Algoritmos basados en nubes de partículas para la determinación de parámetros en un simulador de motores de combustión interna

Tesis de Proyecto de Grado

Marcelo Bazán
Esteban Rodríguez

Tutores: Ing. Martín Pedemonte
Dr. Ing. Pedro Curto

Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Julio 2016

Algoritmos basados en nubes de partículas para la determinación de parámetros en un simulador de motores de combustión interna

Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Julio 2016

A nuestra familia y amigos

Agradecimientos

Agradecemos a nuestro tutor de proyecto de grado Martín Pedemonte por depositar su confianza en nosotros para la realización de este trabajo además de su orientación y colaboración a lo largo del mismo.

También agradecer a Pedro Curto por proporcionarnos el código fuente del simulador del motor de combustión interna y por aportar la información necesaria para la documentación del mismo.

A nuestras familias y amigos por el apoyo y afecto brindado en todo momento sin el cual no hubiera sido posible la realización de este trabajo.

Agradezco especialmente a mi madre y a mi abuela por el apoyo y el cariño de siempre. Y a todos quienes confiaron en mi y me dieron palabras de aliento y motivación para seguir adelante con este trabajo.

Esteban

Agradecer especialmente a Washington por nunca perder la fe en mí y recordarme constantemente que este momento iba a llegar algún día.

Finalmente, agradecer a Mercedes por el empuje final brindado, por la paciencia, tolerancia y cariño recibido, siempre dando una visión positiva de los hechos.

Marcelo

Resumen

La resolución de problemas de optimización complejos representa uno de los mayores retos de la informática desde sus inicios. Periódicamente se proponen algoritmos de optimización para enfrentar a estos problemas, procurando brindar resultados de calidad con un bajo costo computacional. Dentro de estos algoritmos o técnicas de resolución, las metaheurísticas se destacan por su capacidad de aplicación a problemas generales, permitiendo comúnmente la obtención de soluciones de buena calidad.

Existe una importante familia de metaheurísticas denominada inteligencia de enjambre (Swarm Intelligence) que se caracteriza por utilizar el comportamiento colectivo de agentes simples en la obtención de información y aprendizaje para la resolución de los problemas de optimización. Dentro de esta familia se destaca la técnica conocida como optimización por nube de partículas (Particle Swarm Optimization), inspirada en el comportamiento social de algunas especies de la naturaleza como las aves y los peces. Actualmente es una de las técnicas más estudiadas y en mayor desarrollo dentro de Swarm Intelligence.

Uno de los objetivos de este trabajo es el estudio en profundidad de la metaheurística Particle Swarm Optimización. En particular, se analiza su origen, el detalle de su funcionamiento y las diferentes variantes existentes. El otro objetivo de este trabajo consiste en estudiar la optimización del rendimiento de un motor de combustión interna desarrollado por el Instituto de Ingeniería Mecánica y Producción Industrial mediante el algoritmo Particle Swarm Optimization. La evaluación experimental toma como punto de comparación una configuración del motor sugerida por un experto, así como también los resultados obtenidos mediante la utilización de una búsqueda aleatoria.

Índice

Agradecimientos VII

Resumen IX

1. Introducción 1

2. Metaheurísticas y Swarm Intelligence 5
2.1. Estructura del capítulo 5
2.2. Conceptos previos 6
2.3. Metaheurísticas 7
2.3.1. Metaheurísticas basadas en trayectoria 11
2.3.2. Metaheurísticas basadas en poblaciones 12
2.4. Swarm Intelligence 12
2.4.1. Ant Colony Optimization 14

3. Particle Swarm Optimization 19
3.1. Estructura del capítulo 19
3.2. Antecedentes de PSO 20
3.3. Idea detrás de PSO 21
3.4. Vecindario de una partícula 22
3.5. Topología de la nube de partículas 23
3.6. Clases de PSO 24
3.7. Áreas de aplicación de PSO 26
3.8. Algoritmo básico de PSO 27
3.8.1. Inicialización de la velocidad 30
3.8.2. Restricción del cálculo de la velocidad 31
3.9. Modos de actualización 33
3.10. Variantes del algoritmo PSO 34
3.10.1. Peso de inercia 35
3.10.2. PSO con factor de constricción 36
3.10.3. PSO basada en tribus (tribes) 37
3.10.4. ARPSO 37

3.10.5. Cooperative PSO (CPSO) 39

4. Motores de combustión y su resolución mediante PSO 43
4.1. Estructura del capítulo 43
4.2. Reducción de la variabilidad cíclica en motores de combustión 44
4.2.1. Contexto del problema 44
4.2.2. Planteo del problema 46
4.3. Enfoque para la resolución del problema mediante PSO . . . 47
4.4. Detalles de implementación de PSO 49
4.4.1. Arquitectura de la implementación 50
4.4.2. Módulo de Configuración 51
4.4.3. Módulo de Fitness 54
4.4.4. Módulo de Gráficos 54
4.4.5. Módulo del Modelo numérico del motor 54
4.4.6. Módulo Generador de Números Aleatorios 56
4.4.7. Módulo de Tipos 56
4.4.8. Módulo de Topologías 57
4.4.9. Módulo de PSO 58

5. Calibración de Parámetros 61
5.1. Estructura del capítulo 61
5.2. Parámetros de los algoritmos 62
5.3. Detalle de la calibración de parámetros 68
5.4. Resultados de la calibración 71

6. Evaluación de los algoritmos propuestos 75
6.1. Estructura del capítulo 75
6.2. Detalle de la evaluación 76
6.2.1. Primer experimento 76
6.2.2. Segundo experimento 87
6.2.3. Tercer experimento 89
6.3. Análisis físico de los resultados 93

7. Conclusiones y trabajo futuro 95
7.1. Estructura del capítulo 95
7.2. Conclusiones 95
7.3. Trabajo futuro 98

A. Resultados de la calibración de parámetros y evaluación 101
A.1. Resultados de la calibración de parámetros 101
A.2. Resultados de la etapa de evaluación 104
A.2.1. Resultados del primer experimento 104

A.2.2. Resultados del segundo experimento 104
A.2.3. Resultados del tercer experimento 104

Bibliografía **115**

Lista de acrónimos **121**

Índice de figuras

2.1. Banco de peces 13
2.2. Inspiración de una colonia de hormigas buscando un camino óptimo entre el hormiguero y la comida 15

3.1. Topologías de la nube de partículas 24
3.2. Gráfico de actualización de la posición de una partícula. . . . 30
3.3. Gráfico de trayectoria de una partícula sin restricción de velocidad. 32

4.1. Descripción básica de los elementos del motor (imagen extraída de [34]) 45
4.2. Interacción del algoritmo PSO con el modelo numérico del motor. 48
4.3. Diagrama de arquitectura. 51

6.1. Comparación de mejora en fitness promedio para PSO con Peso de Inercia (5 a 25 ciclos) 83
6.2. Comparación de mejora en fitness promedio para PSO con Factor de Constricción (5 a 25 ciclos) 84
6.3. Comparación de mejora en los mejores fitness para PSO con Peso de Inercia (5 a 25 ciclos) 84
6.4. Comparación de mejora en los mejores fitness para PSO con Factor de Constricción (5 a 25 ciclos) 85
6.5. Evolución de fitness 85
6.6. Evolución de fitness - Intervalo de patrón de mejora 86
6.7. Promedios de fitness en función de la instancia 89
6.8. Mejores fitness en función de la instancia 90
6.9. Mejoras de PSO (100 vs 200 iteraciones) 91

Índice de Tablas

3.1. Modo Sincrónico vs Modo Asincrónico 35

4.1. Dominios de las variables de decisión. 47
4.2. Valores de referencia del modelo numérico del motor. 48
4.3. Parámetros de configuración y sus valores asociados 51

5.1. Funciones de benchmark para el estudio del tamaño de la población 63
5.2. Valores seleccionados de parámetros que no se calibrarán . . . 66
5.3. Valores seleccionados de parámetros a calibrar 67
5.4. Configuraciones - Peso de Inercia 69
5.5. Configuraciones - Factor de Constricción 69
5.6. Resultados Calibración - Peso de Inercia 73
5.7. Resultados Calibración - Factor de Constricción 74

6.1. Valores de referencia para la evaluación 76
6.2. Configuración de PSO con Peso de Inercia 77
6.3. Configuración de PSO con Factor de Constricción 77
6.4. Promedios de los resultados para PSO con Peso de Inercia (5 a 25 ciclos) 80
6.5. Promedios de los resultados para PSO con Factor de Constricción (5 a 25 ciclos) 80
6.6. Mejores valores para PSO con Peso de Inercia (5 a 25 ciclos) . 81
6.7. Mejores valores para PSO con Factor de Constricción (5 a 25 ciclos) 81
6.8. Fitness promedio, mejor fitness y tiempo promedio para 25 y 50 ciclos. 88
6.9. Mejoras contra valores de experto y Random Search para 25 y 50 ciclos. 88
6.10. Fitness promedio, mejor fitness y tiempo promedio para 100 y 200 iteraciones. 92
6.11. Mejoras de fitness promedio para 100 vs 200 iteraciones. . . . 92

6.12. Mejoras contra valores de experto y Random Search para 100 y 200 iteraciones. 92
6.13. Valores de los parámetros para la mejor solución de las ocho ejecuciones a partir de las mejores soluciones encontradas en cada ejecución. 93

A.1. Resultados detallados de la calibración - Peso de Inercia . . . 102
A.2. Resultados detallados de la calibración - Factor de Constricción 103
A.3. Resultados de la evaluación para PSO con Peso de Inercia (5 a 25 ciclos). 105
A.4. Resultados de la evaluación para PSO con Factor de Constricción (5 a 25 ciclos). 106
A.5. Resultados de la evaluación para Random Search (5 a 25 ciclos). 107
A.6. Resultados de la evolución de fitness (1 a 50 iteraciones). . . . 108
A.7. Resultados de la evolución de fitness (51 a 100 iteraciones). . 109
A.8. Resultados de la evaluación para 100 iteraciones y 25 ciclos. . 110
A.9. Resultados de la evaluación para 100 iteraciones y 50 ciclos. . 111
A.10. Resultados de la evaluación para 25 ciclos y 100 iteraciones. . 112
A.11. Resultados de la evaluación para 25 ciclos y 200 iteraciones. . 113

Capítulo 1

Introducción

Un camino de mil millas comienza con un paso.

Benjamin Franklin

Resolver un problema de optimización consiste en maximizar o minimizar una cierta función objetivo dado un conjunto de restricciones, es decir, encontrar la solución que produzca el mejor valor funcional. Los problemas de optimización se suelen categorizar según el dominio sobre el que las variables de decisión toman sus valores. En base a esto, se distinguen dos categorías, las que utilizan un dominio discreto y las que utilizan un dominio continuo. A la primera categoría se le suele llamar problemas de optimización discreta mientras que a la segunda se le llama problemas de optimización continua.

Resolver problemas de optimización continua suele ser en general más sencillo que resolver problemas de optimización discreta, ya que en el primer tipo de problemas se suele contar con métodos exactos, como el método del gradiente, que permiten explotar información de la función objetivo.

Es importante destacar que no siempre se conoce la función objetivo a la hora de resolver un problema de optimización. Por tal motivo, resulta impracticable aplicar métodos como los antes mencionados. Este tipo de problemas se los conoce como problemas de optimización de caja negra. Esto se debe a que la función objetivo puede ser vista como una caja negra a la cual se le suministran entradas (valores de las variables de decisión) y la cual produce una salida (valor funcional correspondiente) sin saber realmente cómo este fue producido.

A la hora de resolver este tipo de problemas, las metaheurísticas han demostrado ser una poderosa herramienta que logra obtener soluciones cercanas al óptimo en tiempos de cómputo razonables. Dentro de este universo, los algoritmos basados en nubes de partículas (Particle Swarm Optimization, PSO) son una metaheurística de la familia Swarm Intelligence inspirada en

la trayectoria producida por el movimiento que realizan las bandadas de pájaros y/o los bancos de peces. La idea en la que está basada dicha técnica es recorrer el espacio de búsqueda con una población de soluciones candidatas, imitando el movimiento de estos animales, con el objetivo de encontrar la solución óptima.

A partir del contexto antes mencionado, en este proyecto se plantea la aplicación de la metaheurística Particle Swarm Optimization a la determinación de parámetros para un simulador de un motor de combustión interna implementado por docentes del Instituto de Ingeniería Mecánica y Producción Industrial (IIMPI). La idea es encontrar, a partir de la utilización de PSO, la combinación de parámetros que produzcan el mejor desempeño del motor de combustión interna tomando al simulador como una caja negra. Por lo tanto, este proyecto involucra la implementación de dicha metaheurística para la resolución del problema planteado, así como también proveer la integración con el simulador a fin de poder evaluar las diferentes soluciones.

Los objetivos del proyecto son: realizar un relevamiento del estado del arte de PSO, identificando sus variantes más destacadas; estudiar el problema de la determinación de parámetros para el simulador del motor de combustión interna desarrollado por el IIMPI; y diseñar un algoritmo PSO para la resolución de dicho problema.

El documento se estructura de la siguiente forma.

En el Capítulo 2, se presenta la definición de los conceptos previos necesarios para la comprensión del documento, dentro de los cuales se encuentran el de método exacto y heurística, entre otros. También se aborda el concepto de metaheurísticas, brindando su definición junto a una visión general de los diferentes tipos existentes. Además se introduce el concepto de Swarm Intelligence, importante familia de metaheurísticas entre las que se destacan Particle Swarm Optimization y Ant Colony Optimization.

Por su parte, el Capítulo 3 realiza la introducción a la técnica de Particle Swarm Optimization. Posteriormente, se brindan detalles sobre sus antecedentes, la descripción de la idea general de la técnica, la explicación del concepto de vecindario, las topologías potenciales con que trabaja, las diferentes clases existentes del algoritmo y los posibles campos de aplicación.

El Capítulo 4 comienza por la introducción al problema de reducción de la variabilidad cíclica en motores de combustión interna, dando pie al contexto y planteo del mismo. Luego, se presenta el enfoque para la resolución del problema mediante PSO. Finalmente, se brindan los detalles de la implementación, mencionando la arquitectura propuesta, así como también los diferentes módulos que la componen.

En el Capítulo 5, se detallan los parámetros de las variantes de PSO considerados en la etapa de calibración de parámetros, posteriormente se presentan los parámetros que serán excluidos del proceso de calibración y a

continuación aquellos que están incluidos en dicho proceso. Luego, se describe en detalle los pasos que conforman el mencionado proceso. Finalmente, se presentan los resultados obtenidos en la ejecución de la calibración dando lugar a los valores de parámetros a ser utilizados en la etapa de evaluación.

Por su parte, el Capítulo 6 describe brevemente los experimentos que serán realizados, mostrando el detalle de los datos de configuración y ambiente que estos tienen en común. Posteriormente, para cada experimento, se incluye el detalle del procedimiento a ser realizado, los resultados obtenidos luego de su ejecución, observaciones y conclusiones pertinentes.

En el Capítulo 7, se proveen las conclusiones resultantes de la finalización del proyecto. Además, se intenta contextualizar, describiendo los objetivos y los resultados obtenidos asociados a cada uno de los mismos. Posteriormente se enumeran algunos puntos pendientes y trabajos a futuro que este proyecto motiva a realizar.

Finalmente, el Apéndice A brinda el detalle de los resultados obtenidos en la etapa de calibración de parámetros así como también los de la etapa de evaluación.

Capítulo 2

Metaheurísticas y Swarm Intelligence

Swarm Itelligence: La inteligencia colectiva emergente de grupos de agentes simples.

Eric Bonabeau

El objetivo de este capítulo es sentar las bases para poder explicar el funcionamiento del algoritmo *Particle Swarm Optimization*. Aquí se presentan los conceptos fundamentales que permiten su comprensión. En primer lugar se introducen una serie de conceptos previos necesarios para abordar las ideas detrás de *Metaheurísticas* y *Swarm Intelligence*. Posteriormente se introduce el concepto de *Metaheurísticas*, describiendo las principales características de este tipo de técnicas. Finalmente se profundiza sobre el concepto de *Swarm Intelligence*, el cual conforma una gran familia de metaheurísticas entre las que se encuentra *Particle Swarm Optimization*.

2.1. Estructura del capítulo

A continuación se presenta la definición de los conceptos previos necesarios para la comprensión del documento, dentro de los cuales se encuentran el de *Método exacto* y *Heurística*. La siguiente sección aborda el concepto de *Metaheurísticas*, brindando su definición junto a una visión general de los diferentes tipos existentes. Se destina además una sección al concepto de *Swarm Intelligence*, importante familia de metaheurísticas entre las que se destacan *Particle Swarm Optimization* (a la cual es destinado todo el capítulo 3) y *Ant Colony Optimization* (la que se describe en una subsección aparte).

2.2. Conceptos previos

En esta sección se presentarán los conceptos básicos previos que conforman los pilares necesarios para la introducción de la técnica con la que se trabajará en este proyecto.

Resolver un problema de optimización consiste en encontrar la mejor solución dentro de un conjunto de posibles soluciones candidatas (o factibles). Esta idea es formalizada mediante la siguiente definición:

Definición 2.1. *Una instancia de un problema de optimización puede ser representada como una pareja (X,f) donde X representa el espacio de búsqueda (o conjunto de soluciones factibles) y f una función denominada función objetivo, definida como:*

$$f : X \to \mathbb{R}. \tag{2.1}$$

Por lo tanto, resolver una instancia de un problema de optimización consiste en encontrar una solución $x^* \in X$ tal que:

$$f(x^*) \leq f(x), \forall x \in X. \tag{2.2}$$

A x^* se lo denomina comúnmente solución óptima global (o simplemente solución óptima[1]) para una instancia de problema dada.

Si bien la Definición 2.1 se describe en términos de minimización de la función objetivo, la misma se puede formular sin pérdida de generalidad aplicando maximización. La analogía de ambas definiciones se ve reflejada mediante la siguiente equivalencia:

$$max\{f(x)/x \in X\} \equiv min\{-f(x)/x \in X\} \tag{2.3}$$

Es importante destacar que al conjunto de variables que componen x se las denomina *variables de decisión.*

Definición 2.2. *Un método exacto es una técnica que permite encontrar la solución a un problema de optimización garantizando la optimalidad de la misma.*

La optimalidad de la solución es garantizada debido a que estos métodos se basan comúnmente en búsquedas matemáticas o en la explotación de propiedades matemáticas del problema de optimización[2], o en búsquedas exhaustivas en el espacio de búsqueda hasta encontrar el óptimo.

Estos métodos suelen ser utilizados para la resolución de problemas de clase de complejidad P, donde el tiempo computacional requerido crece de

[1] Cuando no ocurren ambigüedades respecto a la existencia de óptimos locales.

[2] Encontrar máximos o mínimos mediante la resolución analítica de la función objetivo.

forma polinomial [48]. Sin embargo, en problemas donde el tiempo de resolución crece con un orden superior al polinomial (por ejemplo, problemas de clase de complejidad *NP-Difícil*), estas técnicas dejan de ser aplicables dado que no pueden ser computadas en tiempos razonables. Otro detalle relevante al respecto que determina la factibilidad o no del uso de estas técnicas refiere al espacio de memoria requerido para su ejecución. Por lo tanto, es necesario contar con otras alternativas para la resolución de problemas con este nivel de complejidad. Es aquí donde entran en juego los métodos aproximados, que se concentran en encontrar soluciones aceptables[3] en tiempos de computo razonables. Las heurísticas forman parte de esta familia.

Definición 2.3. *El término "Heurística" deriva del griego "euriskein" que significa "encontrar" o "descubrir". Son técnicas que se basan en procedimientos conceptualmente simples, capaces de encontrar soluciones de buena calidad (no necesariamente óptimas) a problemas complejos de un modo sencillo y eficiente. Estos procedimientos usualmente están basados en el sentido común o en la emulación de fenómenos bien conocidos.*

Dado que las heurísticas se construyen en base al conocimiento específico de los problemas que pretenden abordar, estas son difíciles de utilizar en contextos de naturaleza diferente. Esta debilidad dio pie al surgimiento del concepto *Metaheurística*, siendo estas estrategias de mayor nivel de generalidad (abstracción) que serán detalladas a continuación.

2.3. Metaheurísticas

Antes de definir el concepto de metaheurística es necesario presentar un par de conceptos previos utilizados por este tipo de técnicas durante el proceso de búsqueda: *exploración* y *explotación*.

El término *exploración* refiere a inspeccionar diferentes áreas del espacio de búsqueda. Este término está fuertemente ligado al de *diversificación*, el cual refiere al nivel de diversidad que presentan las soluciones factibles que explora el algoritmo.

Por otro lado, el término *explotación* se enfoca en examinar con mayor intensidad un área especifica del espacio de búsqueda. Análogamente, a este término también se lo asocia con el término *intensificación*, el cual refiere a incrementar la intensidad de búsqueda sobre un área específica.

Los términos *diversificación* e *intensificación* provienen de la técnica *Tabu Search* [18]. Es importante destacar que si bien los términos *exploración* y *explotación* se suelen utilizar en lugar de estos (por ejemplo, en el campo de *Evolutionary Computation*), formalmente no significan lo mismo. De cualquier manera, a lo largo del presente documento haremos uso indistinto de los términos antes mencionados.

[3] Soluciones óptimas o cercanas al óptimo.

En contraposición a los métodos exactos, y con fuerte base en las técnicas heurísticas, las metaheurísticas son técnicas aproximadas que permiten abordar apropiadamente instancias de problemas de optimización de gran tamaño, obteniendo soluciones satisfactorias en tiempos razonables [8]. Esta ventaja respecto a los métodos tradicionales permitió que cobraran gran popularidad en los últimos 20 años. Las metaheurísticas combinan métodos heurísticos básicos en frameworks de alto nivel con el objetivo de brindar una exploración y explotación eficiente del espacio de búsqueda del problema atacado.

No existe actualmente una definición comúnmente aceptada del término metaheurística, sin embargo varios autores han realizado diversas propuestas para su definición. A continuación brindamos algunas de estas definiciones tal y como fueron realizadas.

De acuerdo a Osman y Laporte [38], una metaheurística se define formalmente como un proceso iterativo de generación que guía una heurística subordinada, combinando de manera inteligente diferentes conceptos para explorar y explotar el espacio de búsqueda, aprendiendo estrategias para estructurar la información con el objetivo de encontrar a corto plazo soluciones casi-óptimas de manera eficiente.

Otra definición propuesta por Osman junto a Voss y Roucairol [59], afirma que una metaheurística es un proceso iterativo maestro que guía y modifica las operaciones de heurísticas subordinadas para producir de manera eficiente soluciones de alta calidad. Puede manejar una única solución (completa o parcial) simple o un conjunto de soluciones en cada iteración. Las heurísticas subordinadas pueden ser procedimientos de alto (o bajo) nivel, búsquedas locales simples, o tan solo un método de construcción.

Una tercera definición posible propuesta por Blum y Manfrin [7] establece que una metaheurística es un conjunto de conceptos que pueden ser usados para definir métodos heurísticos que pueden ser aplicados para un amplio conjunto de diferentes problemas. En otras palabras, una metaheurística puede ser vista como un framework algorítmico general, el cual puede ser aplicado a diferentes problemas de optimización con relativamente pocas modificaciones para adaptarlo al problema específico.

De las definiciones anteriores se puede realizar el siguiente compilado de propiedades principales que formarían parte de la definición de metaheurística:

- Las metaheurísticas son estrategias que "guían" el proceso de búsqueda.
- Su principal propósito consiste en explorar de un modo eficiente el espacio de búsqueda para lograr encontrar soluciones cercanas al óptimo o eventualmente óptimas.
- Sus algoritmos varían desde simples procedimientos de búsqueda local hasta complejos métodos de aprendizaje automático.
- Son algoritmos aproximados y usualmente no determinísticos.

- Pueden incorporar mecanismos para evitar quedar atrapados en áreas confinadas del espacio de búsqueda.
- Los conceptos básicos de metaheurísticas permiten una descripción de nivel abstracto.
- Las metaheurísticas no fueron concebidas para un problema específico.
- Pueden hacer uso de conocimiento específico del dominio utilizando heurísticas guiadas por estrategias de alto nivel.
- Las metaheurísticas más avanzadas de hoy usan experiencia de búsqueda (incorporado algún mecanismo de memoria) para guiar el proceso de búsqueda.
- Son procesos estocásticos que no garantizan la obtención de soluciones óptimas para el problema abordado.

Estos aspectos pueden integrarse en la siguiente definición que pretende resumir los conceptos recabados anteriormente.

Definición 2.4. *Una metaheurística es una estrategia de búsqueda de alto nivel de abstracción que permite abordar de manera general distintas clases de problemas de optimización utilizando un conjunto de técnicas heurísticas subordinadas para la resolución de subproblemas específicos.*

Existen diferentes maneras de clasificar las metaheurísticas. Cada una de estas clasificaciones es determinada por alguna característica particular de este tipo de técnicas.

Son estas características las que nos permiten diferenciar dichas metaheurísticas desde algún punto de vista asociado. A continuación se presentan algunas de las clasificaciones más relevantes [8, 55]:

- **Inspiradas en la naturaleza versus no inspiradas en la naturaleza:** Esta clasificación se basa en el origen del algoritmo. Es decir, existen algoritmos inspirados en procesos naturales tales como GA (Genetic Algorithms), ACO (Ant Colony Optimization) y PSO (Particle Swarm Optimization), entre otros. Por otro lado, existen algoritmos cuyo origen no está inspirado en la naturaleza como lo son BLS (Basic Local Search), TS (Tabu Search), GRASP (Greedy Randomized Adaptative Search Procedure), etc. Es importante hacer notar que esta clasificación ha perdido peso en la actualidad ya que se han propuesto gran cantidad de algoritmos híbridos que bien podrían pertenecer a ambas categorías.
- **Con uso de memoria versus sin uso de memoria:** Una de las más importantes características para clasificar metaheurísticas es el uso de memoria. Esto refiere a la utilización o no de la historia de la búsqueda por parte del algoritmo a la hora de determinar la próxima acción en el proceso de búsqueda. Entonces, por un lado están los algoritmos

que no utilizan memoria, los cuales pueden ser vistos como procesos Markovianos en donde la próxima acción es solamente determinada por el estado de la acción previa. En contraste están los algoritmos que sí utilizan memoria. Aquí es importante mencionar que existen dos tipos de memoria: a corto y a largo plazo. La primera se preocupa de tener accesible la información más reciente del proceso de búsqueda mientras que la segunda se encarga de mantener una acumulación más permanente de dicha información.

- **Basadas en población versus basadas en una única solución:** Otra característica utilizada para clasificar metaheurísticas es el número de soluciones manipuladas simultáneamente durante el proceso de búsqueda. Esto es, por un lado tenemos los algoritmos basados en una única solución, los cuales se denominan *metaheurísticas basadas en trayectoria* debido a la trayectoria que describe la solución a través del espacio de búsqueda. Esta clase de metaheurísticas está orientada a la explotación, es decir, permiten una mejor *intensificación* en regiones más promisorias del espacio de búsqueda. En contraposición se tienen las *metaheurísticas basadas en población*. Estas recorren el espacio de búsqueda utilizando un conjunto de soluciones denominado población. Claramente esta clase de metaheurísticas está orientada a la exploración permitiendo así una mejor *diversificación* en el espacio de búsqueda.
- **Función objetivo estática versus función objetivo dinámica:** El modo en que se utiliza la función objetivo por parte de las metaheurísticas constituye otra posible clasificación. Es decir, existen algoritmos que mantienen fija la función objetivo definida inicialmente por el problema de optimización durante todo el proceso de búsqueda mientras otros algoritmos modifican dicha función. El fin de dicha modificación radica en evitar quedar atrapado en óptimos locales alterando así la superficie del problema de optimización. Esta alteración de la función objetivo (superficie) se lleva a cabo incorporando a la misma información de interés recolectada durante el proceso de búsqueda.
- **Iterativas versus greedy:** La forma en que se manipula y/o construye una solución (o población de soluciones) determina otra característica en la cual las metaheurísticas se pueden diferenciar. Los *algoritmos iterativos* siempre trabajan con una solución (o población de soluciones) completa y manipulan (transforman) la misma aplicando las reglas impuestas por el algoritmo en cuestión a través del proceso de búsqueda. Los *algoritmos greedy* en contraste parten de una solución inicial vacía. Luego, en cada paso un nuevo elemento[4] es seleccionado (utilizando alguna heurística local apropiada) para formar parte de la solución. El proceso se repite hasta contar con una solución completa

[4]Una solución puede ser vista como un conjunto de elementos que la componen.

del problema que se pretende resolver.

- **Deterministas versus estocásticas:** Esta clasificación se basa en el modo de tomar decisiones por parte del algoritmo. Esto es, si los algoritmos toman decisiones en forma determinista entonces se trata de metaheurísticas deterministas. Si por el contrario, los algoritmos toman decisiones en forma aleatoria entonces se trata de metaheurísticas estocásticas. En consecuencia, los algoritmos deterministas obtienen siempre la misma solución partiendo de una misma solución inicial mientras que en los algoritmos estocásticos esto no sucede ya que se pueden obtener diferentes soluciones partiendo de una misma solución inicial.
- **Vecindario único versus múltiples vecindarios:** Esta clasificación refiere al mantenimiento o no de una única estructura de vecindarios utilizados por dicha metaheurística. Esto es, existen algoritmos que mantienen una topología de vecindario invariante durante todo el proceso, mientras que otros la modifican de manera dinámica. Cada vecindario determina una estrategia diferente para explorar el espacio de búsqueda (dominio de la función objetivo) así como también la forma en que se recorre la superficie del problema de optimización. La mayoría de los algoritmos utilizan una estrategia de vecindario único, sin embargo el uso de múltiples vecindarios brinda la posibilidad de poder intercambiar estrategias de búsqueda.

Dentro de las clasificaciones presentadas, se desarrollará sobre el punto de vista *Basadas en población versus basadas en única solución* dado que este permite describir a las metaheurísticas (o familias de estas) de un modo más natural e intuitivo. A continuación se profundiza en estas dos familias.

2.3.1. Metaheurísticas basadas en trayectoria

Las *metaheuristicas basadas en trayectoria* (o *basadas en una única solución*) son denominadas de esta manera aludiendo a la "trayectoria" de búsqueda que realizan sobre el espacio de soluciones factibles del problema [55]. Esto es, partiendo de una solución inicial única (de aquí la segunda denominación) mejoran de manera iterativa la solución actual hasta alcanzar una solución que cumpla con cierto criterio de parada previamente establecido.

El proceso iterativo se compone de dos fases, una fase de generación y una fase de reemplazo. La fase de generación consiste en generar para la solución única actual un conjunto de soluciones candidatas, construido en base a transformaciones locales de la solución actual. Mientras que la fase de reemplazo se encarga de la selección de la mejor solución candidata para reemplazar la solución actual en el siguiente paso de iteración. El conjunto de soluciones intermedias del proceso determina así la "trayectoria" antes mencionada.

Como ejemplos de metaheurísticas que forman parte de esta familia se pueden citar a *Basic Local Search*, *Simulated Annealing* y *Tabu Search* entre otros, las cuales no serán detalladas ya que escapan al alcance del presente documento.

2.3.2. Metaheurísticas basadas en poblaciones

Las *metaheurísticas basadas en población* manipulan un conjunto de potenciales soluciones (denominado población) durante el proceso de búsqueda, en contraposición a las descriptas en la sección anterior [55]. En primer lugar el proceso inicializa la población. Posteriormente dicha población se transforma de manera iterativa a través del proceso de búsqueda hasta que es alcanzado algún criterio de parada previamente establecido.

Cada iteración consta de dos fases, una de generación y otra de reemplazo. La fase de generación se encarga de generar una nueva población de soluciones candidatas. Mientras que la fase de reemplazo selecciona los individuos de la población actual que serán reemplazados por individuos de la nueva.

