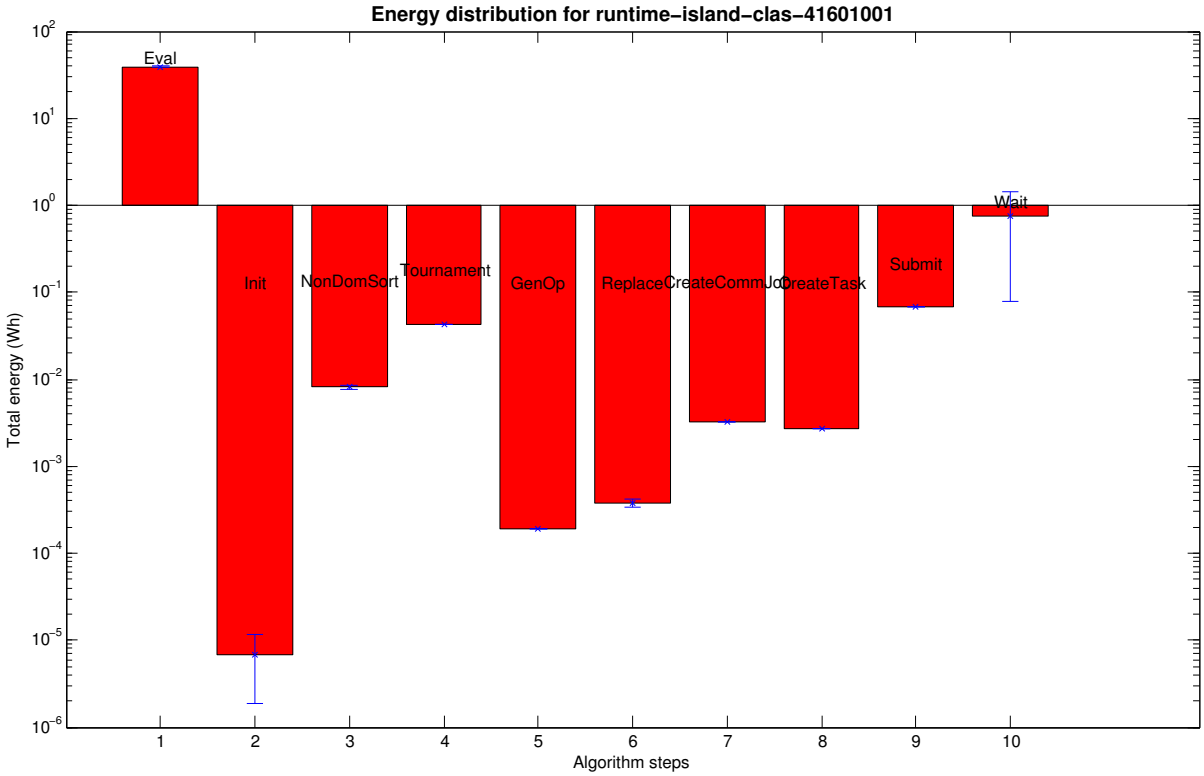


Figura 4.5: Distribución energética para experimento 4-160-100-1

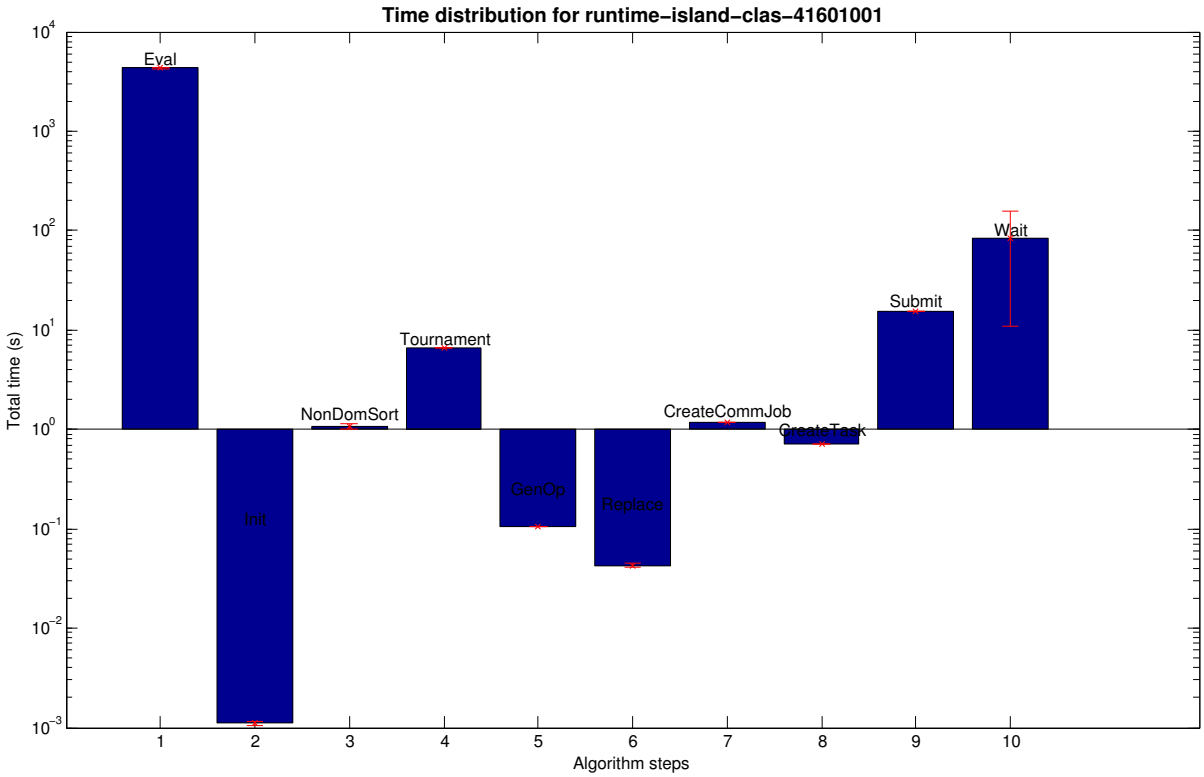


Figura 4.6: Distribución energética para experimento 4-160-100-1

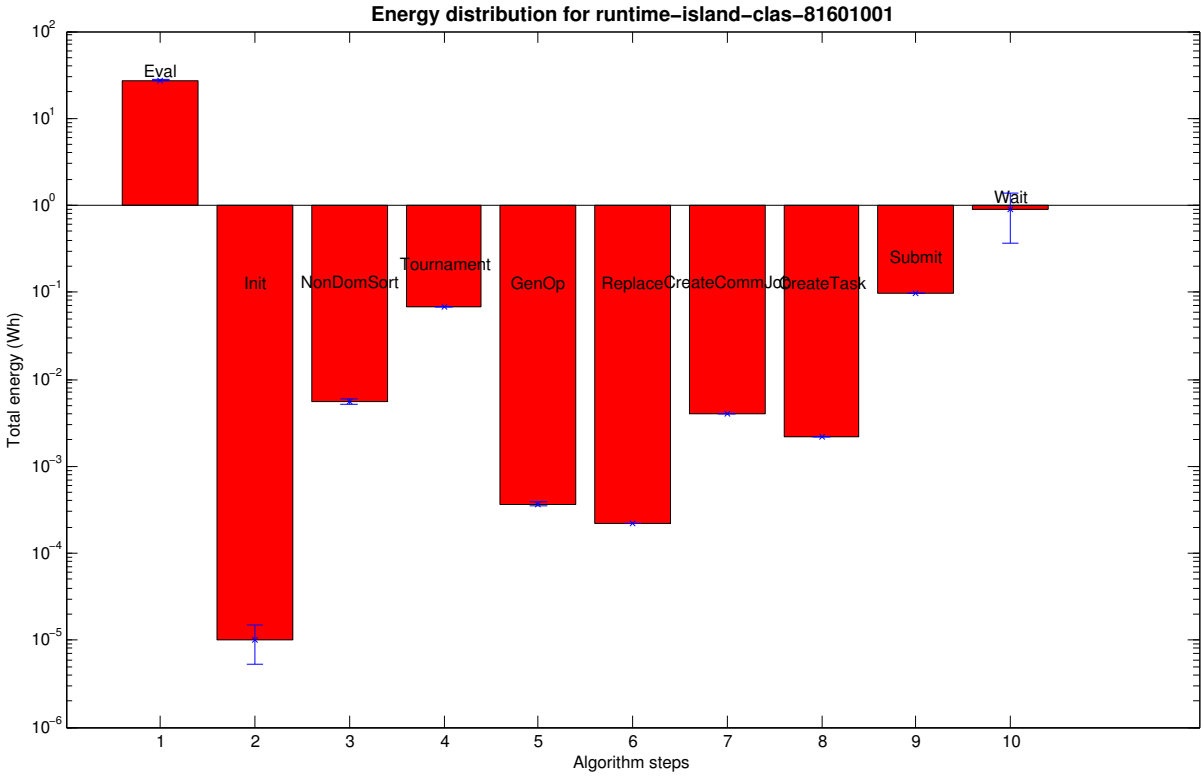


Figura 4.7: Distribución energética para experimento 8-160-100-1

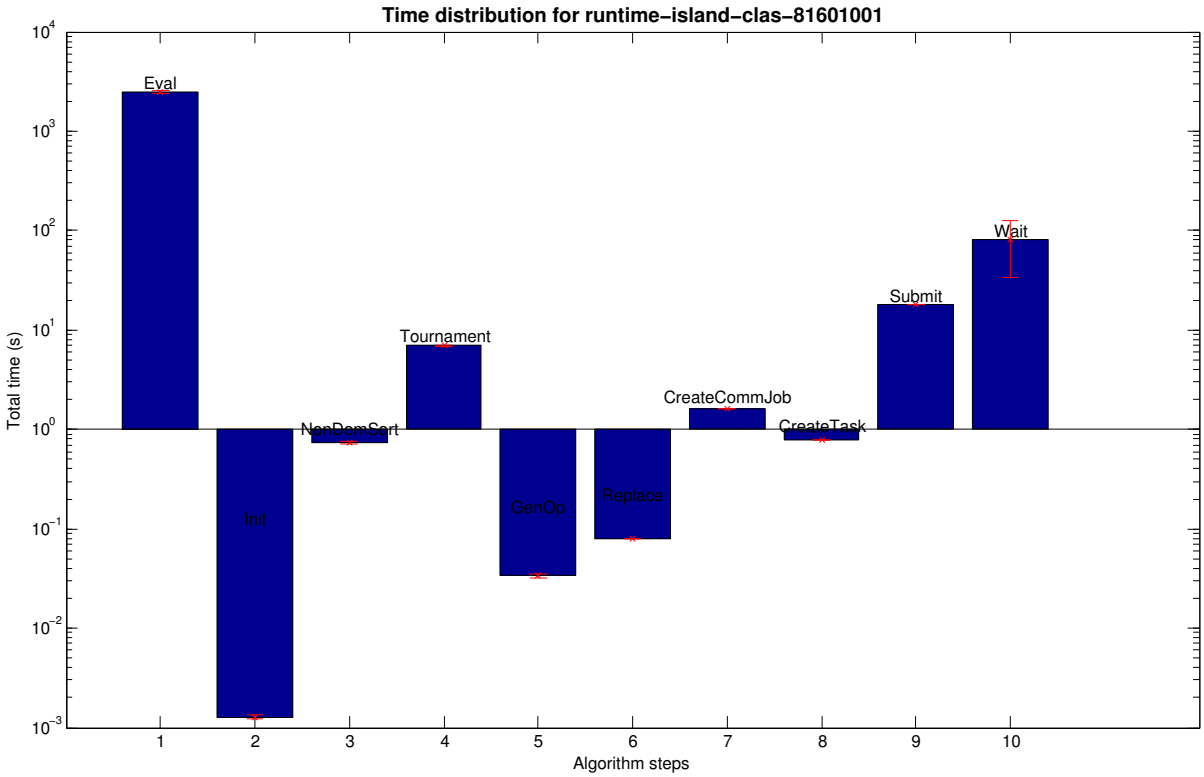


Figura 4.8: Distribución de tiempo para experimento 8-160-100-1

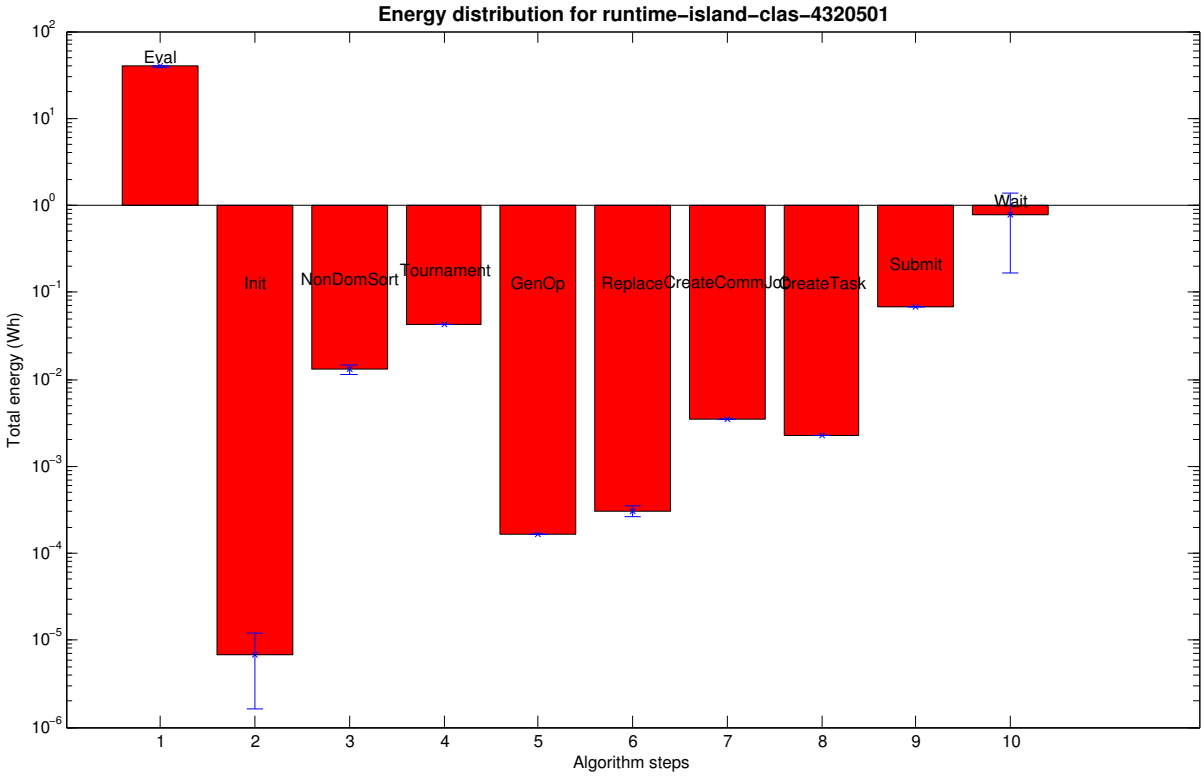


Figura 4.9: Distribución energética para experimento 4-320-50-1

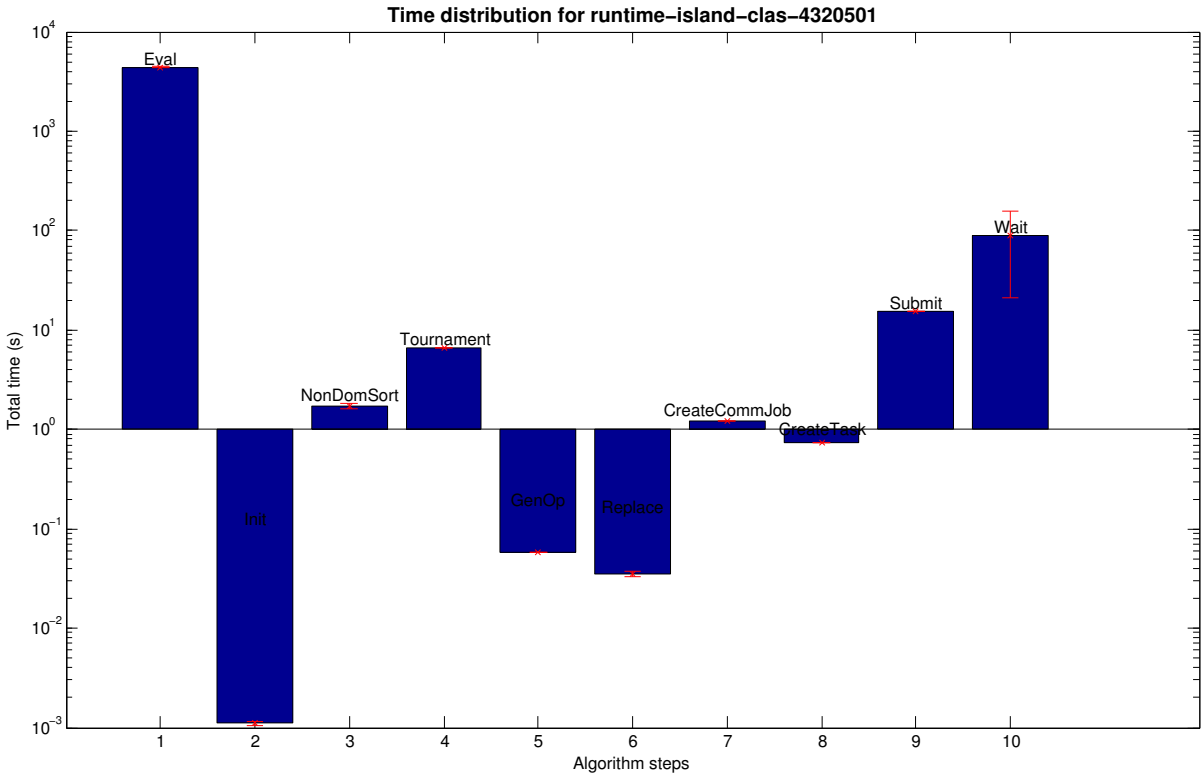


Figura 4.10: Distribución de tiempo para experimento 4-320-50-1

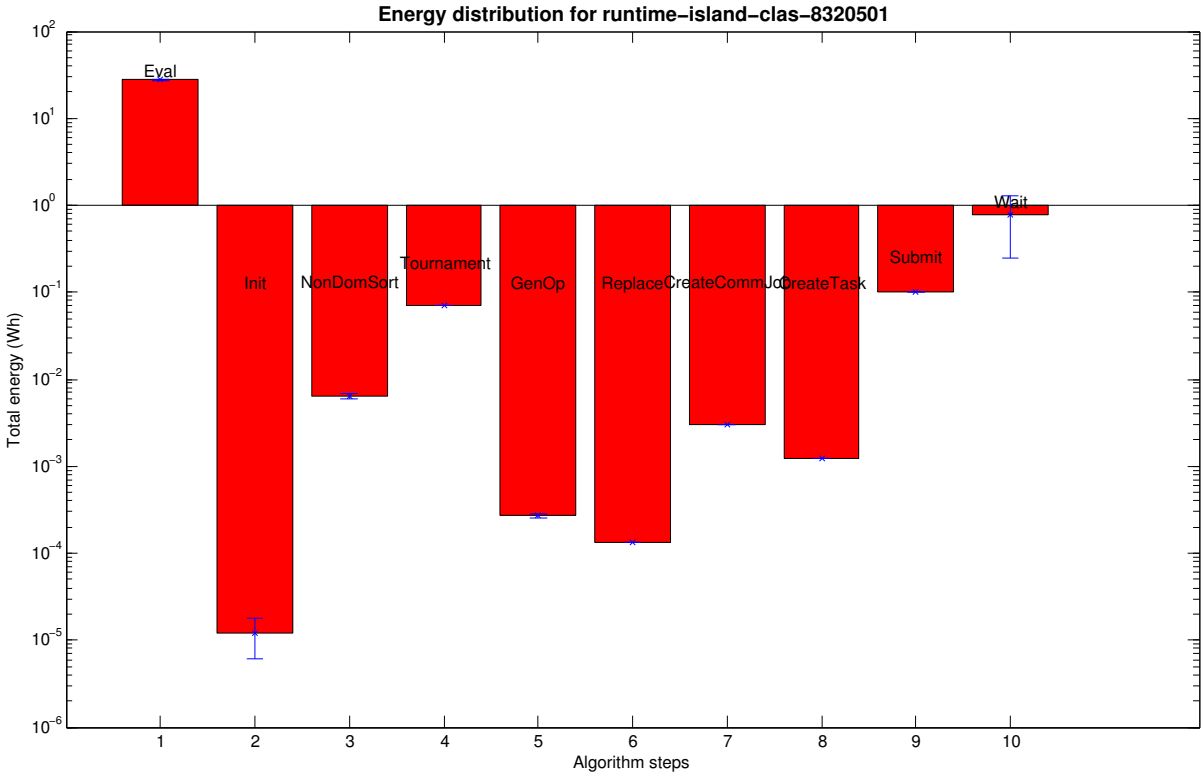


Figura 4.11: Distribución energética para experimento 8-320-50-1

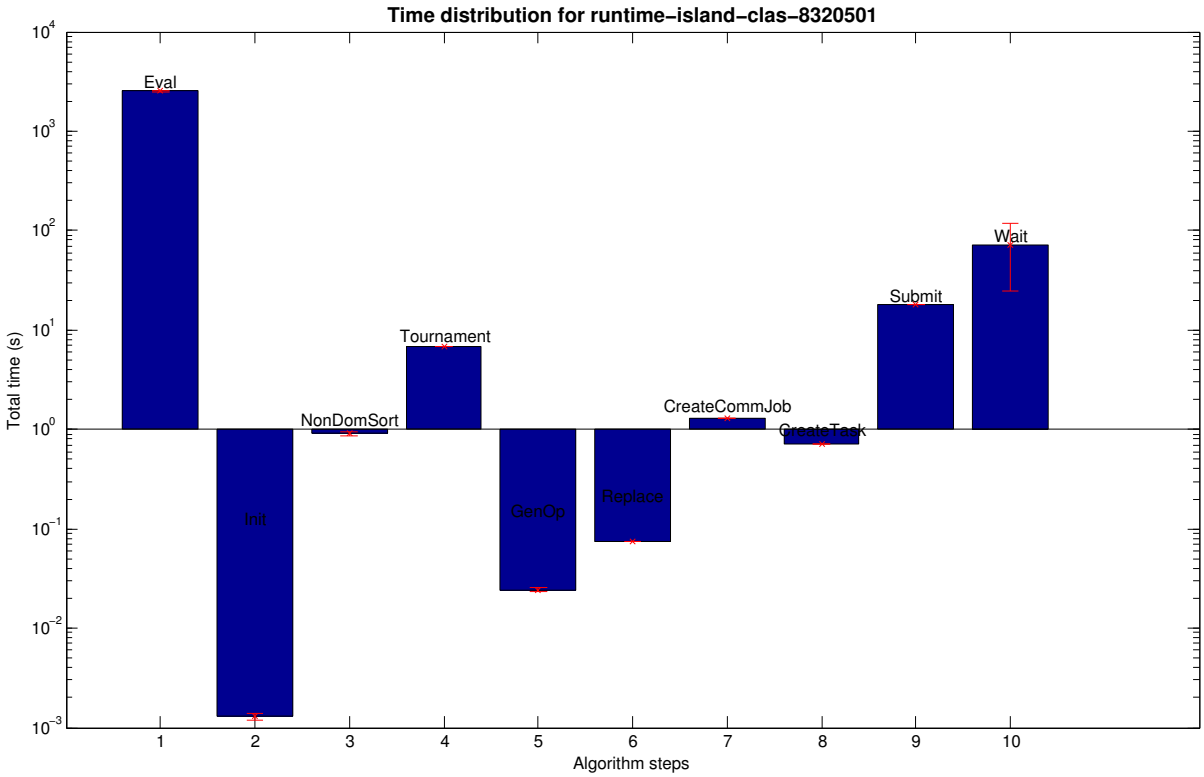


Figura 4.12: Distribución de tiempo para experimento 8-320-50-1

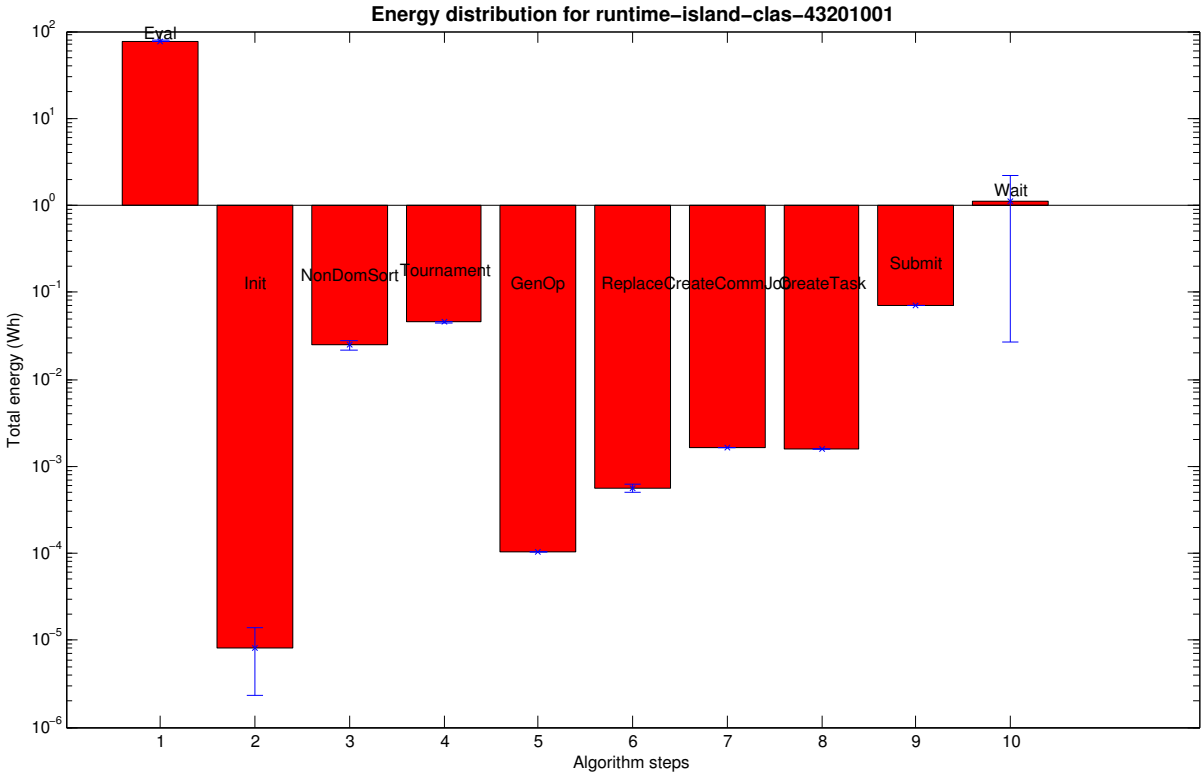


Figura 4.13: Distribución energética para experimento 4-320-100-1

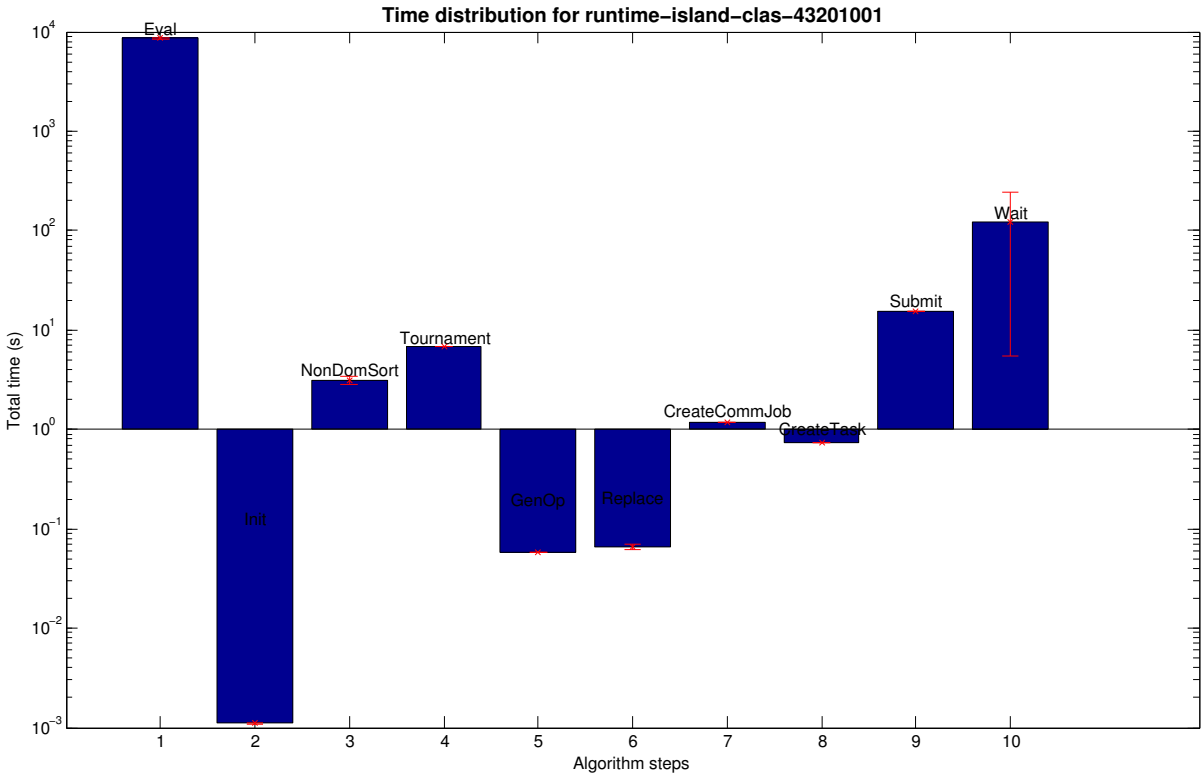


Figura 4.14: Distribución de tiempo para experimento 4-320-100-1

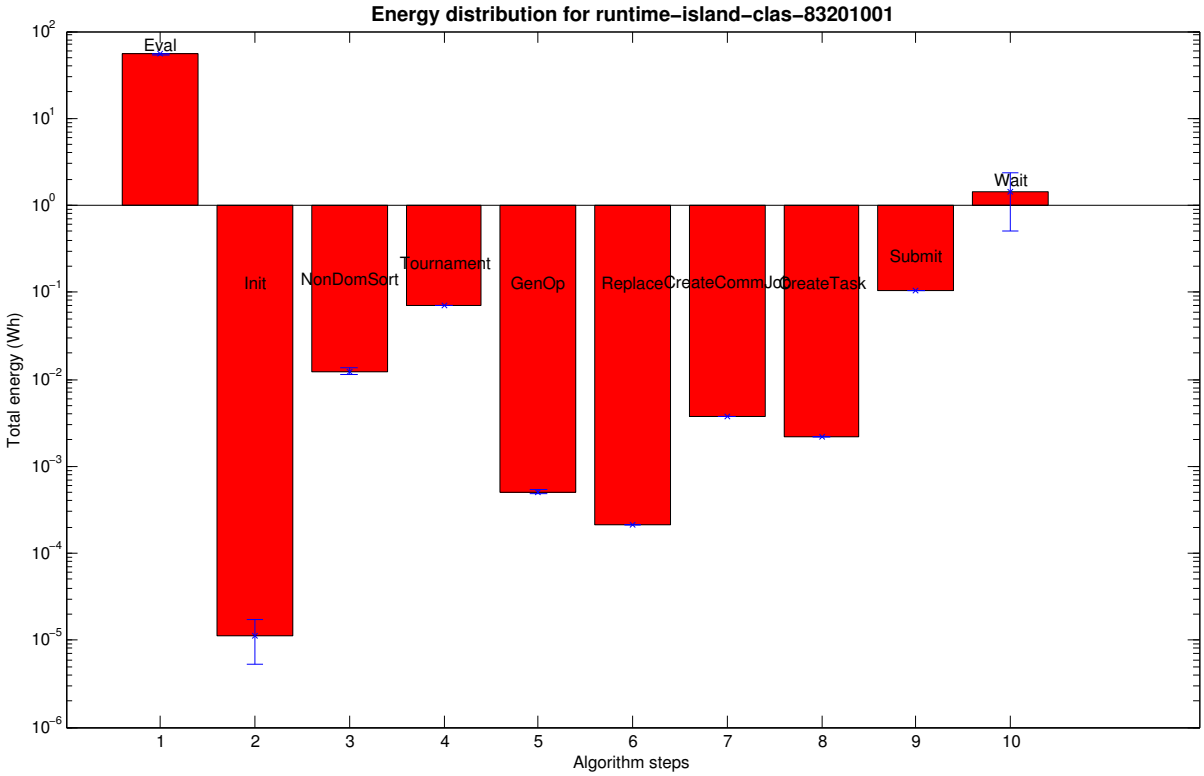


Figura 4.15: Distribución energética para experimento 8-320-100-1

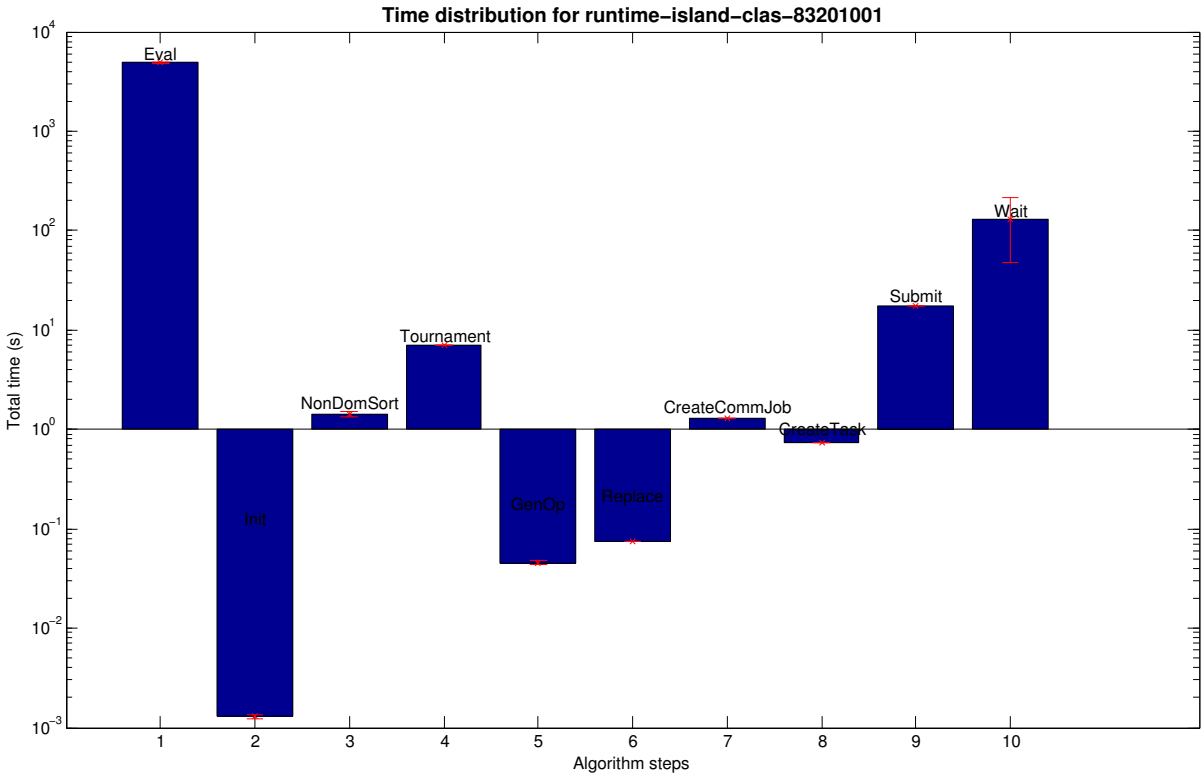


Figura 4.16: Distribución de tiempo para experimento 8-320-100-1

4.2. Tests estadísticos según experimento

En esta sección se van a realizar tests de hipótesis entre los ocho experimentos realizados para contrastar diferencias entre cada una de las fases del algoritmo según el número de hilos, la población a evaluar y el número de generaciones. Se trata de muestras independientes que no presentan normalidad, por ello, se utiliza el test U de Mann-Whitney.

4.2.1. Test según número de hilos

Se plantean las siguientes hipótesis H0 y H1.

H0: Las muestras para los experimentos ejecutados en 4 y 8 hilos pertenecen a la misma distribución.
H1: Las muestras para los experimentos ejecutados en 4 y 8 hilos provienen de distribuciones diferentes.