Algunos ejemplos de esta familia son *Evolutionary Algorithms*, *Scatter Search*, ACO, BC (Bee Colony) y PSO entre otros. Tanto ACO, BC como PSO se engloban en una subclase que se conoce con el nombre de SI (Swarm Intelligence). Esta última será detallada en la siguiente sección.

2.4. Swarm Intelligence

Resulta sorprendente observar cómo determinados conjuntos de insectos de una misma especie resuelven problemas aparentemente difíciles (desde el punto de vista humano) de manera relativamente sencilla. Esto ha llevado al ser humano a sentirse cautivado por entender cómo estos insectos resuelven determinados tipos de problemas que presentan cierta analogía con los que enfrenta el ser humano en su vida.

Bajo esta premisa es que surge el concepto de *Swarm Intelligence*, introducido por Beni y Wang en 1989. Dicho concepto se inspira en la simulación de mecanismos biológicos, más específicamente se basa en el comportamiento colectivo que presentan ciertos sistemas naturales (y también artificiales) compuestos por muchos individuos [27]. Este comportamiento colectivo se centra en la coordinación que resulta de las interacciones locales de los individuos de la población entre sí y con su entorno. Dicha coordinación es sustentada por dos pilares: el control descentralizado y la auto-organización.

Estos dos pilares producen un fenómeno muy interesante, a pesar de que no exista una estructura de control centralizada que dicte como se debe comportar cada individuo, las interacciones entre tales individuos dan lugar a la aparición de una conducta global inteligente, la cual obviamente no es perci-

bida por los mismos de forma individual. El concepto de Swarm Intelligence en sí mismo se puede resumir mediante las palabras de Bonabeau et al, 1999: *"La inteligencia colectiva emergente de grupos de agentes simples"* [9].

En la naturaleza existen varios ejemplos de sistemas que pueden ser considerados y que se comportan como Swarm Intelligence. Algunos de estos son: colonias de hormigas y/o termitas, bancos de peces, bandadas de aves, rebaños de animales y crecimiento bacteriano. Además existen otros sistemas de índole artificial que también pueden ser considerados, sistemas de robots que cooperan entre sí para la realización de una determinada tarea y ciertos programas de computación que son escritos para tratar con problemas de optimización y análisis de datos.

Figura 2.1: Banco de peces

Millonas propuso cinco principios básicos de Swarm Intelligence [35]:

- **Principio de proximidad**: la población debe ser capaz de realizar cálculos elementales de espacio y tiempo. Estos cálculos elementales deben entenderse como la respuesta o comportamiento que presentan los individuos ante los estímulos recibidos desde el entorno. En base a estos, los individuos deben tener la habilidad de evaluar la utilidad de los mismos en pos de maximizar la utilidad total asociada a la población vista como un todo.
- **Principio de calidad**: la población debe ser capaz de responder a factores de calidad en el entorno. Esto es, ante los diferentes factores de calidad (estímulos) provistos por el entorno, la población debe ser capaz de optar (responder) por el que brinde la mejor utilidad. Algunos ejemplos de estos factores incluyen: calidad de los alimentos, nivel de seguridad en una ubicación determinada, entre otros.
- **Principio de respuesta diversa**: la población no debe destinar todos sus recursos (individuos) a un conjunto reducido de acciones. Aquí lo que se pretende es que cada individuo de la población no responda de la misma manera ante los estímulos del entorno, sino por el contrario contar con diversos tipos de respuestas. El propósito es brindar seguridad ante el cambio imprevisto de cualquiera de los individuos originado por fluctuaciones en el entorno.
- **Principio de estabilidad**: la población no debe cambiar su modo de comportarse cada vez que el entorno cambia. Esta condición establece que la población cambie su modo de comportarse sólo cuando "valga la pena" (cuando el gasto de energía sea de utilidad) y no en cada fluctuación del entorno.
- **Principio de adaptabilidad**: la población debe ser capaz de cambiar

el modo de comportarse cuando la inversión de energía (costo computacional) vale la pena. Este principio claramente se contrapone con el de estabilidad. La determinación de la mejor respuesta de la población a los estímulos del entorno se encuentra en un punto intermedio entre el orden completo y el caos total (determinando estos el rango de posibles respuestas). Lo anterior combinado con cierto grado de aleatoriedad (factor de ruido fundamental para la generación de diversidad) dan lugar al punto intermedio (respuesta) buscado. Cabe destacar que introducir un nivel de ruido suficiente conlleva a la obtención de respuestas diversas, mientras que un excesivo nivel de ruido podría causar el deterioro del comportamiento cooperativo. De lo anterior se desprende que una respuesta totalmente ordenada por parte de los individuos al entorno tampoco es deseable, dado que no aporta diversidad.

Dentro de esta familia se destacan las técnicas PSO (la cual será detallada en el capítulo 3) y ACO que se presenta en la siguiente subsección.

2.4.1. Ant Colony Optimization

Un excelente ejemplo de trabajo colaborativo e inteligencia de enjambre puede observarse en la naturaleza en el trabajo de las hormigas. Estos seres vivos que por sí solos cuentan con una habilidad cognitiva limitada son capaces sin embargo de realizar tareas complejas actuando en conjunto. Particularmente se destacan en la resolución del problema asociado a la búsqueda y traslado de alimentos, el que de manera implícita incluye encontrar la ruta de menor distancia entre puntos [55].

La primera observación importante de esta realidad y que cabe destacar en primer término, es que las hormigas logran llevar a cabo este tipo de tareas de manera eficiente aún cuando muchas de las especies tiene escasa visión o incluso nula.

Con el objetivo de analizar el comportamiento específico y la forma de operar de estos insectos en la búsqueda de caminos de menor longitud entre puntos, Goss [20] realizó un experimento. El mismo consistía en reunir una colonia de hormigas, la cual tenía acceso a un recurso alimenticio unido al hormiguero mediante dos caminos posibles (uno más corto que el otro). En él observó de qué manera estas se organizaban para llegar a dichos alimentos y de qué forma lo trasladaban hacia su hormiguero.

Cada hormiga deja durante su recorrido un rastro en el piso utilizando una sustancia química denominada *feromona*. Esta sustancia volátil y olfativa tiene el objetivo de guiar al resto de las hormigas hacia el destino, es decir, a la fuente de alimento encontrada. Este recurso representa entonces un método de comunicación indirecta entre las hormigas.

Los caminos de mayor concentración de feromona resultan más atractivos para las hormigas y por lo tanto tienen una mayor probabilidad de ser utilizados. La volatilidad de la sustancia refiere a que esta pierde intensidad con el paso del tiempo, por lo que el rastro dejado por cada hormiga irá decreciendo a menos de que sea remarcado.

En el experimento se pudo observar como las hormigas comenzaban realizando un recorrido al azar al inicio desde el hormiguero hasta encontrar la fuente de alimento, pero como paulatinamente comenzaban a utilizar en mayor medida el camino más corto para llegar a esta y luego retornar. La Figura 2.2 ilustra lo antes descripto.

La cantidad de feromona que es liberada por las hormigas depende de la calidad de la fuente de alimento encontrada, condición que conjuntamente a la volatilidad antes mencionada favorece al no estancamiento en un camino, y por ende a una mayor exploración del territorio. Si los rastros perdurasen de manera indefinida estimularían la utilización de caminos que tal vez ya no conducen a fuentes de alimento e interferirían en el descubrimiento de nuevas fuentes.

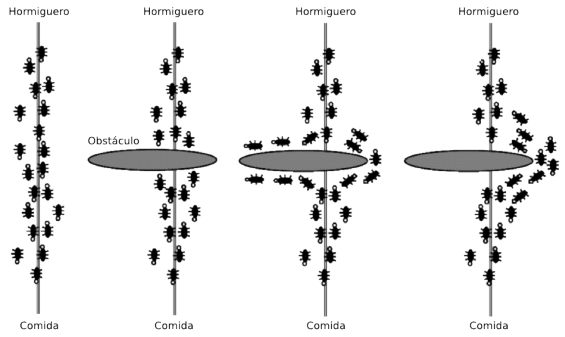


Figura 2.2: Inspiración de una colonia de hormigas buscando un camino óptimo entre el hormiguero y la comida

Inspirada en este comportamiento surge la metaheurística ACO (*Ant Colony Optimization*). Fue propuesta por Dorigo en su tesis de doctorado del año 1991 [3] y pertenece a la familia de algoritmos de inteligencia de enjambre. En la misma, se propone el uso de agentes simples denominados *hormigas artificiales* (en adelante se hará referencia sólo al término hormigas), los cuales imitan el comportamiento de su contraparte en el mundo real. Cada hormiga construye de manera independiente e incremental una solución para un problema de optimización dado [39]. Inicialmente cada hormiga comienza con una solución vacía. Progresivamente estas irán construyendo soluciones parciales mediante el agregado de *componentes*[5] verificando siempre la fac-

[5]En el contexto de ACO, un componente refiere al conjunto de valores que puede tomar una variable de decisión para un problema de optimización dado.

tibilidad de las mismas. Cada componente factible[6] que formará parte de una solución parcial es seleccionado de manera probabilística ponderando la experiencia adquirida (rastros de feromona) hasta el momento así como también la información heurística[7] del problema.

En Pseudocódigo 1 se presenta el algoritmo general de ACO correspondiente a problemas de optimización combinatoria estáticos[8]. El primer paso del algoritmo (línea 1) corresponde a la inicialización de los valores asociados a los rastros de feromona (experiencia adquirida). También es inicializada (línea 2) la mejor solución global (s_{best}) encontrada para el algoritmo con un valor arbitrario y un fitness asociado suficientemente grande.

Pseudocódigo 1 Algoritmo general de ACO

```
1: T = inicializarRastrosFeromona()
2: s_best = s | f(s) = +∞
3: while no se alcanzó el criterio de parada do
4:    pop = construirSolucionesHormigas(T)
5:    pop'= aplicarBusquedaLocal(pop) {opcional}
6:    T = actualizarRastrosFeromona(T,pop')
7:    s = seleccionarMejorDeLaPoblacion(pop')
8:    if f(s) < f(s_best) then
9:       s_best = s
10:   end if
11: end while
12: return s_best
```

La estructura principal del algoritmo se compone de un ciclo que itera mientras no se cumpla con el criterio de parada preestablecido. Dentro de este se distinguen cuatro etapas principales.

En la primera etapa (línea 4), cada hormiga (de forma independiente) parte de una solución inicial vacía e incrementalmente incorpora componentes generando así soluciones parciales hasta completar una solución. La selección de cada componente es realizada mediante el uso de una regla probabilística en la que es ponderada la experiencia adquirida (utilizando la información asociada a los rastros de feromona) así como también información heurística de los componentes candidatos (favoreciendo la exploración del espacio de búsqueda).

[6]Se denomina componente factible a aquel que al ser incorporado a una solución parcial mantiene la factibilidad de la solución parcial resultante.

[7]Por información heurística se entiende al valor estático asociado a una solución, el cual representa la adecuación relativa de una solución entre sus pares basada en intuición, experiencia previa y sentido común.

[8]Un problema de optimización se dice que es estático cuando el dominio de las variables de decisión, el conjunto de restricciones y la función objetivo son conocidos y se mantienen inalterados durante la resolución del problema.

La segunda etapa (línea 5) es opcional y corresponde a la aplicación de una búsqueda local con el objetivo de mejorar potencialmente las soluciones encontradas hasta el momento.

La tercer etapa (línea 6) corresponde a la actualización de los rastros de feromona asociados a cada componente de las distintas soluciones encontradas, teniendo en cuenta la calidad de las mismas. Para dicha actualización existen dos criterios, el primero se basa en el uso de la estrategia de evaporación (decrementando el nivel de feromona) con el objetivo de evitar la convergencia prematura del algoritmo, favoreciendo así la exploración mediante la utilización de información heurística. El segundo criterio consiste en incrementar el depósito de feromona asociado a las distintas componentes utilizadas en las soluciones construidas. Este último brinda a los componentes una mayor probabilidad a la hora de ser seleccionados para la construcción de futuras soluciones.

Finalmente, la cuarta etapa (líneas 7 a 10) consiste en la actualización de la mejor solución (global) encontrada hasta el momento.

Capítulo 3

Particle Swarm Optimization

Individuals interact with one another while learning from their own experience, and gradually the population members move into better regions of the problem space.

James Kennedy and Russell Eberhart

Optimización de nubes de partículas (PSO por su sigla en inglés, Particle Swarm Optimization) es una metaheurística estocástica basada en población que sirve para la resolución de problemas de optimización tanto continuos como discretos. PSO forma parte de la familia de metaheurísticas de Swarm Intelligence y fue desarrollada por Kennedy y Eberhart en el año 1995 [22]. Esta metaheurística está inspirada en el comportamiento social que presentan algunas especies de la naturaleza a la hora de resolver determinados problemas concernientes a sus necesidades. Más específicamente centra su atención en cómo una multitud de aves o peces resuelven el problema de la obtención de alimento de una forma aparentemente coordinada. Cabe destacar que estos dos ejemplos anteriores no son los únicos inspiradores, también se puede mencionar a las abejas, las cuales a la hora de buscar polen, lo realizan en las zonas donde existe una mayor concentración de flores.

3.1. Estructura del capítulo

A continuación se realiza la introducción a la técnica de Particle Swarm Optimization. Posteriormente, se brindan detalles sobre sus antecedentes, la descripción de la idea general de la técnica, la explicación del concepto de vecindario, las topologías potenciales con que trabaja, las diferentes clases existentes del algoritmo y sus posibles campos de aplicación.

3.2. Antecedentes de PSO

Dentro del área de la computación gráfica es donde aparecen los primeros trabajos que sirvieron como base para la creación posterior de PSO. El primer antecedente se asocia a un trabajo realizado por Reeves [45] en 1983, el que proponía el uso de un sistema de partículas destinadas a modelar objetos de comportamiento dinámico complejo. Algunos de estos objetos eran el fuego, las nubes, el humo, líquidos, entre otros ejemplos. El modelo se componía de un conjunto de partículas de comportamiento independiente y cuyo movimiento se basaba en el cumplimiento de un determinado conjunto de reglas.

Un segundo acercamiento a PSO se produce en 1987 cuando Reynolds desarrolla un modelo simple llamado *Boids* [44], que simulaba el comportamiento de una bandada de aves. En este modelo, el movimiento de cada ave estaba basado en tres reglas fundamentales:

- Prevención de colisiones: las aves se alejan antes de chocar entre sí.
- Concordancia de velocidad: las aves intentan mantener la velocidad de los vecinos de la bandada.
- Bandada centrada: las aves intentan movilizarse hacia el centro de la bandada.

En 1990, Heppner y Grenander [21] complementan el modelo de Reynolds con el agregado de un componente "roost" (lugar de paraje o descanso de las aves) con el objetivo de atraer a las aves del modelo. Las reglas en las que están basados los modelos anteriores fueron utilizadas para la construcción de la versión original de la metaheurística PSO.

Otra de las áreas de investigación donde se realizaron trabajos que influenciaron la creación de PSO es la de la investigación de la psicología social, en particular se destaca el trabajo de Nowak, Szamrej y Latané [37], basado en la teoría dinámica del impacto social de 1990. En este caso, las reglas asociadas a las partículas rigen el comportamiento social de las mismas en lugar de su movimiento. Este comportamiento simula la conducta social humana, sus creencias y sus actitudes.

Otro de los antecesores de PSO es el modelo de AC (Autómata Celular). Un AC es un modelo matemático para un sistema dinámico[1] que evoluciona en pasos discretos [60]. Está conformado por un conjunto de componentes idénticos y simples que son capaces de interactuar entre sí y evolucionar logrando como resultado un comportamiento grupal complejo. Permite modelar adecuadamente sistemas naturales que puedan ser descriptos como una

[1] Son sistemas cuyos parámetros internos (variables de estado) siguen una serie de reglas temporales. Se llaman sistemas porque están descritos por un conjunto de ecuaciones (sistema) y dinámicos porque sus parámetros varían con respecto a alguna variable que generalmente es el tiempo.

colección masiva de objetos simples que interactúan entre sí localmente.

Un AC consiste en una grilla o reticulado (lattice) de celdas a las que se denomina "célula", donde cada una puede tomar un número finito de valores. El valor de cada célula evoluciona de manera sincrónica a lo largo de pasos de tiempo discretos y es determinado de acuerdo a un conjunto de reglas idéntico para todas estas células y utilizando el valor anterior de las que conforman su vecindario.

3.3. Idea detrás de PSO

Antes de presentar la idea detrás de esta metaheurística se hará una pequeña digresión con respecto al término "partícula" [27]. Algunos autores utilizan el término "agentes" para referirse a las partículas debido a que abordan el tema desde una perspectiva que sólo concierne a especies de la naturaleza. Por otro lado, existen autores más puristas que prefieren el término "partícula" ya que este engloba tanto a especies como a individuos más generales donde PSO también aplica. Además, como se verá más adelante estos individuos contarán con dos atributos (posición y velocidad) y estos claramente se corresponden más con las partículas que con los agentes. Por lo tanto de aquí en más se utilizará el término "partícula" cuando se quiera referir a un individuo de la población (nube o multitud).

La idea detrás de esta metaheurística es básicamente tomar una población de partículas de manera aleatoria y recorrer (con estas) el espacio de búsqueda imitando el movimiento de las especies antes mencionadas con el fin de hallar una aproximación a la solución óptima del problema de optimización que se pretende resolver. Para lograr este objetivo, es necesario contar con dos aspectos esenciales, el primero corresponde a identificar reglas que permitan a las partículas imitar tales movimientos mientras que el segundo implica modelar esta realidad con el objetivo de poder ejecutar simulaciones y constatar el desempeño de dicha metaheurística.

Algunos autores han propuesto ciertas reglas con respecto a los movimientos que las partículas deben imitar. Tal es el caso de Reynolds [44] que introdujo las siguientes reglas:

- Cada partícula debe evitar colisionar con sus partículas vecinas.
- Cada partícula debe tratar de avanzar a la misma velocidad de sus partículas vecinas.
- Cada partícula debe intentar moverse hacia el centro de la multitud (nube) de partículas según su percepción.

Por otro lado, los autores Kennedy y Eberhart [26] realizaron simulaciones en donde el objetivo de las partículas era alcanzar una determinada zona

puntual del espacio de búsqueda, al que se conoce como "lugar de descanso" de las partículas. En tales simulaciones se identificaron las siguientes reglas:

- Cada partícula fue atraída hacia la ubicación del lugar de descanso.
- Cada partícula recordó donde estuvo más cerca del lugar de descanso.
- Cada partícula compartió información con sus partículas vecinas acerca de su ubicación más cercana al lugar de descanso.

Utilizando todas estas reglas se puede observar que las partículas interactúan unas con otras aprendiendo de su propia experiencia y que gradualmente estas se mueven hacia regiones más promisorias del espacio de búsqueda. A continuación, se presenta el modelo propuesto por los creadores de esta metaheurística, a partir de estas reglas.

En este modelo, la nube es conformada por M partículas que se moverán a través de un espacio de búsqueda D (con $D \subseteq \mathbb{R}^d$)[2]. Cada partícula i representará una solución factible del problema y será modelada como un vector $\vec{x}_i = (x_{i1}, x_{i2}, ..., x_{id})^T / \vec{x}_i \in D$, el cual representa la posición actual de dicha partícula.

Otro atributo importante de una partícula es la velocidad con la que esta se desplaza. La misma se modela con un vector $\vec{v_i} = (v_{i1}, v_{i2}, ..., v_{id})^T / \vec{v_i} \in D$. Además, el modelo registra cual fue la mejor posición visitada (la que obtuvo mejor valor de la función objetivo también conocida como función de fitness) de una partícula i a lo largo de toda su trayectoria. Esta mejor posición se denomina $\vec{p_i}$ y se modela como un vector $\vec{p_i} = (p_{i1}, p_{i2}, ..., p_{id})^T / \vec{p_i} \in D$.

En este modelo (y sin entrar en detalle en el algoritmo) se puede observar cómo se impone la colaboración entre partículas, es decir, el hecho de que una partícula tenga un buen fitness influenciará el comportamiento de sus partículas vecinas en la búsqueda del óptimo global, dado que la actualización de la posición de cada partícula dependerá de factores propios (de la misma partícula) y de factores externos (de su vecindad).

3.4. Vecindario de una partícula

Antes de continuar es necesario profundizar en el concepto de vecindario de una partícula, el cual está íntimamente ligado a otro importante concepto, la topología de la nube de partículas. Como se ha mencionado antes, cada partícula cuenta con cierta cantidad de vecinos que son los que influencian el comportamiento de esta. Aquí es importante destacar que la relación de vecindad entre dos partículas no depende de sus posiciones en el espacio

[2]Más formalmente el dominio D puede expresarse como $D = [x_d^{min}, x_d^{max}]^{n(D)}$, donde $n(D)$ representa el número de dimensiones de D, siendo x_d^{min} y x_d^{max} las cotas mínima y máxima respectivamente en la dimensión d.

de búsqueda sino más bien de aspectos puramente sociales. Estos aspectos sociales son los que diferencian a cada partícula de sus pares. Los mismos logran que dichas partículas tiendan a agruparse con pares de similares características. A continuación se presenta la definición formal de vecindario de una partícula.

Definición 3.1. Vecindario de una partícula: *El vecindario de una partícula es una función* $N : S \rightarrow 2^S$ *que asigna a cada* $s \in S$ *un conjunto de vecinos* $N(s) \subseteq S$. $N(s)$ *es llamado el vecindario de* s.

Cabe mencionar que en las diferentes versiones propuestas por la mayoría de los autores se incluye en el vecindario de la partícula a la propia solución. Por este motivo, a lo largo de este documento se adopta dicho criterio.

La relación de vecindad entre las partículas puede representarse como un grafo $G = (V, E)$, donde cada vértice v con $v \in V$, se corresponde con una partícula de la nube y cada arista e con $e \in E$, se corresponde con una relación de vecindad establecida entre un par de partículas.

El concepto de vecindario asocia además otro importante atributo al modelo, este es el llamado $\vec{p_l}$ o mejor posición local, modelado como un vector $\vec{p_l} = (p_{l1}, p_{l2}, ..., p_{ld})^T / \vec{p_l} \in D$, el cual corresponde a la mejor posición visitada por el vecindario de la partícula i. En caso de que dicho vecindario este compuesto por todas las partículas de la nube, esta mejor posición local se transforma en la mejor posición global denominándose $\vec{p_g}$ y modelándose de la misma forma.

3.5. Topología de la nube de partículas

El grafo subyacente de la sección anterior es lo que define la topología de la nube de partículas. En la literatura se pueden encontrar diferentes tipos de topologías [28]. A continuación se presentan algunas de estas:

- Topología de anillo: cada partícula tiene dos vecinos.
- Topología completamente conectada: se corresponde con un grafo completo.
- Topología aleatoria: los vecinos de cada partícula son elegidos al azar.
- Topología Von Neumann: cada partícula cuenta con vecinos al norte, sur, este y oeste, tomando dichas partículas sobre un plano bidimensional (aplicable a dimensiones superiores).
- Topología de estrella: cada partícula está conectada a una partícula central.
- Topología de pirámide: las partículas componen una pirámide tridimensional[3].

[3]También puede ser vista como malla triangular en tres dimensiones.

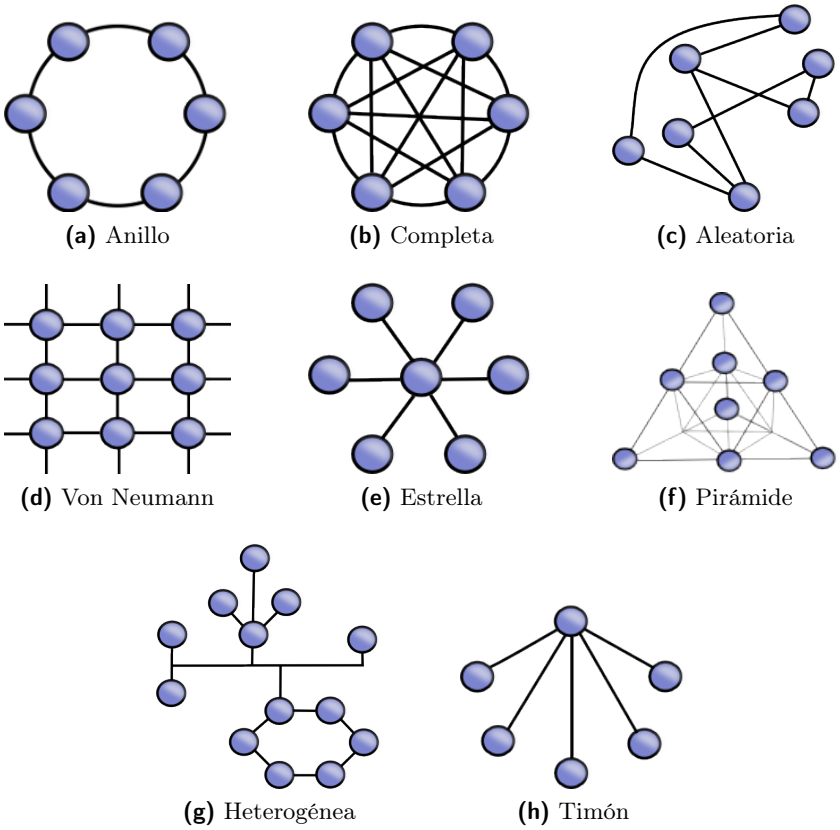


Figura 3.1: Topologías de la nube de partículas

- Topología heterogénea: las partículas se agrupan en conjuntos de diferentes características.

En la Figura 3.1 se presentan gráficamente algunas de dichas topologías.

La topología de la nube de partículas también puede cambiar a lo largo del tiempo debido a cambios en los vecindarios que la componen. A este tipo de topologías se las denomina topologías dinámicas.

3.6. Clases de PSO

La elección de una buena topología no sólo es clave para el éxito de esta metaheurística sino que determina la clase de PSO que estamos utilizando. Esto es, existen dos clases de PSO [54], las cuales se presentan a continuación:

- **Basadas en** $\vec{p_l}$: el comportamiento de cada partícula sólo es influenciado de forma local, es decir por sus vecinos, siendo la cantidad de vecinos menor que la cantidad total de partículas de la nube. Este escenario

ocurre con cualquier topología distinta de la topología completamente conectada.

- **Basadas en** $\vec{p_g}$: el comportamiento de cada partícula es influenciado de forma global, es decir por la totalidad de las partículas de la nube (sus vecinos). Este escenario se da únicamente con la topología completamente conectada.

Estas dos clases de PSO tienen sus ventajas y desventajas, la cuales están muy ligadas con aspectos de convergencia, así como también de exploración y explotación del espacio de búsqueda. Esto es, cuando el PSO está basado en $\vec{p_g}$ se cuenta con una convergencia rápida debido a que todas las partículas son influenciadas por la mejor posición de la nube (rápida propagación de la mejor información) y por ende estas son atraídas hacia la mejor región del espacio de búsqueda visitada por la nube en su totalidad. Este comportamiento viene acompañado por una pobre exploración del espacio de búsqueda y el posible problema de quedar atrapado en un mínimo local siempre y cuando el óptimo global se encuentre lejos de la mejor posición global. Si este no es el caso, se contará con una buena explotación del espacio de búsqueda además de la obtención del óptimo global.

Por otro lado, si se considera un PSO basado en $\vec{p_l}$, la convergencia resultará más lenta (tardía propagación de la mejor información) debido a que se exploran varios vecindarios (cubriendo de buena forma) del espacio de búsqueda, pero la explotación de cada región se hará con menor densidad de partículas. Este comportamiento puede incurrir en un mayor costo computacional pero puede redundar en una mejor oportunidad de encontrar el óptimo global.

De las ventajas y desventajas anteriores de ambas clases de PSO se puede concluir que para atacar problemas de optimización multimodales conviene utilizar la PSO basada en $\vec{p_l}$ por sobre la PSO basada en $\vec{p_g}$.

Sin embargo para abordar problemas unimodales la elección no es tan obvia, es decir, se debe realizar un balance entre exploración y explotación del espacio de búsqueda. En diferentes artículos se deja constancia que las topologías de Von Neumann y de pirámide producen mejores resultados en la práctica que el resto de las topologías [23, 49].

A partir del funcionamiento descripto para el PSO, parece razonable elaborar una estrategia como la que sigue: comenzar explorando el espacio de búsqueda con alguna topología apropiada para ello y luego dinámicamente a través del tiempo cambiar hacia una topología completamente conectada con el fin de explotar los mejores resultados del espacio de búsqueda obtenidos hasta el momento. Si bien esta idea parece atractiva, su implementación resulta compleja y por este motivo escapa al alcance del presente trabajo.

3.7. Áreas de aplicación de PSO

Existe una amplia variedad de áreas de aplicación de PSO que abarcan desde la medicina, el diseño gráfico, la aeronáutica, identificación de personas, base de datos entre muchas otras. A continuación, se describen algunos ejemplos destacados.

La industria del cine representa una de las áreas donde primero fue utilizada la metaheurística PSO, en dicho caso posibilitando la realización de efectos visuales que simulan la interacción de múltiples agentes e individuos, recreando manadas de animales, enjambre de insectos, batallas multitudinarias, entre otras. Algunos ejemplos de esta aplicación pueden observarse en la película animada "El Rey León", donde fue recreado el movimiento de una manada de bisontes, logrando un gran realismo en la escena [47]. Por otro lado, un ejemplo de aplicación en un film con actores reales puede observarse en la película "El Señor de Los Anillos", donde la metaheurística es aplicada para la simulación de escenas con batallas multitudinarias [29].

Otro de los ejemplos de aplicación de PSO es el entrenamiento de redes neuronales, apoyando la resolución de un gran número de problemas en múltiples áreas. Problemas como el estudio de temblores en seres humanos para la predicción de enfermedades como el Mal de Parkinson [15], estimación de carga de baterías para evitar sobrecargas [40], predicción de niveles de polución [33], problemas de tráfico [53], son algunos de los ejemplos de esta aplicación.

La clasificación (clustering) de elementos, que tiene como objetivo la agrupación de datos que comparten ciertas características, representa otra de las áreas donde PSO es frecuentemente aplicada. Para ello se define en primer lugar una función de distancia entre datos basada en algún criterio preestablecido. En base al conjunto de características de dichos datos, la función de distancia retornará valores más pequeños cuanto mayor sea la cantidad de características que los datos tengan en común. Un posible ejemplo se da en la clasificación automática de imágenes con características similares.

El área de diseño también ha sido abordada mediante PSO. En esta, es aplicada para apoyar la decisión en la selección de mejores modelos de componentes, como es el caso de tipos de alas para aviones, antenas, circuitos, estabilizadores de potencia entre otros ejemplos.

PSO también ha sido aplicada a problemas de planificación de tareas. Algunos ejemplos de esta aplicación refieren a la planificación de rutas de vuelo, asignación de tareas en actividades industriales o control de válvulas para motores de combustión interna.

Una de las áreas de mayor actividad en lo que refiere al procesamiento digital de imágenes es el de la detección y localización de rostros. Esto se debe a la gran importancia que juega esta práctica en lo referente a los sistemas de

identificación y seguimiento de rostros, así como para las interfaces hombre-máquina. Por tal motivo ha adquirido especial relevancia la búsqueda de nuevos métodos para la detección de rostros, de manera de aumentar la velocidad de la detección, así como mejorar la precisión con que estos son detectados, adaptándose a ambientes y condiciones de detección complejas de forma robusta. Se ha demostrado que las aplicaciones de reconocimiento de rostros dependen en gran medida de la exactitud con que se realiza su localización, motivo por el cual es necesario contar con métodos que permitan un mayor grado de certeza en la estimación de la posición del rostro. Se ha utilizado PSO para la optimización en la selección y desempeño de los llamados moldes direccionales, que consisten en una representación gráfica que modela la forma de los rostros, posiciones de los ojos y boca obtenidos durante el proceso de detección.