Los resultados del test se encuentran en la Tabla 4.9 para las distribuciones de tiempo, y en la Tabla 4.10 para las de energía. El contraste de hipótesis es el siguiente:

- H0 se rechaza para las fases *Evaluation, Init, NonDomSort, Tournament, GenOp, Replace, CreateCommJob* y *Submit*. Esto arroja el importante resultado de que la evaluación de los individuos, inicialización, ordenación y la aplicación de operadores genéticos difieren al hacerlo en 4 o en 8 núcleos. Esto mismo ocurre con las órdenes de MATLAB para crear el trabajo de comunicación y su envío. Todo esto encaja perfectamente con lo que se tenía previsto, ya que todos estos procesos vienen directamente ligados a la subpoblación con N/P individuos de cada hilo.
- H0 se cumple para las fases *CreateTask* y *Wait*. Del mismo modo, era un resultado previsto ya que son funciones internas de MATLAB a nivel de creación del proceso y sincronización final.

Tabla 4.9: Test de hipótesis (U de Mann-Whitney) para distribuciones de Tiempo.

p-valores			
Evaluation	$3.95 \cdot 10^{-03}$***	Replace	$1.44 \cdot 10^{-14}$***
Init	$1.43 \cdot 10^{-14}$***	CreateCommJob	$3.77 \cdot 10^{-06}$***
NonDomSort	$1.07 \cdot 10^{-04}$***	CreateTask	$3.63 \cdot 10^{-01}$
Tournament	$4.37 \cdot 10^{-10}$***	Submit	$2.18 \cdot 10^{-13}$***
GenOp	$1.44 \cdot 10^{-14}$***	Wait	$5.87 \cdot 10^{-01}$

Tabla 4.10: Test de hipótesis (U de Mann-Whitney) para distribuciones de Energía.

p-valores			
Evaluation	$3.95 \cdot 10^{-03}$***	Replace	$4.41 \cdot 10^{-12}$***
Init	$1.61 \cdot 10^{-07}$***	CreateCommJob	$7.37 \cdot 10^{-03}$***
NonDomSort	$1.92 \cdot 10^{-04}$***	CreateTask	$9.50 \cdot 10^{-01}$
Tournament	$1.44 \cdot 10^{-14}$***	Submit	$1.44 \cdot 10^{-14}$***
GenOp	$6.21 \cdot 10^{-11}$***	Wait	$2.02 \cdot 10^{-01}$

4.2.2. Test según número de individuos en la población

Se plantean las siguientes hipótesis H0 y H1.

H0: Las muestras para los experimentos ejecutados sobre 160 o 320 individuos pertenecen a la misma distribución.
H1: Las muestras para los experimentos ejecutados sobre 160 o 320 individuos provienen de distribuciones diferentes.

Los resultados del test se encuentran en la Tabla 4.11 para las distribuciones de tiempo, y en la Tabla 4.12 para las de energía. El contraste de hipótesis es el siguiente:

- H0 se rechaza para las fases *Evaluation* y *NonDomSort* tanto en tiempo como consumo energético. Esto quiere decir que, como cabía esperar, para distintos números de individuos en la población, el consumo energético y el tiempo utilizado difieren tanto para el proceso de evaluación como de ordenación no-dominante. También puede verse que H0 se rechaza para *Wait* en tiempo de ejecución, aunque esto puede ser simplemente debido a ruido en las medidas, se trata de una función interna de MATLAB sobre la que no se tiene control.
- H0 se cumple para el resto de pasos del algoritmo, en consumo energético y de tiempo. Esto arroja el importante resultado de que la diferencia de población propuesta no afecta a los pasos de inicialización, selección por torneo, aplicación de operadores genéticos ni reemplazo. Estos resultados encajan con el paradigma SPMD implementado en el algoritmo Las funciones de MATLAB que generan el proceso de comunicación entre hilos, las tareas y su envío tampoco se ven afecatadas.

Tabla 4.11: Test de hipótesis (U de Mann-Whitney) según población para distribuciones de Tiempo.

p-valores			
Evaluation	$1.10 \cdot 10^{-10}$***	Replace	$5.16 \cdot 10^{-01}$
Init	$7.80 \cdot 10^{-01}$	CreateCommJob	$6.55 \cdot 10^{-01}$
NonDomSort	$1.92 \cdot 10^{-11}$***	CreateTask	$4.05 \cdot 10^{-01}$
Tournament	$4.11 \cdot 10^{-01}$	Submit	$5.22 \cdot 10^{-01}$
GenOp	$9.35 \cdot 10^{-01}$	Wait	$1.59 \cdot 10^{-05}$***

Tabla 4.12: Test de hipótesis (U de Mann-Whitney) según población para distribuciones de Energía.

p-valores			
Evaluation	$2.52 \cdot 10^{-02}$***	Replace	$6.34 \cdot 10^{-01}$
Init	$7.25 \cdot 10^{-01}$	CreateCommJob	$8.06 \cdot 10^{-01}$
NonDomSort	$2.06 \cdot 10^{-02}$***	CreateTask	$8.89 \cdot 10^{-01}$
Tournament	$6.13 \cdot 10^{-01}$	Submit	$9.35 \cdot 10^{-01}$
GenOp	$6.20 \cdot 10^{-01}$	Wait	$1.70 \cdot 10^{-01}$

4.2.3. Test según número de generaciones

Se plantean las siguientes hipótesis H0 y H1.

H0: Las muestras para los experimentos ejecutados durante 50 y 100 generaciones pertenecen a la misma distribución.
H1: Las muestras para los experimentos ejecutados durante 50 y 100 generaciones provienen de distribuciones diferentes.

Los resultados del test se encuentran en la Tabla 4.13 para las distribuciones de tiempo, y en la Tabla 4.14 para las de energía. El contraste de hipótesis es el siguiente:

- H0 se rechaza para *Evaluation, NonDomSort* y *Replace* tanto en tiempo como en energía, es decir, tan sólo estas tres partes del proceso se ven afectadas por haber duplicado el número de generaciones de 50 a 100. Además, H0 también se rechaza para *Wait* en cuanto a distribuciones de tiempo.
- H0 se acepta para el resto de partes del algoritmo. El proceso de inicialización (*Init*) no sufre diferencias dependiendo del número de generaciones a evolucionar, tal y cómo se esperaba, pero se puede destacar que las fases de selección por torneo (*Tournament*) y aplicación de operadores genéticos (*GenOp*) tampoco lo han hecho. Esto puede explicarse por el ínfimo coste energético y de tiempo que estas fases llevan, además de venir implementadas como SIMD/SPMD, junto a las altas desviaciones estándar que presentan respecto a sus media.

Tabla 4.13: Test de hipótesis (U de Mann-Whitney) según generaciones para distribuciones de Tiempo.

p-valores			
Evaluation	$3.67 \cdot 10^{-07}$***	Replace	$1.86 \cdot 10^{-02}$***
Init	$4.88 \cdot 10^{-01}$	CreateCommJob	$7.47 \cdot 10^{-01}$
NonDomSort	$2.33 \cdot 10^{-04}$***	CreateTask	$4.00 \cdot 10^{-01}$
Tournament	$4.16 \cdot 10^{-01}$	Submit	$5.41 \cdot 10^{-01}$
GenOp	$1.11 \cdot 10^{-01}$	Wait	$4.54 \cdot 10^{-06}$***

Tabla 4.14: Test de hipótesis (U de Mann-Whitney) según generaciones para distribuciones de Energía.

p-valores			
Evaluation	$4.27 \cdot 10^{-07}$***	Replace	$3.09 \cdot 10^{-03}$***
Init	$6.00 \cdot 10^{-01}$	CreateCommJob	$7.18 \cdot 10^{-01}$
NonDomSort	$5.42 \cdot 10^{-05}$***	CreateTask	$2.09 \cdot 10^{-01}$
Tournament	$8.66 \cdot 10^{-01}$	Submit	$8.81 \cdot 10^{-01}$
GenOp	$1.67 \cdot 10^{-01}$	Wait	$1.40 \cdot 10^{-05}$

4.3. Clustering y detección de anomalías

Como se ha visto en el apartado anterior, la evaluación de los individuos supone el grueso en tiempo y energía del desarrollo del algoritmo. Por ello, a continuación, se presentan los resultados de aplicar clustering mediante el algoritmo de las k-medias [24] a ese conjunto de datos (Figuras 4.17 - 4.24), con los siguientes objetivos:

1. Clasificación de los distintos comportamientos encontrados.
2. Detección de anomalías (outliers)

Los experimentos realizados en 4 hilos han sido distribuidos en 3 clusters, mientras que los que se han desarrollado en 8 hilos se han distribuido perfectamente sólo en 2. Esto es así por la evidente presencia de dos comportamientos crecientes $\Delta tiempo - \Delta energía$ superpuestos que parecen provocar una generación de un cluster intermedio extra en los experimentos ejecutados en 4 hilos, quedando a la izquierda un único comportamiento creciente, en medio dos comportamientos crecientes superpuestos, y a la derecha un clúster disperso de datos. Para 8 núcleos estos dos clusters quedan perfectamente definidos como uno a la izquierda compacto y otro a la derecha disperso.

Como hipótesis de esta diferencia sustancial en la distribución de los datos para 4 y 8 núcleos se plantean varias casuísticas, desde la presencia de cambios de contexto en la ejecución del algoritmo (al no ocuparse todos los núcleos, se favorecen los cambios de contexto desde la propia gestión del sistema operativo) hasta fallos de caché, requiriendo así accesos a memoria principal (distribuir la población en 4 hilos en lugar de 8 implicaría la formación de subpoblaciones más grandes con las que lidiar). Estas hipótesis quedan aún pendientes de validar una vez se realicen las pruebas con una mayor recogida de datos a nivel de kernel mediante herramientas como PERF. Además, hay que tener en cuenta también las capas extra que añade MATLAB (y Java por detrás) dentro de la gestión del programa.

Otro comportamiento curioso se presenta en la Figura 4.18 donde la distribución tiene el doble de "anchura"que el resto de experimentos desarrollados con 4 y 8 hilos. Por ello, se dejará fuera de la elaboración del modelo en apartados posteriores.

A continuación, en los siguientes subapartados, se van a ir desarrollando los diferentes métodos de detección y eliminación de anomalías probados.