3.8. Algoritmo básico de PSO

En esta sección se describe el algoritmo básico de PSO [22, 27]. Cabe aclarar que el término "básico" en este caso hace alusión a "original" dado que existen muchas variantes del algoritmo, las cuales serán presentadas más adelante. Además es importante destacar que este algoritmo utiliza el modo de actualización sincrónico que será detallado en la Sección 3.9.

El Pseudocódigo 2 resume de manera simplificada el algoritmo básico de PSO, el cual será explicado a continuación. Sea P el enjambre de partículas, f la función de aptitud o fitness a optimizar y p_n la mejor posición visitada por el vecindario de la partícula.

Las líneas 1 a 5 del Pseudocódigo 2 describen la inicialización de las posiciones, velocidades y mejores posiciones iniciales de las partículas.

Al comienzo de este algoritmo todas las posiciones actuales de las partículas son inicializadas aleatoriamente de forma tal que: $\vec{x}_i \in D, \forall i \in P$, siendo P la población total (nube de partículas) [22] y D el dominio del problema. Dicha inicialización asegura que las posiciones iniciales de todas las partículas de la nube sean soluciones factibles del problema que se pretende resolver. Posteriormente, se procede a la inicialización de la mejor posición de cada partícula, la cual en principio toma el mismo valor que la posición inicial que le fue asignada. Por otro lado, también se deben inicializar aleatoriamente las velocidades actuales de cada una de las partículas de la siguiente manera: $\vec{v}_i \in D, \forall i \in P$. Sin embargo aquí cabe acotar que dicha inicialización debe prevenir que las partículas abandonen el espacio de búsqueda en tempranas instancias del algoritmo (debido a velocidades iniciales excesivas). Para evitar este inconveniente, dichas velocidades se suelen inicializar en el vector nulo $0_{n,1}$ ó cercanos a $0_{n,1}$ (en la sección 3.8.1 se brindarán detalles acerca de este punto).

El núcleo del algoritmo se basa en un ciclo (líneas 6 a 18) que itera hasta

Pseudocódigo 2 Algoritmo básico de PSO

1: **for** $i \in P$ **do**
2: $\vec{x}_i$ = posición aleatoria válida $\in D$
3: $\vec{v}_i$ = velocidad aleatoria $\in D$ cercana a $\vec{0}$
4: $\vec{p}_i = \vec{x}_i$
5: **end for**
6: **repeat**
7: **for** $i \in P$ **do**
8: **if** $f(\vec{x}_i)$ es mejor que $f(\vec{p}_i)$ **then**
9: $\vec{p}_i = \vec{x}_i$
10: **end if**
11: **end for**
12: Sea $\vec{p}_n$ la mejor posición $\vec{p}_i / i \in N(i)$, para cada vecindario.
13: **for** $i \in P$ **do**
14: $\vec{v}_i = \vec{v}_i + c_1 * R_1(t) \otimes (\vec{p}_i - \vec{x}_i) + c_2 * R_2(t) \otimes (\vec{p}_n - \vec{x}_i)$
15: $\vec{x}_i = \vec{x}_i + \vec{v}_i$
16: **end for**
17: $t = t + 1$
18: **until** se cumple la condición de parada del algoritmo
19: **end**

cumplir cierta condición de parada previamente establecida con el objetivo de que las partículas se desplacen sobre el espacio de búsqueda. Para realizar dicha iteración resulta necesario contar con la noción de tiempo, la cual será modelada con la variable t.

El primer paso de la iteración consiste en realizar para cada partícula las siguientes acciones (líneas 7 a 11):

- Evaluar la función de fitness para cada partícula.
- Actualizar la mejor posición de cada partícula ($\vec{p}_i$). Esto corresponde a comparar los valores de fitness de la posición actual de la partícula ($f(\vec{x}_i)$) y la de su mejor posición visitada hasta el momento ($f(\vec{p}_i)$). En caso de que se produzca una mejora en el fitness, se realiza dicha actualización.

Una vez actualizadas las mejores posiciones de cada partícula ($\vec{p}_i$) se procede a la actualización de la mejor posición global de cada vecindario ($\vec{p}_n$), lo que puede apreciarse en la línea 12. Esto implica comparar las mejores posiciones de cada partícula dentro del vecindario con la mejor posición global del mismo calculada hasta el momento. En caso de detectarse una mejora en el fitness, es actualizada la mejor posición global de cada vecindario.

El último paso de la iteración (líneas 13 a 16) consiste en la actualización de la posición y la velocidad de cada partícula. El cálculo de la nueva posición

de cada partícula puede ser visto como:

$$\vec{x}_i(t+1) = h(\vec{x}_i(t), \vec{v}_i(t), \vec{p}_i, \vec{p}_n) \tag{3.1}$$

Es decir, que la nueva posición actual de la partícula i es una función de su posición anterior ($\vec{x}_i(t)$), su velocidad anterior ($\vec{v}_i(t)$), su mejor posición ($\vec{p}_i$) y la mejor posición de su vecindario ($\vec{p}_n$).

Antes de continuar se hará una pequeña digresión sobre el término $\vec{p}_n$ (mejor posición visitada por el vecindario de una determinada partícula). Se utilizará esta notación con el objetivo de abstraerse de la clase específica de PSO que se utiliza debido a que la misma no influye en la descripción del algoritmo. Es decir que $\vec{p}_n$ puede adoptar el valor de $\vec{p}_l$ o bien el de $\vec{p}_g$ indistintamente.

Continuando con la descripción del algoritmo se presentan las reglas de movimiento de las partículas que se describen a continuación:

$$\vec{v}_i(t+1) = \vec{v}_i(t) + c_1.R_1(t) \otimes (\vec{p}_i - \vec{x}_i(t)) + c_2.R_2(t) \otimes (\vec{p}_n - \vec{x}_i(t)) \tag{3.2}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{3.3}$$

Donde cada componente que forma parte de las ecuaciones anteriores se describe a continuación:

- $\vec{v}_i(t+1)$ es la nueva velocidad de la partícula i.
- $\vec{v}_i(t)$ es la velocidad anterior de la partícula i.
- $\vec{x}_i(t+1)$ es la nueva posición de la partícula i.
- $\vec{x}_i(t)$ es la posición anterior de la partícula i.
- c_1 y c_2 son factores de aprendizaje.
- $R_1(t)$ y $R_2(t)$ son matrices diagonales aleatorias.
- $\vec{p}_i$ es la mejor posición visitada de la partícula i.
- $\vec{p}_n$ es la mejor posición visitada por el vecindario de la partícula i.

La representación vectorial de las Ecuaciones 3.2 y 3.3 se ilustran en la Figura 3.2.

Como se mencionó anteriormente, $R_1(t)$ y $R_2(t)$ son matrices diagonales cuyos elementos de la diagonal son valores aleatorios uniformemente distribuidos en el intervalo $[0, 1]$. Dichas matrices son las encargadas de proporcionar aleatoriedad en lo que concierne a la exploración del espacio de búsqueda. Notar que R_1 y R_2 son funciones de la variable t razón por la cual las mismas se regeneran en cada iteración.

Por otro lado, también fue mencionado que c_1 y c_2 son factores de aprendizaje, los cuales representan la atracción que posee una partícula de retornar

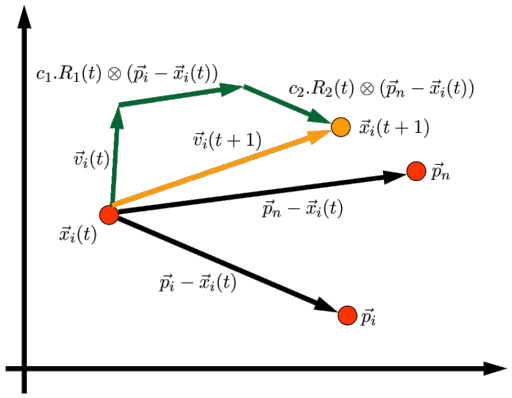


Figura 3.2: Gráfico de actualización de la posición de una partícula.

hacia su posición más exitosa o hacia la posición más exitosa de su vecindario. Esto es, c_1 es el factor de aprendizaje cognitivo, el cual representa la atracción que posee una partícula de retornar hacia su posición más exitosa. Mientras que c_2 es el factor de aprendizaje social, el cual representa la atracción que posee una partícula de retornar hacia la posición más exitosa de su vecindario.

Finalmente, nótese que la Ecuación 3.2 también se puede reescribir de la siguiente forma:

$$\vec{v}_i(t+1) = inercia + componente\ cognitiva + componente\ social \quad (3.4)$$

Donde cada componente se detalla a continuación:

- Inercia: esta componente tiende a mantener el movimiento de la partícula en la misma dirección con la que se venía desplazando, evitando así movimientos bruscos. La misma se corresponde con la componente $\vec{v}_i(t)$ de la Ecuación 3.2.
- Componente cognitiva: esta componente representa la tendencia de una partícula a retornar a su mejor posición visitada (también llamada componente de influencia personal). La misma se corresponde con la componente $c_1.R_1(t).(\vec{p}_i - \vec{x}_i(t))$ de la Ecuación 3.2.
- Componente social: esta componente representa la tendencia de una partícula a retornar a la mejor posición visitada por su vecindario (también llamada componente de influencia social). La misma se corresponde con la componente $c_2.R_2(t).(\vec{p}_n - \vec{x}_i(t))$ de la Ecuación 3.2.

3.8.1. Inicialización de la velocidad

La inicialización de la velocidad de las partículas juega un rol muy importante en el desempeño del algoritmo, dado que afecta significativamente la

exploración inicial y la convergencia del mismo. A continuación se analizan diferentes enfoques de inicialización de la velocidad [17], clasificados por el valor que estos pueden tomar:

- $\vec{v}_i(0) = \mathbf{0}_{n,1}$: La principal crítica acerca de este enfoque refiere a la supuesta limitante en lo relativo a la cobertura inicial del espacio de búsqueda, dando lugar así a una pobre exploración inicial. Sin embargo, esta afirmación es refutada mediante el argumento de que las posiciones iniciales de las partículas son distribuidas de manera uniforme en el dominio y por lo tanto la cobertura inicial no se ve afectada por este enfoque.
- **$\vec{v}_i(0)$ con valores aleatorios dentro del dominio**: lo que significa asignar valores al azar a $\vec{v}_i(0)$ asegurando que los mismos estén comprendidos de manera uniforme dentro de $[x_d^{min}, x_d^{max}]^{n(D)}$. Quienes sostienen este enfoque comúnmente afirman que este favorece a una mayor exploración, debido a que en su mayoría se tendrán tamaños de paso (velocidades iniciales) grandes. Por otro lado este enfoque tiene como desventaja el hecho de que existe una alta probabilidad de que las partículas abandonen el espacio de búsqueda en tempranas iteraciones del algoritmo (incluso en la primera).
- **$\vec{v}_i(0)$ con valores aleatorios cercanos a $\mathbf{0}_{n,1}$**: este enfoque es similar al anterior, pero con la salvedad de elegir un tamaño de paso pequeño. Esto claramente tiene el objetivo de evitar que las partículas abandonen el dominio en tempranas etapas además de proveer diversidad al enjambre en lo que refiere a exploración. Sin embargo, esto introduce un nuevo problema, el de decidir cuan pequeños deben ser los valores apropiados para el tamaño de los pasos.

3.8.2. Restricción del cálculo de la velocidad

Como se detalló anteriormente, el cálculo de $\vec{x}_i(t+1)$ está fuertemente ligado al de $\vec{v}_i(t+1)$. Este último cálculo tiende a tomar grandes valores haciendo que la partícula realice trayectorias oscilantes (ver Figura 3.3) cada vez más amplias dificultando así la convergencia de la misma hacia la mejor posición del espacio de búsqueda [14]. A fin de evitar este fenómeno, se limita el valor de cada componente del vector $\vec{v}_i(t+1)$ a v_{max} ó $-v_{max}$ según corresponda. En el Algoritmo 3 se presenta la restricción antes mencionada, siendo $v_{ik}(t+1)$ la k-ésima componente del vector $\vec{v}_i(t+1)$.

La restricción anterior tiene claras desventajas, entre las cuales se destacan:

- Restringir todas las componentes del vector $\vec{v}_i(t+1)$ con el mismo valor $\pm v_{max}$, genera los mismos rangos de valores en todas las dimensiones. Esto claramente no es deseable porque quizás podrían existir com-

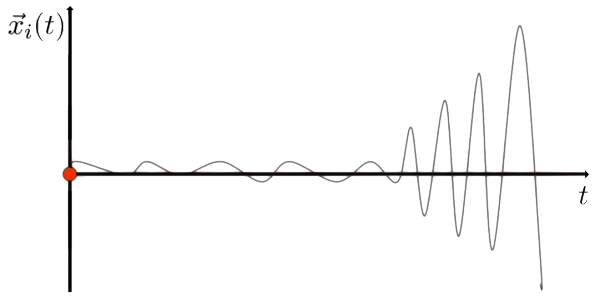


Figura 3.3: Gráfico de trayectoria de una partícula sin restricción de velocidad.

Pseudocódigo 3 Restricción del cálculo de la velocidad

```
1: if v_ik(t+1) > v_max then
2:     v_ik(t+1) = v_max
3: else
4:     if v_ik(t+1) < −v_max then
5:         v_ik(t+1) = −v_max
6:     end if
7: end if
```

ponentes que tomen valores en diferentes rangos de una determinada dimensión.

- Seleccionar un valor muy bajo de v_{max} tiene como consecuencia limitar el movimiento de la partícula haciendo que esta pueda no alcanzar la solución óptima.
- Por otro lado, utilizar un valor muy alto de v_{max} conlleva a que la partícula realice movimientos muy bruscos, alejando a la misma de la solución óptima.

Más allá de todas estas desventajas, generalmente el valor de v_{max} es seleccionado empíricamente dependiendo de las características del problema. Por otro lado, algunos autores han propuesto mejoras para tratar dichas desventajas. Tal es el caso de Fan y Shi [14] que han mostrado que cambiar v_{max} de manera dinámica puede redundar en una mejora en el rendimiento del algoritmo de PSO. Abido [2], en cambio propone asegurar que la velocidad sea uniforme a lo largo de todas las dimensiones tomando:

$$v_k^{max} = \frac{(x_k^{max} - x_k^{min})}{N} \tag{3.5}$$

Siendo v_k^{max} la velocidad máxima en la dimensión k, x_k^{max} la componente k de la mejor posición encontrada hasta el momento, x_k^{min} la componente k de la peor posición encontrada hasta el momento y N un número de intervalos elegidos en la dimensión k (el cual brinda la uniformidad deseada).

3.9. Modos de actualización

Existen dos alternativas a la hora de actualizar la posición y la velocidad de las partículas, a saber *modo sincrónico* y *modo asincrónico*. A continuación se brinda la descripción de cada uno de estos.

El *modo de actualización sincrónico* [1] refiere a que la actualización de la posición y la velocidad de cada partícula es realizada de manera independiente y luego de contar con la totalidad de la información. Esto es, que todas las partículas hayan sido evaluadas (cálculo de fitness) y que se conozcan las mejores posiciones de cada partícula, así como también las mejores posiciones de cada vecindario. Vilela en [43] realizó un estudio en el que concluye que el modo de actualización sincrónico es el que brinda mejores resultados y además una mayor velocidad de convergencia. No obstante, este modo resulta costoso en términos de tiempo [4] debido a que el movimiento de cada partícula hacia su nueva posición necesita que la totalidad de la información mencionada anteriormente haya sido calculada. El algoritmo descripto es detallado por el Pseudocódigo 4.

Pseudocódigo 4 Modo de actualización sincrónico

```
1: Inicializar la población de partículas
2: repeat
3:    Calcular los valores de fitness para cada una de las partículas
4:    Para cada una de las partículas, actualizar p_i si el nuevo valor de fitness
      de i es mejor que este
5:    Para cada vecindario, calcular los p_n como el mejor de los p_i de dicho
      vecindario
6:    for partícula i perteneciente a la población P do
7:       Actualizar v_i
8:       Actualizar x_i
9:    end for
10: until se cumple la condición de parada del algoritmo
```

El *modo de actualización asincrónico* [1] por su parte refiere a que la actualización de la posición y velocidad de cada partícula se realice con la información más reciente. Esto significa que las actualizaciones para cada partícula se realizan inmediatamente después de su evaluación, actualización de la mejor posición y mejor posición de su vecindario. Por lo tanto, en este tipo de actualización el proceso de búsqueda empleado por las partículas se apoya en información parcial (incompleta) de su vecindario. Es decir que se cuenta con parte de la información de la iteración actual (partículas que ya han sido evaluadas) y con información desactualizada de la iteración previa de aquellas partículas que aún no han sido evaluadas en la iteración en curso. Esta característica favorece la diversidad del enjambre, promoviendo una

mayor exploración [36].

Mussi [36] también realiza un estudio comparativo de los modos de actualización sincrónico y asincrónico, utilizando diferentes topologías, algunas que se corresponden con PSO basado en $\vec{p_l}$ (topología de vecindario local) y topología completa para PSO basado en $\vec{p_g}$ (topología de vecindario global). Como resultado de dicho estudio encontró que para la topología de vecindario global en general el modo de actualización asincrónico obtuvo mejores resultados que el sincrónico, mientras que en vecindarios locales los resultados fueron similares para ambos modos. Este fenómeno se debe a que las topologías de vecindario local tienden a preservar la diversidad, dado que las partículas están influenciadas sólo por la información de su vecindario. Otro trabajo realizado por Xue [61] comprobó que el modo de actualización sincrónico obtiene menores tiempos de ejecución. El algoritmo descripto es detallado por el Pseudocódigo 5.

Pseudocódigo 5 Modo de actualización asincrónico

```
1: Inicializar la población de partículas
2: repeat
3:    for partícula i perteneciente a la población P do
4:       Calcular el valor de fitness para la partícula i
5:       Actualizar p_i si el nuevo valor de fitness de i es mejor que este
6:       Actualizar p_n si el nuevo valor de fitness de i es mejor que este
7:       Actualizar v_i y x_i
8:    end for
9: until se cumple la condición de parada del algoritmo
```

Finalmente se presenta la Tabla 3.1 que detalla las diferencias más relevantes de los modos de actualización presentados previamente.

3.10. Variantes del algoritmo PSO

En esta sección se analizan las variantes propuestas más relevantes del algoritmo PSO. Estas fundamentalmente están vinculadas a la modificación de diferentes aspectos del funcionamiento del algoritmo, como son: valoraciones de los parámetros del algoritmo o las estrategias utilizadas para la actualización de la posición y velocidad de las partículas. Es importante destacar que existen un gran número de variantes, por lo que se optó por detallar sólo un conjunto reducido de estas seleccionando aquellas que resultaron más populares o con particularidades interesantes. Las siguientes subsecciones detallan las mencionadas variantes.

Propiedad	Modo Sincrónico	Modo Asincrónico
Actualización	En el inicio de cada iteración se actualizan todas las partículas	Actualizar la partícula tan pronto como su fitness sea evaluado
Información	Todas las partículas usan información de la misma iteración (información completa y un feedback más lento)	Las partículas usan una combinación de información de la iteración actual y la previa
Valor de $\vec{p}_n$	Todas las partículas son actualizadas basadas en los $\vec{p}_n$ de la iteración previa brindando un feedback inmediato	Las partículas usan la información disponible del momento, mejor diversidad

Tabla 3.1: Modo Sincrónico vs Modo Asincrónico

3.10.1. Peso de inercia

Como fue visto anteriormente el cálculo de la actualización de la velocidad $\vec{v}_i(t+1)$ se ve influenciado principalmente por el valor de la velocidad actual $\vec{v}_i(t)$. Este comportamiento provoca que la dirección del movimiento de las partículas tienda a mantenerse a lo largo del tiempo. Esto no es deseable dado que conlleva a una baja exploración y explotación del dominio. Con el objetivo de evitar esta situación Shi y Eberhart [50] proponen una variante que considera un coeficiente para la velocidad, el cual controla el nivel de influencia de la velocidad actual en el cálculo de la nueva, como puede observarse en la Ecuación 3.6. Dicho coeficiente se denomina peso de inercia y se nota como w.

$$\vec{v}_i(t+1) = \mathbf{w}\vec{v}_i(t) + c_1.R_1(t) \otimes (\vec{p}_i - \vec{x}_i(t)) + c_2.R_2(t) \otimes (\vec{p}_n - \vec{x}_i(t)) \quad (3.6)$$

Variando el valor de dicho parámetro, es posible controlar el nivel de exploración y explotación del espacio de búsqueda. Esto es, valores grandes de w estimulan la búsqueda global (exploración), mientras que valores pequeños promoverán la búsqueda local (explotación) [51]. Por este motivo se propusieron variantes en las que se altera de manera dinámica el valor de w a lo largo de la ejecución en contraposición con la versión original que lo mantiene invariante. En este documento se hará hincapié únicamente en esta última variante. En algunos trabajos (como [5]) se establece que al incluir el peso de inercia se elimina la necesidad del uso de v_{max} como limitante de la velocidad, mientras que otros autores utilizan ambos parámetros conjuntamente.

A continuación se brinda un análisis de convergencia[4] de esta variante [51, 58, 62], de acuerdo a los posibles valores que puede tomar el coeficiente de peso de inercia:

- Un valor $w >= 1$ provoca un incremento de la velocidad a lo largo del tiempo, lo que conduce a la divergencia del enjambre.
- Un valor $0 < w < 1$ por otro lado provoca una desaceleración de las partículas, lo que dependiendo de los valores asignados a c_1 y c_2 puede promover a la convergencia del enjambre.
- Mientras que un valor $w < 0$ provoca una desaceleración a lo largo del tiempo, llevando eventualmente a la velocidad a la cercanía del cero, generando un comportamiento de convergencia.

Algunos resultados empíricos sugieren valores para w, c_1 y c_2 que proveen buenos resultados [16].

3.10.2. PSO con factor de constricción

Esta variante propuesta por Clerc [11] introduce el uso de un coeficiente adicional en la ecuación de velocidad del algoritmo básico con el objetivo de evitar la explosión[5] del enjambre así como también asegurar la convergencia del mismo. Este trabajo surge del esfuerzo de Clerc por brindar una fundamentación matemática adecuada de la convergencia del algoritmo básico de PSO. En el mismo, plantea que las Ecuaciones 3.2 y 3.3 pueden considerarse un sistema de *ecuaciones en diferencias*[6] y por tal motivo el procedimiento de búsqueda (dinámica del sistema) puede ser analizado mediante el estudio de los valores propios permitiendo así controlar los aspectos relacionados a la convergencia del algoritmo y brindando una exploración eficiente, evitando la convergencia prematura. El fundamento matemático que conduce a esta conclusión escapa del alcance de este trabajo.

La nueva versión propone agregar un coeficiente χ que multiplica a todos los componentes de la ecuación de velocidad original(Ecuación 3.2), llamado *factor de constricción* [30]. Por lo tanto se obtiene una nueva versión de la ecuación de velocidad que se presenta en la Ecuación 3.7.

$$\vec{v}_i(t+1) = \chi[\vec{v}_i(t) + c_1.R_1(t) \otimes (\vec{p}_i - \vec{x}_i(t)) + c_2.R_2(t) \otimes (\vec{p}_n - \vec{x}_i(t))] \quad (3.7)$$

[4] El término convergencia alude al caso en que una secuencia progresa hacia un estado terminal. En términos matemáticos, dada una secuencia $\{x_k\}_{k=1}^{\infty}$, esta converge si se cumple que $\lim_{k\to\infty} x_k = p$ siendo p un valor arbitrario.

[5] El término explosión refiere al estado en el cual la posición y la velocidad de las partículas tienden a infinito.

[6] En términos generales, una ecuación en diferencias es una expresión que relaciona diferentes sucesiones, en las que una de ellas es desconocida.

Donde $\chi = \dfrac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}$, con $\varphi = c_1 + c_2$, $\varphi > 4$.

En [30] se sugiere un valor de $\varphi = 4{,}0$ con el objetivo de garantizar la estabilidad. Pero típicamente suele establecerse $\varphi = 4{,}1$ con $c_1 = c_2 = 2{,}05$.

Finalmente una de las consecuencias a destacar de esta variante es la de brindar un mayor control de la explosión de la velocidad de las partículas, y de esta manera prescindir de la restricción de la velocidad.

3.10.3. PSO basada en tribus (tribes)

Esta variante propuesta por Clerc en 2006 consiste en adaptar de manera dinámica el tamaño y la topología del enjambre de partículas (generalmente parámetros del algoritmo) a lo largo del tiempo [41]. Es decir, estos parámetros dejan de ser datos de entrada del algoritmo para transformarse en variables que serán recalculadas a lo largo de las iteraciones.

La población de partículas se divide en subpoblaciones (tribus), cada una de las cuales mantiene su propio orden y estructura. Las tribus "buenas" (que presentan buenas soluciones) pueden beneficiarse eliminando a su miembro o miembros más débiles, dado que ya presentan buenas soluciones al problema y por tal motivo pueden disminuir su cantidad de individuos. Es importante destacar que las partículas eliminadas de una tribu son también eliminadas de la población total. Por otro lado, las tribus "malas" pueden beneficiarse agregando nuevas partículas, aumentando su capacidad de mejora. La generación de nuevas partículas es realizada de manera aleatoria.

El algoritmo propuesto por esta variante es análogo al básico, con la salvedad de la inclusión del cálculo para la regeneración de la topología y tamaño de las tribus. Esta transformación sucede periódicamente cada $L/2$ iteraciones, siendo L el número de relaciones existentes entre las partículas de la población.

Esta variante es un claro ejemplo de un algoritmo de PSO basado en una topología dinámica y heterogénea.

3.10.4. ARPSO

La variante ARPSO (Attractive and Repulsive Particle Swarm Optimization), como su nombre lo indica, propone alternar entre dos fases, una atractiva y otra repulsiva, para sobreponerse al problema de la convergencia prematura a la hora de resolver un problema de optimización [46]. La idea principal es controlar la diversidad del enjambre, es decir manipular la forma en que este explora, así como también explota el espacio de búsqueda.

Para esto, se cuenta en primera instancia con una *fase atractiva* la cual promueve que las partículas se atraigan unas a otras empleando información acerca de las mejores soluciones encontradas hasta el momento. Como efecto

inmediato, el enjambre se contrae haciendo que la diversidad del mismo disminuya, favoreciendo la explotación y desfavoreciendo la exploración. Este fenómeno sucede hasta que se alcanza una cota mínima para la diversidad, a saber d_{low}. Aquí es importante destacar que esta fase utiliza como modelo subyacente el mismo que la variante de peso de inercia (ver Ecuación 3.6) ya vista en una sección previa.

Cuando el nivel de diversidad del enjambre alcanza el valor d_{low}, esta variante propone alternar hacia una *fase repulsiva*, en la cual las partículas se repelen de sus mejores posiciones, tanto propia ($\vec{p}_i$), como global ($\vec{p}_n$). Esto tiene como efecto que el enjambre se expanda incrementando así la diversidad del mismo, lo cual pone el foco en la exploración y no tanto en la explotación. Este incremento de la diversidad se da hasta que la misma alcanza una cota superior d_{high}. Es en este momento donde se produce nuevamente un cambio de fase, esta vez hacia la fase atractiva. Para lograr este cometido, la fase repulsiva invierte tanto el signo de la componente cognitiva como también el de la componente social, tal como se muestra en la Ecuación 3.8.

$$\vec{v}_i(t+1) = w\vec{v}_i(t) - \vec{\varphi}_1(t) \otimes (\vec{p}_i - \vec{x}_i(t)) - \vec{\varphi}_2(t) \otimes (\vec{p}_n - \vec{x}_i(t)) \tag{3.8}$$

Por lo tanto, el algoritmo irá alternando entre fases de exploración y explotación, atracción y repulsión, baja y alta diversidad hasta alcanzar la convergencia del mismo. El funcionamiento de este algoritmo se puede apreciar en el Pseudocódigo 6. Aquí se observan claramente dos diferencias con respecto al algoritmo PSO original: las funciones *establecerDireccion* y *calcularDiversidad*, las cuales se explicarán más adelante.

Pseudocódigo 6 Algoritmo ARPSO

```
1: inicializar()
2: while no completado do
3:    establecerDireccion()
4:    actualizarVelocidad()
5:    nuevaPosicion()
6:    asignarFitness()
7:    calcularDiversidad()
8: end while
```

Antes de pasar a describir las nuevas funciones previamente mencionadas, se presenta la Ecuación 3.9 la cual determina la nueva forma en la que la velocidad es actualizada:

$$\vec{v}_i(t+1) = w\vec{v}_i(t) + dir(\vec{\varphi}_1(t) \otimes (\vec{p}_i - \vec{x}_i(t)) + \vec{\varphi}_2(t) \otimes (\vec{p}_n - \vec{x}_i(t))) \tag{3.9}$$

Aquí la variable *dir* determina la fase actual en la cual se encuentra

el algoritmo. Para calcular el valor que esta adopta, se utiliza la función *establecerDireccion* presentada en el Pseudocódigo 7.

Pseudocódigo 7 Función Establecer Dirección

```
1: if dir > 0 and diversity < d_low then
2:     dir = −1
3: end if
4: if dir < 0 and diversity > d_high then
5:     dir = 1
6: end if
```

Notar que el Pseudocódigo 7 depende de otra variable, a saber: $diversity$. La misma se utiliza para medir la diversidad del enjambre y su valor puede ser determinado utilizando la Ecuación 3.10 (usada por la función *calcular-Diversidad*), donde S es el enjambre, $|S|$ es el tamaño del enjambre, $|L|$ es la longitud de la diagonal más larga del espacio de búsqueda[7], N es la dimensionalidad del problema, p_{ij} es el valor de la j-ésima componente de la partícula i y $\bar{p}_j$ es el valor de la j-ésima componente del punto promedio $\bar{p}$.

$$diversity(S) = \frac{1}{|S|\,|L|} \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^{N} (p_{ij} - \bar{p}_j)^2} \tag{3.10}$$

3.10.5. Cooperative PSO (CPSO)

Esta variante propone mejorar el rendimiento del algoritmo original de PSO mediante una estrategia de cooperación de múltiples enjambres optimizando diferentes componentes del vector solución.

La mayoría de las técnicas de optimización estocástica se enfrentan a un problema conocido como la "maldición de dimensionalidad", la que refiere a una caída en su performance cuando crece el número de dimensiones del dominio del problema que intentan resolver. Con el objetivo de resolver este problema, Potter propuso en el contexto de GA, dividir el espacio de búsqueda del problema en diferentes subespacios, mediante el particionado de los vectores solución, de manera tal que el algoritmo de optimización utilizado garantice el cubrimiento de la totalidad del espacio de búsqueda original [42]. Además, la función de fitness es evaluada siempre mediante la combinación de las soluciones encontradas para cada subespacio. Este modelo se conoce como CCGA (Cooperative Coevolutionary Genetic Algorithm).

Basada en esta idea surge CPSO (Cooperative Particle Swarm Optimization), propuesta por Van den Bergh y Engelbrecht [57], la cual utiliza la misma estrategia sobre la metaheurística PSO.

[7]Esto también puede verse como la distancia más larga que dos partículas cualesquiera pueden tomar dentro de los límites del espacio de búsqueda.

En el modelo $CPSO\text{-}S_K$ se propone que el espacio de búsqueda del problema sea dividido en K subespacios[8], mediante el particionado del vector solución. Para cada subespacio, asigna un enjambre P_j (con $j \in [1..K]$) diferente, conformado por un total de s partículas, siendo s un número arbitrario. Cada enjambre P_j tiene el objetivo de optimizar uno de los subespacios, esto es, un conjunto de componentes (del vector solución) asociadas al mismo, utilizando con tal fin un PSO independiente.

Un aspecto muy importante a destacar refiere a que la función de fitness utilizada por los diferentes PSOs es la original, cuyo dominio continúa siendo de dimensión n (dimensión del espacio de búsqueda). Notar que los diferentes enjambres sólo conocen un subespacio de menor dimensión (un subconjunto de componentes del vector solución), por lo que surge la necesidad de compartir información entre las distintas particiones con el fin de completar un vector solución de dimensión n. Esta información compartida se encuentra disponible en el llamado *vector de contexto*, el que será detallado a continuación.

El *vector de contexto* es un vector solución del problema, el cual oficia a modo de memoria compartida entre los diferentes PSOs que intentan optimizar las diferentes particiones del espacio de búsqueda. Dicho vector se compone de la concatenación de las mejores soluciones encontradas hasta el momento por cada uno de los diferentes PSOs[9]. Para ilustrar mejor esta idea se presenta a continuación la nomenclatura necesaria. Sean:

- $P_j.x_i$: posición de la partícula i en el enjambre P_j.
- $P_j.v_i$: velocidad de la partícula i en el enjambre P_j.
- $P_j.y_i$: mejor posición de la partícula i en el enjambre P_j.
- $P_j.\hat{y}$: mejor posición global del enjambre P_j.
- n: cantidad de dimensiones del espacio de búsqueda.
- $b(j, z)$: función que retorna un vector de dimensión n resultado de la concatenación de la totalidad de las mejores posiciones globales de los diferentes vectores de los enjambres, a excepción del j-ésimo que es sustituido por z, donde z corresponde a la posición de cualquier partícula del enjambre P_j.

El *vector de contexto* descripto anteriormente puede ser expresado como $(P_1.\hat{y}, P_2.\hat{y}, ..., P_K.\hat{y})$, siendo K el número de particiones del espacio de búsqueda y tal que la suma de las dimensiones de cada una de estas coincide con la dimensión del espacio de búsqueda, es decir $\sum_{j=1}^{K} dim(P_j) = n$.

Como enfoque inicial, se propone la división en subespacios unidimensionales, lo que implica utilizar una cantidad de particiones $K = n$. De esta

[8] La suma de los K subespacios coincide con el espacio original del problema.

[9] Cada PSO colabora con los mejores valores asociados al conjunto de componentes con el que opera.

manera se obtienen n subespacios 1-D, cada uno de los cuales será optimizado por un PSO diferente. A esta versión particular de la variante se la denomina $CPSO\text{-}S$ y en Algoritmo 8 se muestra su pseudocódigo.

La primera línea define la función $b(j, z)$ antes descripta, la cual permite a cada PSO el acceso al vector de contexto, mientras que la segunda inicializa los diferentes PSO a ejecutar. La estructura principal ejecuta hasta alcanzar algún criterio de parada preestablecido, iterando por cada partición del espacio (líneas 4 a 14) y para cada una de estas, procesando la información de cada una de sus partículas (líneas 5 a 12). Las líneas 6 a 8 y 9 a 11, corresponden a la actualización de la mejor posición de cada partícula y mejor posición global de cada enjambre respectivamente, utilizando la función de fitness. Finalmente en la línea 13 se ubican las llamadas a las funciones de actualización de posición y velocidad del algoritmo PSO.

Pseudocódigo 8 Algoritmo CPSO-S

1: Sea $b(j, z) = (P_1.\hat{y}, P_2.\hat{y}, ..., P_{j-1}.\hat{y}, z, P_{j+1}.\hat{y}, ..., P_n.\hat{y})$
2: Crear e inicializar n PSOs unidimensionales: $P_j, j \in [1..n]$
3: **repeat**
4: **for** cada enjambre $j \in [1..n]$ **do**
5: **for** cada partícula $i \in [1..s]$ **do**
6: **if** $f(b(j, P_j.x_i)) < f(b(j, P_j.y_i))$ **then**
7: $P_j.y_i = P_j.x_i$
8: **end if**
9: **if** $f(b(j, P_j.y_i)) < f(b(j, P_j.\hat{y}))$ **then**
10: $P_j.\hat{y} = P_j.y_i$
11: **end if**
12: **end for**
13: Actualizar PSO para P_j usando las ecuaciones 3.2 y 3.3.
14: **end for**
15: **until** se cumple la condición de parada del algoritmo

De manera más general, el modelo $CPSO\text{-}S_K$ propone dividir el vector solución en K particiones[10]. En Algoritmo 9 se muestra el pseudocódigo asociado a éste modelo, cuya estructura coincide con la detallada en Algoritmo 8, con la diferencia de que se agrega previo a la iteración principal, la creación de las particiones del espacio de búsqueda (líneas 2 a 5).

Notar que el modelo $CPSO\text{-}S$ puede ser visto como un caso particular del modelo $CPSO\text{-}S_K$, donde $K = n$.

[10] Al número K también se lo conoce como factor de división.

Pseudocódigo 9 Algoritmo $CPSO\text{-}S_K$

Sea $b(j, z) = (P_1.\hat{y}, P_2.\hat{y}, ..., P_{j-1}.\hat{y}, z, P_{j+1}.\hat{y}, ..., P_n.\hat{y})$
$K_1 = n \bmod K$
$K_2 = K - (n \bmod K)$
Inicializar K_1 $\lceil n/K \rceil$-dimensional PSOs: $P_j, j \in [1..K_1]$
Inicializar K_2 $\lfloor n/K \rfloor$-dimensional PSOs: $P_j, j \in [(K_1 + 1)..K]$
repeat
  for cada enjambre $j \in [1..K]$ **do**
    for cada partícula $i \in [1..s]$ **do**
      if $f(b(j, P_j.x_i)) < f(b(j, P_j.y_i))$ **then**
        $P_j.y_i = P_j.x_i$
      end if
      if $f(b(j, P_j.y_i)) < f(b(j, P_j.\hat{y}))$ **then**
        $P_j.\hat{y} = P_j.y_i$
      end if
    end for
    Actualizar PSO para P_j usando las Ecuaciones 3.2 y 3.3.
  end for
until se cumple la condición de parada del algoritmo

Capítulo 4

El problema de reducción de la variabilidad cíclica en motores de combustión interna y su resolución mediante PSO

Elegancia es la ciencia de no hacer nada igual que los demás, pareciendo que se hace todo de la misma manera que ellos.

Honoré de Balzac

El principal objetivo de este capítulo es presentar el problema de la reducción de la variabilidad cíclica en motores de combustión interna. Adicionalmente, se describe como este problema fue modelado mediante PSO y finalmente se brindan los detalles de la implementación realizada.

4.1. Estructura del capítulo

En este capítulo se comienza por la introducción al problema de reducción de la variabilidad cíclica en motores de combustión interna, dando pie al contexto y planteo del mismo. Luego, se presenta el enfoque para la resolución del problema mediante PSO. Finalmente, se brindan los detalles de la implementación, mencionando la arquitectura propuesta, así como también los diferentes módulos que la componen.

4.2. El problema de reducción de la variabilidad cíclica en motores de combustión interna

Cuando el espacio de búsqueda de un problema está contenido en $\mathbb{R}^n$ ($\mathcal{S} \subseteq \mathbb{R}^n$), el problema de optimización se clasifica como un problema de optimización continua. Trabajar en un espacio de búsqueda continua tiene la ventaja de que, si la función objetivo es suave, permite el uso de métodos como el descenso de gradiente [6]. Por este motivo, los problemas de optimización continua, por lo general, pueden ser resueltos rápidamente. Sin embargo, si la función objetivo no se conoce a priori, es imposible aplicar ese tipo de métodos. Este tipo de problemas se suelen conocer como problemas de optimización de caja negra, ya que la función objetivo puede ser vista como una caja negra, cuyos componentes internos son desconocidos y que sólo puede ser vista en términos de sus entradas y salidas.

Un caso particular de optimización de caja negra es cuando la función objetivo se calcula utilizando una simulación. En ese caso, la evaluación de cada solución tiene que ejecutar una simulación que puede llevar, dependiendo del caso, desde unos pocos segundos hasta varias horas. Esto puede ser problemático, ya que el algoritmo de optimización sólo puede obtener información del espacio de búsqueda a través de la ejecución de la simulación. El algoritmo de optimización tiene que elegir una solución candidata del espacio de búsqueda para evaluar en base únicamente a la información obtenida de las soluciones que ya han sido evaluadas. El problema de reducción de la variabilidad cíclica en motores de combustión interna será modelado como un problema de optimización de caja negra.

4.2.1. Contexto del problema

Los motores monocilíndricos de ciclo Otto son motores de cuatro tiempos (etapas) de combustión interna de encendido por chispa [34]. En la Figura 4.1 se muestran los elementos básicos de un motor de combustión interna. La primera etapa del ciclo corresponde a la admisión que es cuando entra el aire y el combustible en el cilindro. Luego, en la etapa de compresión, el movimiento del pistón comprime la mezcla de gases, se provoca la detonación de la chispa y por la elevada presión que ejercen los gases al quemarse, se pasa a la etapa de expansión. En la etapa de expansión la energía liberada en la combustión es dirigida al eje del motor mediante la biela y el cigüeñal. Finalmente, en la etapa de escape, se liberan los gases quemados a la atmósfera.

Curto-Risso et al. [12, 13] propusieron y validaron una simulación numérica que modela el funcionamiento de un motor de encendido por chispa monocilíndrico, teniendo en cuenta aspectos mecánicos, termodinámicos y de modelado de la combustión. El modelo numérico propuesto considera dos zo-

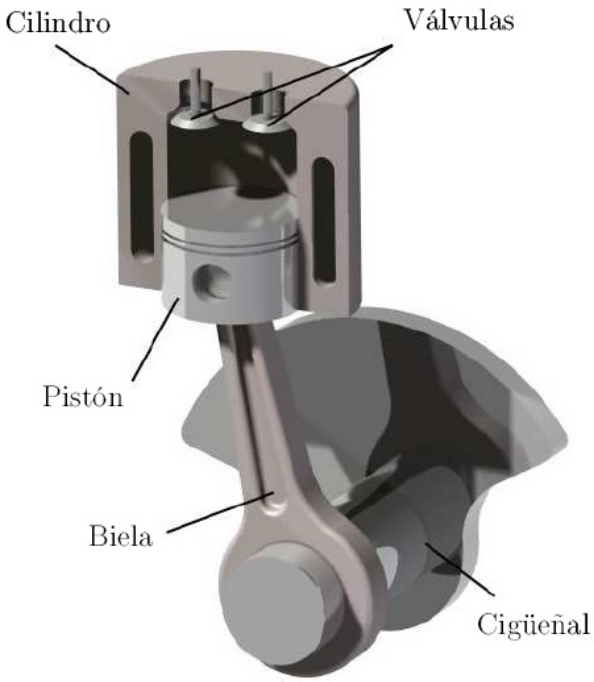


Figura 4.1: Descripción básica de los elementos del motor (imagen extraída de [34])

nas para la combustión distinguiendo entre los gases quemados y sin quemar que están separados por un frente de llama delgado. El modelo involucra un sistema de ecuaciones diferenciales ordinarias para la presión, la temperatura y las masas que considera las particularidades de cada una de las etapas de funcionamiento del motor. Dicho sistema es resuelto utilizando el método de Runge-Kutta, por más detalles se puede consultar [34].

Existe una gran variabilidad de un ciclo al siguiente en las propiedades termodinámicas del motor, esencialmente asociadas a la combustión turbulenta que tiene lugar en el cilindro. A este fenómeno se lo conoce con el nombre de variabilidad cíclica. La variabilidad cíclica es un problema fundamental de los motores de combustión interna que se ve influenciado por muchas variables del motor y operacionales, como las propiedades del combustible, la composición de la mezcla cerca de la bujía, etc. En el análisis de este tipo de fluctuaciones, se han realizado muchos trabajos examinando diferentes variables como la presión máxima o la liberación de calor por medio de diferentes métodos estadísticos [31, 32].

La variabilidad cíclica limita el rango de operaciones de los motores de encendido por chispa, incluso se estima que su eliminación podría aumentar en un 10 % la potencia de salida del motor para un mismo consumo de combustible [34]. En algunos casos, la variabilidad ciclo a ciclo produce grandes diferencias en la velocidad del motor, lo que da lugar a una baja

manejabilidad [34].

El estudio de la variabilidad cíclica en los motores de combustión interna tiene algunas dificultades. Desde un punto de vista experimental, se necesita adquirir y almacenar grandes volúmenes de información en tiempos de muestreo muy pequeños. Además, desde el punto de vista de la simulación numérica, los modelos de dinámica de los fluidos computacional suelen involucrar un muy alto tiempo de cómputo para procesar un sólo ciclo, por lo que es imposible estudiar series de tiempo muy largas.

En particular, Curto-Risso et al. [34] incorporaron la variabilidad al modelo numérico a través de considerar la longitud característica como un parámetro aleatorio del modelo. El modelo incorpora la aleatoriedad únicamente al inicio de la combustión de cada ciclo. El valor de la longitud característica se modifica durante la evolución de la combustión a partir de la relación de las densidades de los gases no quemados y una distribución aleatoria lognormal. Por detalles adicionales se puede consultar [34].

4.2.2. Planteo del problema

El problema de reducción de la variabilidad cíclica en motores de combustión interna consiste en encontrar la configuración de parámetros óptima para diversas variables de diseño y operación de un motor de encendido por chispa con el objetivo de minimizar la variabilidad cíclica de la serie de tiempo de la eficiencia del motor. El objetivo es minimizar la variabilidad cíclica a través de la minimización del coeficiente de variación (RCOV, por su sigla en inglés) de la eficiencia del motor. Se define a RCOV como la razón entre la desviación estándar y media de la serie de tiempo de la eficiencia del motor.

A partir de la experiencia en el trabajo sobre el modelo numérico del Prof. Curto-Risso, se identificaron cinco parámetros que tienen incidencia en la variabilidad cíclica: el ángulo de apertura de la válvula de admisión (IVO, por su sigla en inglés), el ángulo de cierre de la válvula de admisión (IVC, por su sigla en inglés), el ángulo de apertura de la válvula de escape (EVO, por su sigla en inglés), el ángulo de cierre de la válvula de escape (EVC, por su sigla en inglés), y el avance de la chispa (SA, por su sigla en inglés). Todos los parámetros identificados se miden en grados y toman valores reales. Dichos parámetros son las variables de decisión del problema de reducción de la variabilidad cíclica en motores de combustión interna.

Las variables de decisión del problema están definidas solamente en un intervalo de los reales que se detalla en la Tabla 4.1.

Variable	Límite inferior	Límite superior
IVO	−50,0	−10,0
IVC	210,0	250,0
EVO	480,0	500,0
EVC	735,0	750,0
SA	IVC	360,0

Tabla 4.1: Dominios de las variables de decisión.

4.3. Enfoque para la resolución del problema mediante PSO

Una vez presentados el contexto y el planteo del problema, se pasa a detallar el enfoque adoptado para el modelado y resolución del mismo utilizando PSO. Tal como fuera detallado anteriormente, dicho problema será planteado como uno de optimización continua de caja negra.

En primer lugar, es necesario definir el dominio sobre el cual será evaluada la función objetivo del problema. En este caso, dicho dominio se compone de 5 dimensiones, cada una asociada a los distintos intervalos reales definidos para las variables de decisión ya presentados anteriormente. Formalmente, el dominio D puede definirse como:

$$\begin{aligned} D = \{(IVO, IVC, EVO, EVC, SA) \in \mathbb{R}^5 / \\ -50{,}0 \leq IVO \leq -10{,}0 \wedge \\ 210{,}0 \leq IVC \leq 250{,}0 \wedge \\ 480{,}0 \leq EVO \leq 500{,}0 \wedge \\ 735{,}0 \leq EVC \leq 750{,}0 \wedge \\ IVC \leq SA \leq 360{,}0\} \end{aligned} \tag{4.1}$$

Tal como fue mencionado en la Sección 3.3, cada partícula tiene asociada una posición, la cual representa una solución factible que toma valores en el dominio D. Es importante destacar, que en cada iteración del algoritmo PSO, el movimiento de las partículas puede ocasionar que esta abandone el espacio de búsqueda definido por el dominio. Cuando esto ocurre, es necesario entonces que el algoritmo corrija la posición de las partículas de manera de preservar la factibilidad de las soluciones. La estrategia propuesta para estos casos será la siguiente: cuando en una determinada dimensión del dominio, la variable de decisión toma un valor que supera alguno de sus límites, esta adopta el valor del límite excedido. Al mismo tiempo, se establece en cero la velocidad de la partícula en dicha dimensión.

Para completar el modelo, es necesario definir la función objetivo. Dado que el problema se modela como uno de caja negra, la función objetivo no es más que el valor de RCOV que se obtiene mediante una ejecución del

modelo numérico antes mencionado. Dicha función evalúa soluciones en el dominio D y retorna un valor real que corresponde al coeficiente de variación ($f : D \to \mathbb{R}$).

En la Figura 4.2, se muestra como el algoritmo PSO interactúa con el modelo numérico del motor de combustión interna, reflejando el enfoque antes mencionado.

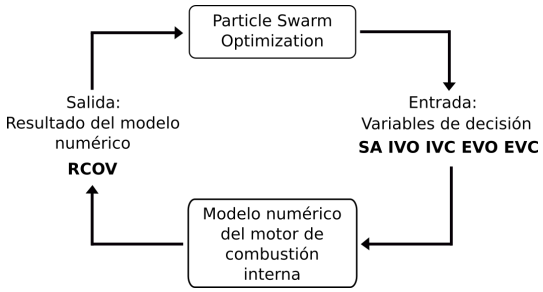


Figura 4.2: Interacción del algoritmo PSO con el modelo numérico del motor.

En el capítulo anterior fueron presentadas algunas de las variantes existentes de PSO. De estas, fueron seleccionadas las variantes de *Peso de Inercia* y *Factor de Constricción* para ser implementadas. El motivo por el cual dichas variantes fueron seleccionadas se debe principalmente a que gran parte de la bibliografía en que se basa este trabajo destaca su buen rendimiento y además en ambos casos resulta sencilla su implementación.

Actualmente se cuenta con valores de referencia que surgen del trabajo realizado por Curto-Risso, los cuales se presentan en la Tabla 4.2. Estos valores son los mejores resultados conocidos hasta el momento y servirán como punto de referencia para medir la calidad de los resultados.

Variable de decisión	**Valor de referencia**
IVO	−50,00
IVC	214,00
EVO	490,00
EVC	750,00
SA	330,00

Tabla 4.2: Valores de referencia del modelo numérico del motor.

Con el objetivo de evaluar la eficiencia numérica de PSO, los resultados obtenidos por ambas variantes serán comparados con los valores de referencia, así como también con los resultados que se obtengan mediante el uso de una búsqueda completamente aleatoria (*Random Search*).

El motivo de utilizar una técnica diferente para la resolución del problema fue el de contar con otro factor para poder evaluar la calidad de los resultados obtenidos por PSO.

El funcionamiento de *Random Search* consiste en el muestreo aleatorio del dominio de la función a optimizar durante un ciclo iterativo que actualiza el mejor resultado obtenido desde el inicio del proceso. La condición de finalización de la búsqueda consiste en un máximo preestablecido de soluciones muestreadas. El Pseudocódigo 10 presenta el algoritmo *Random Search*.

Pseudocódigo 10 Algoritmo Random Search

```
1: mejorSolucion = obtenerSolucionAleaoria()
2: mejorFitness = evaluarSolucion(mejorSolucion)
3: repeat
4:    solucion = obtenerSolucionAleaoria()
5:    fitness = evaluarSolucion(solucion)
6:    if fitness < mejorFiness then
7:       mejorSolucion = solucion
8:       mejorFitness = fitness
9:    end if
10: until no se alcanzó el criterio de parada
11: return  mejorSolucion
```

4.4. Detalles de implementación de PSO

Las particularidades del enfoque antes mencionado junto con la escasez de implementaciones o frameworks que se adaptaran a las necesidades del problema, fueron los principales motivadores para realizar una implementación propia del algoritmo PSO. Además, se identifican un conjunto de puntos importantes que apoyaron esta decisión, los cuales se detallan a continuación:

- **Recolección de datos:** contar con la capacidad de flexibilizar y personalizar la forma en que los datos son recolectados.
- **Configuración:** se deseaba contar con un conjunto de parámetros de configuración que permitieran alternar fácilmente entre las dos variantes de PSO[1], así como también ajustar el comportamiento de cada una de estas.
- **Fácil implementación:** dada la baja complejidad del algoritmo PSO, resultaba interesante y conveniente realizar una implementación propia que contara con todos los aspectos que se consideraran relevantes.
- **Integración:** era necesario seleccionar un lenguaje apropiado para la fácil integración con el código del motor provisto.
- **Performance:** partiendo de la premisa de que los tiempos de invocación a la función de fitness son muy significativos, fue necesario tener

[1]De esta manera es posible realizar una única implementación parametrizada para ambas variantes.

la libertad de seleccionar los detalles para la implementación con el fin de reducir los tiempos totales de cómputo. Esto es, definir el lenguaje de programación a utilizar, el paradigma, la arquitectura entre otros.

Habiendo presentado los puntos que motivaron la realización de una implementación propia, se destacan algunas de las decisiones relevantes para su desarrollo. Como lenguaje de programación se optó por la utilización de *C*, por cumplir este con los requisitos de performance e integración con el código del motor. Esta decisión a su vez motivó el diseño de una arquitectura modular que se detallará más adelante.

Previo a la decisión anterior, fueron evaluadas algunas implementaciones existentes. Dado que ninguna se adaptaba a las necesidades del problema, las mismas fueron descartadas. A continuación se nombran dos de estas implementaciones y los motivos por los que se descartó su utilización.

Una de las implementaciones disponibles es *PSO Standard* [10]. Se trata de un código fuente en lenguaje *C* desarrollado por Kentzoglanakis en 2010. Algunas de las carencias de esta implementación fueron, que no contaba con las topologías con las que se iba a trabajar y sólo ofrecía la implementación de la variante de peso de inercia.

Otra implementación evaluada fue *Particle Swarm Optimizer* [10], desarrollada por Shi en 1998. En este caso, la misma se descartó dado que también carece de soporte para la variante de factor de constricción, sólo provee el modo de actualización asincrónico y no dispone de un conjunto de topologías a utilizar.

Finalmente, luego de evaluar las diferentes implementaciones disponibles y analizar sus carencias, el esfuerzo requerido para adaptarlas a las necesidades del problema en cuestión comparado con el de desarrollar una versión propia desde cero no parecía significativo. Además, este enfoque brinda la libertad tanto en flexibilidad como personalización de la implementación del algoritmo.

4.4.1. Arquitectura de la implementación

Para la implementación se optó por un enfoque de programación modular. Esto es, en primera instancia se identificaron y categorizaron los diferentes aspectos funcionales que componen la implementación del algoritmo PSO. Posteriormente, se agrupan en módulos los diferentes aspectos funcionales pertenecientes a una misma categoría. El conjunto de estos módulos conforman la arquitectura de la solución propuesta. La Figura 4.3 muestra el conjunto de módulos de la arquitectura, así como también la interacción entre los mismos.

En las siguientes subsecciones se detallará el objetivo y las características de cada uno de los módulos que componen esta arquitectura.

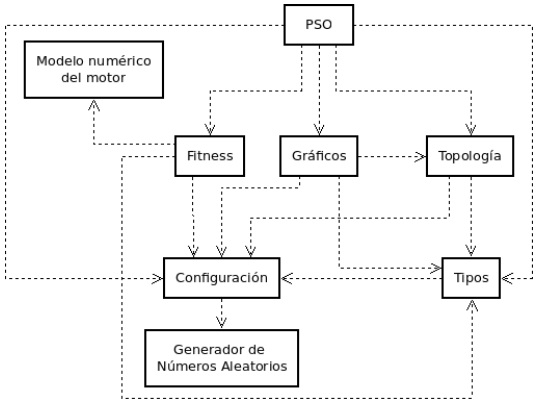


Figura 4.3: Diagrama de arquitectura.

4.4.2. Módulo de Configuración

Con el fin de permitir variar el comportamiento de la implementación del algoritmo PSO brindada, se provee un archivo de configuración (*parametros.cfg*). Este archivo permite alterar parámetros inherentes al algoritmo así como también parámetros específicos de dicha implementación. En la Tabla 4.3 se presentan los diferentes parámetros a configurar con sus posibles valores asociados.

Parámetro	Descripción	Valores asociados
tamPoblacion	Tamaño de la población	$\mathbb{N}^*$
dimension	Dimensión del dominio de la función de fitness	$\mathbb{N}^*$
w	Peso de inercia	[0.4, 0.9]
cantMaxEjecuciones	Cantidad máxima de ejecuciones independientes del algoritmo	$\mathbb{N}^*$
cotasDominio	Conjunto de rangos asociados a cada una de las dimensiones del dominio	conjunto de rangos
politicaInitVel	Política de inicialización de la velocidad	{0, 1, 2}
tipoTopologia	Topología de partículas	{0, 1, 2, 3}
c_1	Factor de aprendizaje cognitivo	{2.0, 2.05}
c_2	Factor de aprendizaje social	{2.0, 2.05}
modoActualizacion	Modo de actualización	{0, 1}
umbral	Límite a partir del cual una solución se considera aceptable	$\mathbb{R}^*$
cantMaxIter	Cantidad máxima de iteraciones permitidas para cada ejecución del algoritmo	$\mathbb{N}^*$
cantMaxIterSinMejoras	Cantidad máxima de iteraciones permitidas sin producirse mejoras de fitness	$\mathbb{N}^*$
habilitarGraficos	Habilitar/deshabilitar ventana de gráficos	{0, 1}
funcionFitness	Función de fitness a optimizar	{0, 1, 2, 3}
nVmax	Número de intervalos en cada dimensión para el cálculo dinámico de v_{max}	$\mathbb{N}^*$
habilitarFactorConst	Habilitar factor de constricción	{0, 1}
rutaArchivoSalida	Ruta del archivo donde se guardará la información de salida	{ruta válida, NONE}

Tabla 4.3: Parámetros de configuración y sus valores asociados

En la Tabla 4.3 se utilizaron valores constantes para algunos de los parámetros. A continuación se detalla la semántica de los mismos:

- **politicaInitVel**: Cero(0), Uniforme(1), No Uniforme(2).
- **tipoTopologia**: Completa(0), Anillo(1), Aleatoria_6(2), Von Neumann(3).

- **modoActualizacion**: Sincrónico(0), Asincrónico(1).
- **funcionFitness**: Motor(0), Sphera(1), Rosembrock(2), Griewank (3).
- **habilitarFactorConst**: Deshabilitar(0), Habilitar(1).
- **habilitarGraficos**: Deshabilitar(0), Habilitar(1).

Antes de continuar con la descripción del módulo se harán algunas aclaraciones y/o restricciones sobre los valores que pueden adoptar algunos de los parámetros. En primera instancia se tiene el caso del tamaño de la población, el cual requiere que el número de partículas sea múltiplo de 3 si la topología utilizada es de *Von Neumann*. Por otro lado, para los factores de aprendizaje cognitivo y social (c_1 y c_2 respectivamente) se tiene que ambos deben tomar el valor 2,0 para la variante de *Peso de Inercia*, mientras que para la variante de *Factor de Constricción* estos deben tomar el valor 2,05. También vale la pena destacar que cada valor del parámetro *cotasDominio* representa un conjunto de rangos, cada uno de los cuales se corresponde con una dimensión del dominio. Finalmente, para que el parámetro de la habilitación de gráficos funcione, la dimensión del problema a resolver debe ser 2.

En el Capítulo 5 se pueden visualizar en mayor detalle los valores adoptados para los demás parámetros. Un ejemplo de archivo de configuración se presenta en el Código 4.1.

Código 4.1: Ejemplo de archivo de configuración de parámetros

```
1  tamPoblacion = 30
2  dimension = 2
3  w = 0.4
4  cantMaxEjecuciones = 5
5  cotasDominio = {(-100,100);(-100,100)}
6  politicaInitVel = 0
7  tipoTopologia = 3
8  c1 = 2.05
9  c2 = 2.05
10 modoActualizacion = 0
11 umbral = -10
12 cantMaxIter = 200
13 cantMaxIterSinMejoras = 20
14 habilitarGraficos = 0
15 funcionFitness = 1
16 nVmax = 10
17 habilitarFactorConst = 0
18 rutaArchivoSalida = salida.txt
```

La idea de este módulo es tomar un archivo de configuración como el antes descripto y cargar su información en las estructuras de datos asociadas al mismo. Para lograr este cometido, se cuenta con los archivos *configuracion.h* y *configuracion.c*. En el Código 4.2 se pueden apreciar las estructuras de datos soportadas por el módulo mientras que en el Código 4.3 se proveen las funciones necesarias para realizar la carga de la configuración. Es importante destacar que este módulo también proporciona funciones para el manejo de números aleatorios, las cuales utilizan servicios provistos por el módulo de

números aleatorios descripto en la Sección 4.4.6.

Código 4.2: Estructuras de datos de configuración

```
1  typedef struct
2  {
3    double max;
4    double min;
5  } CotasDimension;
6
7  typedef CotasDimension * CotasDominio;
8
9  typedef struct
10 {
11    int tamPoblacion;
12    int dimension;
13    double w;
14    int cantMaxEjecuciones;
15    CotasDominio * cotasDominio;
16    int politicaInitVel;
17    int tipoTopologia;
18    double c1;
19    double c2;
20    int modoActualizacion;
21    double umbral;
22    int cantMaxIter;
23    int cantMaxIterSinMejoras;
24    int habilitarGraficos;
25    int funcionFitness;
26    int nVmax;
27    int habilitarFactorConst;
28    double factorConst;
29    char * rutaArchivoSalida;
30 } Configuracion;
```

Como puede apreciarse en el Código 4.2, el módulo define tres estructuras. En primer lugar *CotasDimension*, la cual modela las cotas mínima y máxima para una determinada dimensión. Posteriormente aparece la estructura *CotasDominio*, que modela la totalidad de cotas del dominio del problema, mediante un arreglo de *CotasDimension*. Finalmente la estructura *Configuracion*, es la encargada de mantener la información contenida en el archivo de configuración.

Código 4.3: Funciones del módulo de configuración

```
1  // Inicializa el generador de números aleatorios
2  void inicializarGeneradorAleatorio();
3
4  // Retorna un número aleatorio entre min y max
5  double obtenerNumeroAleatorio(double min, double max);
6
7  // Retorna la configuración de acuerdo a lo especificado en el
8  // archivo pathConf.
9  Configuracion * obtenerConfiguracion(char * pathConf);
```

4.4.3. Módulo de Fitness

Este módulo provee la funcionalidad para evaluar la función de fitness asociada al problema a resolver. La misma puede ser seleccionada de un conjunto disponible de estas en el archivo de configuración, tal como es detallado en la Sección 4.4.2. Se dispone de la función del motor (objetivo principal del trabajo) así como también de un conjunto de funciones de benchmark. La implementación de dicho módulo se compone de los archivos *fitness.h* y *fitness.c*.

Código 4.4: Funciones del módulo de Fitness

```
1 // Retorna el valor de fitness para la posición actual dentro del
2 // espacio de búsqueda de acuerdo a lo especificado en config
3 double fitness(Posicion * pos, Configuracion * config);
```

El código fuente del archivo *fitness.c* incluye las diferentes implementaciones de las funciones de fitness antes mencionadas. El objetivo de la función *fitness* es el de seleccionar una de estas utilizando con tal fin la información contenida en el parámetro de entrada *config*.