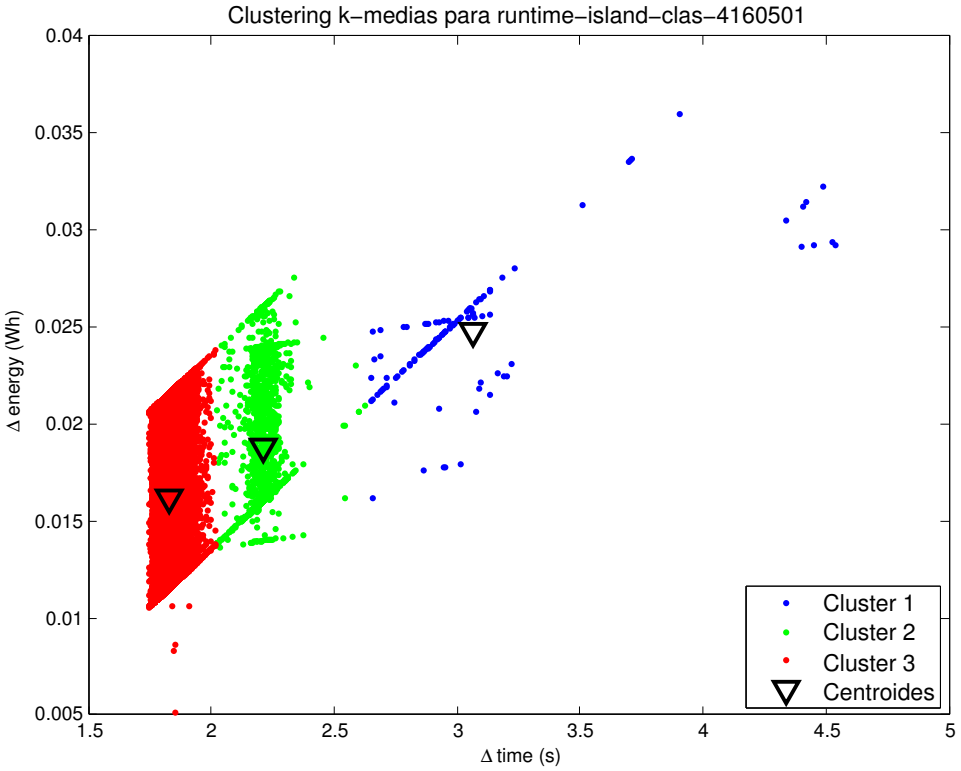


Figura 4.17: Clustering para experimento 4-160-50-1

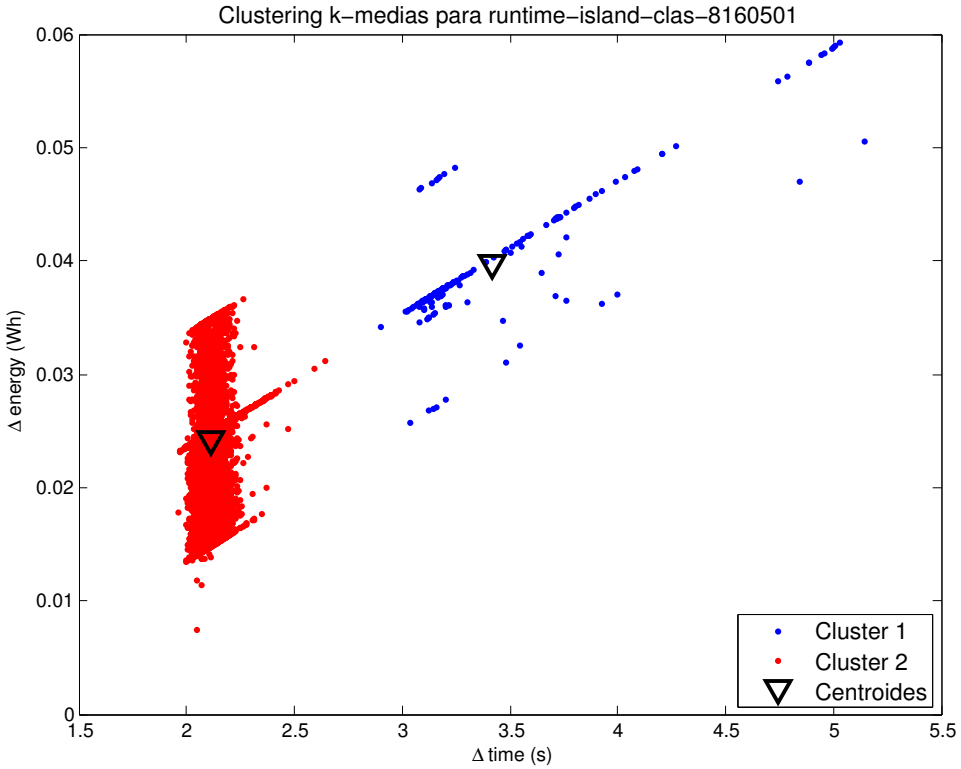


Figura 4.18: Clustering para experimento 8-160-50-1

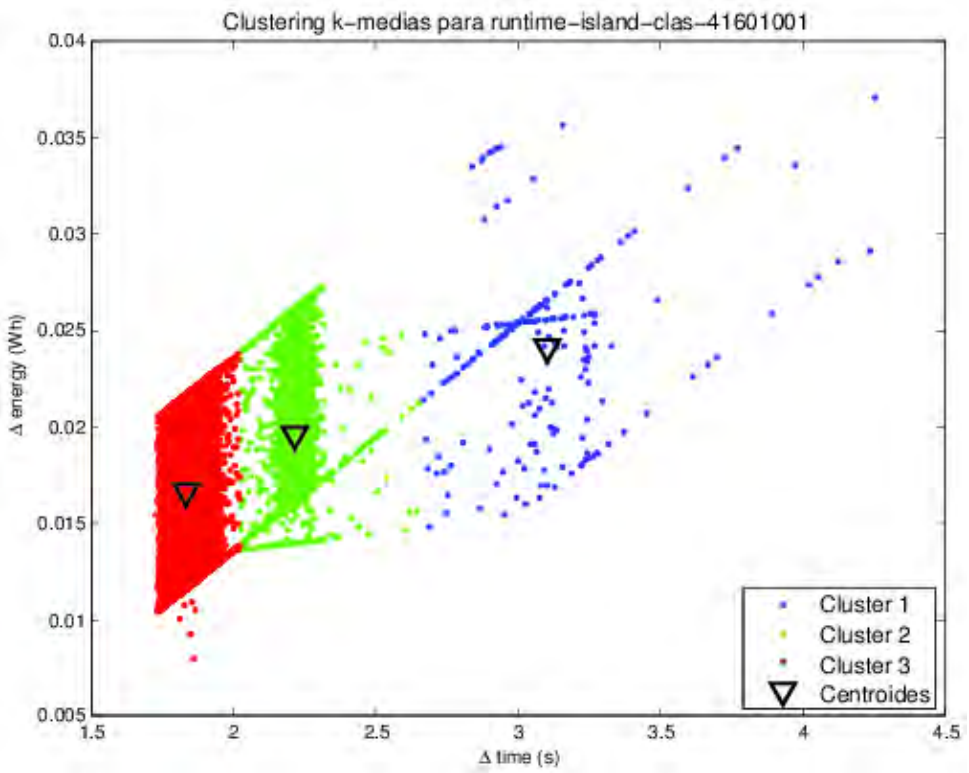


Figura 4.19: Clustering para experimento 4-160-100-1

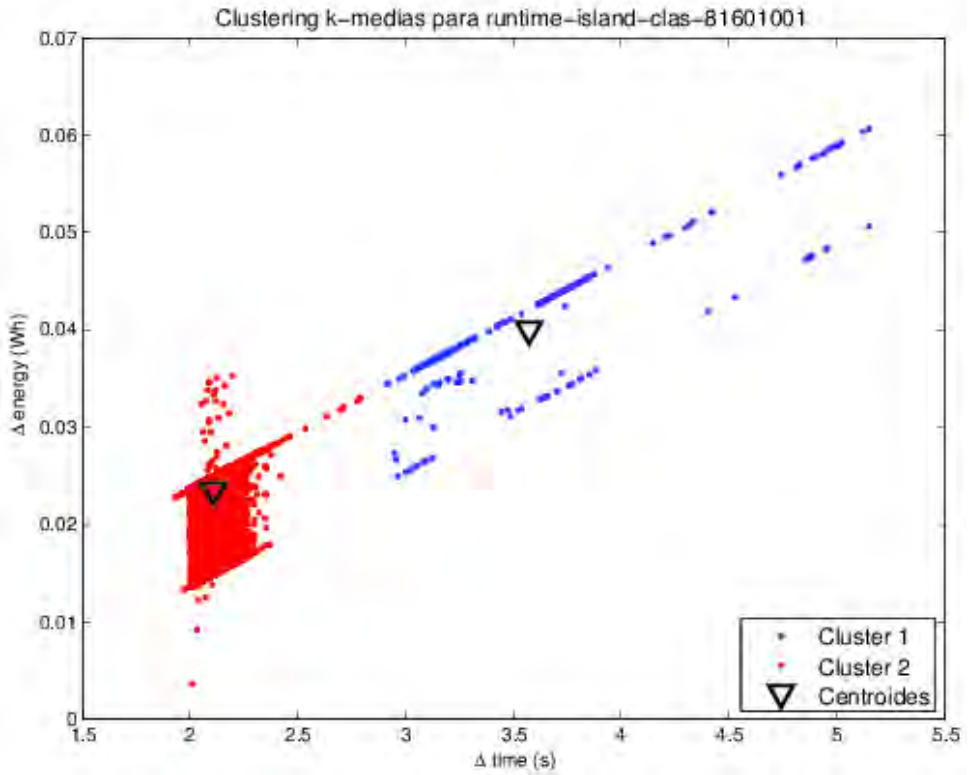


Figura 4.20: Clustering para experimento 8-160-100-1

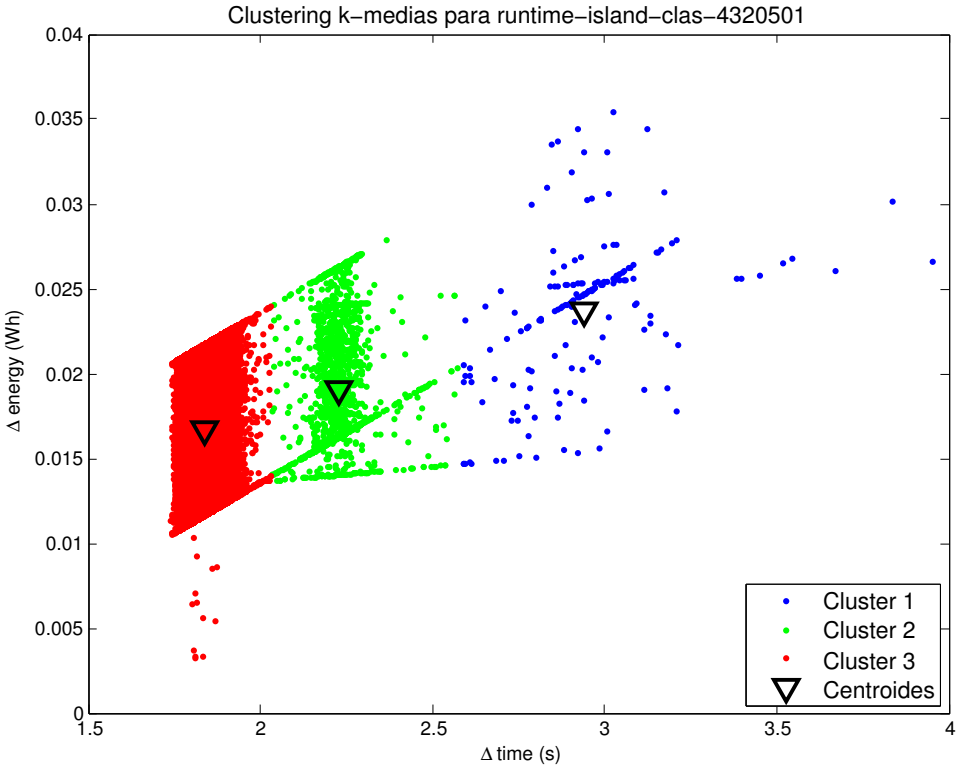


Figura 4.21: Clustering para experimento 4-320-50-1

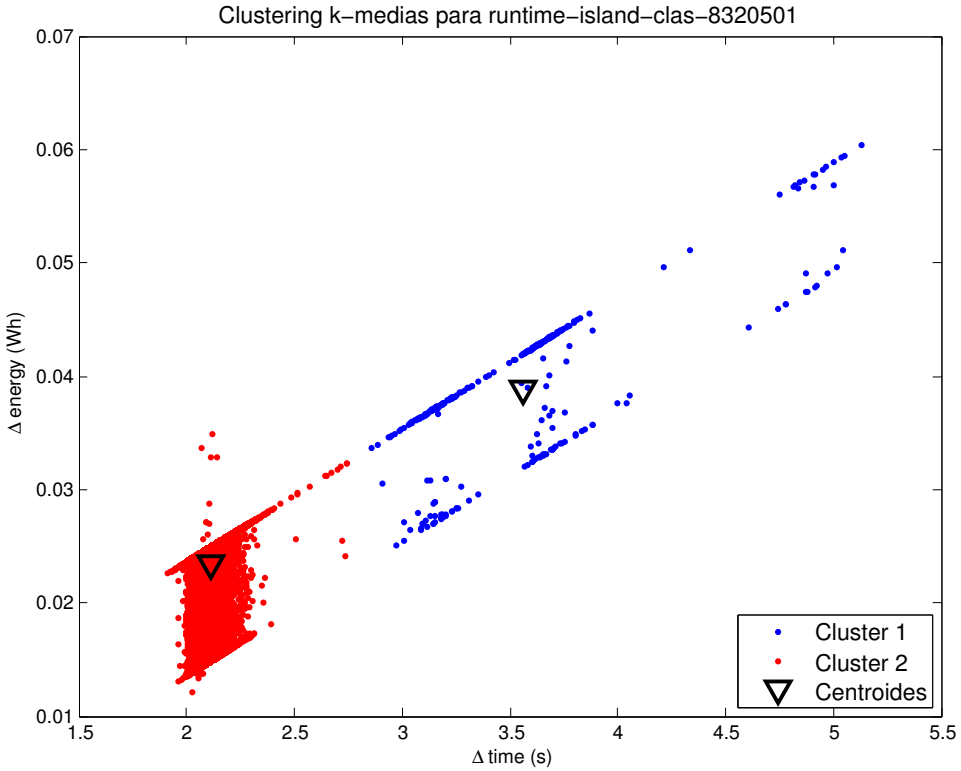


Figura 4.22: Clustering para experimento 8-320-50-1

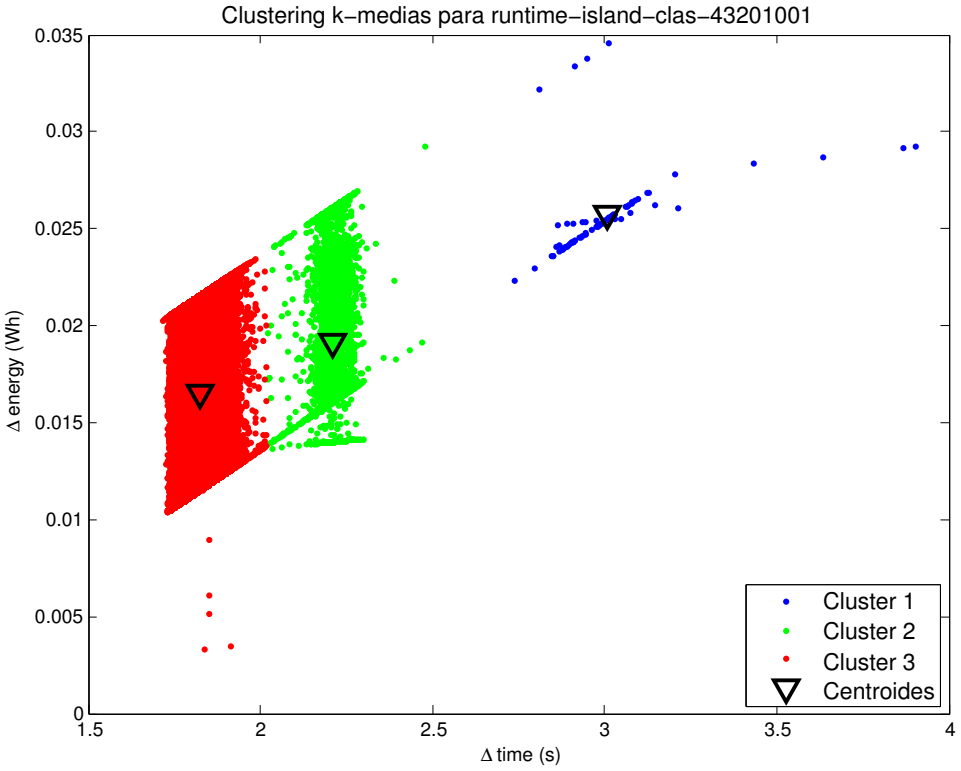


Figura 4.23: Clustering para experimento 4-320-100-1

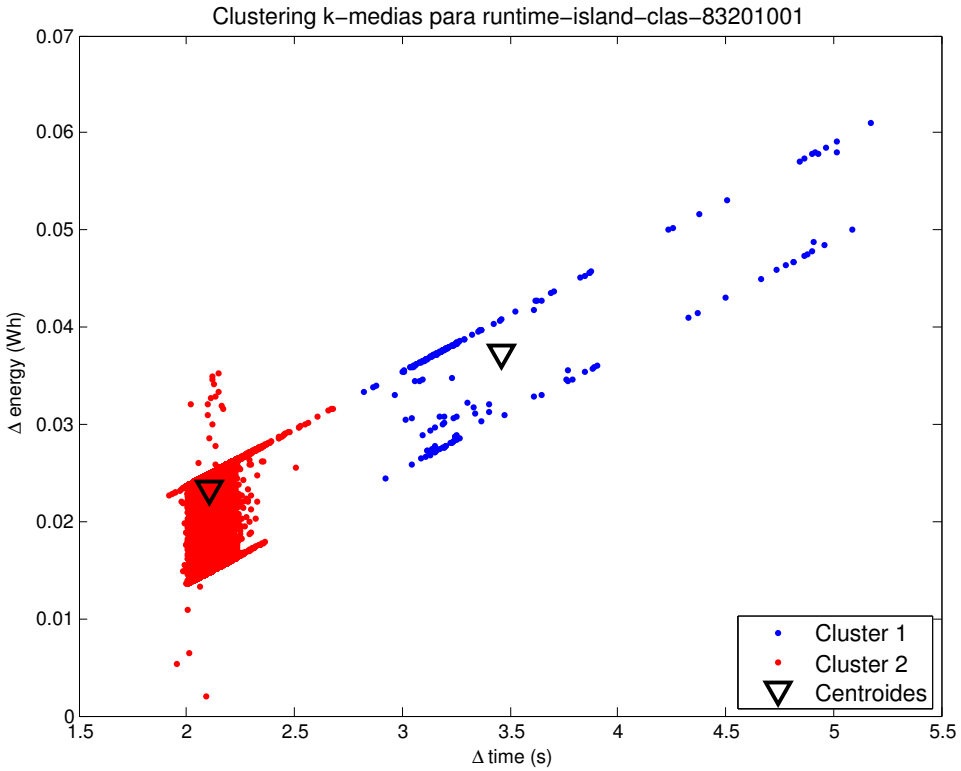


Figura 4.24: Clustering para experimento 8-320-100-1

4.3.1. Detección de anomalías mediante IQR

InterQuartile Range (IQR) o rango intercuartil ha sido la primera de las opciones escogidas entre todos los métodos de búsqueda de anomalías. Se trata de un método de detección de anomalías univariable que asume distribución normal de los datos. Se ha aplicado tanto sobre el eje X (Δt) como el eje Y (ΔE) [25].

$$extremeOutliers_{left} = Q1 - 3 \cdot IQR \tag{4.1}$$
$$extremeOutliers_{right} = Q3 + 3 \cdot IQR \tag{4.2}$$

Se aplica sobre el experimento *4-160-50-1* para ejemplificar sus resultados. Analicemos los resultados según cluster:

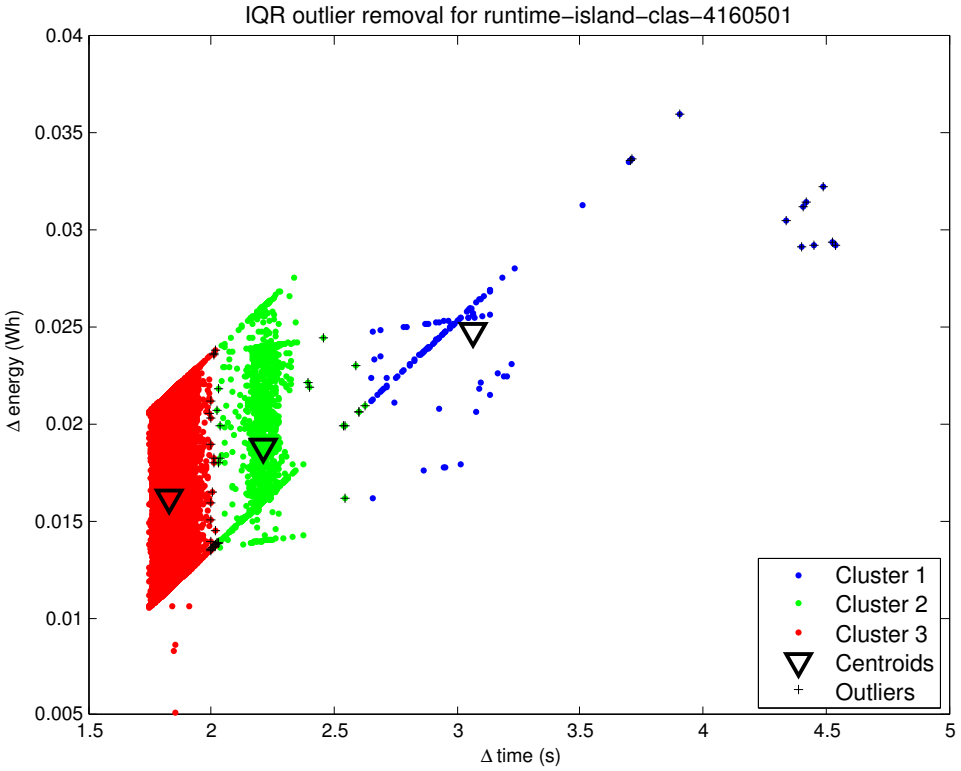
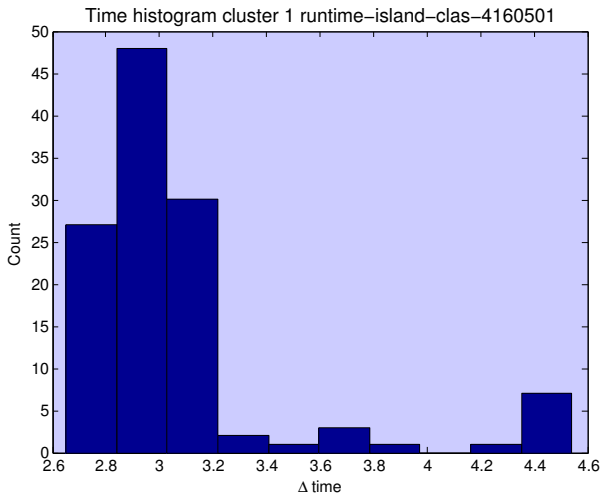


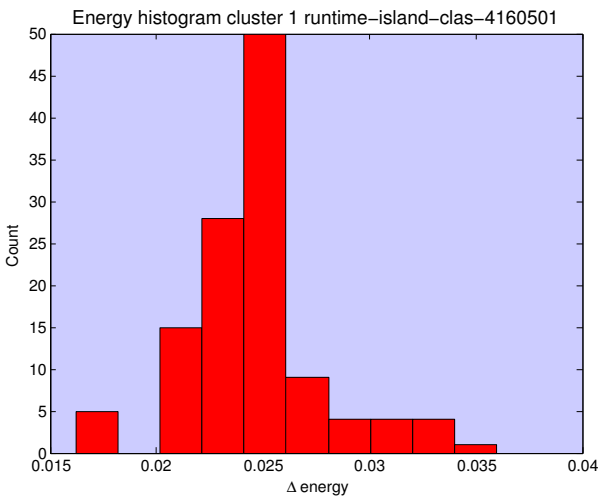
Figura 4.25: Detección de outliers (IQR) para experimento 4-160-50-1

- **Cluster 1**: Cluster disperso. Parece hacer una selección buena de outliers, eliminando los más alejados del centroide. El test de *Shapiro-Wilk* no asegura normalidad de los datos (Figura 4.26).
- **Cluster 2**: Cluster con combinación de comportamientos. La selección de outliers también es coherente con lo que se busca, esa mezcla de los dos comportamientos avistados. El test de *Shapiro-Wilk* no asegura normalidad de los datos (Figura 4.27).
- **Cluster 3**: Cluster de comportamiento único. La detección de outliers no es correcta, puede verse cómo hay puntos con un consumo energético por debajo de los 0.01Wh, aproximadamente, que no han sido marcados. Esto es debido a la agrupación de los datos de energía que se muestra en el histograma de la Figura 4.28b, que no se asemeja en nada a una distribución normal, mostrando una gran acumulación de datos de energía en los laterales. El test de *Shapiro-Wilk* no asegura normalidad de los datos.

Otros métodos basados en la distancia al centroide dan resultados similares, con los mismos problemas de detección en el cluster 3, y sin diferencias apreciables para los clusters 1 y 2.