4.4.4. Módulo de Gráficos

Este módulo consiste en un conjunto de funciones utilizadas para ilustrar el movimiento de las partículas durante la ejecución del algoritmo. Los gráficos están disponibles sólo para funciones de fitness con dominio de 2 dimensiones y su ejecución es opcional mediante la variación de un parámetro en el archivo de configuración. Su implementación se encuentra disponible en los archivos *graficos.h* y *graficos.c*.

Código 4.5: Funciones del módulo de gráficos

```
1  // Inicializa el módulo de gráficos.
2  void inicializarGraficos(Configuracion * config);
3
4  // Dibuja en pantalla la estructura de vecindarios.
5  void dibujarVecindarios(Particula * particulas,
6  Vecindarios * vecindarios, Configuracion * config);
7
8  // Dibuja en pantalla las partículas.
9  void dibujarParticulas(Particula * particulas,
10 Configuracion * config);
11
12 // Cierra el módulo de gráficos.
13 void finalizarGraficos(Configuracion * config);
```

4.4.5. Módulo del Modelo numérico del motor

Este módulo corresponde al código fuente del motor de combustión (implementado en Fortran dentro del archivo *modelo18f.f*) que se desea optimizar. Se utilizó un enfoque similar al de caja negra para su manipulación,

con la salvedad de la modificación de la variable de número de ciclos (*ciclos*) alterada durante la etapa de calibración.

Si bien no se profundizará en el código de este componente (dado el enfoque de caja negra antes mencionado), resulta de interés mostrar un ejemplo de cómo fue accedido para la implementación de la función de fitness asociada al problema objetivo de este trabajo. En el código 4.6 se muestra una sección que indica el modo en el que es invocada la función del motor.

Código 4.6: Invocación de función del motor

```
1 ...
2 double pcov, ivo, ivc, evo, evc, sa;
3 ...
4 wmodelof_(&sa,&ivo,&ivc,&evo,&evc,&pcov,&resultado);
5 ...
```

La función *wmodelof_* es provista por el componente del motor recibiendo la misma 7 parámetros. Los 5 primeros (*sa, ivo, ivc, evo* y *evc*) corresponden a las variables independientes del problema (parámetros de entrada) y determinan el dominio de la función. De los dos parámetros restantes (ambos de salida), el primero (*pcov*) no será tenido en cuenta, mientras que el segundo (*resultado*) brinda el valor de fitness asociado a los parámetros de entrada antes mencionados.

Código 4.7: Funciones del módulo de números aleatorios

```
1  /* initializes mt[N] with a seed */
2  void init_genrand(unsigned long s);
3
4  /* initialize by an array with array-length */
5  /* init_key is the array for initializing keys */
6  /* key_length is its length */
7  /* slight change for C++, 2004/2/26 */
8  void init_by_array(unsigned long init_key[], int key_length);
9
10 /* generates a random number on [0,0xffffffff]-interval */
11 unsigned long genrand_int32(void);
12
13 /* generates a random number on [0,0x7fffffff]-interval */
14 long genrand_int31(void);
15
16 /* These real versions are due to Isaku Wada, 2002/01/09 added */
17 /* generates a random number on [0,1]-real-interval */
18 double genrand_real1(void);
19
20 /* generates a random number on [0,1)-real-interval */
21 double genrand_real2(void);
22
23 /* generates a random number on (0,1)-real-interval */
24 double genrand_real3(void);
25
26 /* generates a random number on [0,1) with 53-bit resolution*/
27 double genrand_res53(void);
```

4.4.6. Módulo Generador de Números Aleatorios

Este módulo contiene el conjunto de funciones que proveen la generación de números aleatorios. Los fuentes que componen el módulo no corresponden a la implementación de este trabajo, sino que se trata de la implementación de un tercero. El código fuente se encuentra disponible en los archivos *mt19937ar.h* y *mt19937ar.c*.

De las funciones provistas en dicho módulo se utilizó *init_genrand* que permite la inicialización del generador y *genrand_real1* que permite obtener un número aleatorio real en el intervalo $[0, 1]$. El detalle completo de las funciones provistas por el módulo es mostrado en el Código 4.7.

Código 4.8: Estructuras de datos de tipos

```
1  typedef double * Posicion;
2  typedef double * Velocidad;
3
4  typedef struct
5  {
6     int numero;
7     Posicion * pos;
8     Velocidad * vel;
9     double fitness;
10    Posicion * mejorPos;
11    double mejorFitness;
12 } Particula;
```

4.4.7. Módulo de Tipos

Aquí se define la estructura de datos asociada a las partículas (como es mostrado en el Código 4.8) así como el conjunto de funciones que permiten su manipulación, las que se muestran en el Código 4.9. La implementación de este módulo se encuentra disponible en los archivos *tipos.h* y *tipos.c*.

La estructura de la partícula encapsula un conjunto de datos que permiten describirla, los que incluyen, un número de identificación (*numero*), posición de la partícula (*pos*), velocidad (*vel*), valor de fitness para la posición actual (*fitness*) e información de mejor posición (*mejorPos*) y fitness (*mejorFitness*) encontrados para la partícula.

Código 4.9: Funciones del módulo de tipos

```
1  // Copia los datos de posOrigen a posDestino
2  void copiarDatosPosicion(Posicion * posOrigen,
3  Posicion* posDestino, int dimension);
4
5  // Retorna una copia de la posición "pos".
6  // dimension indica el número de dimensiones del espacio
7  // de búsqueda.
8  Posicion * crearPosicion(Posicion * pos, int dimension);
9
10 // Retorna una posición aleatoria válida dentro del
11 // espacio de búsqueda.
12 Posicion * obtenerPosicionAleatoria(Configuracion * config);
13
14 // Retorna una velocidad inicial de acuerdo a la política empleada.
15 Velocidad * obtenerVelocidadInicial(Configuracion * config);
16
17 // Libera la memoria reservada por posicion
18 void liberarMemoriaPosicion(Posicion * posicion, int dimension);
19
20 // Libera la memoria reservada por velocidad
21 void liberarMemoriaVelocidad(Velocidad * velocidad, int dimension);
22
23 // Imprime en pantalla la posición pos
24 void imprimirPosicion(Posicion * pos, int dimension);
25
26 // Imprime en pantalla la velocidad vel
27 void imprimirVelocidad(Velocidad * vel, int dimension);
28
29 // Imprime en pantalla los datos de la partícula especificada.
30 void imprimirParticula(Particula * particula, int dimension);
31
32 // Retorna el texto descriptivo de la posición especificada.
33 char* obtenerDatosPosicion(Posicion * pos, int dimension);
```

4.4.8. Módulo de Topologías

Este módulo define las estructuras de datos para la representación de los vecindarios y el conjunto de funciones necesarias para su manipulación. Además define otra estructura utilizada para mantener la información acerca de los mejores vecindarios así como las funciones que permiten operar sobre esta.

La implementación de este módulo se encuentra disponible en los archivos *topologia.h* y *topologia.c*. El Código 4.10 muestra el segmento de código correspondiente a las estructuras de datos antes mencionadas, mientras que el Código 4.11 detalla los cabezales de las funciones.

Código 4.10: Estructuras de datos de topologías

```
1  typedef struct
2  {
3     int * vecinos;
4     int tope;
5  } Vecinos;
6
7  typedef Vecinos* Vecindarios;
8
9  typedef struct
10 {
11    Posicion * mejorPos;
12    double mejorFitness;
13    Posicion * peorPos;
14    double peorFitness;
15 } MejorVecino;
16
17 typedef MejorVecino * MejoresVecinos;
```

Como se puede observar en el Código 4.10, se optó por modelar los vecindarios (estructura de datos *Vecindarios*) como un arreglo de conjuntos de vecinos, cada uno de los cuales es modelado por la estructura *Vecinos*. Por otro lado, se define una estructura con el propósito de mantener los datos de los mejores vecinos encontrados (*MejoresVecinos*), definida como un arreglo de *MejorVecino*.

Código 4.11: Funciones del módulo de topologías

```
1  // Retorna un mejor vecino a partir de una posición y un fitness.
2  MejorVecino * crearMejorVecino(Posicion * mejorPosicion,
3  double mejorFitness, Posicion * peorPosicion,
4  double peorFitness, int dimension);
5
6  // Carga la topología especificada y retorna los vecindarios.
7  Vecindarios * obtenerVecindarios(Configuracion * config);
8
9  // Imprime la información de los mejores vecinos.
10 void imprimirMejoresVecinos(MejoresVecinos * mejoresVecinos,
11 Configuracion * config);
12
13 // Imprime la información del mejor vecino.
14 void imprimirMejorVecino(MejorVecino * mejorVecino, int dimension);
15
16 // Retorna el texto que describe el tipo de topología.
17 char* obtenerDescripcionTopologia(int tipoTopologia);
```

4.4.9. Módulo de PSO

Contiene el programa principal, la estructura general del algoritmo PSO así como también el conjunto de funciones necesarias para inicialización y manipulación de las estructuras de datos utilizadas. Su implementación es provista en los archivos *pso.h* y *pso.c*.

Código 4.12: Estructuras de datos de PSO

```
1  typedef struct
2  {
3     double mejorFitness;
4     Posicion * mejorPos;
5     double mejorFitnessAnterior;
6     int idMejorParticula;
7     int iter;
8     int iterSinMejoras;
9     double peorFitness;
10    Posicion * peorPos;
11    int idPeorParticula;
12    double tiempoEjecucion;
13 } DatosEjecucion;
```

El Código 4.12 muestra la estructura de datos definida para el proceso, cuyo propósito es el de mantener el conjunto de datos de ejecución, información del mejor y peor fitness encontrado, mejor partícula, entre otros.

Del conjunto de funciones listadas en el Código 4.13 se destaca la función *actualizarPosicionYVelocidad*, en la cual son aplicadas las ecuaciones de actualización de posición y velocidad de cada partícula, tal como fueran detalladas en la Ecuación 3.2 y la Ecuación 3.3. Además proporciona (mediante el uso de los parámetros de configuración definidos en 4.4.2) la posibilidad de alternar entre las ecuaciones correspondientes a las variantes de *Peso de inercia* y *Factor de constricción* (detalladas en las secciones 3.6 y 3.7, respectivamente), seleccionadas para el estudio de este trabajo.

Código 4.13: Funciones del módulo de PSO

```
1  // Retorna un vector de partículas inicializadas al azar.
2  Particula* inicializarParticulas(Configuracion * config);
3
4  // Inicializa la estrucura de mejores vecinos.
5  MejoresVecinos* inicializarMejoresVecinos(Particula * particulas,
6  Vecindarios * vecindarios, Configuracion * config);
7
8  // Actualiza la mejor posición de la partícula i.
9  void actualizarMejorPosicion(int i, Particula * particulas,
10 Configuracion * config);
11
12 // Actualiza la mejor posición del vecindario asociado a la
13 // particula i.
14 void actualizarMejorPosicionVencindario(int i,
15 MejoresVecinos* mejoresVecinos, Particula* particulas,
16 Vecindarios* vecindarios, Configuracion* config,
17 DatosEjecucion * datosEjecucion);
18
19 // Actualiza las mejores posiciones encontradas para cada
20 // vecindario.
21 void actualizarMejoresPosicionesVencindarios(
22 MejoresVecinos* mejoresVecinos, Particula* particulas,
23 Vecindarios* vecindarios, Configuracion* config,
24 DatosEjecucion * datosEjecucion);
25
26 // Auxiliar para obtener la velocidad maxima admitida.
27 Velocidad* obtenerVelocidadMaxima(Configuracion* config,
28 MejoresVecinos * mejoresVecinos, int idParticula,
29 DatosEjecucion * datosEjecucion);
30
31 // Actualiza la posición y la velocidad de la particula i
32 void actualizarPosicionYVelocidad(int i, Particula * particulas,
33 MejoresVecinos * mejoresVecinos, Configuracion * config,
34 DatosEjecucion * datosEjecucion);
35
36 // Retorna true sólo si se cumple la condición de parada del
37 // algoritmo
38 int seCumpleCondicionParada(Configuracion * config,
39 DatosEjecucion * datosEjecucion);
40
41 // Actualiza las distintas iteraciones de los datos de ejecución.
42 void actualizarIteraciones(DatosEjecucion * datosEjecucion);
43
44 // Inicializa las estructuras de datos de ejcución.
45 DatosEjecucion * inicializarDatosEjecucion(Particula * particulas,
46 Configuracion * config);
47
48 // Libera la totalidad de recursos utilizados durante la ejecución.
49 void liberarRecursos(Configuracion* config,  Particula* particulas,
50 MejoresVecinos* mejoresVecinos,  Vecindarios* vecindarios,
51 DatosEjecucion* datosEjecucion);
52
53 // Imprime el resultado de las ejecuciones en archivo
54 // o en pantalla.
55 void imprimirResultados(Configuracion * config,
56 DatosEjecucion ** datosEjecuciones);
```

Capítulo 5

Calibración de Parámetros

The pessimist complains about the wind;
the optimist expects it to change; the
realist adjusts the sails.

William George Ward

El objetivo de este capítulo es determinar los valores apropiados para cada uno de los parámetros de los algoritmos *PSO* implementados. Con tal motivo serán seleccionados los parámetros para los que se realizará un proceso de calibración, mientras que se brindará la justificación de los valores seleccionados para aquellos que no formen parte de dicho proceso. Se cuenta con la implementación de dos variantes de *PSO*, razón por la cual serán planteadas justificaciones sensiblemente diferentes en cada caso. Finalmente serán detallados los procedimientos de calibración, los criterios para la selección de valores y los resultados obtenidos.

5.1. Estructura del capítulo

En primer lugar se detallan los parámetros de las variantes de *PSO* considerados en esta calibración de parámetros, posteriormente se presentan los parámetros que serán excluidos del proceso de calibración y a continuación aquellos que están incluidos en dicho proceso. Luego, se describe en detalle los pasos que conforman el mencionado proceso. Finalmente, se presentan los resultados obtenidos en la ejecución de la calibración dando lugar a los valores de parámetros a ser utilizados en la etapa de evaluación.

5.2. Parámetros de los algoritmos

Para lograr un tiempo razonable para la etapa de calibración, fue necesario optar por la selección de un conjunto reducido de parámetros a calibrar. Para ello se seleccionaron valores apropiados para el resto de los parámetros, a partir de valores sugeridos en diferentes trabajos del área de PSO. Cada parámetro seleccionado para ser calibrado aumenta el conjunto de pruebas necesarias, ya que incrementa exponencialmente el número de combinaciones de las pruebas que deben ser ejecutadas.

A continuación se comentan los parámetros para los que se toman valores de la bibliografía relevada y que no forman parte del proceso de calibración.

Uno de los parámetros que no es tenido en cuenta en la calibración es el tamaño de la población. Carlisle y Dozier [4] realizaron un estudio experimental en el cual analizaron los efectos del tamaño de la población en la calidad de las soluciones obtenidas. Dicho experimento consistió en variar el tamaño de la población desde 5 hasta 200 partículas considerando incrementos de 5 partículas, manteniendo el resto de los parámetros inalterados y utilizando un conjunto de funciones de benchmark.

Para dicho experimento se realizaron 20 ejecuciones, cada una de las cuales fue limitada a un máximo de 100000 iteraciones, registrando al término de las mismas los datos de tasa de éxito[1], promedio y media de ejecuciones exitosas y promedio y media del número de invocaciones a la función evaluada.

Las funciones de benchmark usadas fueron las propuestas por Kennedy [25], a saber: *Sphere*, *Rosenbrock*, *Rastrigrin*, *Griewank* y *Schaffer F6*, las cuales se muestran en la Tabla 5.1. Por otra parte, los valores utilizados para los parámetros invariantes fueron propuestos por Shi y Eberhart [16], esto es, usando el *factor de constricción* K con un valor de $\varphi = 4{,}1$ (coeficiente de aprendizaje cognitivo = coeficiente de aprendizaje social = 2,05) y $v_{max} = x_{max}$.

Los resultados del experimento muestran que no todos los tamaños de población (para todas las funciones tratadas) permitieron obtener una tasa de éxito del 100 %. Las funciones de menor exigencia[2] para el algoritmo resultaron ser *Griewank*, *Rosenbrock* y *Sphere*, mientras que *Schaffer F6* y *Rastrigrin* condicionaron el tamaño de la población a 30 para conseguir tasas de éxito próximas a 100 %.

Otro resultado importante del experimento es que mostró que cuando se usa una población de mayor tamaño se requiere un menor número de iteraciones para alcanzar resultados exitosos. Esto conduciría a incentivar el uso de poblaciones de mayor tamaño para la resolución de problemas con mayor

[1] El éxito alude al caso en que el algoritmo converge al óptimo del problema.

[2] Por exigencia se entiende a la cantidad de partículas necesarias para la obtención de tasas de éxito prometedoras.

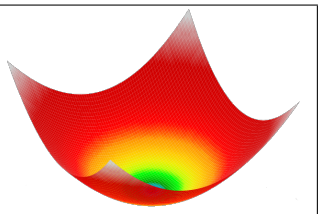
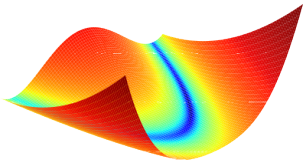
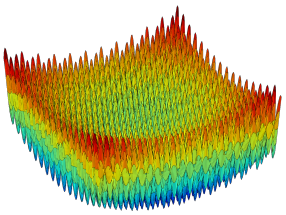
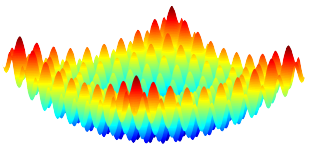
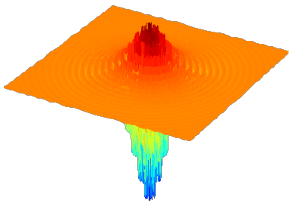
Función	Ecuación	Gráfico
Sphere (De Jong F1)	$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$	
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	
Grienwank	$f(\vec{x}) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$	
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i) + 10)$	
Schaffer's F6	$f(\vec{x}) = 0{,}5 - \frac{sin^2(\sqrt{x_1^2 + x_2^2}) - 0{,}5}{(1 + 0{,}001(x_1^2 + x_2^2))^2}$	

Tabla 5.1: Funciones de benchmark para el estudio del tamaño de la población

rapidez, sin embargo esto conlleva a un mayor costo computacional debido al aumento de evaluaciones de la función objetivo. Por lo tanto, resulta necesario encontrar un punto de equilibrio en lo que respecta a la cantidad de evaluaciones de la función de fitness y a la calidad de las soluciones obtenidas.

En lo que respecta a la afirmación "PSO no es sensible al tamaño de la población" formulada por Shi y Eberhart en [52], el experimento reveló que si bien esto es cierto en términos de calidad de los resultados, no lo es en términos de performance computacional del algoritmo.

Como conclusión del experimento, se obtiene que un tamaño de población de 30 partículas brinda la mejor relación *calidad de los resultados - costo computacional*, es decir que representa un valor lo suficientemente pequeño para ser eficiente pero suficientemente grande como para producir buenos resultados. Por este motivo, se usa 30 partículas como tamaño de la población durante la calibración para las variantes de peso de inercia y factor de constricción.

Los parámetros *factor de aprendizaje cognitivo* (c_1) y *factor de aprendizaje social* (c_2) tampoco serán calibrados, sin embargo se usarán valores diferentes para las variantes de peso de inercia y factor de constricción. En base al trabajo de Kennedy [24] se fundamentarán los valores escogidos para la variante de peso de inercia. En dicho trabajo se realiza un análisis de la trayectoria de las partículas que permite obtener un rango útil para el valor de los factores de aprendizaje.

Para ello se estudia una simplificación del modelo original de PSO para centrarse únicamente en el impacto del llamado "parámetro de control"[3]. Las simplificaciones involucran utilizar un dominio de una sola dimensión, limitar el tamaño de la población a una única partícula (prescindiendo de la notación vectorial), eliminar la aleatoriedad del modelo transformándolo en uno determinista y dejar constantes las variables p y φ ($= c_1 + c_2$). De este modo se obtiene la simplificación dada por las Ecuaciones 5.1 y 5.2 para el modelo original de PSO.

$$v(t+1) = v(t) + \varphi * (p - x(t)) \tag{5.1}$$

$$x(t+1) = x(t) + v(t+1) \tag{5.2}$$

Donde φ es el ya mencionado parámetro de control, el cual permite obtener información acerca del comportamiento de la partícula mediante la variación de este en diferentes rangos que Kennedy evaluó de manera empírica. Del estudio de los rangos de φ se determinaron diferentes comportamientos en la trayectoria de la partícula (para cada uno de estos) permitiendo así diversificar el modo en que esta explora y/o explota el espacio de búsqueda.

[3]Este parámetro refiere a una simplificación de los factores de aprendizaje (c_1 y c_2) que será explicada más adelante.

En el estudio se concluye que cuando $\varphi >= 4{,}0$ la trayectoria de la partícula "explota"[4] sugiriendo que valores superiores a 4,0 no serían adecuados.

Finalmente, se concluye que el rango útil para φ varía desde 0 a 4,0. Por otro lado, se suele recomendar un balance entre los factores de aprendizaje (social y cognitivo) y como $\varphi = c_1 + c_2$ se suele utilizar $c_1 = c_2 = \varphi/2 = 2{,}0$ [24].

Por otra parte, la selección de los valores para c_1 y c_2 de la variante de factor de constricción se basó en los trabajos de Eberhart y Shi [16] y de Tewolde et al. [56], en donde se destacan como valores comúnmente utilizados $c_1 = c_2 = 2{,}05$ implicando esto un valor de factor de constricción $\chi = 0{,}729$. Por otro lado, Clerc y Kennedy [11] señalan que usar un valor de 0,729 para χ promueve al decrecimiento de la oscilación de las partículas, asegurando la convergencia y estabilidad del algoritmo.

La política de inicialización de la velocidad es otro de los parámetros que no serán calibrados. En este caso la decisión se apoya en lo fundamentado por Engelbrecht y Andries [17], donde se realiza un análisis empírico de diferentes políticas de inicialización de la velocidad, las cuales fueran descriptas en la subsección 3.8.1. A partir de dicho análisis se concluye que la opción correspondiente a la inicialización de la velocidad en el vector nulo o cercano al mismo supera al resto en términos de rendimiento y convergencia. Por este motivo se optó por inicializar la velocidad en cero, es decir $\vec{v}_i = 0_{n,1}, \forall i \in P$.

Otro de los parámetros que no será calibrado es N utilizado para el cálculo de la velocidad máxima $\vec{v}_{max}$ (ver subsección 3.8.2). En este caso, la decisión se apoya en el propio trabajo donde Abido [2] propone el uso de este parámetro con la intención de brindar uniformidad en el control de la velocidad entre las diferentes dimensiones del dominio del problema. En dicho trabajo se utiliza un valor de $N = 10$ para la implementación del algoritmo PSO.

Vale la pena destacar que los algoritmos PSO implementados utilizan también algunos parámetros adicionales que tampoco serán calibrados, dado que son inherentes a la condición de parada del algoritmo PSO. Dichos parámetros se detallan a continuación.

El parámetro *umbral* es utilizado como criterio para definir el límite a partir del cual se considera aceptable el resultado obtenido en la búsqueda, permitiendo así detener el algoritmo. Este valor es definido para cada problema particular en que es aplicado el algoritmo y está fuertemente asociado a un criterio de calidad preestablecido.

Por otra parte se tiene la *cantidad máxima de iteraciones* permitidas para cada ejecución del algoritmo, la cual también está asociada a un criterio de

[4]En términos del documento citado, la explosión de la trayectoria refiere al alejamiento de la media (tendiendo a infinito), de manera lineal cuando $\varphi = 4{,}0$ y exponencial cuando supera dicho valor.

parada. Sin embargo, este parámetro está mayormente relacionado al tiempo máximo tolerable que se desea ejecutar el algoritmo. Su valor (al igual que el caso del umbral) dependerá del problema concreto ya que determina el tiempo máximo que es factible o razonable esperar hasta obtener el resultado. Para la etapa de calibración se optó por utilizar una cantidad máxima de iteraciones = 100.

En último lugar, la *cantidad máxima de iteraciones sin mejoras* representa otro parámetro que establece una condición de parada. Dicho parámetro tiene un comportamiento análogo al parámetro anterior, pero su objetivo es controlar que el algoritmo no se haya estancado en óptimos locales, evitando así iteraciones innecesarias.

Es importante destacar que los parámetros *umbral* y *cantidad máxima de iteraciones sin mejoras* no serán tenidos en cuenta durante la etapa de calibración. Esta decisión se debe a que se desea contar con una cantidad de ejecuciones invariante y equitativa para todas las pruebas de calibración a ejecutar, de manera que los resultados obtenidos correspondan a contextos equivalentes y no se vean condicionados por estos factores. Por tal motivo no son indicados valores a utilizar para ellos en esta etapa.

Para finalizar, se presentan en la Tabla 5.2 los valores seleccionados para todos los parámetros mencionados (no calibrados), distinguiendo entre las dos variantes de PSO tratadas.

Parámetro	**Peso de Inercia**	**Factor de Constricción**
Tamaño de población	30	30
c_1	2,0	2,05
c_2	2,0	2,05
$N(v_{max})$	10	10
Política de inicialización de $\vec{v}_i$	$0_{n,1}$	$0_{n,1}$
Umbral	No aplica	No aplica
Cant. máx. iter.	100	100
Cant. máx. iter. sin mejoras	No aplica	No aplica
χ	No aplica	0,729

Tabla 5.2: Valores seleccionados de parámetros que no se calibrarán

A continuación se detallan los parámetros que se decidió calibrar para las dos variantes de PSO implementadas. Además, se brinda el detalle de los valores seleccionados como candidatos a utilizar en la etapa de calibración para cada uno de estos. La selección de parámetros a calibrar fue realizada en base al impacto que se considera estos tienen en el funcionamiento del algoritmo PSO.

La *topología del enjambre* determina el modo en que las partículas comparten información, además de definir la clase de PSO con la cual se trabaja.

Es por tal motivo que se considera de alto impacto en el rendimiento del algoritmo y que se decidió que forme parte del proceso de calibración.

Respecto a la decisión de los tipos de topologías seleccionadas, la misma se basó en primer término en cubrir las distintas clases de PSO, a saber *basada en* $\vec{p_g}$ y *basada en* $\vec{p_l}$. La primer clase determina el uso de la topología *Completa*, mientras que la segunda motivo a la selección de otras dos topologías representativas, *Von Neumman* y *Aleatoria 6*[5]. La topología *Von Neumman* es sugerida en diferentes trabajos como es el caso de [28], en donde Mendes y Kennedy afirman que la misma supera los resultados obtenidos por diferentes topologías que también formaron parte de dicho estudio. Por otro lado, la topología *Aleatoria 6* fue extraída de [19] por resultar atractiva ya que presenta un comportamiento no determinístico.

El siguiente parámetro seleccionado para la calibración es el *modo de actualización*. Aquí sólo existen dos valores posibles, *modo sincrónico* y *modo asincrónico*. Ambos modos son determinantes en la estructura del algoritmo y por ende en su funcionamiento por lo que resulta de interés que formen parte de este proceso.

Finalmente, el parámetro *peso de inercia (w)* resulta decisivo para el funcionamiento de la variante *PSO con peso de inercia* ya que es inherente a su definición. Por tal motivo también se tomó la decisión de incluirlo en el proceso de calibración. Respecto al rango de valores posibles para este parámetro, el trabajo [16] propone que w varíe en el rango $[0{,}4; 0{,}9]$. De este rango fueron seleccionados los valores candidatos 0,4, 0,7 y 0,9 con la intención de cubrir lo mejor posible al mismo.

La Tabla 5.3 resume los valores seleccionados para cada parámetro a calibrar así como también la inclusión de los mismos en las diferentes variantes tratadas.

Parámetro	**Peso de Inercia**	**Factor de Constricción**
Topología	[Completa, Aleatoria 6, Von Neumann]	[Completa, Aleatoria 6, Von Neumann]
Modo de actualización	[Sincrónico, Asincrónico]	[Sincrónico, Asincrónico]
Peso de inercia (w)	$[0{,}4; 0{,}7; 0{,}9]$	No aplica

Tabla 5.3: Valores seleccionados de parámetros a calibrar

La motivación de la cantidad de valores candidatos para cada uno de los parámetros obedece a la necesidad de reducir la cantidad de configuraciones de parámetros, ya que esta impacta directamente en la cantidad de ejecuciones requeridas para completar el proceso de calibración. Las pruebas iniciales del proceso de calibración revelaron tiempos de cómputo excesivos en computadoras personales. Por tal motivo, fue necesario reducir la cantidad de configuraciones además de escoger un hardware de mayor capacidad de

[5]Esta topología es un caso particular de la topología aleatoria, incorporando la restricción de que cada partícula cuenta con exactamente 6 vecinos.

cómputo (se trabajó en un nodo de un cluster que se detallará más adelante) de manera de lograr ejecutar la evaluación de todas las configuraciones en un tiempo razonable.

5.3. Detalle de la calibración de parámetros

Para la etapa de calibración se tomaron dos instancias diferentes del problema objetivo de este trabajo. Este punto se logra mediante la configuración de dos valores diferentes de la variable de cantidad de ciclos, dando lugar a una instancia con 5 ciclos y otra con 10. Mediante esta estrategia es posible modelar dos escenarios diferentes del mismo problema con el objetivo de que la calibración no quede vinculada a un único contexto. Otro punto a destacar es la simplificación del problema mediante la selección de valores de cantidad de ciclos pequeña para el motor, dado que los tiempos de ejecución crecen de manera muy significativa de acuerdo a este valor y no se desea invertir un tiempo demasiado prolongado en la etapa de calibración.

El proceso de calibración comienza por identificar las distintas configuraciones de parámetros calibrables, teniendo en cuenta las variantes de PSO a utilizar. De acuerdo a las decisiones realizadas en las secciones anteriores se cuenta con un total de 24 configuraciones de parámetros diferentes, 18 para la variante de *Peso de Inercia*[6] y 6 para *Factor de Constricción*[7].

La Tabla 5.4 y la Tabla 5.5, describen las configuraciones a utilizar para la calibración de la variante de peso de inercia y factor de constricción respectivamente.

Para cada uno de los posibles pares configuración - instancia[8] del problema serán realizadas tres ejecuciones independientes con el fin de recabar los datos relevantes. Por lo tanto, se deben realizar 72 ejecuciones para la instancia con 5 ciclos y 72 ejecuciones para la instancia con 10 ciclos.

El ambiente de ejecución donde se realiza la calibración se compone de un servidor con ocho procesadores Intel(R) Xeon(R) CPU E5530, cada uno con una velocidad de reloj de 1600 MHz, que dispone además de un total de memoria de 48 GB. En el servidor se dispone de un sistema operativo Linux CentOS versión 6.5.

[6]Este número surge de multiplicar la cantidad de valores seleccionados para peso de inercia, tipo de topología y modo de actualización (3, 3 y 2 respectivamente).

[7]Este número surge de multiplicar la cantidad de valores seleccionados para el tipo de topología y modo de actualización (3 y 2 respectivamente).

[8]Cada uno de los pares corresponde a una configuración para una variante y una instancia determinada.

Nro. config.	w	Topología	Modo de actualización
1	0.4	Completa	Sincrónico
2	0.4	Completa	Asincrónico
3	0.4	Aleatoria6	Sincrónico
4	0.4	Aleatoria6	Asincrónico
5	0.4	Von Neumann	Sincrónico
6	0.4	Von Neumann	Asincrónico
7	0.7	Completa	Sincrónico
8	0.7	Completa	Asincrónico
9	0.7	Aleatoria6	Sincrónico
10	0.7	Aleatoria6	Asincrónico
11	0.7	Von Neumann	Sincrónico
12	0.7	Von Neumann	Asincrónico
13	0.9	Completa	Sincrónico
14	0.9	Completa	Asincrónico
15	0.9	Aleatoria6	Sincrónico
16	0.9	Aleatoria6	Asincrónico
17	0.9	Von Neumann	Sincrónico
18	0.9	Von Neumann	Asincrónico

Tabla 5.4: Configuraciones - Peso de Inercia

Nro. config.	Topología	Modo de actualización
1	Completa	Sincrónico
2	Completa	Asincrónico
3	Aleatoria6	Sincrónico
4	Aleatoria6	Asincrónico
5	Von Neumann	Sincrónico
6	Von Neumann	Asincrónico

Tabla 5.5: Configuraciones - Factor de Constricción

Para estimar el tiempo total de ejecución de la calibración se midió el tiempo requerido para evaluar una solución. En el caso de la instancia de 5 ciclos el tiempo de evaluación de una solución fue del orden de 6 segundos, mientras que para la instancia de 10 ciclos tomó aproximadamente 10,1 segundos. Como se trabaja con una población de 30 partículas y se realizan 100 iteraciones por ejecución, el tiempo estimado para cada ejecución independiente es de aproximadamente 5 horas para la instancia de 5 ciclos ($100 * 30 * 6 = 18000$ segundos) y 8,5 horas para la de 10 ciclos ($100 * 30 * 10{,}1 = 30300$ segundos). A partir de la cantidad total de ejecuciones y el tiempo requerido por cada ejecución, es posible estimar el tiempo total del proceso de calibración. Esto es, 72 ejecuciones de 5 horas más otras 72 de 8,5 horas, resultando un total de tiempo de computo de 972 horas aproximadamente ($72 * 5 + 72 * 8{,}5$).

En base al cálculo del total de ejecuciones y el hardware disponible se opta por correr 8 lotes (uno por cada procesador) de 18 (144 ejecuciones / 8 procesadores) ejecuciones. Para cada lote de ejecución se utilizó un script con el formato mostrado en Código 5.1. De este modo, es posible reducir el tiempo de calibración a 121,5 horas.

Código 5.1: Script de ejemplo correspondiente al primer lote de calibración.

```
1 ./pso5 calib_w_5_1_1/parametros_w_5_1_1.cfg
2 ./pso5 calib_w_5_1_2/parametros_w_5_1_2.cfg
3 ./pso5 calib_w_5_1_3/parametros_w_5_1_3.cfg
4 ./pso5 calib_w_5_2_1/parametros_w_5_2_1.cfg
5                 ...
6 ./pso5 calib_w_5_18_1/parametros_w_5_18_1.cfg
7 ./pso5 calib_w_5_18_2/parametros_w_5_18_2.cfg
8 ./pso5 calib_w_5_18_3/parametros_w_5_18_3.cfg
```

Cada línea del script corresponde a una ejecución de un par configuración - instancia. Aquí *pso5* es el nombre del archivo binario correspondiente a la instancia del problema con 5 ciclos (análogamente existe un *pso10* asociado al caso de 10 ciclos) cuyo parámetro de entrada corresponde a la ruta del archivo de configuración asociado a dicha ejecución. El nombre del archivo de configuración y el directorio donde este reside está conformado por la concatenación de un identificador de la variante, cantidad de ciclos de la instancia, número de configuración y número de ejecución en el orden mencionado.

El script fue realizado de esta manera con el objetivo de desfasar las diferentes ejecuciones, evitando así la generación de una misma semilla en el generador de números aleatorios[9] lo que causaría ejecuciones con idénticos resultados.

[9]El algoritmo de PSO implementado utiliza un generador de números aleatorios cuya semilla se construye a partir de la hora del sistema.

5.4. Resultados de la calibración

Para cada ejecución se considera el mejor fitness encontrado para el par configuración - instancia. En base a la información obtenida por las 3 ejecuciones de cada par, se calcula el fitness promedio y el mejor fitness. El cálculo de dichos valores permite posteriormente determinar el gap al mejor fitness de la instancia. Este valor representa el porcentaje de diferencia que hay entre el promedio de los mejores fitness de cada par y el mejor fitness obtenido por alguno de los pares. El modo en que este valor se calcula es expresado como:

$$Gap\ al\ mejor\ fitness = 100 \times \left(\frac{Promedio\ de\ los\ mejores\ fitness}{Mejor\ fitness\ de\ los\ pares} - 1 \right) \quad (5.3)$$

El objetivo de usar este valor es brindar un criterio de selección no sesgado puramente por los mejores valores de fitness, sino ponderando las instancias de acuerdo a la tasa de error de cada una de estas.

La Tabla 5.6 y la Tabla 5.7, resumen los resultados obtenidos para la etapa de calibración de la variante de peso de inercia y factor de constricción respectivamente. Los mejores valores de cada métrica están resaltados en negrita en dichas tablas. Es importante mencionar que el detalle completo de los resultados puede ser visto en el Apéndice A, en la Tabla A.1 y la Tabla A.2.

En la Tabla 5.6 y la Tabla 5.7 también se presenta el gap promedio. Este valor se calcula como el promedio de los gaps al mejor fitness de cada instancia para cada una de las distintas configuraciones. Una vez calculados todos los gaps promedio, es posible visualizar la configuración que en promedio se aparta menos de la mejor solución encontrada para las dos instancias.

El criterio de selección para la configuración de parámetros a utilizar en la etapa de evaluación para cada variante se basa en elegir aquella cuyo gap promedio entre las dos instancias sea menor. En base a este criterio, la configuración seleccionada para la variante de peso de inercia es la número 1, lo que implica un valor de $w = 0{,}4$, usar la topología Completa y un modo de actualización sincrónico. Mientras que para la variante de factor de constricción se optó por la configuración número 1, la cual supone utilizar la topología Completa y un modo de actualización sincrónico.

Es importante destacar que se podría haber tomado otro criterio para la selección, como es el caso de optar por la configuración que además de presentar un gap promedio bajo, cumpla que los gaps al mejor fitness de cada una de las instancias sean cercanos al promedio. A modo de ejemplo, si se hubiera considerado este criterio en la variante de peso de inercia, se hubiera optado por la configuración número 3 en lugar de la 1, ya que para

la misma, sus gaps al mejor fitness para cada una de las instancias se alejan menos del gap promedio.

Nro. config.	Instancia 5 ciclos			Instancia 10 ciclos			Gap promedio (%)
	Fitness promedio	Mejor fitness	Gap al mejor (%)	Fitness promedio	Mejor fitness	Gap al mejor (%)	
1	$5{,}797 \times 10^{-3}$	$5{,}711 \times 10^{-3}$	5,18	$\mathbf{7{,}730 \times 10^{-3}}$	$\mathbf{7{,}599 \times 10^{-3}}$	**1,72**	**3,45**
2	$5{,}710 \times 10^{-3}$	$\mathbf{5{,}512 \times 10^{-3}}$	3,61	$8{,}121 \times 10^{-3}$	$7{,}891 \times 10^{-3}$	6,88	5,24
3	$5{,}777 \times 10^{-3}$	$5{,}616 \times 10^{-3}$	4,82	$7{,}782 \times 10^{-3}$	$7{,}750 \times 10^{-3}$	2,41	3,61
4	$5{,}777 \times 10^{-3}$	$5{,}703 \times 10^{-3}$	4,82	$8{,}232 \times 10^{-3}$	$8{,}006 \times 10^{-3}$	8,33	6,58
5	$5{,}762 \times 10^{-3}$	$5{,}689 \times 10^{-3}$	4,54	$8{,}113 \times 10^{-3}$	$7{,}794 \times 10^{-3}$	6,76	5,65
6	$\mathbf{5{,}611 \times 10^{-3}}$	$5{,}570 \times 10^{-3}$	**1,80**	$8{,}094 \times 10^{-3}$	$7{,}928 \times 10^{-3}$	6,52	4,16
7	$5{,}753 \times 10^{-3}$	$5{,}559 \times 10^{-3}$	4,39	$7{,}864 \times 10^{-3}$	$7{,}756 \times 10^{-3}$	3,49	3,94
8	$6{,}187 \times 10^{-3}$	$6{,}041 \times 10^{-3}$	12,25	$8{,}432 \times 10^{-3}$	$8{,}262 \times 10^{-3}$	10,97	11,61
9	$5{,}994 \times 10^{-3}$	$5{,}810 \times 10^{-3}$	8,75	$8{,}293 \times 10^{-3}$	$8{,}037 \times 10^{-3}$	9,13	8,94
10	$6{,}031 \times 10^{-3}$	$5{,}822 \times 10^{-3}$	9,43	$8{,}174 \times 10^{-3}$	$8{,}038 \times 10^{-3}$	7,56	8,50
11	$5{,}960 \times 10^{-3}$	$5{,}695 \times 10^{-3}$	8,14	$8{,}244 \times 10^{-3}$	$7{,}942 \times 10^{-3}$	8,50	8,32
12	$5{,}971 \times 10^{-3}$	$5{,}755 \times 10^{-3}$	8,35	$8{,}549 \times 10^{-3}$	$8{,}297 \times 10^{-3}$	12,51	10,43
13	$6{,}026 \times 10^{-3}$	$5{,}951 \times 10^{-3}$	9,33	$8{,}260 \times 10^{-3}$	$8{,}119 \times 10^{-3}$	8,70	9,01
14	$6{,}123 \times 10^{-3}$	$5{,}969 \times 10^{-3}$	11,10	$8{,}211 \times 10^{-3}$	$7{,}964 \times 10^{-3}$	8,06	9,58
15	$6{,}150 \times 10^{-3}$	$6{,}091 \times 10^{-3}$	11,59	$8{,}188 \times 10^{-3}$	$7{,}877 \times 10^{-3}$	7,75	9,67
16	$6{,}100 \times 10^{-3}$	$5{,}979 \times 10^{-3}$	10,68	$8{,}228 \times 10^{-3}$	$8{,}058 \times 10^{-3}$	8,29	9,48
17	$6{,}014 \times 10^{-3}$	$5{,}931 \times 10^{-3}$	9,11	$8{,}230 \times 10^{-3}$	$8{,}067 \times 10^{-3}$	8,30	8,71
18	$6{,}071 \times 10^{-3}$	$5{,}943 \times 10^{-3}$	10,14	$8{,}529 \times 10^{-3}$	$8{,}451 \times 10^{-3}$	12,25	11,19

Tabla 5.6: Resultados Calibración - Peso de Inercia

Nro. config.	Instancia 5 ciclos			Instancia 10 ciclos			Gap promedio (%)
	Fitness promedio	Mejor fitness	Gap al mejor (%)	Fitness promedio	Mejor fitness	Gap al mejor (%)	
1	$5{,}663 \times 10^{-3}$	$5{,}609 \times 10^{-3}$	1,41	$\mathbf{7{,}809 \times 10^{-3}}$	$7{,}745 \times 10^{-3}$	**1,16**	**1,29**
2	$6{,}117 \times 10^{-3}$	$5{,}926 \times 10^{-3}$	9,53	$8{,}222 \times 10^{-3}$	$8{,}015 \times 10^{-3}$	6,51	8,02
3	$5{,}755 \times 10^{-3}$	$\mathbf{5{,}584 \times 10^{-3}}$	3,05	$8{,}048 \times 10^{-3}$	$8{,}028 \times 10^{-3}$	4,27	3,66
3	$5{,}858 \times 10^{-3}$	$5{,}824 \times 10^{-3}$	4,91	$8{,}176 \times 10^{-3}$	$\mathbf{7{,}719 \times 10^{-3}}$	5,92	5,42
5	$\mathbf{5{,}657 \times 10^{-3}}$	$5{,}589 \times 10^{-3}$	**1,30**	$7{,}991 \times 10^{-3}$	$7{,}820 \times 10^{-3}$	3,52	2,41
6	$5{,}834 \times 10^{-3}$	$5{,}774 \times 10^{-3}$	4,47	$8{,}454 \times 10^{-3}$	$8{,}275 \times 10^{-3}$	9,53	7,00

Tabla 5.7: Resultados Calibración - Factor de Constricción

Capítulo 6

Evaluación de los algoritmos propuestos

Evaluation is creation: hear it, you creators! Evaluating is itself the most valuable treasure of all that we value. It is only through evaluation that value exists: and without evaluation the nut of existence would be hollow. Hear it, you creators!

Friedrich Wilhelm Nietzsche

El objetivo de este capítulo consiste en evaluar el funcionamiento de los algoritmos PSO diseñados para la reducción de la variabilidad cíclica en motores de combustión interna. Con tal fin, se exponen los resultados de un conjunto de experimentos conducidos para obtener información acerca del desempeño de cada uno de estos. Dichos experimentos incluyen también la evaluación del algoritmo RS (Random Search) con el objetivo de comparar el desempeño entre ambos algoritmos.

6.1. Estructura del capítulo

En primer lugar se detallan brevemente los experimentos que serán realizados, mostrando el detalle de los datos de configuración y ambiente de ejecución que se utilizará para las pruebas. Posteriormente, se destina una sección a explicar cada experimento, incluyendo el detalle del procedimiento a ser realizado, los resultados obtenidos luego de su ejecución, observaciones y conclusiones pertinentes.

6.2. Detalle de la evaluación

El objetivo de esta etapa es evaluar el desempeño numérico de los algoritmos PSO propuestos. Con tal fin, como ya se mencionó en el Capítulo 4, se compararán los algoritmos *PSO con peso de inercia*, *PSO con factor de constricción* y *Random Search* sobre diferentes instancias del problema de reducción de variabilidad cíclica. También se tienen en cuenta en la evaluación los valores de fitness asociados a los valores de referencia para los parámetros a optimizar provistos por Pedro Curto para la medición de la calidad de los resultados. Estos valores de fitness son detallados en la Tabla 6.1, mientras que los valores de referencia ya fueron presentados en la Tabla 4.2.

Cant. ciclos	**Fitness**	**Tiempo(s)**
5	7.366×10^{-3}	6,15
10	1.026×10^{-2}	11,47
15	9.850×10^{-3}	16,64
20	1.326×10^{-2}	22,19
25	1.319×10^{-2}	28,43
50	1.120×10^{-2}	50,56

Tabla 6.1: Valores de referencia para la evaluación

La evaluación experimental se compone de tres experimentos distintos, el primero de ellos considera un conjunto de instancias del problema que poseen una baja cantidad de ciclos. El segundo enfocado en evaluar el desempeño sobre una instancia con una cantidad de ciclos alta y el tercero que estudia la influencia del aumento del número de iteraciones en los resultados.

La configuración usada por los algoritmos PSO es distinta para cada variante y corresponde a la obtenida en la etapa de calibración. Los detalles de estas configuraciones se muestran en la Tabla 6.2 y Tabla 6.3. El algoritmo *Random Search* carece de configuración. De todas formas, conviene señalar que en todos los experimentos realizados los algoritmos PSO construyen la misma cantidad de soluciones que el algoritmo RS.

Es importante destacar que la cantidad máxima de iteraciones dependerá del experimento, por tal motivo se muestra el conjunto de valores posible a utilizar.

6.2.1. Primer experimento

En este primer experimento, se considera el uso de diferentes instancias del problema, considerando para ello un motor de 5, 10, 15, 20 y 25 ciclos. Al igual que en la etapa de calibración, tiene el objetivo de brindar un criterio de comparación de resultados no sesgado a algún escenario particular del problema.

Parámetro	Valor
Tamaño de población	30
c_1	2,0
c_2	2,0
w	0,4
Topología	Completa
Modo de actualización	Sincrónico
$N(v_{max})$	10
Política de inicialización de $\vec{v}_i$	$0_{n,1}$
Umbral	No aplica
Cant. máx. iter.	100 o 200
Cant. máx. iter. sin mejoras	No aplica

Tabla 6.2: Configuración de PSO con Peso de Inercia

Parámetro	Valor
Tamaño de población	30
c_1	2,05
c_2	2,05
Topología	Completa
Modo de actualización	Sincrónico
$N(v_{max})$	10
Política de inicialización de $\vec{v}_i$	$0_{n,1}$
Umbral	No aplica
Cant. máx. iter.	100 o 200
Cant. máx. iter. sin mejoras	No aplica
χ	0,729

Tabla 6.3: Configuración de PSO con Factor de Constricción

Es importante destacar que a diferencia de la etapa de calibración, donde se buscaba trabajar sobre escenarios chicos de forma de no aumentar los tiempos de ejecución significativamente, en este caso resulta de mayor interés hacer una evaluación más centrada en la calidad de los resultados obtenidos. Por este motivo, algunas de las instancias seleccionadas intentan modelar una cantidad de ciclos más realistas del motor.

Como se puede apreciar en la Tabla 6.2 y Tabla 6.3, ambos algoritmos PSO utilizan una población de 30 partículas y realizan 100 iteraciones por ejecución. Esto se traduce en un total de 3000 (30×100) evaluaciones de la simulación del motor, por lo que el algoritmo *Random Search* itera 3000 veces para igualar el número de evaluaciones y de esta manera lograr una comparación justa entre los algoritmos.

Cada par algoritmo - instancia ($3 \times 5 = 15$ en total) será ejecutado ocho veces para eliminar posibles sesgos en resultados de ejecuciones particulares. Esto produce un total de 120 (15×8) ejecuciones, las cuales se realizarán sobre el mismo hardware utilizado durante la etapa de calibración.

La estimación del tiempo total de ejecución para la etapa de evaluación se realiza de manera similar a como fuera realizada en la etapa de calibración, esto es, en base a los tiempos de evaluación de una única solución para las diferentes instancias del problema. Aquí, los tiempos a utilizar son los detallados en la Tabla 6.1, los cuales corresponden a las distintas evaluaciones de dicha solución, adoptando esta los valores de referencia presentados en la Tabla 4.2. Esto es, se requerirán aproximadamente 6,1 seg, 11,5 seg, 16,6 seg, 22,2 seg y 28,4 seg para las instancias de 5, 10, 15, 20 y 25 ciclos, respectivamente.

Dado que se realizan 3000 evaluaciones por ejecución y abstrayéndose de los algoritmos subyacentes[1], es posible estimar el tiempo requerido para cada una de las diferentes instancias. De esta manera, se cuenta con un total de 5,1 ($3000 \times 6{,}1$ segundos), 9,6 ($3000 \times 11{,}5$ segundos), 13,8 ($3000 \times 16{,}6$ segundos), 18,5 ($3000 \times 22{,}2$ segundos) y 23,7 ($3000 \times 28{,}4$ segundos) horas para los casos de 5, 10, 15, 20 y 25 ciclos, respectivamente. A partir de la cantidad total de ejecuciones y el tiempo requerido por cada una de estas, es posible estimar el tiempo total del proceso de evaluación, resultando en aproximadamente un tiempo total de computo de 1696,8 horas (calculadas como: $24 \times 5{,}1 + 24 \times 9{,}6 + 24 \times 13{,}8 + 24 \times 18{,}5 + 24 \times 23{,}7$).

Basados en el número total de ejecuciones y en el hardware disponible, se opta por correr ocho lotes (uno por cada procesador) de 15 (120 ejecuciones / 8 procesadores) ejecuciones. Para cada lote de ejecución se utilizó un script

[1]Para la estimación de tiempos de cada par algoritmo - instancia, se hace foco únicamente en los tiempos asociados a la instancia, desestimando las diferencias de tiempo inherentes a cada algoritmo.

con el formato mostrado en Código 6.1. De este modo, es posible reducir el tiempo de de ejecución del experimento a aproximadamente 212,1 horas (1696,8 horas / 8 procesadores).

Código 6.1: Script de ejemplo correspondiente al primer lote de evaluación.

```
./pso5 eval_w_5_1/parametros_w_5_1.cfg
./pso10 eval_w_10_1/parametros_w_10_1.cfg
./pso15 eval_w_15_1/parametros_w_15_1.cfg
./pso20 eval_w_20_1/parametros_w_20_1.cfg
./pso25 eval_w_25_1/parametros_w_25_1.cfg
./pso5 eval_x_5_1/parametros_x_5_1.cfg
./pso10 eval_x_10_1/parametros_x_10_1.cfg
./pso15 eval_x_15_1/parametros_x_15_1.cfg
./pso20 eval_x_20_1/parametros_x_20_1.cfg
./pso25 eval_x_25_1/parametros_x_25_1.cfg
./rs5 eval_rs_5_1/salida.txt
./rs10 eval_rs_10_1/salida.txt
./rs15 eval_rs_15_1/salida.txt
./rs20 eval_rs_20_1/salida.txt
./rs25 eval_rs_25_1/salida.txt
```

Cada línea del script corresponde a la ejecución de un par algoritmo - instancia. Existen dos tipos posibles para estas, el primero asociado a la ejecución de algoritmos PSO y el segundo asociado al algoritmo RS. Dentro del primer tipo, las líneas ejecutan el binario correspondiente a un par algoritmo - instancia de PSO, tal como pso5, pso10, pso15, pso20 y pso25 para 5, 10, 15, 20 y 25 ciclos, respectivamente. Cada uno de los binarios recibe como parámetro de entrada la ruta al archivo de configuración necesario, el cual determina principalmente la variante a ejecutar. El nombre del archivo de configuración y el directorio donde este reside está conformado por la concatenación de un identificador de la variante, cantidad de ciclos de la instancia y número de ejecución en el orden mencionado.

Por otra parte, el segundo tipo de línea contiene el binario asociado al par algoritmo - instancia de RS. Estos binarios pueden ser rs5, rs10, rs15, rs20 y rs25 para 5, 10, 15, 20 y 25 ciclos, respectivamente. También cuentan con un parámetro de entrada correspondiente a la ruta del archivo de salida de dicha ejecución. El nombre del directorio donde este reside se conforma de la concatenación del nombre del algoritmo, la cantidad de ciclos de la instancia y el número de ejecución en el orden mencionado.

Ciclos	Experto	Random Search		PSO con Peso de Inercia			
		Promedio	Desviación Estándar	Promedio	Desviación Estándar	Mejora Experto (%)	Mejora RS (%)
5	$7{,}366 \times 10^{-3}$	$6{,}242 \times 10^{-3}$	$1{,}603 \times 10^{-4}$	$5{,}724 \times 10^{-3}$	$1{,}241 \times 10^{-4}$	22,29	8,29
10	$1{,}026 \times 10^{-2}$	$8{,}444 \times 10^{-3}$	$2{,}671 \times 10^{-4}$	$7{,}836 \times 10^{-3}$	$2{,}334 \times 10^{-4}$	23,66	7,20
15	$9{,}850 \times 10^{-3}$	$8{,}544 \times 10^{-3}$	$2{,}606 \times 10^{-4}$	$7{,}997 \times 10^{-3}$	$1{,}638 \times 10^{-4}$	18,81	6,40
20	$1{,}326 \times 10^{-2}$	$1{,}001 \times 10^{-2}$	$2{,}139 \times 10^{-4}$	$9{,}683 \times 10^{-3}$	$1{,}286 \times 10^{-4}$	26,99	3,22
25	$1{,}319 \times 10^{-2}$	$9{,}564 \times 10^{-3}$	$1{,}655 \times 10^{-4}$	$9{,}073 \times 10^{-3}$	$1{,}421 \times 10^{-4}$	31,21	5,13

Tabla 6.4: Promedios de los resultados para PSO con Peso de Inercia (5 a 25 ciclos)

Ciclos	Experto	Random Search		PSO con Factor de Constricción			
		Promedio	Desviación Estándar	Promedio	Desviación Estándar	Mejora Experto (%)	Mejora RS (%)
5	$7{,}366 \times 10^{-3}$	$6{,}242 \times 10^{-3}$	$1{,}603 \times 10^{-4}$	$5{,}714 \times 10^{-3}$	$1{,}956 \times 10^{-4}$	22,43	8,45
10	$1{,}026 \times 10^{-2}$	$8{,}444 \times 10^{-3}$	$2{,}671 \times 10^{-4}$	$7{,}806 \times 10^{-3}$	$1{,}913 \times 10^{-4}$	23,96	7,56
15	$9{,}850 \times 10^{-3}$	$8{,}544 \times 10^{-3}$	$2{,}606 \times 10^{-4}$	$7{,}978 \times 10^{-3}$	$1{,}821 \times 10^{-4}$	19,01	6,63
20	$1{,}326 \times 10^{-2}$	$1{,}001 \times 10^{-2}$	$2{,}139 \times 10^{-4}$	$9{,}600 \times 10^{-3}$	$2{,}218 \times 10^{-4}$	27,62	4,05
25	$1{,}319 \times 10^{-2}$	$9{,}564 \times 10^{-3}$	$1{,}655 \times 10^{-4}$	$9{,}118 \times 10^{-3}$	$1{,}972 \times 10^{-4}$	30,87	4,66

Tabla 6.5: Promedios de los resultados para PSO con Factor de Constricción (5 a 25 ciclos)

Ciclos	Experto	Random Search	PSO con Peso de Inercia		
			Mejor valor	Mejora Experto (%)	Mejora RS (%)
5	$7{,}366 \times 10^{-3}$	$6{,}058 \times 10^{-3}$	$5{,}573 \times 10^{-3}$	24,35	8,01
10	$1{,}026 \times 10^{-2}$	$8{,}077 \times 10^{-3}$	$7{,}599 \times 10^{-3}$	25,97	5,92
15	$9{,}850 \times 10^{-3}$	$8{,}125 \times 10^{-3}$	$7{,}750 \times 10^{-3}$	21,32	4,61
20	$1{,}326 \times 10^{-2}$	$9{,}764 \times 10^{-3}$	$9{,}539 \times 10^{-3}$	28,08	2,31
25	$1{,}319 \times 10^{-2}$	$9{,}249 \times 10^{-3}$	$8{,}809 \times 10^{-3}$	33,21	4,76

Tabla 6.6: Mejores valores para PSO con Peso de Inercia (5 a 25 ciclos)

Ciclos	Experto	Random Search	PSO con Factor de Constricción		
			Mejor valor	Mejora Experto (%)	Mejora RS (%)
5	$7{,}366 \times 10^{-3}$	$6{,}058 \times 10^{-3}$	$5{,}517 \times 10^{-3}$	25,10	8,92
10	$1{,}026 \times 10^{-2}$	$8{,}077 \times 10^{-3}$	$7{,}590 \times 10^{-3}$	26,06	6,03
15	$9{,}850 \times 10^{-3}$	$8{,}125 \times 10^{-3}$	$7{,}791 \times 10^{-3}$	20,90	4,11
20	$1{,}326 \times 10^{-2}$	$9{,}764 \times 10^{-3}$	$9{,}337 \times 10^{-3}$	29,60	4,37
25	$1{,}319 \times 10^{-2}$	$9{,}249 \times 10^{-3}$	$8{,}749 \times 10^{-3}$	33,66	5,41

Tabla 6.7: Mejores valores para PSO con Factor de Constricción (5 a 25 ciclos)

Una vez presentados los detalles del primer experimento, se pasan a presentar los resultados obtenidos en el mismo. La información completa de las ejecuciones puede verse en detalle en la Tabla A.3, la Tabla A.4 y la Tabla A.5 del Apéndice A. Es importante destacar que para cada ejecución se considera únicamente el mejor fitness encontrado para el par algoritmo - instancia. En base a los resultados obtenidos para las ocho ejecuciones de cada par, se calcula el fitness promedio, la desviación estándar y el mejor fitness obtenido. A partir de estos datos se construye la Tabla 6.4, la cual compara el fitness promedio del algoritmo PSO con Peso de Inercia respecto al fitness promedio obtenido por el algoritmo RS, así como también con el fitness asociado a los valores de referencia del experto. Estas comparaciones son brindadas mediante dos columnas que detallan los porcentajes de mejora correspondientes a cada caso. Análogamente, se brinda en la Tabla 6.5 el detalle de los porcentajes de mejoras para PSO con Factor de Constricción.

Posteriormente, se detalla en la Tabla 6.6 la comparación de los mejores valores de fitness del algoritmo PSO con Peso de Inercia respecto a los mejores valores de fitness del algoritmo RS, así como también los fitness asociados a los valores de referencia. Dicha comparación se muestra mediante los porcentajes de mejora correspondientes a cada caso. Análogamente, se detalla en la Tabla 6.7 la comparación correspondiente al algoritmo PSO con Factor de Constricción.

Para ilustrar de mejor manera las comparaciones realizadas en la Tabla 6.4 y Tabla 6.5, se presenta la Figura 6.1 y la Figura 6.2 que resumen los resultados de los porcentajes de mejora de los promedios de fitness para PSO con Peso de Inercia y PSO con Factor de Constricción, respectivamente.

Análogamente, se ilustra en la Figura 6.3 y la Figura 6.4 los porcentajes de mejora para los mejores fitness de PSO con Peso de Inercia y PSO con Factor de Constricción, cuyos valores fueran detallados en la Tabla 6.6 y Tabla 6.7, respectivamente.

Analizando el comportamiento de los resultados obtenidos que muestran las gráficas se pueden formular algunas conclusiones. En primer lugar, es importante resaltar que ambas variantes de PSO superan tanto en promedio como en el mejor valor de fitness a los resultados obtenidos por RS y los valores del experto, para cada una de las instancias analizadas.

En segundo lugar, para la comparación de los fitness promedio de ambas variantes de PSO, se aprecia un aumento en los porcentajes de mejora contra los valores de referencia en función de los ciclos, con la excepción de la instancia de 15 ciclos. De manera contraria, se observa una reducción en los porcentajes de mejora respecto a la variante RS, a excepción de la instancia de 20 ciclos.

Por otra parte, en la comparación de los mejores fitness de ambas variantes de PSO, se aprecia un incremento respecto a los valores de referencia en función de la instancia, a excepción del caso de 15 ciclos. Para la compara-

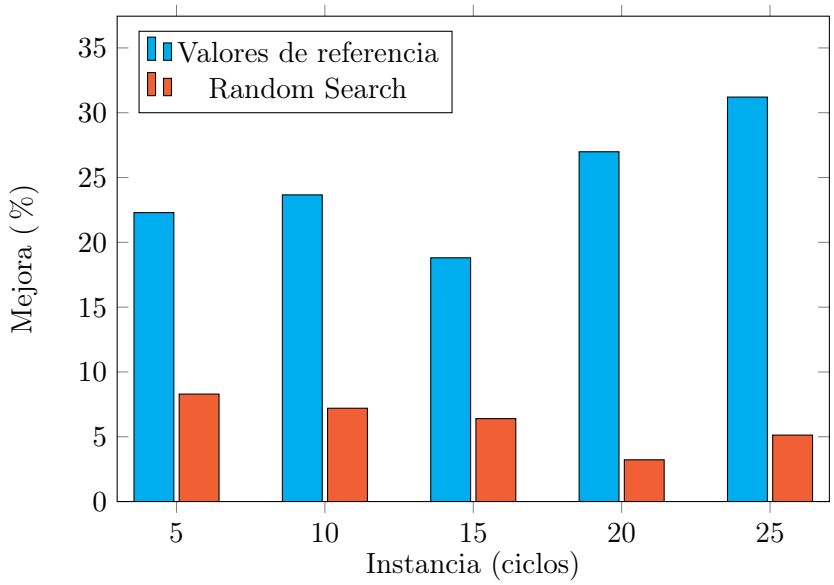


Figura 6.1: Comparación de mejora en fitness promedio para PSO con Peso de Inercia (5 a 25 ciclos)

ción con la variante RS, se observa una reducción inicial en la mejora y un leve crecimiento en las últimas instancias.

Como conclusión general, se obtienen altos y crecientes porcentajes de mejora de ambas variantes de PSO respecto a los valores provistos por el experto. Mientras que contra RS, no son alcanzados porcentajes tan significativos de mejora, decreciendo en general para instancias con un mayor número de ciclos.

Para finalizar el análisis del primer experimento, se estudia la evolución de los mejores fitness encontrados hasta el momento en cada iteración para cada uno de los diferentes algoritmos. El objetivo es mostrar la forma en que estos algoritmos convergen y así detectar posibles puntos de estancamiento de los mismos.