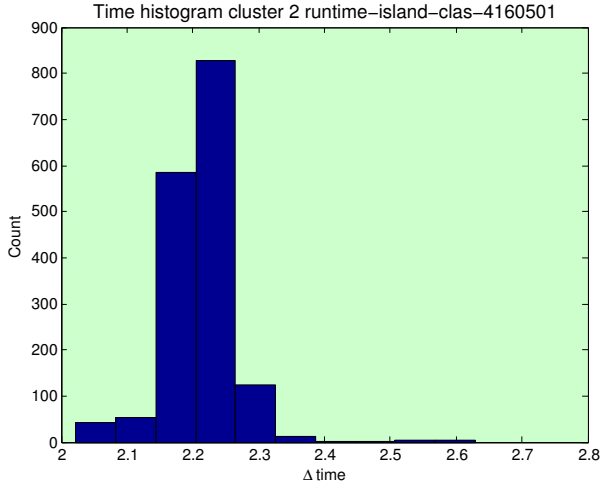


(a) Consumo de tiempo en ejecución 4-160-50-1

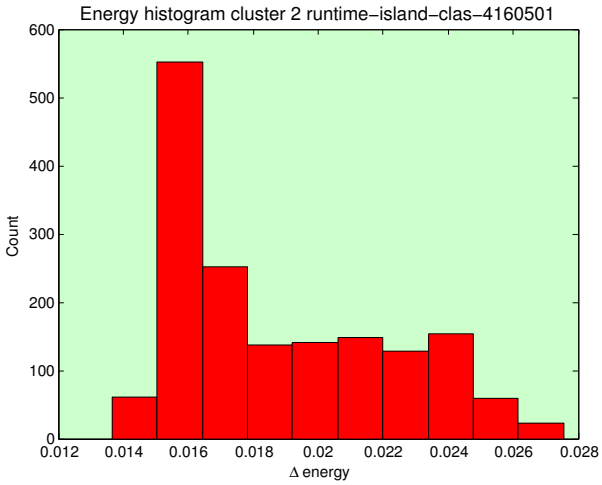


(b) Consumo energético en ejecución 4-160-50-1

Figura 4.26: Histogramas para Cluster 1

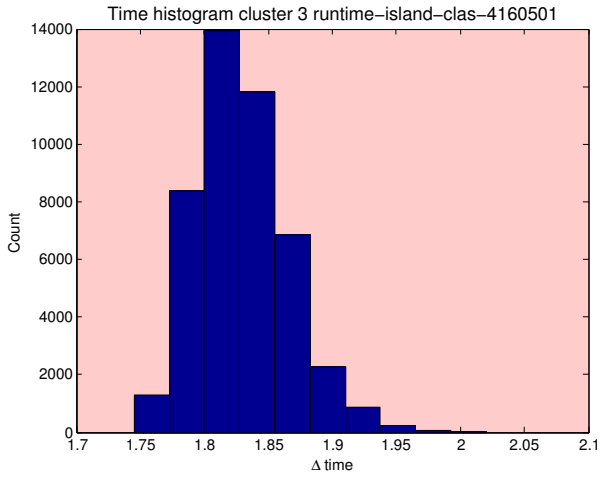


(a) Consumo de tiempo en ejecución 4-160-50-1

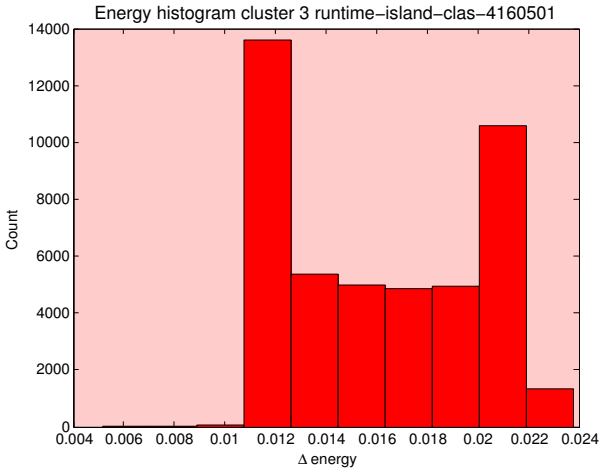


(b) Consumo energético en ejecución 4-160-50-1

Figura 4.27: Histogramas para Cluster 2



(a) Consumo de tiempo en ejecución 4-160-50-1



(b) Consumo energético en ejecución 4-160-50-1

Figura 4.28: Histogramas para Cluster 3

4.3.2. Detección de anomalías mediante distancia k-NN

k-Nearest Neighbours es una técnica normalmente utilizada en clasificación que busca la caracterización de cada punto en función de las propiedades de sus k-vecinos más cercanos [26]. En este caso, se va a aplicar IQR a la distancia k-NN buscando aquellas zonas de menor densidad de puntos. Se han realizado experimentos con 1,3 y 5 vecinos (Figura 4.29 - 4.31). Se aplica sobre el experimento *4-160-50-1* para ejemplificar sus resultados.

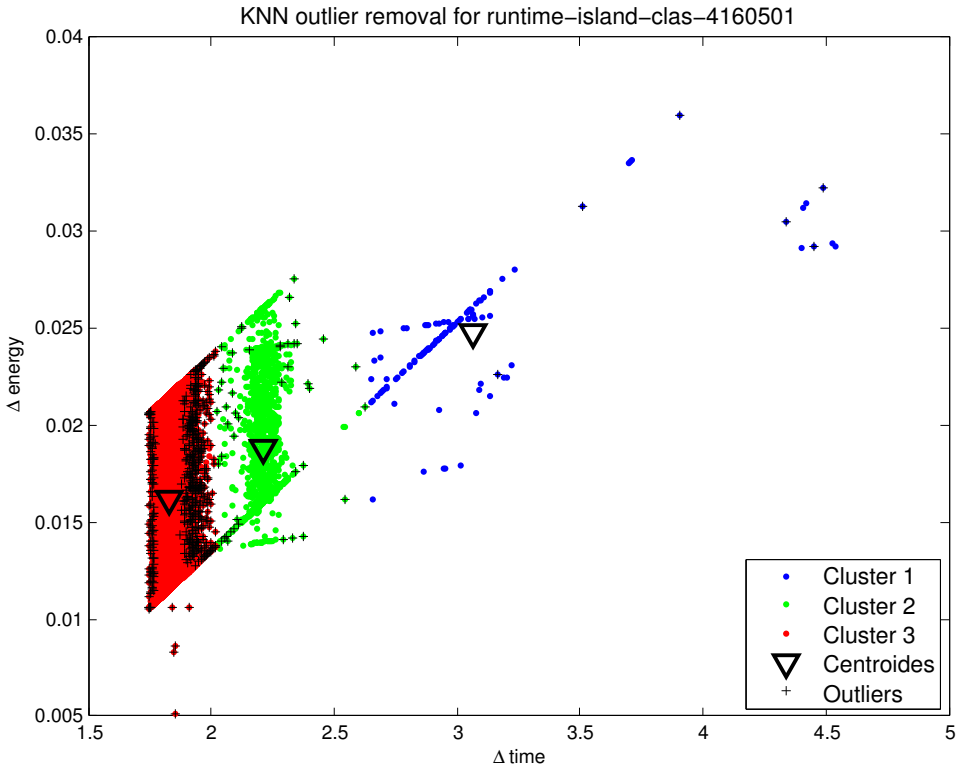


Figura 4.29: Detección de outliers (1NN) para experimento 4-160-50-1

- **Cluster 1**: Cluster disperso. La selección no es buena, hay puntos dispersos que no han sido seleccionados para 1NN, 3NN y 5NN. El test de *Shapiro-Wilk* no asegura normalidad de las distancias kNN.
- **Cluster 2**: Cluster con combinación de comportamientos. La selección de outliers aumenta la dispersión en los laterales del cluster (1NN), e incluso dentro del mismo (3NN, 5NN). El test de *Shapiro-Wilk* no asegura normalidad de las distancias kNN.
- **Cluster 3**: Cluster de comportamiento único. la selección de outliers tampoco es correcta. Aunque se seleccionan las anomalías más destacadas, es decir, los puntos por debajo de 0.01Wh (1NN, 3NN), por otro lado se aumenta la dispersión en los laterales (1NN) e intercluster (3NN, 5NN) al seleccionar una gran cantidad de puntos como anomalías. El test de *Shapiro-Wilk* no asegura normalidad de las distancias kNN.

En definitiva, este método no aporta nada a la selección de outliers de ningún cluster. Esto es debido a la gran heterogeneidad en las densidades intracluster, especialmente en el Cluster 3, donde las partes superior e inferior tienen una densidad altísima en comparación con los puntos intermedios (ver histograma en Figura 4.28b).

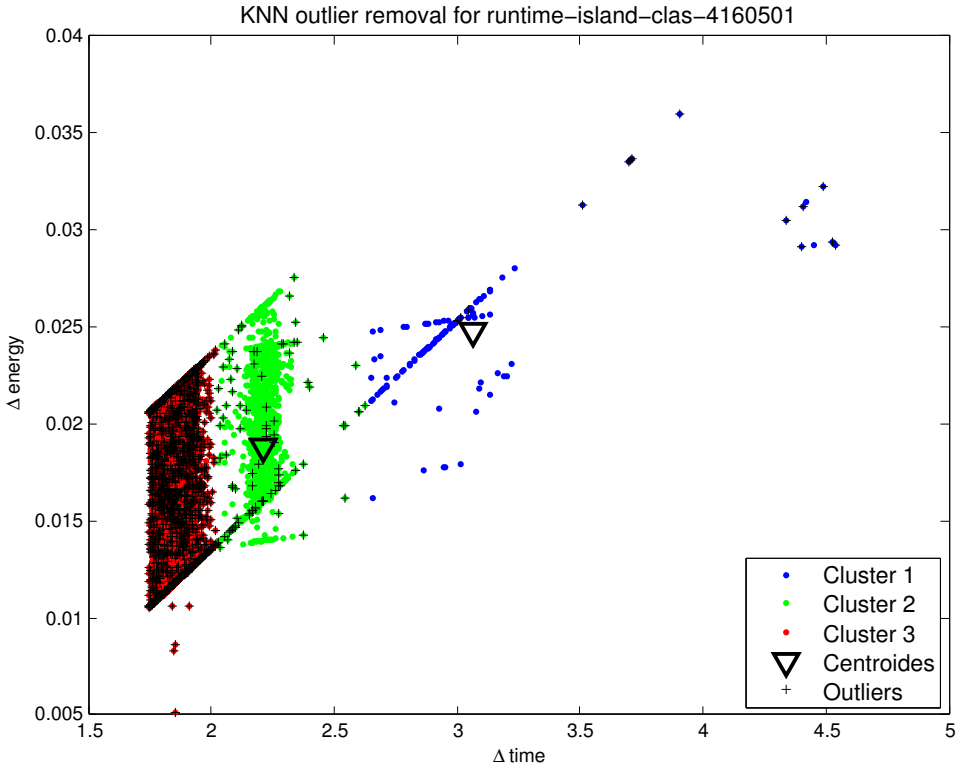


Figura 4.30: Detección de outliers (3NN) para experimento 4-160-50-1

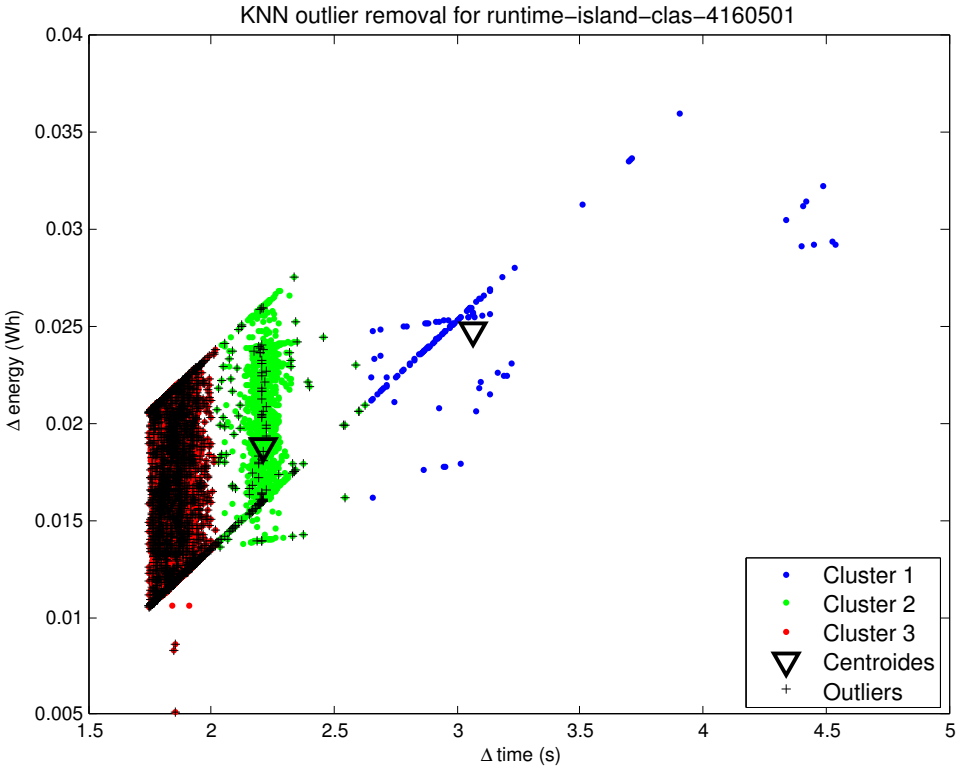


Figura 4.31: Detección de outliers (5NN) para experimento 4-160-50-1

4.3.3. Detección de anomalías mediante LOF

Local Outlier Factor clasifica anomalías en función de la desviación de sus datos respecto a sus vecinos más cercanos [27]. A continuación se expone el pseudocódigo para el cálculo de LOF en un único punto.

$$\textbf{Input} : X, k \mid p, q, kNearestNeighbour \in X; k \in \mathbb{N} \tag{4.3}$$

$$d(p,q) := distance(p,q) \tag{4.4}$$

$$d_k(p) := distance(p, kNearestNeighbour) \tag{4.5}$$

$$N_k(p) \leftarrow ||q \in X \mid d(p,q) \leqslant d_k(p)|| \tag{4.6}$$

$$Rd_k(p,q) = max\{d_k(q), d(p,q)\} \tag{4.7}$$

$$LRD_k(p) = 1/\frac{\sum_{q \in N_k(p)}[Rd_k(p,q)]}{|N_k(p)|} \tag{4.8}$$

$$LOF_k(p) = \frac{\sum_{q \in N_k(p)}[LRD_k(q)]}{|N_k(p)|}/LDR_k(p) \tag{4.9}$$

El cálculo de LOF para un punto viene descrito en las ecuaciones 2.3 - 2.9. Primero se calcula la distancia $d(p,q)$ desde el punto p hasta el resto de puntos q (ec. 2.4), y la k-distancia $d_k(p)$ del mismo punto p (ec. 2.5), es decir, la distancia hasta su k-vecino mas cercano. Se hace un recuento de todos los puntos a menor o igual k-distancia $d_k(p)$ desde p en $N_k(p)$ (ec. 2.6). A continuación, se define la distancia de alcance o *Reachability Distance* $Rd_k(p,q)$ como el máximo entre la k-distancia y la distancia real entre los puntos (ec. 2.7). Por último, se define una distancia local de alcance $LRD_k(p)$ como la inversa de la media de las distancias de alcance de todos puntos recuperados en $N_k(p)$ (ec. 2.8) para poder finalmente calcular el factor local de anomalía (*Local Outlier Factor* o LOF) como la media de las $LRD_k(q)$ calculadas en el paso anterior, divididas por la $LRD_k(p)$ del punto estudiado, es decir, relativizando la medida LOF al punto p.

En la Figura 4.32 se han utilizado los 3 vecinos más cercanos para computar LOF. Se aplica sobre el experimento *4-160-50-1* para ejemplificar sus resultados. Se ha utilizado la *OutlierDetectionToolbox* de MATLAB de G. Erdogan [28].

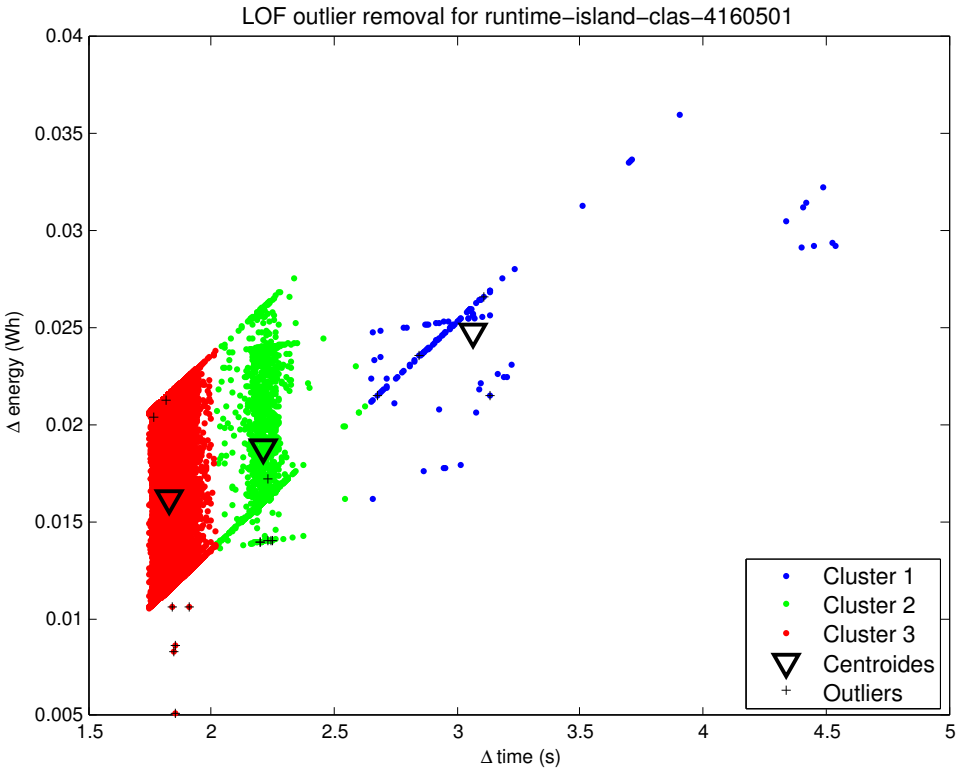


Figura 4.32: Detección de outliers (LOF) para experimento 4-160-50-1

- **Cluster 1**: Cluster disperso. La selección no es buena, hay puntos dispersos que no han sido seleccionados. Umbral LOF mayor que 3.
- **Cluster 2**: Cluster con combinación de comportamientos. La selección de outliers no es buena para distintos umbrales de LOF. Se muestran outliers con LOF mayores que 40.
- **Cluster 3**: Cluster de comportamiento único. La selección de outliers en este caso es suficientemente buena, eliminando aquellos puntos claramente fuera del cluster. Se ha seleccionado el mínimo umbral necesario LOF que abarcase a los cinco outliers referidos, consecuentemente, eliminando otros dos puntos en la parte superior del cluster. Umbral LOF mayor que 300.

LOF se erige como la solución para estudiar los "Cluster 3" de cada uno de los experimentos ejecutados en cuatro hilos y los "Cluster 2" de los ejecutados en 8 hilos. El resto serán procesados mediante detección de outliers IQR.