Para esto, se realizó una única ejecución por algoritmo seleccionando la instancia de 25 ciclos como caso representativo, con una cantidad de 100 iteraciones para los algoritmos PSO y 3000 soluciones construidas para el algoritmo RS.

La Figura 6.5 muestra en escala logarítmica la evolución completa del mejor fitness registrado por cada cada uno de los algoritmos. Por otro lado, la Figura 6.6 muestra una versión reducida de la evolución del mejor fitness en escala logarítmica con el objetivo de visualizar el patrón de mejora, excluyendo para esto el salto inicial y el estancamiento final de cada uno de los algoritmos.

Con el fin de registrar la evolución de los diferentes algoritmos, se reali-

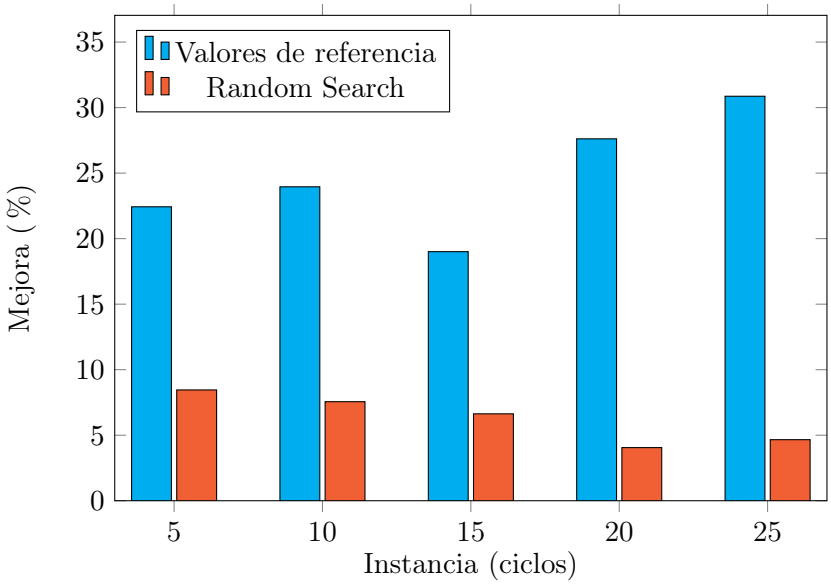


Figura 6.2: Comparación de mejora en fitness promedio para PSO con Factor de Constricción (5 a 25 ciclos)

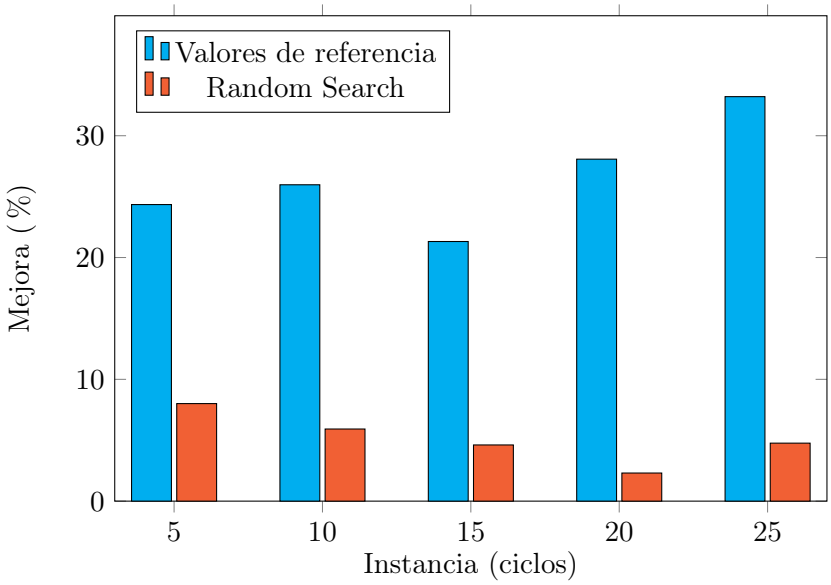


Figura 6.3: Comparación de mejora en los mejores fitness para PSO con Peso de Inercia (5 a 25 ciclos)

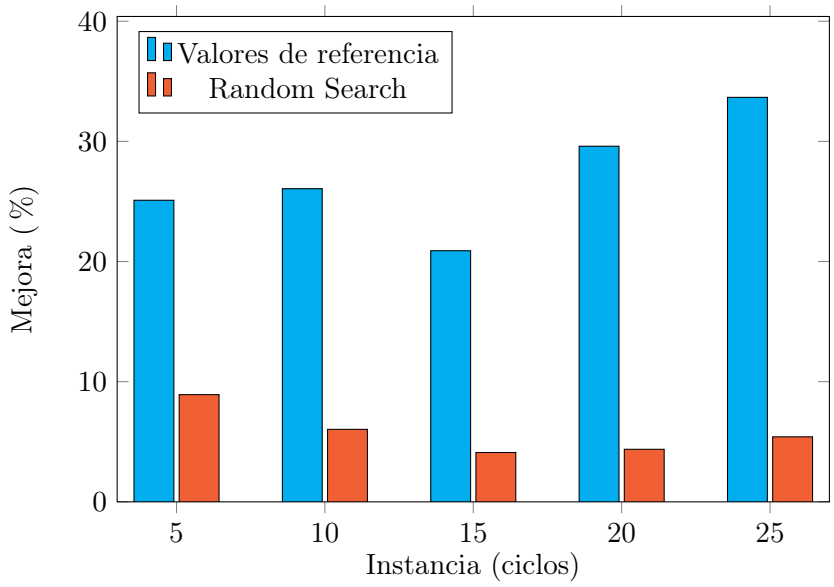


Figura 6.4: Comparación de mejora en los mejores fitness para PSO con Factor de Constricción (5 a 25 ciclos)

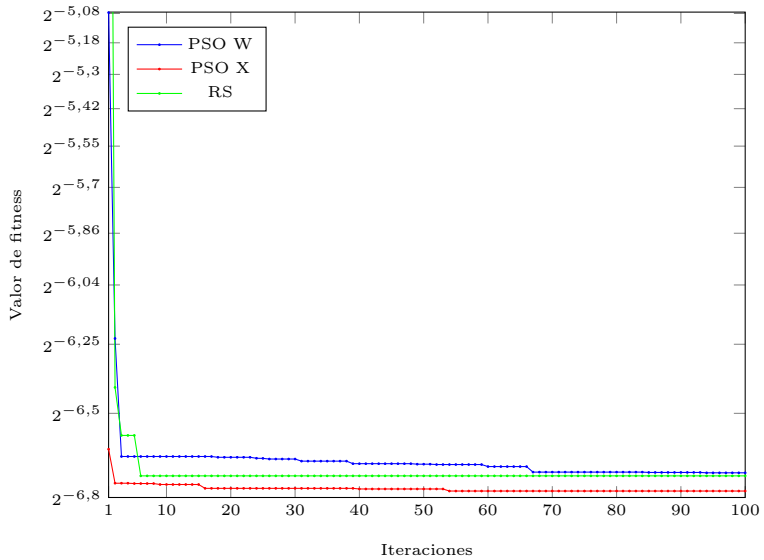


Figura 6.5: Evolución de fitness

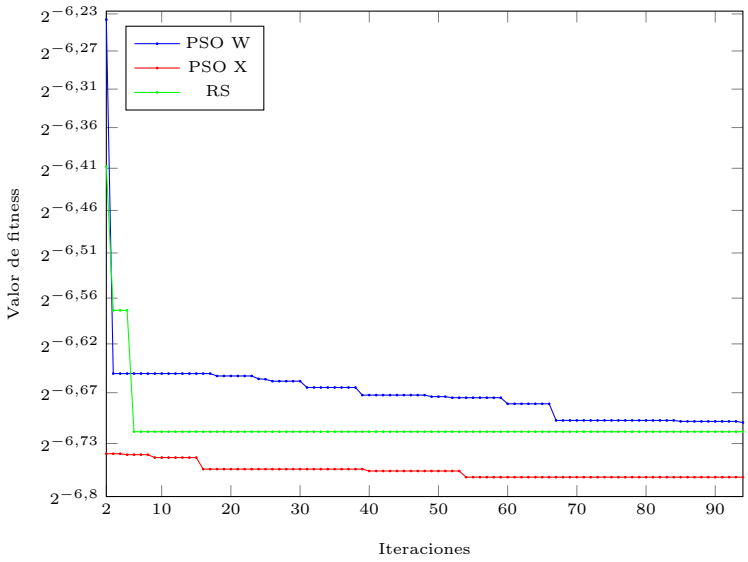


Figura 6.6: Evolución de fitness - Intervalo de patrón de mejora

zarán 100 mediciones para cada uno de estos. Para el caso de PSO se toma una medición por iteración mientras que para el caso de RS las mismas se agrupan cada 30 soluciones construidas para que sea comparable.

La primera observación que se desprende acerca de la forma de las gráficas es que en los tres casos se percibe una importante mejora al inicio de dichas ejecuciones. Esto responde probablemente a que el fitness obtenido con la posición aleatoria inicial es fácilmente superable en las primeras iteraciones de los distintos algoritmos.

Es importante destacar que estas gráficas no brindan información acerca del nivel de exploración y explotación del espacio de búsqueda, dado que no hay datos de la posición en el dominio sino únicamente del fitness obtenido. Sin embargo, es posible realizar algunas conjeturas al respecto.

Para la gráfica asociada al algoritmo PSO con Peso de Inercia, se observan períodos de estancamiento medianos, registrándose un crecimiento lento pero casi continuo a lo largo de las iteraciones. El mejor valor de fitness se encuentra en la iteración 94. Esto puede deberse a un buen nivel de exploración del espacio de búsqueda.

Para el caso de la gráfica de PSO con Factor de Constricción, se destacan dos fases diferentes. En la primera, se producen estancamientos pequeños pero con mejoras más importantes en comparación al algoritmo PSO con Peso de Inercia. Sin embargo, en la segunda fase se aprecia que la mejor solución encontrada se produce aproximadamente en la mitad de la ejecución (iteración 54), estancándose así hasta el final de la misma. Este comportamiento puede deberse a que este algoritmo cuenta con un alto nivel de explotación. Nótese que si bien se producen estancamientos en la mejora del fitness, esto

no permite asegurar que dicho algoritmo no exploró otras zonas del dominio.

Observando la gráfica del algoritmo RS, se puede notar que en la misma se da un estancamiento de la mejora de fitness muy temprano y prolongado a lo largo de la ejecución. Esto probablemente obedezca a la naturaleza aleatoria de dicho algoritmo, dado que este encontró su mejor solución al inicio (sexta medición) y por carecer de un mecanismo de explotación nunca logró superarla, más allá de la capacidad de exploración del espacio de búsqueda del mismo.

6.2.2. Segundo experimento

El objetivo del segundo experimento se focaliza en extender el primer experimento, agregando una instancia con una cantidad bastante más significativa de ciclos (50) para el motor, de manera de analizar el comportamiento de los algoritmos en función de las diferentes instancias. Esto permitirá posteriormente visualizar la evolución del rendimiento de los algoritmos en función de los ciclos. Para ser consistentes con el primer experimento se realizarán ocho ejecuciones de cada algoritmo para la instancia de 50 ciclos.

La Tabla 6.8 resume los resultados de mejor fitness, fitness promedio, desviación estándar y tiempos promedio de ejecución de los diferentes algoritmos para las instancias de 25 y 50 ciclos. La misma tiene el objetivo de reunir la información necesaria para la posterior comparación entre dichas instancias, mostrando el salto que ocurre al pasar de una instancia a la otra.

Por otro lado, la Tabla 6.9 realiza la comparación de los resultados de la Tabla 6.8, mostrando los porcentajes de mejora de los algoritmos PSO respecto a los valores de referencia y RS para las instancias de 25 y 50 ciclos.

Dado que la evolución de las mejoras en función de las instancias fue analizada en el primer experimento, aquí solo se estudia el comportamiento entre las instancias de 25 y 50 ciclos.

Como primera observación, se destaca que para ambas variantes de PSO se registra un leve decrecimiento en el porcentaje de mejora en el cambio de 25 a 50 ciclos, en términos del valor de fitness promedio, tanto contra los valores de experto así como también el algoritmo RS. Además, se pudo percibir un aumento casi proporcional del tiempo de ejecución de los algoritmos en función de la cantidad de ciclos de la instancia. Es importante destacar que empíricamente se verifica este comportamiento, más allá de que las ejecuciones no contaron con la exclusividad del hardware utilizado[2].

[2]La falta de exclusividad refiere a la existencia de otros procesos ejecutando en el mismo hardware y compitiendo por dichos recursos. Este aspecto, claramente hace que los tiempos de ejecución estimados difieran de los obtenidos.

Algoritmo	Ciclos	Fitness promedio	Desviación Estándar	Mejor fitness	Tiempo promedio (s)
PSO W	25	9.073×10^{-3}	1.421×10^{-4}	8.809×10^{-3}	79 762,37
PSO W	50	7.915×10^{-3}	1.827×10^{-4}	7.750×10^{-3}	160 686,83
PSO X	25	9.118×10^{-3}	1.972×10^{-4}	8.749×10^{-3}	83 087,35
PSO X	50	7.829×10^{-3}	1.383×10^{-4}	7.668×10^{-3}	155 473,29
RS	25	9.564×10^{-3}	1.655×10^{-4}	9.249×10^{-3}	86 960,78
RS	50	8.215×10^{-3}	1.355×10^{-4}	7.912×10^{-3}	172 218,03

Tabla 6.8: Fitness promedio, mejor fitness y tiempo promedio para 25 y 50 ciclos.

Algoritmo	Ciclos	Experto	Fitness promedio R.S	Desviación Estándar R.S	Fitness promedio	Desviación Estándar	Mejora experto (%)	Mejora R.S (%)
PSO W	25	1.319×10^{-2}	9.564×10^{-3}	1.655×10^{-4}	9.073×10^{-3}	1.421×10^{-4}	31,21	5,13
PSO W	50	1.120×10^{-2}	8.215×10^{-3}	1.355×10^{-4}	7.915×10^{-3}	1.827×10^{-4}	29,34	3,65
PSO X	25	1.319×10^{-2}	9.564×10^{-3}	1.655×10^{-4}	9.118×10^{-3}	1.972×10^{-4}	30,87	4,66
PSO X	50	1.120×10^{-2}	8.215×10^{-3}	1.355×10^{-4}	7.829×10^{-3}	1.383×10^{-4}	30,10	4,69

Tabla 6.9: Mejoras contra valores de experto y Random Search para 25 y 50 ciclos.

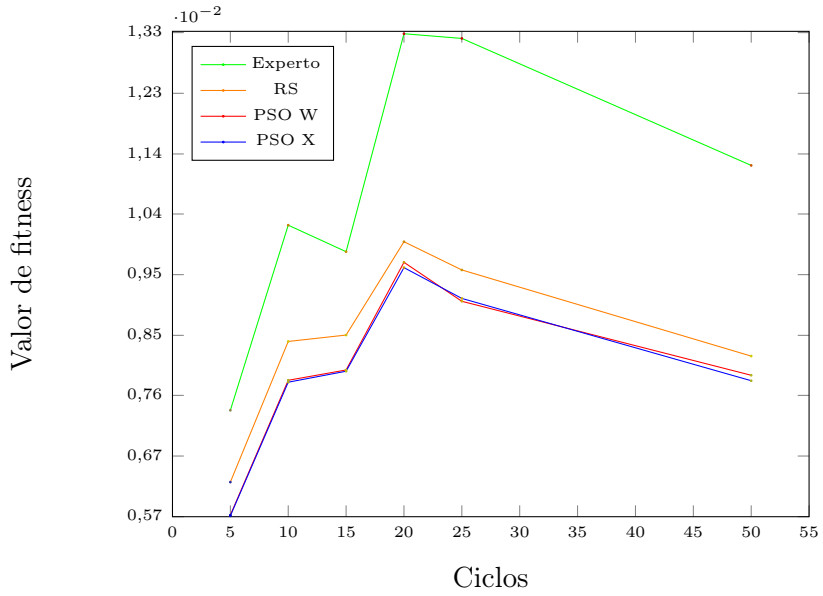


Figura 6.7: Promedios de fitness en función de la instancia

Finalmente se presenta en la Figura 6.7 la evolución de los fitness promedio en función de la instancia encontrados por cada uno de los algoritmos incluyendo los valores de referencia. Y análogamente la evolución de los mejores fitness en la Figura 6.8. En ambas gráficas se destaca la superioridad de los algoritmos PSO, resultando la variante con factor de constricción la que obtiene mejor rendimiento. Además, se puede observar un comportamiento similar en la evolución del valor de fitness en función de la instancia para todos los algoritmos, así como también para los valores de referencia.

6.2.3. Tercer experimento

El tercer experimento tiene el objetivo de estudiar el impacto de aumentar el número de iteraciones de los algoritmos en la calidad de los resultados obtenidos. Con tal fin, para el caso de los algoritmos PSO se decidió comparar las ejecuciones usando 100 y 200 iteraciones. Tal como se vio en secciones anteriores, 100 iteraciones conlleva a realizar 3000 evaluaciones de la función de fitness y 6000 evaluaciones en el caso de 200 iteraciones. Por lo tanto, para el caso del algoritmo RS se consideraran las ejecuciones con 3000 y 6000 iteraciones, dado que en cada iteración este algoritmo realiza una única evaluación de la función de fitness. Cabe destacar que por simplicidad a nivel de las tablas se hará referencia a las versiones de 100 y 200 iteraciones incluso para RS, aunque en este último caso corresponda a 3000 y 6000 iteraciones.

Para realizar este experimento se seleccionó la instancia de 25 ciclos para todas las ejecuciones. Además, cada ejecución se realizó de manera indepen-

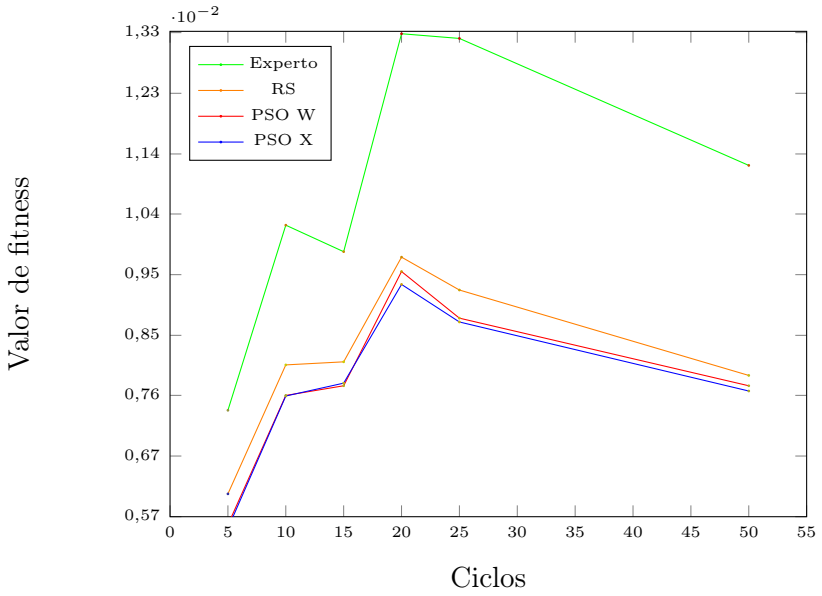


Figura 6.8: Mejores fitness en función de la instancia

diente, es decir, los resultados para 100 iteraciones no se extraen como un resultado intermedio de las ejecuciones de 200 iteraciones.

La Tabla 6.10 resume los resultados de mejor fitness, fitness promedio, desviación estándar y tiempos promedio de ejecución para los diferentes algoritmos con 100 y 200 iteraciones.

Una observación importante respecto a los tiempos registrados es que no son 100 % fieles a las estimaciones previamente realizadas. Como ya fue explicado en el experimento anterior, esto se debe a que cada ejecución no contó con la disponibilidad exclusiva de los recursos de hardware utilizados.

En general, se puede concluir que aumentar al doble el número de iteraciones de los algoritmos implica aproximadamente un aumento al doble también del tiempo de ejecución.

Por su parte, la Tabla 6.11 realiza la comparación de los resultados de la Tabla 6.10, mostrando los porcentajes de mejora de las versiones de 100 y 200 iteraciones para cada uno de los diferentes algoritmos.

Aquí se puede observar que los algoritmos PSO con factor de constricción y RS mejoraron en ambos casos en el entorno de 2 % a 3 %. Sin embargo, el algoritmo PSO con peso de inercia empeora. Esto puede deberse a que las ejecuciones para 100 y 200 iteraciones se realizaron de manera independiente conjuntamente con un número de ejecuciones (ocho) un tanto bajo para realizar los promedios. Este último punto fue una restricción causada por los excesivos tiempos requeridos para la evaluación de la función de fitness. De cualquier manera el porcentaje de desmejora no fue muy significativo (apenas un 0,16 %) y por otra parte, si se analiza el valor del mejor fitness

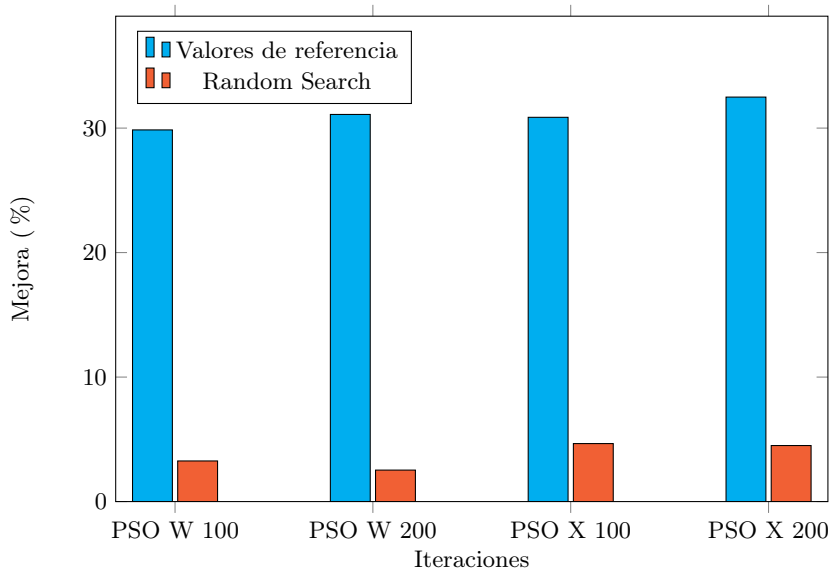


Figura 6.9: Mejoras de PSO (100 vs 200 iteraciones)

existe una leve mejora de la versión de 200 iteraciones respecto a la de 100, tal como se esperaba.

Finalmente, la Tabla 6.12 analiza los porcentajes de mejora de los algoritmos PSO con respecto a los valores del experto y RS para 100 y 200 iteraciones. En primer lugar, conviene señalar que los algoritmos PSO superan tanto los valores de referencia como los hallados por RS para 100, así como también para 200 iteraciones. Particularmente, para el caso del experto se destaca una amplia mejora. Otra observación importante es que PSO con factor de constricción muestra un mejor desempeño que PSO con peso de inercia para ambas cantidades de iteraciones. Por último, se nota que el aumento de iteraciones se traduce en una disminución de los porcentajes de mejora de ambos algoritmos PSO respecto a RS. Esto puede deberse a la baja cantidad de ejecuciones realizadas y a que si bien los algoritmos PSO mejoran con el aumento de iteraciones, RS también lo hace, y por tal motivo el porcentaje de mejora resulta cada vez menos significativo.

Para poder observar de mejor manera el análisis comparativo anterior se presentan los resultados de la Tabla 6.12 expresados de manera gráfica en la Figura 6.9.

Algoritmo	Ciclos	Iteraciones	Fitness promedio	Desviación estándar	Mejor fitness	Tiempo promedio (s)
PSO W	25	100	9.073×10^{-3}	1.421×10^{-4}	8.809×10^{-3}	79 762,37
PSO W	25	200	9.088×10^{-3}	1.709×10^{-4}	8.806×10^{-3}	157 832,45
PSO X	25	100	9.118×10^{-3}	1.972×10^{-4}	8.749×10^{-3}	83 087,35
PSO X	25	200	8.904×10^{-3}	1.447×10^{-4}	8.737×10^{-3}	159 348,15
RS	25	100	9.564×10^{-3}	1.655×10^{-4}	9.249×10^{-3}	86 960,78
RS	25	200	9.323×10^{-3}	2.779×10^{-4}	9.019×10^{-3}	175 984,69

Tabla 6.10: Fitness promedio, mejor fitness y tiempo promedio para 100 y 200 iteraciones.

Algoritmo	Ciclos	100 iteraciones			200 iteraciones			
		Fitness promedio	Desviación estándar	Tiempo promedio	Fitness promedio	Desviación estándar	Tiempo promedio	Mejora vs 100 iter. (%)
PSO W	25	9.073×10^{-3}	1.421×10^{-4}	79 762,37	9.088×10^{-3}	1.709×10^{-4}	157 832,45	−0,16
PSO X	25	9.118×10^{-3}	1.972×10^{-4}	83 087,35	8.904×10^{-3}	1.447×10^{-4}	159 348,15	2,35
RS	25	9.564×10^{-3}	1.655×10^{-4}	86 960,78	9.323×10^{-3}	2.779×10^{-4}	175 984,69	2,51

Tabla 6.11: Mejoras de fitness promedio para 100 vs 200 iteraciones.

Algoritmo	Ciclos	Iteraciones	Experto	Fitness promedio RS	Desviación estándar RS	Fitness promedio	Desviación estándar	Mejora experto (%)	Mejora RS (%)
PSO W	25	100	1.319×10^{-2}	9.564×10^{-3}	1.655×10^{-4}	9.252×10^{-3}	1.421×10^{-4}	29,85	3,26
PSO W	25	200	1.319×10^{-2}	9.323×10^{-3}	2.779×10^{-4}	9.088×10^{-3}	1.709×10^{-4}	31,10	2,53
PSO X	25	100	1.319×10^{-2}	9.564×10^{-3}	1.655×10^{-4}	9.118×10^{-3}	1.972×10^{-4}	30,87	4,66
PSO X	25	200	1.319×10^{-2}	9.323×10^{-3}	2.779×10^{-4}	8.904×10^{-3}	1.447×10^{-4}	32,49	4,50

Tabla 6.12: Mejoras contra valores de experto y Random Search para 100 y 200 iteraciones.

6.3. Análisis físico de los resultados

En esta sección se realiza una comparación entre las configuraciones de parámetros del motor obtenidas mediante la utilización de los algoritmos PSO y la configuración de parámetros del motor que se realizaba en base a la experiencia del experto. En la Tabla 6.13 se presenta el valor de cada uno de los parámetros y del fitness para la mejor solución encontrada en las ocho ejecuciones. Los valores se presentan truncando a tres cifras después de la coma.

Ciclos	Método	IVO	IVC	EVO	EVC	SA	Fitness
5	Experto	−50,000	214,000	490,000	750,000	330,000	7.366×10^{-3}
	w mejor	−30,917	217,888	487,876	739,039	331,219	5.572×10^{-3}
	x mejor	−30,547	214,813	492,936	738,034	331,367	5.517×10^{-3}
10	Experto	−50,000	214,000	490,000	750,000	330,000	1.026×10^{-2}
	w mejor	−18,732	223,313	490,053	739,869	334,675	7.598×10^{-3}
	x mejor	−15,957	214,145	496,311	738,172	335,538	7.589×10^{-3}
15	Experto	−50,000	214,000	490,000	750,000	330,000	9.849×10^{-3}
	w mejor	−15,196	214,201	484,719	740,644	335,077	7.750×10^{-3}
	x mejor	−18,366	221,421	495,948	741,133	334,663	7.791×10^{-3}
20	Experto	−50,000	214,000	490,000	750,000	330,000	1.326×10^{-2}
	w mejor	−26,033	216,646	493,934	736,417	335,608	9.538×10^{-3}
	x mejor	−12,468	225,531	490,024	741,551	337,041	9.337×10^{-3}
25	Experto	−50,000	214,000	490,000	750,000	330,000	1.319×10^{-2}
	w mejor	−10,000	219,883	490,794	742,344	337,012	8.809×10^{-3}
	x mejor	−12,156	222,816	487,892	739,104	337,051	8.749×10^{-3}
50	Experto	−50,000	214,000	490,000	750,000	330,000	1.120×10^{-2}
	w mejor	−10,145	223,646	488,611	742,186	337,019	7.750×10^{-3}
	x mejor	−10,207	218,456	493,210	739,567	337,512	7.668×10^{-3}

Tabla 6.13: Valores de los parámetros para la mejor solución de las ocho ejecuciones a partir de las mejores soluciones encontradas en cada ejecución.

Comenzaremos nuestro análisis por el parámetro IVO. El valor basado en la experiencia del experto se ubica en el extremo inferior del intervalo de valores considerados para dicho parámetro ($-50 \leq IVO \leq -10$). Por el contrario, los valores de IVO obtenidos con los algoritmos PSO, se pegan contra el extremo superior del intervalo a medida que se consideran simulaciones con una mayor cantidad de ciclos. Un valor más grande de IVO corresponde a atrasar la apertura de la válvula de admisión. Dicho atraso tiene como efecto que se reduzcan los gases residuales y esto provoca que las velocidades de combustión sean más altas y alcancen un valor cercano al máximo. Por lo tanto, para un avance de encendido determinado, el proceso de combustión se estabiliza, es decir que se reduce la variabilidad cíclica.

En lo que respecta al parámetro IVC, el valor basado en la experiencia del experto es 214 (siendo el intervalo de valores considerados $210 \leq IVC \leq 250$). Los valores de IVC obtenidos por los algoritmos PSO son algo superiores a los del experto. El cierre de la válvula de admisión condiciona la velocidad media de los gases, que de alguna manera, determina la intensidad de la turbulencia que también influye en la velocidad de la combustión. Por otro lado, también condiciona la carga de masa que entra al sistema, que está relacionado directamente con la producción de potencia.

Para el parámetro EVO, el valor del experto es 490 (siendo el intervalo de valores considerados $480 \leq EVO \leq 500$) y es el parámetro en el que los valores obtenidos mediante los algoritmos PSO se encuentran más próximos al valor definido por el experto. Conviene destacar que el valor del experto fue determinado considerando únicamente la eficiencia del motor, mientras que con los algoritmos PSO lo que se busca es reducir la variabilidad cíclica.

El valor del experto para el parámetro EVC es 750, siendo el intervalo de valores considerados $735 \leq EVC \leq 750$. Los valores que obtienen los algoritmos PSO para las buenas soluciones son cercanos a 740. Tanto el parámetro EVO como el parámetro EVC condicionan la eliminación de gases y afectan al valor de los gases residuales (como se explicó para el parámetro IVO). Estos condicionan las velocidades de quema del combustible y afectan la variabilidad cíclica.

Finalmente, el valor del experto para el parámetro SA es 330, siendo el intervalo de valores considerados $IVC \leq SA \leq 360$. Los valores de SA obtenidos con los algoritmos PSO, se incrementan a medida que se consideran simulaciones con una mayor cantidad de ciclos y toman valores superiores a los utilizados por el experto. Esto es físicamente consistente ya que el parámetro SA está directamente relacionado con la duración de la combustión y es extremadamente sensible a sus cambios. La duración de la combustión afecta la forma en cómo se libera la energía del combustible durante el ciclo y afecta la potencia y la eficiencia.

Capítulo 7

Conclusiones y trabajo futuro

Life is the art of drawing sufficient conclusions from insufficient premises.

Samuel Butler

El objetivo de este capítulo es brindar las conclusiones más relevantes de este proyecto, así como también la propuesta de posibles trabajos a futuro como extensión del mismo.