4.3.4. Detección de anomalías mediante selección manual, LOF e IQR

A continuación se utilizan los métodos de detección LOF e IQR en los distintos clústeres que aparecen en cada uno de los experimentos. Para cada uno de ellos, la k escogida dentro de LOF varía según la naturaleza del cluster. En ellos puede observarse variabilidad de densidades, por lo que es muy importante afinar bien este parámetro. Hay casos en los que esto no es posible por culpa de la alta heterogeneidad de densidades en la agrupación de los datos; en ellos se ha realizado una selección manual. A continuación se exponen las peculiaridades de cada experimento:

- Los experimentos ejecutados en 4 hilos, mostrados en las Figuras 4.33 - 4.36, mantienen una estructura similar en todo momento. La detección de outliers con IQR en los clusters 1 y 2 de cada uno de ellos funciona bastante bien; con LOF, en los clusters 3, hay que ser más preciso, puesto que el número de vecinos es crucial para hacer una correcta detección de los outliers a nivel local. Esto junto a la densidad no homogénea del cluster 1 (ver Figura 4.28b) hace que, incluso encontrando el valor de k óptimo, haya algunos pocos falsos positivos inevitables en la detección.
- Los experimentos ejecutados en 8 hilos no han permitido el correcto uso de LOF en la detección de anomalías por la alta heterogeneidad de densidades en los mismos, por lo que finalmente ha sido un ajuste manual (Figuras 4.37 - 4.40). Esto ha permitido la disección de unos clusters muy compactos que permitirán una modelización posterior más robusta. Cabe destacar cómo IQR ha eliminado pocos o ningún punto dentro de las distribuciones de los clusters 1.

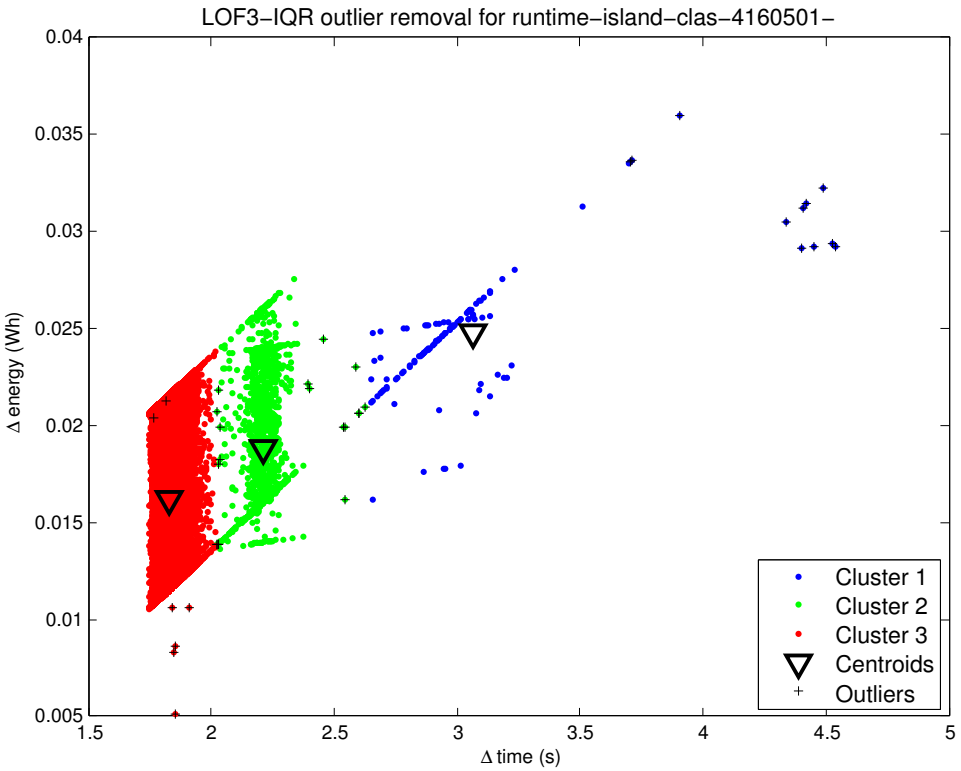


Figura 4.33: Detección de outliers (LOF e IQR) para experimento 4-160-50-1

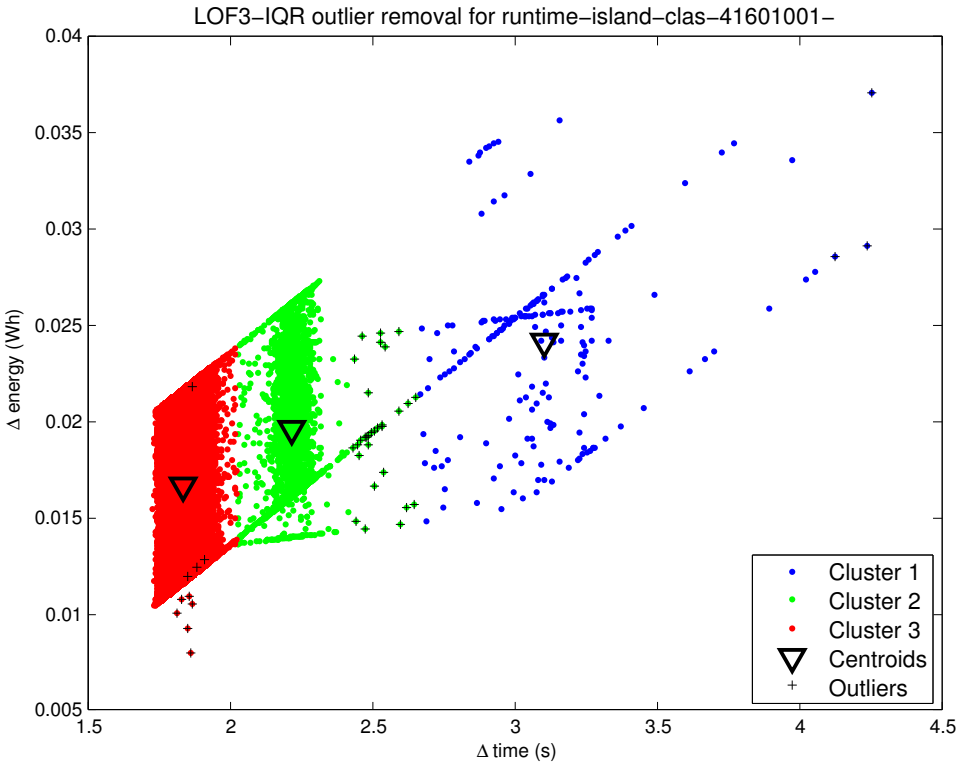


Figura 4.34: Detección de outliers (LOF e IQR) para experimento 4-160-100-1

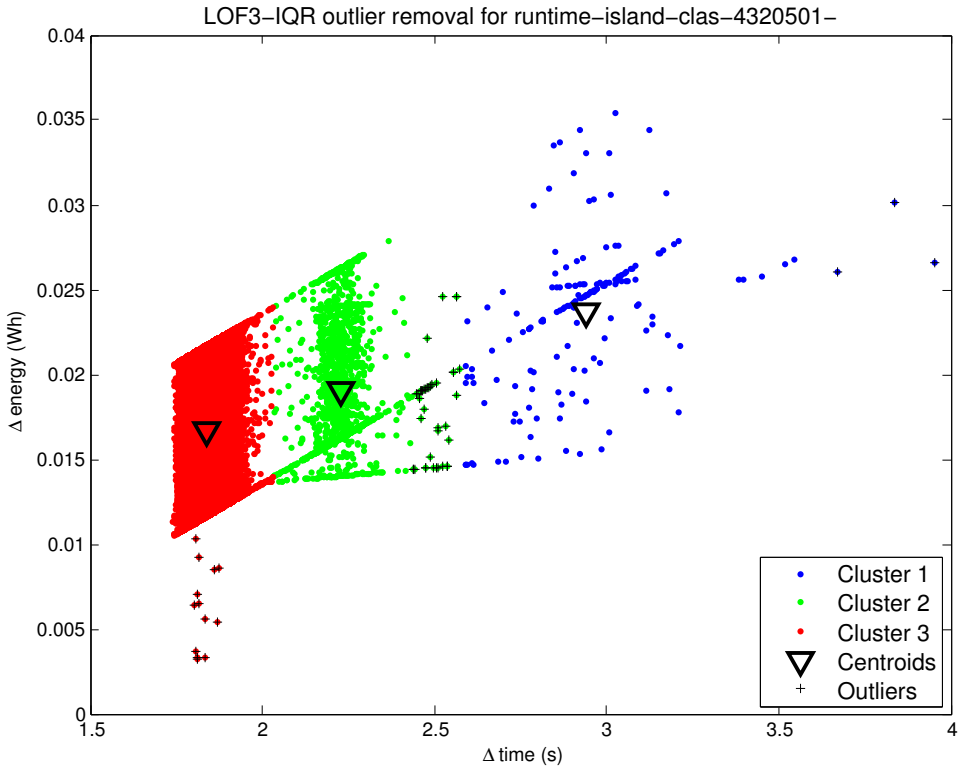


Figura 4.35: Detección de outliers (LOF e IQR) para experimento 4-320-50-1

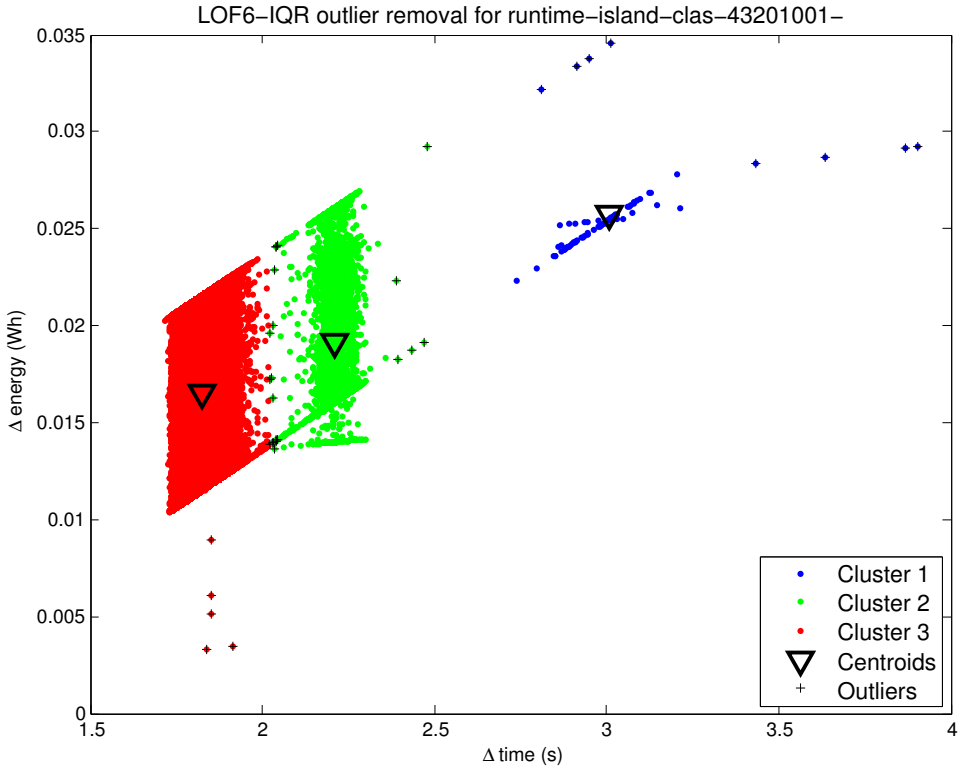


Figura 4.36: Detección de outliers (LOF e IQR) para experimento 4-320-100-1

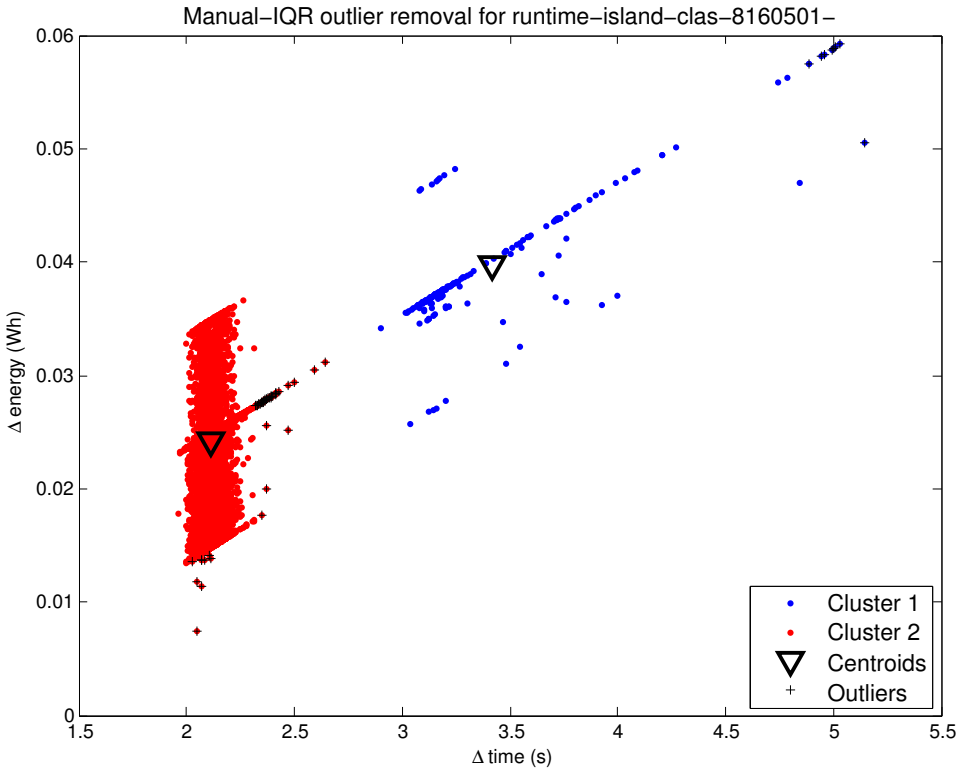


Figura 4.37: Detección de outliers (Manual e IQR) para experimento 8-160-50-1

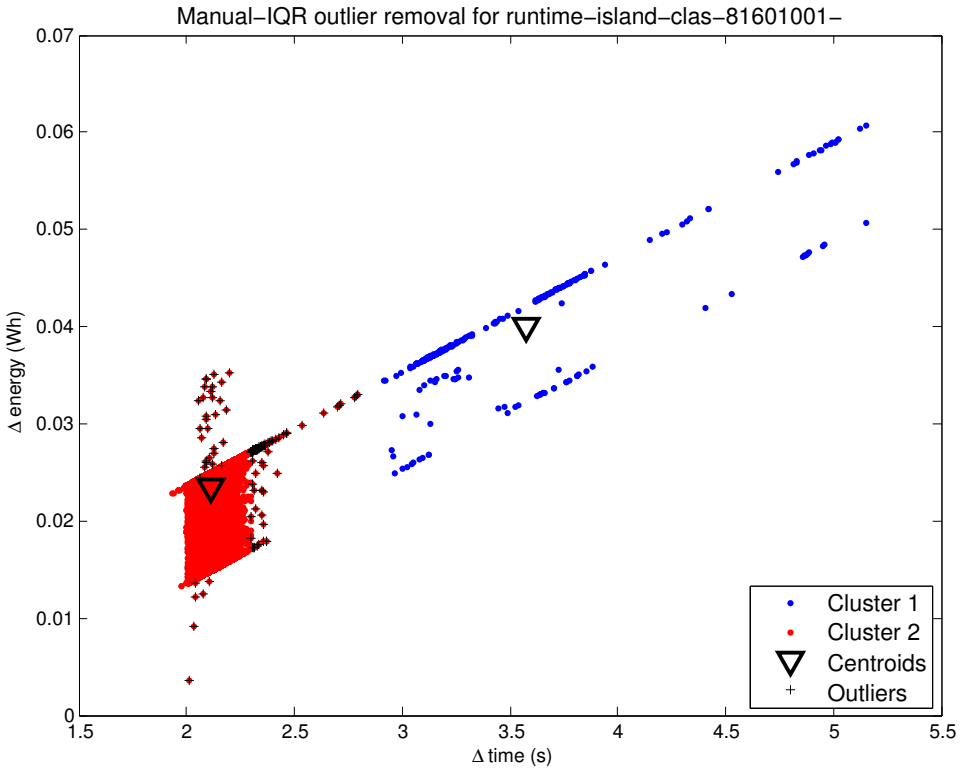


Figura 4.38: Detección de outliers (Manual e IQR) para experimento 8-160-100-1

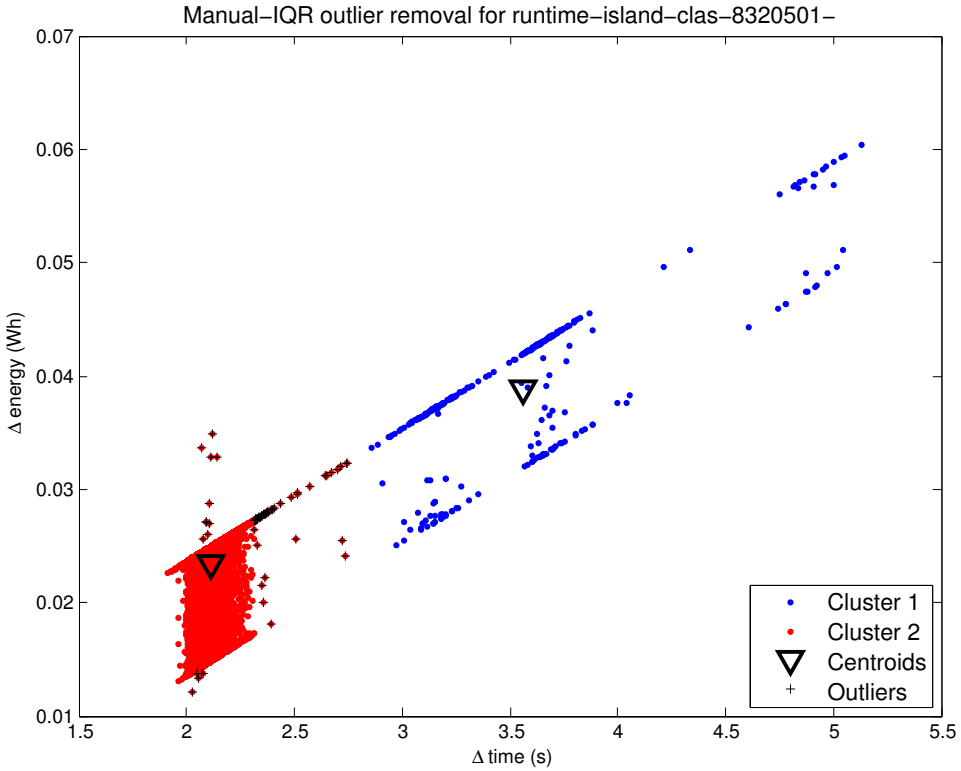


Figura 4.39: Detección de outliers (Manual e IQR) para experimento 8-320-50-1

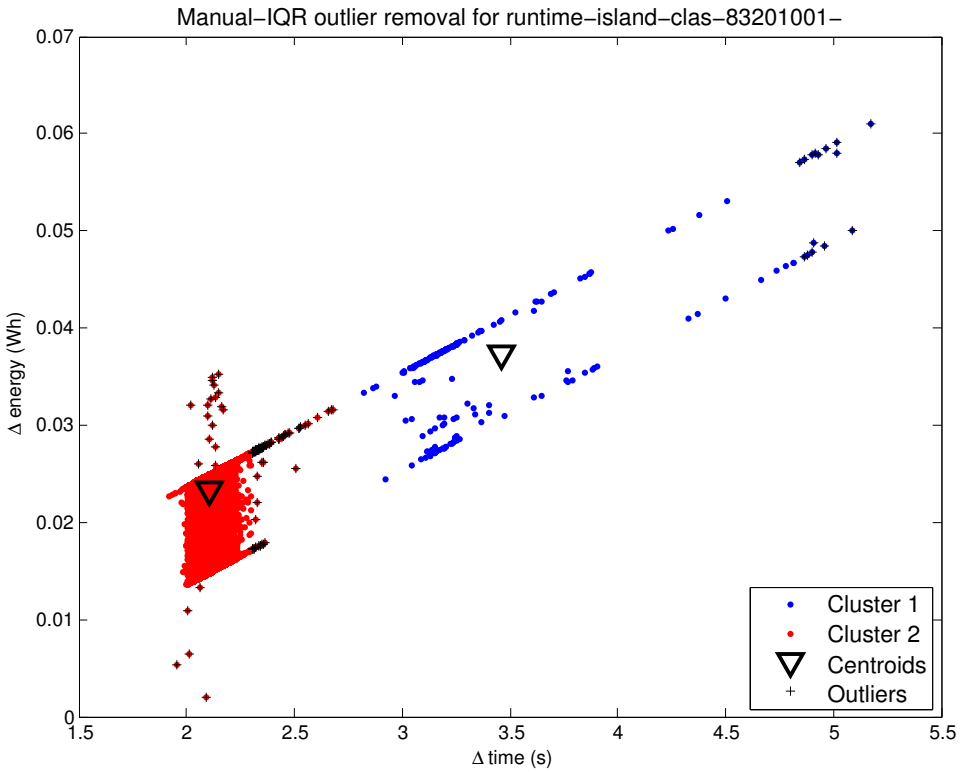


Figura 4.40: Detección de outliers (Manual e IQR) para experimento 8-320-100-1

4.4. Predicción de consumo energético

4.4.1. Construcción del modelo

Para la construcción del modelo se van a utilizar los datos de las 9 primeras repeticiones, una vez eliminadas las anomalías, de los experimentos #1, #3, #4, #5, #6, #7, #8. El experimento #2 (8-160-50-1) se ha dejado fuera de la construcción dado a la acumulación energética diferenciada en su *cluster 2* (ver Figura 4.18). Dado que para 8 núcleos apenas existe el comportamiento lineal extra superpuesto al principal, se asumen 3 partes a partir de las cuales se construyen 3 modelos lineales:

- **LeftModel**. Modelos entrenado con los datos correspondientes a la "parte izquierda" de las gráficas, es decir, a los cluster #2 de los experimentos ejecutados sobre 8 núcleos y a los cluster #3 de los ejecutados sobre 4. El modelo se representa en la Figura 4.41. Pueden diferenciarse dos nubes de datos, a la izquierda los de consumo energético para 4 núcleos y a la derecha los de ejecución en 8 núcleos. También puede apreciarse que su carácter creciente no parece corresponder con la tendencia general de los puntos. Esto es debido a que existe una mayor acumulación de puntos en la frontera superior correspondiente a los experimentos ejecutados sobre 8 núcleos. Sin embargo, como el propósito es la agregación de las predicciones, este detalle no va a afectar a una posterior predicción.
- **MiddleModel**. Modelo entrenado exclusivamente para los experimentos ejecutados sobre 4 núcleos. Se encarga de representar la parte intermedia de los mismos donde existe una superposición de dos comportamientos lineales perfectamente diferenciados a nivel visual, pero que no se pueden separar por carecer de datos referidos a este fenómeno. El modelo se representa en la Figura 4.42.
- **RightModel**. Modelo entrenado con los datos correspondientes a la "parte derecha" de las gráficas. Representa a la minoría de datos dispersa agrupada en los cluster #1 de todos los experimentos seleccionados. El modelo se representa en la Figura 4.43.

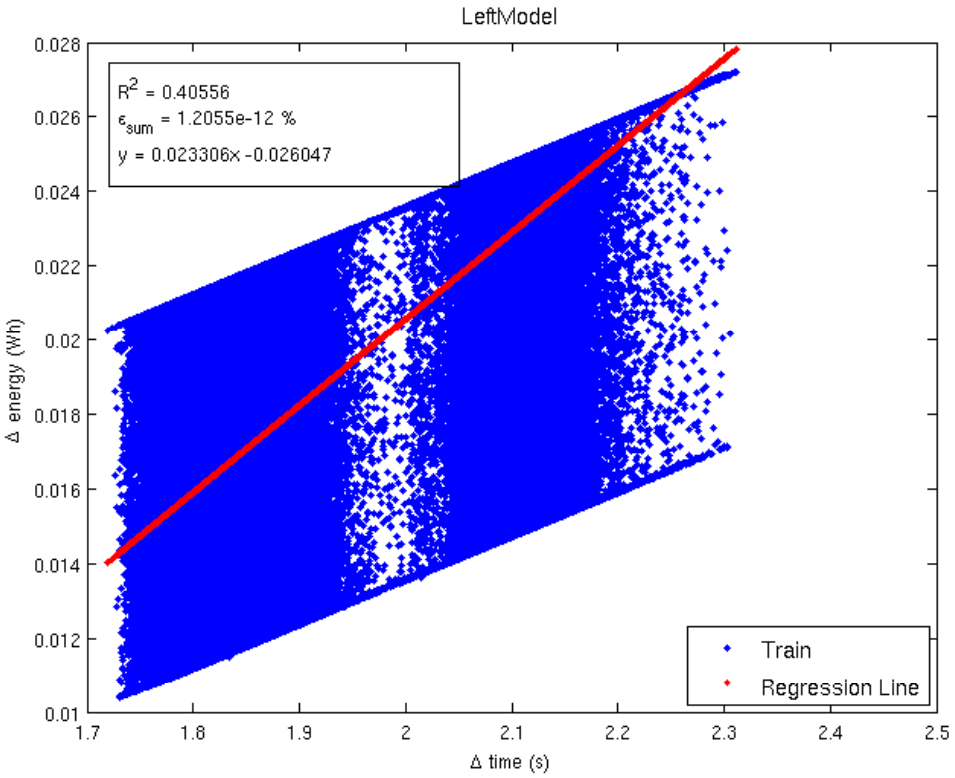


Figura 4.41: LeftModel

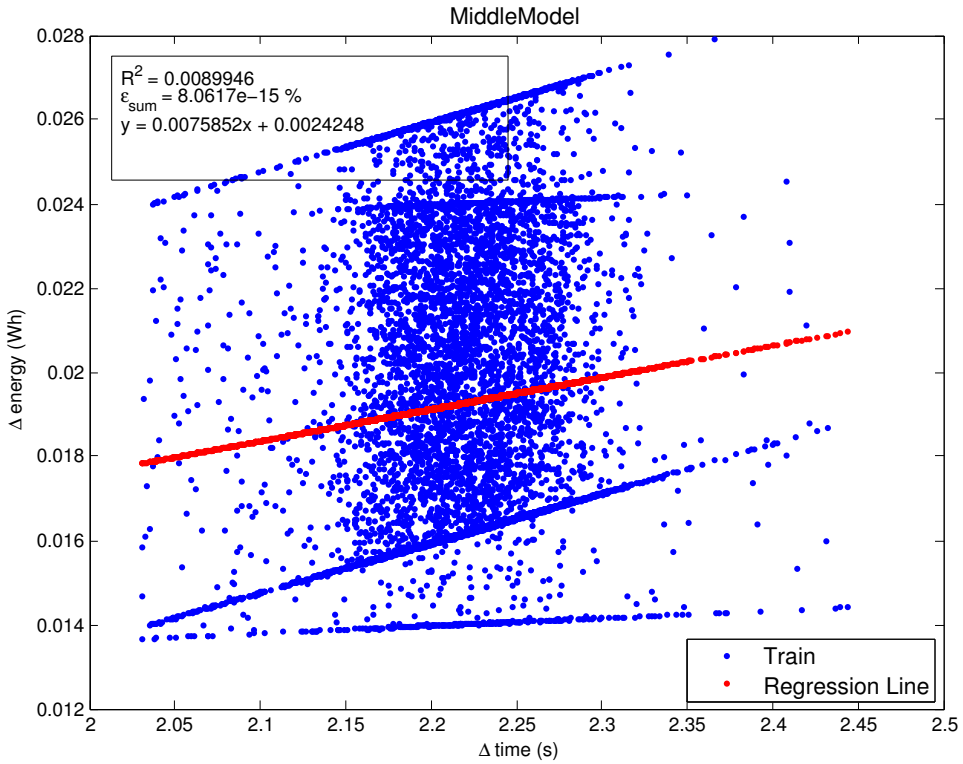


Figura 4.42: MiddleModel

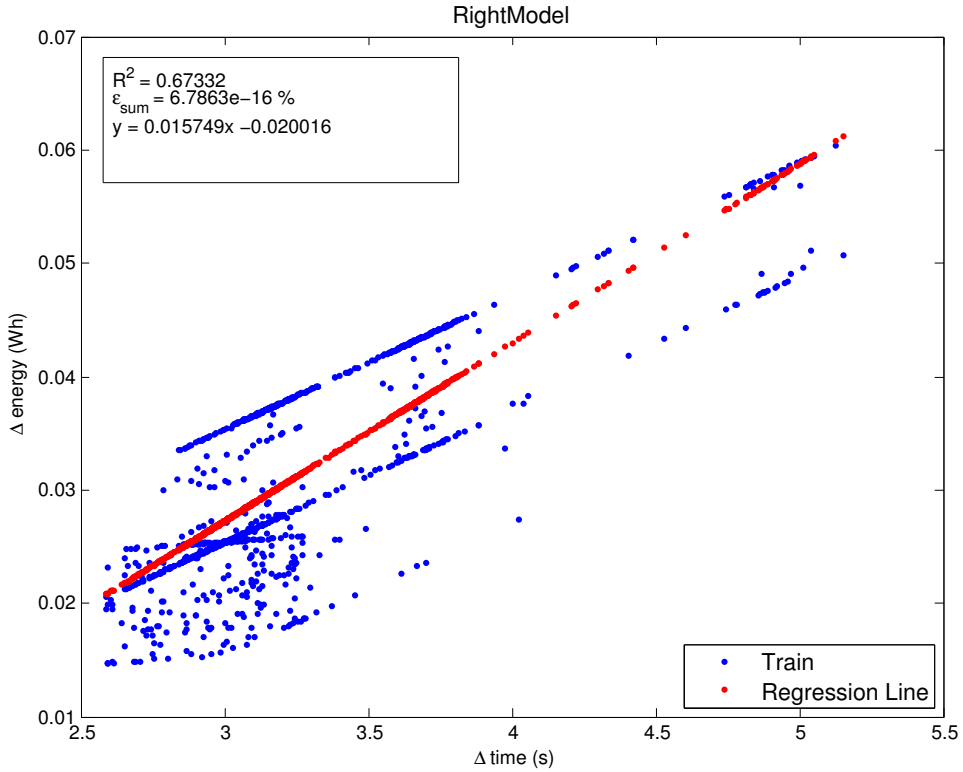


Figura 4.43: RightModel

Como puede observare en las figuras de cada modelo, el coeficiente de determinación R^2 es muy bajo. Esto es debido a que es directamente dependiente de la dispersión de los datos originales respecto a la recta de regresión. En regresión lineal R^2 se define como el cuadrado del coeficiente de correlación de Pearson:

$$R^2 = \frac{\sigma_{XY}^2}{\sigma_X^2 \sigma_Y^2} \tag{4.10}$$

Donde:

- σ_{XY} es la covarianza de (X,Y), en nuestro caso, (E,t).
- σ_X es la desviación típica de (X), en nuestro caso, (E).
- σ_Y es la desviación típica de (Y), en nuestro caso, (t).

Aunque la tendencia de que el consumo energético es directamente proporcional al tiempo empleado es claramente visible, las desviaciones típicas de ambas variables son demasiado altas como para proporcionar un valor de R^2 razonable. Especial mención a la desviación típica de la energía en el cluster 1, donde la acumulación de datos se da en las fronteras (ver Figura 4.28b), por lo que su magnitud es muy relevante y reduce mucho el coeficiente de determinación.

Sin embargo, esto no afecta a nuestro objetivo de determinar cuál va a ser la suma de todos los datos, ya que la dispersión es constante en todo momento.

4.4.2. Validación del modelo

En la Tabla 4.15 pueden verse los resultados después de haber entrenado el modelo con las repeticiones 1-9, y validar con la repetición 10 de cada experimento. Los resultados son buenos y reflejan una media de error relativo entre la predicción y el valor real de 1,1925 %

	Predicción (Wh)	Real (Wh)	Error relativo (%)
Exp. 1-10	79.46	78.33	1.44
Exp. 2-10	109.21	110.77	1.40
Exp. 3-10	152.36	154.93	1.66
Exp. 4-10	220.33	221.55	0.55
Exp. 5-10	159.28	158.11	0.74
Exp. 6-10	224.56	223.65	0.41
Exp. 7-10	301.48	306.58	1.66
Exp. 8-10	430.06	437.40	1.68

Tabla 4.15: Validación de modelos de predicción energética.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este Trabajo de Fin de Máster se ha elaborado un modelo de predicción de consumo energético para el algoritmo genético NSGA-II paralelizado en Islas orientado a la selección de características en BCI. Para ello, en primer lugar, ha sido necesaria una revisión bibliográfica de los métodos empleados habitualmente en la caracterización de consumo energético de sistemas y algoritmos. A continuación, se ha estudiado la última implementación de NSGA-II realizada por Ortega y *cols.* [1] orientada a la selección de características en BCI, y el funcionamiento de la misma con la *Parallel Computing Toolbox* de MATLAB [23] para poder realizar la medida de tiempos de cada una de sus partes a lo largo de su ejecución y sincronizarla con las medidas de consumo energético. De este caso de estudio se ha extraído lo siguiente:

- La evaluación de los individuos representa, frente al total, el 96.65 % del tiempo de ejecución, y el 97.07 % del consumo energético total.
- Teniendo en cuenta ese gran peso de la evaluación, se puede extraer que distribuir la carga de la misma entre el máximo número de núcleos disponibles. Ejecutar un algoritmo en 8 núcleos consume de media el 57.40 % del tiempo, y el 71.26 % de la energía de lo que supone lanzarlo en 4 .
- La aplicación de operadores genéticos (*GenOp*) y del reemplazo generación tras generación (*Replace*) es mucho más robusta en 4 que en 8 procesadores.
- La desviación estándar de las funciones internas de MATLAB *CreateCommJob* y *CreateTask* es prácticamente nula a nivel energético y de tiempo.
- El número de individuos sólo afecta de forma significativa en los procesos de evaluación y ordenación no dominada.
- El número de generaciones sólo afecta de forma significativa en los procesos de evaluación, ordenación no dominada y reemplazo.

Se han estudiado y comparado los datos recogidos en todos los experimentos a partir de un análisis de *clustering* con k-medias, infiriendo comportamientos diferentes para la distribución del mismo problema (población inicial, generaciones a evolucionar, una comunicación entre islas) entre 4 y 8 núcleos. Además, se han comparado distintos métodos de detección de outliers basados en IQR, k-NN, distancia a centroide y LOF, obteniendo diferentes resultados según qué agrupación de datos se estuviese analizando:

- LOF ha sido el método de detección de anomalías más exitoso para los experimentos ejecutados en 4 hilos, a pesar de ser muy costoso computacionalmente. Los *outliers* de los ejecutados en 8 núcleos han sido eliminados manualmente.

Por último, se ha elaborado un modelo de predicción de consumo energético para el algoritmo estudiado que depende exclusivamente de parámetros intrínsecos al propio algoritmo. Esto hace que la metodología desarrollada a lo largo de la memoria sea fácilmente exportable y adaptable a otras máquinas o algoritmos genéticos de los que se quiera desarrollar un modelo de predicción energética similar. Así, por todo lo expuesto con anterioridad y los resultados obtenidos, este modelo de caja-negra cumple con los objetivos

1, 2, 4, 5, 6 y 7 listados en la Subsección 2.1.1: (1) se trata de una aproximación de sistema completo en la que se contempla el comportamiento de todo el sistema como unidad y su respuesta al algoritmo, (2) tiene un error medio de validación del 1.1925 % (4) es genérico y portable por no depender de parámetros internos de la máquina sobre la que se ejecuta el algoritmo, y (5,7) el coste de generación y utilización del modelo se limita a una simple recta de regresión para cada tipo de cluster definido en 4 y 8 núcleos de ejecución. Respecto a los objetivos 3 y 6, no se puede asegurar su cumplimiento, porque el modelo permite realizar *scheduling estático*, es decir, predicciones antes de la ejecución del programa, y porque no se puede asegurar que la medición de tiempos a lo largo del programa no haya alterado el propio flujo del mismo sin ellas, respectivamente.

5.2. Trabajo futuro

Como línea de trabajo futuro y continuación directa de este TFM se plantea lo siguiente:

1. Elaborar modelo que incluya eventos internos a nivel de kernel dentro del sistema con herramientas como PERF. Esto ayudaría a tener datos de contadores internos, y además podría compararse la medida energética obtenida a través de este software con la obtenida a partir de Arduino.
2. Migrar el código a OpenCL/OpenMP, optimizando el código y evitando el máximo número posible de capas intermediarias en la ejecución del algoritmo (Java en el caso de MATLAB).
3. Estudiar el impacto de la medición de tiempos dentro de la ejecución del programa en la predicción de consumo energético con el modelo elaborado.
4. Estudiar el impacto en el consumo energético de más de una comunicación entre islas.
5. Estudar la relación entre consumo energético y bondad de la solución encontrada.

5.3. Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad y los fondos FEDER a través del proyecto TIN2015-67020-P.

Apéndice A

Instrucciones para la recogida de datos.

En este anexo se desarrollan las instrucciones para reproducir la recogida de los datos. Al final del mismo se incluye el código original para la medición de energía (python) y de tiempos (MATLAB).

A.1. Recogida de los datos

Para la recogida de los datos se ha sincronizado la medida de energía con Arduino a través del script *consumo_v2.py*, escrito en Python, con la medición de tiempos directamente realizada en MATLAB integrada en el propio runtime del programa. En la Figura A.1 puede verse un diagrama de flujo de ambos scripts funcionando a la vez. La comunicación entre ambos se realiza a través de la creación/eliminación de los archivos *.init_energy_measure* y *.stop_energy_measure* para iniciar y detener la medida de cada experimento por separado.

A.1.1. Datos de energía

En primer lugar, para poder recoger los datos de tiempos y energía asociados a los mismos ejecutamos el script en python *consumo_v2.py* para tener preparada la medición de energía una vez se lance el algoritmo en MATLAB. El código se muestra más adelante en la Página 59. Pueden destacarse los siguientes detalles en su lanzamiento:

- Al ejecutarlo en *nohup* nos ahorramos mantener las conexiones con el cluster activas.
- El script *consumo_v2.py* tiene como parámetro de entrada el número de mediciones a realizar. En el ejemplo se indican 81 ya que son el total recogido y utilizado a lo largo de la memoria (1 de medición para calibrar el consumo del SO en el cluster y 80 de todas las repeticiones realizadas).
- El operador 2 > permite recoger la salida STDERR en el archivo *nohup_consumo.log*, es decir, los errores que pudieran surgir a lo largo de la medición.

```
1 #!/bin/bash
MEDICIONES = 80
3 nohup ./consumo_v2.py $MEDICIONES 2> nohup_consumo.log &
```

El script queda en ejecución a la espera de encontrar los archivos *.init_energy_measure* y *.stop_energy_measure* para comenzar y parar de medir, respectivamente. Una vez detectados son eliminados automáticamente para evitar problemas en el flujo del programa. Estos archivos serán creados por MATLAB en cada repetición de las distintas ejecuciones del algoritmo, permitiendo que cada una de estas tenga asociada su medición de energía en su correspondiente archivo *outX.txt*, donde X es un número entero entre 0 y el número de mediciones indicado menos uno.

Figura A.1: Medida de energía y tiempos sincronizada.

A.1.2. Datos de tiempo

Una vez preparada la medición de energía podemos ejecutar los experimentos de MATLAB contenidos en el script *experiments_islands.m* en segundo plano gracias a *nohup*. El código fuente se encuentra en la Página 60; éste además depende de *scripts prepare_path.m*, donde se añaden las dependencias del algoritmo original NSGA-II, y de *init_energy_measure.m* y *stop_energy_measure.m* que crean los archivos de inicio y parada de medida de energía, respectivamente. Estos últimos se encuentran en la página Página 61

- Del mismo modo que antes, al ejecutarlo en *nohup* nos ahorramos mantener las conexiones con el cluster activas.
- MATLAB necesita como parámetro el script *experiments_islands.m* donde se detallan todos los experimentos a realizar.
- El operador & > permite recoger la salida STDOUT en el archivo *nohup_matlab.log*, es decir, todos los mensajes de salida que MATLAB vaya aportando tras la ejecución de cada comando.

```
1  !#/bin/bash
   nohup matlab -nodisplay -nosplash < experiments_islands.m &> nohup_matlab.log &
```

A.1.3. Construcción del dataset: unión de datos de tiempo y energía

Ya con los dos conjuntos de datos, es necesario unirlos para formar un único dataset mediante el script *code:energy_profile_islands* en la Página 61. No sólo los une, sino que construye un dataset diferencial que recoge incrementos de tiempo junto al consumo energético medido de ese intervalo. Se tiene en cuenta la corrección de frecuencia de muestreo inferior a 1Hz teóricamente dispuesta en Arduino.

A.2. Código fuente

consumo_v2.py

```
#!/usr/bin/python
import os.path
import zmq
import signal
import sys

BASE_FILENAME = 'out'

# Functions
def sighandler(signum, frame):
  print 'Stopping now...'
  if os.path.isfile('.init_energy_measure'):
    os.remove('.init_energy_measure')
  if os.path.isfile('.stop_energy_measure'):
    os.remove('.stop_energy_measure')
  sys.exit(1)

def main():
  signal.signal(signal.SIGINT, sighandler)

  for i in range(0, int(sys.argv[1])):
    while not os.path.isfile('.init_energy_measure'):
      continue

    os.remove('.init_energy_measure')

    # TCP connexion
    context = zmq.Context()
    socket = context.socket(zmq.SUB)
    socket.connect("tcp://localhost:5214")
    socket.setsockopt(zmq.SUBSCRIBE, "")

```

```
    with open(BASE_FILENAME + str(i) + ".txt", 'w') as f:
      while not os.path.isfile('.stop_energy_measure'):
        data = socket.recv().split(" ")
        if len(data)>0:
          #print ' '.join(data[0:])
          data = ' '.join(data[0:])
          f.write(data + '\n')

    os.remove('.stop_energy_measure')

# Program
if __name__ == "__main__":
  main()
```

code/consumo_v2.py

experiments_islands.m

```
%% experiments_islands.m
%Run every test for energy and time measuring.
run prepare_path.m

% Test #0 - Energy measurement of OS
init_energy_measure();
pause(4764);
stop_energy_measure;

% Test #1
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(160,50,1,i,4);
stop_energy_measure();
end

% Test #2
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(160,50,1,i,8);
stop_energy_measure();
end

% Test #3
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(160,100,1,i,4);
stop_energy_measure();
end

% Test #4
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(160,100,1,i,8);
stop_energy_measure();
end

% Test #5
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(320,50,1,i,4);
stop_energy_measure();
end

% Test #6
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(320,50,1,i,8);
stop_energy_measure();
end

% Test #7
for i = 1:10
init_energy_measure();
```

```
hpmoon_islands_classifiers_time(320,100,1,i,4);
stop_energy_measure();
end

% Test #8
for i = 1:10
init_energy_measure();
hpmoon_islands_classifiers_time(320,100,1,i,8);
stop_energy_measure();
end

exit
```

code/experiments_islands.m

prepare_path.m

```
%% prepare_path.m
%    Prepares path for executing hpmoon_islands_classifiers_time
addpath(genpath('~/hpmoon_CAPCO/NSGA2'))  %NSGA-II base scripts
addpath(genpath('~/hpmoon_CAPCO/somtoolbox'))    %In case somtoolbox is used
```

code/prepare_path.m

init_energy_measure.m

```
function [ ] = init_energy_measure(  )
%init_energy_measure
%   Writes empty file .init_energy_measure in current directory.
    f = fopen('.init_energy_measure', 'w');
    fprintf(f, '\0');
    fclose(f);
end
```

code/init_energy_measure.m

stop_energy_measure.m

```
function [ ] = stop_energy_measure(  )
%stop_energy_measure
%   Writes empty file .stop_energy_measure in current directory.
    f = fopen('.stop_energy_measure', 'w');
    fprintf(f, '\0');
    fclose(f);
end
```

code/stop_energy_measure.m

energy_profile_islands.m

```
function [EnergyProfileCell] = energy_profile_islands(powerLog,RuntimeCell,N)
%energy_profile_islands
%   energy_profile_islands(powerLog,RuntimeCell,N) adequates the energy
%   measurement to the runtime of the algorithm.
%   @param powerLog: energy log
%   @param RuntimeCell: time log for all workers (Cell object)
%   @param N: sample right before the energy actually starts rising its value
%over the idle state.

%% Initialize variables
RUNTIME_LENGTH = cellfun(@length, RuntimeCell);
WORKERS = length(RUNTIME_LENGTH);
COMM = RuntimeCell{1,1}(1,3);
POP = RuntimeCell{1,1}(1,4);
GEN = RuntimeCell{1,1}(1,5);

```

```
%% powerGround assignment
% In this version 'N' equals '0' due to measurement perfectly synchronised
% with MATLAB algorithms' execution.
if nargin < 3
    powerGround = 65.38; % mean measured 30 June 2017 (1.5 hours)
    N = 1;
else
    powerGround = mean(powerLog(1:N,1));
end

%% powerLog preparation
% Is assumed to be reading the compute-0-1 columns (3rd and 7th)
powerLog = powerLog(:,[3,7]);

%% Runtime preparation for all Workers

for w = 1:WORKERS
    %% diff_runtime cell-matrix construction:
    % First row is header where the algorithm's initial conditions are set.
    %   (:,1) is for deltaTime
    %   (:,2) is for deltaEnergy
    %   (:,3) is for #thread
    %   (:,4) is for #communication
    %   (:,5) is for population
    %   (:,6) is for #generation
    %   (:,7) is for step in the runtime
    runtime = []; diffRuntime = []; % Clear variables for loop
    runtime = RuntimeCell{w,1};
    diffRuntime(:,1) = diff(runtime(:,1));
    diffRuntime(:,3:7) = runtime(2:end,2:6);

    %% Energy estimation
    realTime = []; En = []; energy = []; % clear for loop in WORKERS
    En0=powerLog(N,2);
    powerLogLength = length(powerLog) - 1; % Last measurement is reserved
    timeElapsed = runtime(end,1);
    delay = (timeElapsed + 1) - powerLogLength;
    for (i=1:RUNTIME_LENGTH(w)-1)
        %A correction is needed due to imperfect measurement of energy,
  % it means, f ~= 1Hz, actually, f < 1Hz.
        realTime(i) = N + runtime(i+1,1) - delay*runtime(i+1)/timeElapsed;
        if (i==1)
            En(i)=( (powerLog(floor(realTime(i)) + 1, 2) - ...
                     powerLog(floor(realTime(i)), 2)) * ...
                     (realTime(i) - floor(realTime(i))) ) + ...
                     powerLog(floor(realTime(i)), 2);
            energy=En(i)-En0;
        else
            En(i)=( (powerLog(floor(realTime(i))+1,2) - ...
                powerLog(floor(realTime(i)),2)) * ...
                (realTime(i) - floor(realTime(i))) ) + ...
                powerLog(floor(realTime(i)),2);
            energy=En(i)-En(i-1);
        end
        diffRuntime(i,2)=energy-(powerGround*diffRuntime(i,1)/3600);
    end

    %% Create Cell with all workers
    EnergyProfileCell{w,1} = diffRuntime;
end
end
```

code/energy_profile_islands.m

Apéndice B

Instrucciones para la predicción de consumo energético.

En primer lugar necesitamos conocer cuál es la proporción de los datos de cada cluster para según qué experimento. Se muestran en la Tabla B.1:

	Cluster 1 (%)	Cluster 2 (%)	Cluster 3 (%)
4 núcleos	0.14	2.43	97.43
8 núcleos	0.22	99.78	0.00

Tabla B.1: Proporción de datos según cluster y número de núcleos de ejecución

A continuación, conociendo la proporción de la población que se va a evaluar como consecuencia de las probabilidades de que un individuo se cruce o sufra una mutación, calculamos el número de evaluaciones que se va a realizar.

$$Evaluaciones = pop \cdot gen \cdot f \tag{B.1}$$

Donde:

- *pop*: número de individuos de la población a evolucionar.
- *gen*: número de generaciones a evolucionar la población.
- f: proporción de evaluaciones finales tras aplicar probabilidades de cruce y mutación. En este caso $f = 0{,}5851$.

Ahora, decidiendo si lo vamos a ejecutar en 4 u 8 núcleos, hacemos una distribución uniforme con el tanto por ciento correspondiente a cada cluster entre sus valores máximos y mínimos (Tablas B.2 y B.3).

	Cluster 1	Cluster 2	Cluster 3
Tiempo min. (s)	1.73	2.03	2.59
Tiempo max. (s)	1.98	2.44	5.15

Tabla B.2: Tiempos máximos y mínimos según cluster para ejecución en 4 hilos.

	Cluster 1	Cluster 2
Tiempo min. (s)	1.98	2.59
Tiempo max. (s)	2.30	5.15

Tabla B.3: Tiempos máximos y mínimos según cluster para ejecución en 8 hilos.

Por último, a cada distribución uniforme se le aplica su modelo de regresión y se agregan todas las predicciones para realizar la suma total. En MATLAB el código a ejecutar sería el siguiente:

```
%@param tmin: tiempo mínimo según cluster.
%@param tmax: tiempo máximo según cluster.
%@param evaluaciones: número de evaluaciones de acuerdo al tanto por
%ciento del cluster escogido.
%@param model: modelo según cluster, por ejemplo, [0.023306, -0.026047] para cluster
  1.

x = tmin + (tmax-tmin)*rand(evaluaciones) %Vector de tiempos
y = polyval(model, x)
prediccion = sum(y)
```

Bibliografía

[1] J. Ortega, D. Kimovski, J. Q. Gan, A. Ortiz, and M. Damas, "A Parallel Island Approach to Multiobjective Feature Selection for Brain-Computer Interfaces," in *International Work-Conference on Artificial Neural Networks (IWANN): Advances in Computational Intelligence*, vol. 10305. Cham: Springer International Publishing, 2017, pp. 16–27. [Online]. Available: http://link.springer.com/10.1007/978-3-319-59153-7_2

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/996017/

[3] S. M. Rivoire, "Models and metrics for energy-efficient computer systems," Ph.D. dissertation, Stanford University, 2008. [Online]. Available: http://search.proquest.com/openview/ff08fcba635951b94fa064edf55a7b8d/1?pq-origsite=gscholar&cbl=18750&diss=y

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-level Power Analysis and Optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ser. ISCA '00. New York, NY, USA: ACM, 2000, pp. 83–94. [Online]. Available: http://doi.acm.org/10.1145/339647.339657

[5] "SimpleScalar LLC." [Online]. Available: http://www.simplescalar.com/

[6] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Paging: Faster Translations (TLBs)," in *Operating Systems: Three Easy Pieces*, 0th ed. Arpaci-Dusseau Books, May 2015.

[7] H. Shafi, P. J. Bohrer, J. Phelan, C. A. Rusu, and J. L. Peterson, "Design and validation of a performance and power simulator for powerpc systems," *IBM J. Res. Dev.*, vol. 47, no. 5-6, pp. 641–651, Sep. 2003. [Online]. Available: http://dx.doi.org/10.1147/rd.475.0641

[8] W. Bakkali, M. Tlich, P. Pagani, and T. Chonavel, "A measurement-based model of energy consumption for PLC modems," in *18th IEEE International Symposium on Power Line Communications and Its Applications*, Mar. 2014, pp. 42–46.

[9] R. Azimi, M. Stumm, and R. W. Wisniewski, "Online Performance Analysis by Statistical Sampling of Microprocessor Performance Counters," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ser. ICS '05. New York, NY, USA: ACM, 2005, pp. 101–110. [Online]. Available: http://doi.acm.org/10.1145/1088149.1088163

[10] A. Jaiantilal, Y. Jiang, and S. Mishra, "Modeling CPU energy consumption for energy efficient scheduling," in *Proceedings of the 1st Workshop on Green Computing*. ACM, 2010, pp. 10–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1925015

[11] J. I. Aliaga, M. Barreda, M. F. Dolz, A. F. Martin, R. Mayo, and E. S. Quintana-Orti, "Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems," *Cluster Computing*, vol. 17, no. 4, pp. 1335–1348, Dec. 2014. [Online]. Available: http://link.springer.com/10.1007/s10586-014-0402-z

[12] "Qué es C-state | Dell España." [Online]. Available: http://www.dell.com/support/article/es/es/esdhs1/qna41893/qu%C3%A9-es-c-state?lang=es

[13] "What exactly is a P-state? (Pt. 1) | Intel® Software." [Online]. Available: https://software.intel.com/en-us/blogs/2008/05/29/what-exactly-is-a-p-state-pt-1

[14] R. Barik, N. Farooqui, B. T. Lewis, C. Hu, and T. Shpeisman, "A black-box approach to energy-aware scheduling on integrated CPU-GPU systems." ACM Press, 2016, pp. 70–81. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2854038.2854052

[15] S. Raudys and A. Jain, "Small sample size effects in statistical pattern recognition: recommendations for practitioners," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 3, pp. 252–264, 1991. [Online]. Available: http://dataclustering.cse.msu.edu/papers/RaudysJainPAMI91.pdf

[16] D. Kimovski, J. Ortega, A. Ortiz, and R. Banos, "Feature selection in high-dimensional EEG data by parallel multi-objective optimization," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 314–322. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6968782/

[17] D. Kimovski and al., "Leveraging cooperation for parallel multi-objective feature selection in high-dimensional EEG data: Cooperative parallel MOEAs for feature selection in EEG," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 18, pp. 5476–5499, Dec. 2015. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3594

[18] D. Kimovski, J. Ortega, A. Ortiz, and R. Banos, "Parallel alternatives for evolutionary multi-objective optimization in unsupervised feature selection," *Expert Systems with Applications*, vol. 42, no. 9, pp. 4239–4252, Jun. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0957417415000846

[19] Y. Saeys, I. n. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, Oct. 2007. [Online]. Available: https://academic.oup.com/bioinformatics/article/23/19/2507/185254/A-review-of-feature-selection-techniques-in

[20] J. Handl and J. Knowles, "Feature subset selection in unsupervised learning via multiobjective optimization," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 217–238, 2006. [Online]. Available: https://www.researchgate.net/profile/Joshua_Knowles/publication/228670597_Feature_Subset_Selection_in_Unsupervised_Learning_via_Multiobjective_Optimization/links/0912f509274b1d8ea8000000.pdf

[21] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[22] J. Asensio-Cubero, J. Q. Gan, and R. Palaniappan, "Multiresolution analysis over simple graphs for brain computer interfaces," *Journal of Neural Engineering*, vol. 10, no. 4, p. 046014, Aug. 2013.

[23] "Parallel Computing Toolbox Documentation - MathWorks España." [Online]. Available: https://es.mathworks.com/help/distcomp/

[24] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[25] W. H. Swallow and F. Kianifard, "Using Robust Scale Estimates in Detecting Multiple Outliers in Linear Regression," *Biometrics*, vol. 52, no. 2, pp. 545–556, 1996. [Online]. Available: http://www.jstor.org/stable/2532894

[26] B. W. Silverman and M. C. Jones, "E. Fix and J.L. Hodges (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951)," *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 233–238, 1989. [Online]. Available: http://www.jstor.org/stable/1403796

[27] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-based Local Outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 93–104. [Online]. Available: http://doi.acm.org/10.1145/342009.335388

[28] G. Erdogan, "OutlierDetectionToolbox: Outlier Detection Toolbox for MATLAB," Apr. 2017, original-date: 2015-05-10T13:53:19Z. [Online]. Available: https://github.com/gokererdogan/OutlierDetectionToolbox