7.1. Estructura del capítulo

Para comenzar se destina una sección a las conclusiones resultantes de la finalización del proyecto. Aquí se intenta contextualizar, describiendo los objetivos y los resultados obtenidos asociados a cada uno de los mismos. Posteriormente se destina una nueva sección para enumerar algunos puntos pendientes y trabajos a futuro que este proyecto motiva a realizar.

7.2. Conclusiones

Antes de comenzar a enumerar las diferentes conclusiones de la realización de este trabajo, es conveniente repasar los principales objetivos planteados al inicio y a lo largo del proyecto. El primer objetivo del proyecto fue relevar el estado del arte de la metaheurística Particle Swarm Optimization. En segundo lugar, se estudió el problema de la determinación de parámetros para un simulador de un motor de combustión interna. Finalmente, se diseñó un algoritmo PSO para su resolución que fue evaluado sobre un conjunto amplio de escenarios.

Durante el relevamiento del estado del arte, fueron encontradas un gran número de variantes de la metaheurística PSO, de las cuales por motivos de

alcance del trabajo, fueron seleccionadas solamente un grupo representativo. De este grupo, se eligieron dos variantes a implementar. La primera llamada PSO con Peso de Inercia, la cual agrega un coeficiente que tiene el objetivo de controlar el nivel de influencia de la velocidad actual en el cálculo de la próxima posición. La segunda variante seleccionada fue PSO con Factor de Constricción, la cual agrega un coeficiente adicional en la ecuación de velocidad con el objetivo de evitar la explosión del enjambre así como también asegurar la convergencia del algoritmo.

El problema abordado consiste en la determinación de parámetros para un simulador de un motor de combustión interna diseñado y desarrollado por docentes del Instituto de Ingeniería Mecánica y Producción Industrial (IIMPI), de forma de encontrar la combinación de parámetros que produce el mejor desempeño del motor mediante la utilización de PSO. El objetivo consiste en obtener la configuración de parámetros que minimiza la variabilidad cíclica del motor de combustión interna.

Para ello se consideran cinco parámetros de entrada del simulador: el ángulo de apertura de la válvula de admisión (IVO), el ángulo de cierre de la válvula de admisión (IVC), el ángulo de apertura de la válvula de escape (EVO), el ángulo de cierre de la válvula de escape (EVC) y el avance de la chispa (SA); y un único valor de salida: el coeficiente de variación de la eficiencia del motor (RCOV).

El problema se pude ver como un problema de optimización continua con enfoque de caja negra, que toma el valor de salida producido por el simulador como función de fitness, los cinco parámetros de entrada como variables de decisión y cuyo objetivo consiste en encontrar el menor valor de RCOV a través de la minimización de la función de fitness.

La evaluación experimental de los algoritmos PSO propuestos fue dividida en tres experimentos, el primero que considera el uso de diferentes instancias del problema (motor configurado con 5, 10, 15, 20 y 25 ciclos). Un segundo experimento que extiende el primero, agregando una instancia con una cantidad significativamente mayor de ciclos del motor. Finalmente, el tercer experimento que tiene el objetivo de estudiar el impacto de aumentar el número de iteraciones de los algoritmos en la calidad de los resultados obtenidos. En la evaluación experimental, con el objetivo de tener valores de referencia que permitieran comparar los resultados, se consideró además una búsqueda completamente aleatoria (Random Search) y un conjunto de valores de los parámetros provistos por un experto.

Del primer experimento, se concluye que los algoritmos PSO superan los valores sugeridos por el experto, así como también a los obtenidos por RS. En el primero caso por una diferencia muy significativa, mientras que en el segundo por un margen razonable. Como conclusión del segundo experimento se destaca un leve decrecimiento en el porcentaje de mejora entre las instancias de 25 y 50 ciclos de los algoritmos PSO en relación a RS y los

valores de referencia, en términos del valor de fitness promedio. Finalmente, el planteo del tercer experimento esperaba obtener resultados de mejor calidad, objetivo que también fue logrado. Sin embargo, un punto a destacar es que la variante PSO con Peso de Inercia no obtuvo la mejora esperada en promedio (no así para el caso de los mejores fitness). Esto puede deberse al bajo número de ejecuciones realizado en dicho experimento.

Un objetivo fundamental de este trabajo era lograr obtener mejores valores de la variabilidad cíclica del motor que los que se obtienen a partir de los valores de los parámetros propuestos por el experto. El mismo fue cumplido con un amplio margen de mejora para cada una de las instancias. Por otro lado, se esperaba también lograr buenos resultados compitiendo con otras técnicas para la resolución del problema. Este punto también fue alcanzado, consiguiendo ambas variantes de PSO mejores resultados que el algoritmo Random Search.

Dentro de las variantes de PSO implementadas se comprobó que PSO con Factor de Constricción obtuvo un mejor desempeño numérico en comparación al logrado por PSO con Peso de Inercia, tal como se afirmaba en los distintos trabajos relevados. Esta conclusión se basa en los resultados obtenidos por ambos algoritmos tanto en la etapa de calibración como en la de evaluación. Este comportamiento puede deberse a que la variante PSO con Factor de Constricción brinda un mayor control de la explosión de la velocidad, lo que favorece una explotación más focalizada de áreas promisorias del espacio de búsqueda. En cambio, PSO con Peso de Inercia se ve limitada en este aspecto, ya que utiliza un único parámetro para controlar los niveles de exploración y explotación.

Ciertas decisiones tomadas a lo largo de este trabajo fueron producto de algunos factores limitantes, tal como es el caso de contar con una función de fitness con un alto costo de evaluación en términos de tiempo. Esta limitante afecto considerablemente las etapas de calibración y evaluación, determinando un requerimiento muy alto de tiempo de cómputo, así como también limitando el número de ejecuciones e iteraciones a realizar.

En el caso de la calibración, no fue posible contar con un número adecuado de combinaciones de valores de los parámetros a calibrar, dado que cada nueva combinación exige una gran cantidad adicional de tiempo de cómputo. Esto claramente no favorece a la calidad esperada de los resultados de esta etapa, ya que podrían estar siendo no considerados valores potencialmente buenos. Otro factor con posible impacto en este punto fue la carencia de un método formal de calibración de parámetros desestimado por motivos de alcance.

Una posible consecuencia de estas limitantes, puede ser la elección de la topología Completa por sobre la de Von Neumann, siendo que ésta última es sugerida en muchos trabajos por su buen desempeño numérico.

En la etapa de evaluación, la cantidad de iteraciones y ejecuciones reali-

zadas fue significativamente baja. Para el caso de las ejecuciones, un número mayor de estas (al menos 30, pero idealmente 100) hubiese permitido obtener resultados con una mayor certeza, y hubieran permitido la realización de test estadísticos para comprobar si las diferencias en los resultados obtenidos son significativas estadísticamente. Por otro lado, el número de iteraciones condiciona el recorrido realizado por las partículas, acotando el área del dominio explorada.

Durante esta etapa, se percibió en los resultados una posible convergencia prematura. Esto probablemente se deba a que la topología utilizada en esta etapa no favorece a la exploración del espacio de búsqueda. Tal vez, la utilización de una topología diferente hubiese provisto una mayor capacidad en este aspecto, brindando eventualmente mejores resultados.

Finalmente, conviene señalar que el éxito relativo que tiene una simple búsqueda aleatoria para la resolución del problema planteado puede indicar que el problema es relativamente sencillo. Se debería conducir un estudio más profundo sobre el problema para entender sus características particulares que permitan entender si los resultados obtenidos en este trabajo se deben al problema ó si puede mejorarse el diseño de los algoritmos PSO.

7.3. Trabajo futuro

Esta sección tiene el objetivo de presentar algunos de los posibles trabajos a futuro, como extensión del presente proyecto.

Por motivos de alcance fueron desestimadas algunas propuestas que hubiera sido interesante abordar. En este contexto se destacan, la cantidad de variantes de PSO relevadas, el uso de una metaheurística más competitiva a la hora de comparar desempeños y el incremento tanto del número de ejecuciones como de iteraciones en las etapas de calibración y evaluación.

Como fue mencionado anteriormente, este trabajo relevó un número representativo de variantes de PSO, sin embargo existen una gran cantidad de estas. Una propuesta interesante sería implementar alguna de las variantes no seleccionadas, así como también, relevar nuevas variantes, para su posterior implementación y evaluación de su desempeño.

Otro trabajo interesante a futuro, sería exigir a PSO, comparándola con una o varias metaheurísticas más competitivas que RS. Esto permitiría visualizar mejor el posicionamiento de PSO en el universo de metaheurísticas para la resolución de un problema de optimización específico.

Tal como fue mencionado en las conclusiones, resulta de interés realizar las ejecuciones con un mayor número de iteraciones. Esto permitiría ampliar el tiempo de exploración de las partículas, permitiendo examinar más regiones del espacio de búsqueda. De manera similar, se propone aumentar el número de ejecuciones con el fin de mejorar los resultados en promedio, mejorando también desde otro enfoque el cubrimiento y la exploración del

dominio.

La utilización de un método más sofisticado de calibración de parámetros resulta un desafío interesante ya que mejoraría el criterio de selección de valores de los parámetros, lo cual impactaría directamente en la calidad de los resultados obtenidos en la etapa de evaluación.

Del comportamiento observado de los algoritmos PSO implementados, resulta interesante desarrollar una nueva variante que cuente con la capacidad de alternar de manera dinámica el tipo de topología del enjambre de partículas a lo largo de la ejecución. Esto es, comenzar con una topología que brinde un mayor nivel de exploración y luego cambiar a otra que proporcione un mayor nivel de explotación.

Apéndice A

Resultados de la calibración de parámetros y evaluación

Locura es hacer la misma cosa una y otra vez esperando obtener diferentes resultados.

Albert Einstein

El objetivo de este apéndice es brindar el detalle de los resultados obtenidos en la etapa de calibración de parámetros así como también los de la etapa de evaluación.

A.1. Resultados de la calibración de parámetros

Luego de ejecutados los scripts de calibración descriptos en la Sección 5.3 se obtuvieron los resultados para la variante de *Peso de Inercia* y *Factor de Constricción* detallados en las tablas Tabla A.1 y Tabla A.2 respectivamente.

Nro. config.	Instancia 5 ciclos			Instancia 10 ciclos		
	Fitness ejec. 1	**Fitness ejec. 2**	**Fitness ejec. 3**	**Fitness ejec. 1**	**Fitness ejec. 2**	**Fitness ejec. 3**
1	$5{,}711 \times 10^{-3}$	$5{,}854 \times 10^{-3}$	$5{,}825 \times 10^{-3}$	$7{,}599 \times 10^{-3}$	$7{,}779 \times 10^{-3}$	$7{,}811 \times 10^{-3}$
2	$6{,}003 \times 10^{-3}$	$5{,}512 \times 10^{-3}$	$5{,}617 \times 10^{-3}$	$8{,}497 \times 10^{-3}$	$7{,}976 \times 10^{-3}$	$7{,}891 \times 10^{-3}$
3	$5{,}789 \times 10^{-3}$	$5{,}616 \times 10^{-3}$	$5{,}926 \times 10^{-3}$	$7{,}838 \times 10^{-3}$	$7{,}757 \times 10^{-3}$	$7{,}750 \times 10^{-3}$
4	$5{,}915 \times 10^{-3}$	$5{,}713 \times 10^{-3}$	$5{,}703 \times 10^{-3}$	$8{,}149 \times 10^{-3}$	$8{,}541 \times 10^{-3}$	$8{,}006 \times 10^{-3}$
5	$5{,}689 \times 10^{-3}$	$5{,}714 \times 10^{-3}$	$5{,}882 \times 10^{-3}$	$8{,}408 \times 10^{-3}$	$8{,}136 \times 10^{-3}$	$7{,}794 \times 10^{-3}$
6	$5{,}574 \times 10^{-3}$	$5{,}570 \times 10^{-3}$	$5{,}688 \times 10^{-3}$	$7{,}928 \times 10^{-3}$	$8{,}042 \times 10^{-3}$	$8{,}313 \times 10^{-3}$
7	$5{,}968 \times 10^{-3}$	$5{,}559 \times 10^{-3}$	$5{,}732 \times 10^{-3}$	$7{,}896 \times 10^{-3}$	$7{,}756 \times 10^{-3}$	$7{,}940 \times 10^{-3}$
8	$6{,}195 \times 10^{-3}$	$6{,}325 \times 10^{-3}$	$6{,}041 \times 10^{-3}$	$8{,}356 \times 10^{-3}$	$8{,}679 \times 10^{-3}$	$8{,}262 \times 10^{-3}$
9	$6{,}150 \times 10^{-3}$	$5{,}810 \times 10^{-3}$	$6{,}022 \times 10^{-3}$	$8{,}563 \times 10^{-3}$	$8{,}277 \times 10^{-3}$	$8{,}037 \times 10^{-3}$
10	$6{,}220 \times 10^{-3}$	$6{,}052 \times 10^{-3}$	$5{,}822 \times 10^{-3}$	$8{,}047 \times 10^{-3}$	$8{,}038 \times 10^{-3}$	$8{,}436 \times 10^{-3}$
11	$6{,}179 \times 10^{-3}$	$6{,}006 \times 10^{-3}$	$5{,}695 \times 10^{-3}$	$7{,}942 \times 10^{-3}$	$8{,}686 \times 10^{-3}$	$8{,}106 \times 10^{-3}$
12	$5{,}755 \times 10^{-3}$	$6{,}052 \times 10^{-3}$	$6{,}107 \times 10^{-3}$	$8{,}981 \times 10^{-3}$	$8{,}297 \times 10^{-3}$	$8{,}370 \times 10^{-3}$
13	$6{,}029 \times 10^{-3}$	$6{,}097 \times 10^{-3}$	$5{,}951 \times 10^{-3}$	$8{,}119 \times 10^{-3}$	$8{,}330 \times 10^{-3}$	$8{,}330 \times 10^{-3}$
14	$6{,}379 \times 10^{-3}$	$6{,}022 \times 10^{-3}$	$5{,}969 \times 10^{-3}$	$8{,}705 \times 10^{-3}$	$7{,}964 \times 10^{-3}$	$7{,}964 \times 10^{-3}$
15	$6{,}091 \times 10^{-3}$	$6{,}213 \times 10^{-3}$	$6{,}147 \times 10^{-3}$	$8{,}211 \times 10^{-3}$	$8{,}476 \times 10^{-3}$	$7{,}877 \times 10^{-3}$
16	$5{,}979 \times 10^{-3}$	$6{,}077 \times 10^{-3}$	$6{,}243 \times 10^{-3}$	$8{,}058 \times 10^{-3}$	$8{,}286 \times 10^{-3}$	$8{,}342 \times 10^{-3}$
17	$5{,}931 \times 10^{-3}$	$6{,}131 \times 10^{-3}$	$5{,}979 \times 10^{-3}$	$8{,}315 \times 10^{-3}$	$8{,}067 \times 10^{-3}$	$8{,}308 \times 10^{-3}$
18	$5{,}968 \times 10^{-3}$	$5{,}943 \times 10^{-3}$	$6{,}301 \times 10^{-3}$	$8{,}451 \times 10^{-3}$	$8{,}672 \times 10^{-3}$	$8{,}465 \times 10^{-3}$

Tabla A.1: Resultados detallados de la calibración - Peso de Inercia

Nro. config.	Instancia 5 ciclos			Instancia 10 ciclos		
	Fitness ejec. 1	**Fitness ejec. 2**	**Fitness ejec. 3**	**Fitness ejec. 1**	**Fitness ejec. 2**	**Fitness ejec. 3**
1	$5{,}609 \times 10^{-3}$	$5{,}688 \times 10^{-3}$	$5{,}693 \times 10^{-3}$	$7{,}745 \times 10^{-3}$	$7{,}837 \times 10^{-3}$	$7{,}844 \times 10^{-3}$
2	$6{,}140 \times 10^{-3}$	$6{,}283 \times 10^{-3}$	$5{,}926 \times 10^{-3}$	$8{,}250 \times 10^{-3}$	$8{,}015 \times 10^{-3}$	$8{,}399 \times 10^{-3}$
3	$5{,}584 \times 10^{-3}$	$5{,}800 \times 10^{-3}$	$5{,}881 \times 10^{-3}$	$8{,}055 \times 10^{-3}$	$8{,}061 \times 10^{-3}$	$8{,}028 \times 10^{-3}$
4	$5{,}824 \times 10^{-3}$	$5{,}835 \times 10^{-3}$	$5{,}916 \times 10^{-3}$	$8{,}360 \times 10^{-3}$	$8{,}450 \times 10^{-3}$	$7{,}719 \times 10^{-3}$
5	$5{,}752 \times 10^{-3}$	$5{,}630 \times 10^{-3}$	$5{,}589 \times 10^{-3}$	$7{,}973 \times 10^{-3}$	$7{,}820 \times 10^{-3}$	$8{,}178 \times 10^{-3}$
6	$5{,}774 \times 10^{-3}$	$5{,}940 \times 10^{-3}$	$5{,}789 \times 10^{-3}$	$8{,}275 \times 10^{-3}$	$8{,}442 \times 10^{-3}$	$8{,}645 \times 10^{-3}$

Tabla A.2: Resultados detallados de la calibración - Factor de Constricción

A.2. Resultados de la etapa de evaluación

Esta sección se divide en tres subsecciones que brindan toda la información recabada por los experimentos realizados en la etapa de evaluación.

A.2.1. Resultados del primer experimento

Luego de realizadas todas las ejecuciones del primer experimento fueron recabados los datos que se presentan en esta subsección. En la Tabla A.3 se muestran los resultados obtenidos por el algoritmo PSO con Peso de Inercia. De manera análoga, la Tabla A.4 brinda los resultados obtenidos por el algoritmo PSO con Factor de Constricción. Finalmente, la Tabla A.5 muestra los resultados obtenidos por el algoritmo RS.

Además, se presentan los resultados obtenidos en el estudio de la evolución de los mejores fitness encontrados hasta el momento en cada iteración para cada uno de los diferentes algoritmos. Esto es, la Tabla A.6 y la Tabla A.7 brindan los resultados obtenidos por los diferentes algoritmos incluyendo éstas, la primera y la segunda mitad de las iteraciones respectivamente.

A.2.2. Resultados del segundo experimento

En esta subsección son presentados los datos recabados durante las ejecuciones del segundo experimento. En la Tabla A.8 se muestran los resultados obtenidos por todos los algoritmos utilizando la instancia de 25 ciclos, realizando 100 iteraciones. Por otro lado, la Tabla A.9 brinda los resultados obtenidos por los algoritmos utilizando la instancia de 50 ciclos, realizando también 100 iteraciones.

A.2.3. Resultados del tercer experimento

En esta subsección son presentados los datos recabados durante las ejecuciones del tercer experimento. En la Tabla A.10 se muestran los resultados obtenidos por todos los algoritmos realizando 100 iteraciones, utilizando la instancia de 25 ciclos. Por otro lado, la Tabla A.11 brinda los resultados obtenidos por los algoritmos realizando 200 iteraciones, utilizando también la instancia de 25 ciclos.

Algoritmo	Ciclos	Nro de ejec.	Fitness	Tiempo (s)
PSO W	5	1	5.711×10^{-3}	30 931,81
PSO W	5	2	5.854×10^{-3}	31 416,68
PSO W	5	3	5.825×10^{-3}	31 024,46
PSO W	5	4	5.605×10^{-3}	18 043,30
PSO W	5	5	5.573×10^{-3}	18 392,67
PSO W	5	6	5.909×10^{-3}	18 555,13
PSO W	5	7	5.668×10^{-3}	18 490,50
PSO W	5	8	5.647×10^{-3}	18 264,67
PSO W	10	1	7.599×10^{-3}	283 306,81
PSO W	10	2	7.779×10^{-3}	284 099,02
PSO W	10	3	7.811×10^{-3}	288 543,76
PSO W	10	4	8.320×10^{-3}	34 870,55
PSO W	10	5	7.695×10^{-3}	33 052,07
PSO W	10	6	8.022×10^{-3}	32 717,71
PSO W	10	7	7.805×10^{-3}	33 914,01
PSO W	10	8	7.658×10^{-3}	33 843,20
PSO W	15	1	8.153×10^{-3}	50 096,30
PSO W	15	2	7.964×10^{-3}	48 062,49
PSO W	15	3	7.912×10^{-3}	49 601,92
PSO W	15	4	7.981×10^{-3}	49 738,05
PSO W	15	5	7.912×10^{-3}	48 743,84
PSO W	15	6	7.750×10^{-3}	48 155,41
PSO W	15	7	8.291×10^{-3}	49 398,41
PSO W	15	8	8.017×10^{-3}	51 241,88
PSO W	20	1	9.854×10^{-3}	65 381,21
PSO W	20	2	9.539×10^{-3}	64 602,98
PSO W	20	3	9.751×10^{-3}	65 898,87
PSO W	20	4	9.577×10^{-3}	64 665,86
PSO W	20	5	9.874×10^{-3}	66 243,95
PSO W	20	6	9.669×10^{-3}	64 707,51
PSO W	20	7	9.599×10^{-3}	65 113,10
PSO W	20	8	9.604×10^{-3}	64 820,67
PSO W	25	1	8.954×10^{-3}	78 336,19
PSO W	25	2	9.099×10^{-3}	79 805,79
PSO W	25	3	8.809×10^{-3}	77 893,09
PSO W	25	4	9.112×10^{-3}	79 575,02
PSO W	25	5	9.141×10^{-3}	81 545,87
PSO W	25	6	9.252×10^{-3}	81 113,18
PSO W	25	7	9.020×10^{-3}	78 906,24
PSO W	25	8	9.198×10^{-3}	80 923,59

Tabla A.3: Resultados de la evaluación para PSO con Peso de Inercia (5 a 25 ciclos).

Algoritmo	Ciclos	Nro de ejec.	Fitness	Tiempo (s)
PSO X	5	1	5.609×10^{-3}	186 487,15
PSO X	5	2	5.688×10^{-3}	192 981,78
PSO X	5	3	5.693×10^{-3}	188 834,18
PSO X	5	4	5.595×10^{-3}	19 127,23
PSO X	5	5	5.517×10^{-3}	17 775,11
PSO X	5	6	6.084×10^{-3}	18 967,46
PSO X	5	7	5.937×10^{-3}	18 685,49
PSO X	5	8	5.588×10^{-3}	18 518,15
PSO X	10	1	7.745×10^{-3}	287 585,96
PSO X	10	2	7.837×10^{-3}	288 047,78
PSO X	10	3	7.844×10^{-3}	287 673,84
PSO X	10	4	8.082×10^{-3}	34 216,17
PSO X	10	5	7.590×10^{-3}	32 750,99
PSO X	10	6	7.615×10^{-3}	33 524,25
PSO X	10	7	8.071×10^{-3}	34 362,77
PSO X	10	8	7.664×10^{-3}	31 249,41
PSO X	15	1	7.791×10^{-3}	64 688,12
PSO X	15	2	8.013×10^{-3}	64 947,32
PSO X	15	3	7.982×10^{-3}	66 290,04
PSO X	15	4	8.388×10^{-3}	66 525,12
PSO X	15	5	7.892×10^{-3}	63 789,18
PSO X	15	6	8.000×10^{-3}	65 083,99
PSO X	15	7	7.878×10^{-3}	64 864,59
PSO X	15	8	7.878×10^{-3}	63 754,97
PSO X	20	1	9.484×10^{-3}	72 701,75
PSO X	20	2	9.449×10^{-3}	73 068,84
PSO X	20	3	9.671×10^{-3}	72 614,02
PSO X	20	4	9.676×10^{-3}	71 975,32
PSO X	20	5	9.841×10^{-3}	62 462,82
PSO X	20	6	9.388×10^{-3}	60 620,00
PSO X	20	7	9.337×10^{-3}	61 442,15
PSO X	20	8	9.954×10^{-3}	63 141,82
PSO X	25	1	9.311×10^{-3}	82 266,91
PSO X	25	2	9.021×10^{-3}	78 099,64
PSO X	25	3	8.749×10^{-3}	77 776,36
PSO X	25	4	9.215×10^{-3}	79 177,77
PSO X	25	5	9.287×10^{-3}	87 223,53
PSO X	25	6	9.157×10^{-3}	87 184,24
PSO X	25	7	9.259×10^{-3}	87 930,90
PSO X	25	8	8.945×10^{-3}	85 039,46

Tabla A.4: Resultados de la evaluación para PSO con Factor de Constricción (5 a 25 ciclos).

Algoritmo	Ciclos	Nro de ejec.	Fitness	Tiempo (s)
RS	5	1	6.270×10^{-3}	20 916,82
RS	5	2	6.229×10^{-3}	20 837,83
RS	5	3	6.181×10^{-3}	20 855,54
RS	5	4	6.186×10^{-3}	19 750,88
RS	5	5	6.058×10^{-3}	19 794,90
RS	5	6	6.538×10^{-3}	19 772,14
RS	5	7	6.394×10^{-3}	19 782,55
RS	5	8	6.077×10^{-3}	19 813,08
RS	10	1	8.361×10^{-3}	36 927,03
RS	10	2	8.400×10^{-3}	37 129,30
RS	10	3	8.310×10^{-3}	36 949,30
RS	10	4	8.454×10^{-3}	36 656,16
RS	10	5	8.783×10^{-3}	37 254,85
RS	10	6	8.280×10^{-3}	36 829,26
RS	10	7	8.887×10^{-3}	36 739,66
RS	10	8	8.077×10^{-3}	36 349,53
RS	15	1	8.251×10^{-3}	53 419,70
RS	15	2	8.758×10^{-3}	53 467,68
RS	15	3	8.483×10^{-3}	53 191,01
RS	15	4	8.473×10^{-3}	53 445,68
RS	15	5	8.753×10^{-3}	53 333,18
RS	15	6	8.641×10^{-3}	53 271,68
RS	15	7	8.871×10^{-3}	53 497,92
RS	15	8	8.125×10^{-3}	53 544,18
RS	20	1	9.781×10^{-3}	70 365,89
RS	20	2	1.006×10^{-2}	70 292,71
RS	20	3	9.899×10^{-3}	70 655,90
RS	20	4	1.025×10^{-2}	70 638,62
RS	20	5	9.824×10^{-3}	70 437,10
RS	20	6	1.026×10^{-2}	70 505,61
RS	20	7	9.764×10^{-3}	70 649,68
RS	20	8	1.020×10^{-2}	70 339,29
RS	25	1	9.249×10^{-3}	87 171,71
RS	25	2	9.681×10^{-3}	86 530,75
RS	25	3	9.798×10^{-3}	87 174,63
RS	25	4	9.676×10^{-3}	86 868,08
RS	25	5	9.515×10^{-3}	86 880,72
RS	25	6	9.516×10^{-3}	87 357,00
RS	25	7	9.484×10^{-3}	87 029,27
RS	25	8	9.591×10^{-3}	86 674,12

Tabla A.5: Resultados de la evaluación para Random Search (5 a 25 ciclos).

Iteración	PSO W	PSO X	RS
1	2.966×10^{-2}	1.006×10^{-2}	1.899×10^{-1}
2	1.331×10^{-2}	1.013×10^{-2}	1.180×10^{-2}
3	9.955×10^{-3}	9.322×10^{-3}	1.048×10^{-2}
4	9.955×10^{-3}	9.322×10^{-3}	1.048×10^{-2}
5	9.955×10^{-3}	9.322×10^{-3}	1.048×10^{-2}
6	9.955×10^{-3}	9.314×10^{-3}	9.492×10^{-3}
7	9.955×10^{-3}	9.314×10^{-3}	9.492×10^{-3}
8	9.955×10^{-3}	9.314×10^{-3}	9.492×10^{-3}
9	9.955×10^{-3}	9.314×10^{-3}	9.492×10^{-3}
10	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
11	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
12	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
13	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
14	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
15	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
16	9.955×10^{-3}	9.292×10^{-3}	9.492×10^{-3}
17	9.955×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
18	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
19	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
20	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
21	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
22	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
23	9.935×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
24	9.912×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
25	9.909×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
26	9.893×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
27	9.893×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
28	9.893×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
29	9.893×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
30	9.893×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
31	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
32	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
33	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
34	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
35	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
36	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
37	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
38	9.842×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
39	9.781×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
40	9.781×10^{-3}	9.204×10^{-3}	9.492×10^{-3}
41	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
42	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
43	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
44	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
45	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
46	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
47	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
48	9.781×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
49	9.769×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
50	9.769×10^{-3}	9.191×10^{-3}	9.492×10^{-3}

Tabla A.6: Resultados de la evolución de fitness (1 a 50 iteraciones).

Iteración	PSO W	PSO X	RS
51	9.769×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
52	9.760×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
53	9.760×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
54	9.760×10^{-3}	9.191×10^{-3}	9.492×10^{-3}
55	9.760×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
56	9.760×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
57	9.760×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
58	9.760×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
59	9.760×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
60	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
61	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
62	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
63	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
64	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
65	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
66	9.711×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
67	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
68	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
69	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
70	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
71	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
72	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
73	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
74	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
75	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
76	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
77	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
78	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
79	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
80	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
81	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
82	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
83	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
84	9.580×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
85	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
86	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
87	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
88	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
89	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
90	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
91	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
92	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
93	9.572×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
94	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
95	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
96	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
97	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
98	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
99	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}
100	9.563×10^{-3}	9.144×10^{-3}	9.492×10^{-3}

Tabla A.7: Resultados de la evolución de fitness (51 a 100 iteraciones).

Algoritmo	Ciclos	Nro de ejec.	Fitness	Tiempo (s)
PSO W	25	1	8.954×10^{-3}	78 336,19
PSO W	25	2	9.099×10^{-3}	79 805,79
PSO W	25	3	8.809×10^{-3}	77 893,09
PSO W	25	4	9.112×10^{-3}	79 575,02
PSO W	25	5	9.141×10^{-3}	81 545,87
PSO W	25	6	9.252×10^{-3}	81 113,18
PSO W	25	7	9.020×10^{-3}	78 906,24
PSO W	25	8	9.198×10^{-3}	80 923,59
PSO X	25	1	9.311×10^{-3}	82 266,91
PSO X	25	2	9.021×10^{-3}	78 099,64
PSO X	25	3	8.749×10^{-3}	77 776,36
PSO X	25	4	9.215×10^{-3}	79 177,77
PSO X	25	5	9.287×10^{-3}	87 223,53
PSO X	25	6	9.157×10^{-3}	87 184,24
PSO X	25	7	9.259×10^{-3}	87 930,90
PSO X	25	8	8.945×10^{-3}	85 039,46
RS	25	1	9.249×10^{-3}	87 171,71
RS	25	2	9.681×10^{-3}	86 530,75
RS	25	3	9.798×10^{-3}	87 174,63
RS	25	4	9.676×10^{-3}	86 868,08
RS	25	5	9.515×10^{-3}	86 880,72
RS	25	6	9.516×10^{-3}	87 357,00
RS	25	7	9.484×10^{-3}	87 029,27
RS	25	8	9.591×10^{-3}	86 674,12

Tabla A.8: Resultados de la evaluación para 100 iteraciones y 25 ciclos.

Algoritmo	Ciclos	Nro de ejec.	Fitness	Tiempo (s)
PSO W	50	1	8.184×10^{-3}	166 885,62
PSO W	50	2	8.205×10^{-3}	163 726,92
PSO W	50	3	7.845×10^{-3}	158 120,57
PSO W	50	4	7.839×10^{-3}	160 364,83
PSO W	50	5	7.788×10^{-3}	160 274,04
PSO W	50	6	7.942×10^{-3}	160 008,74
PSO W	50	7	7.765×10^{-3}	158 279,08
PSO W	50	8	7.750×10^{-3}	157 834,81
PSO X	50	1	7.668×10^{-3}	153 874,80
PSO X	50	2	7.778×10^{-3}	154 552,54
PSO X	50	3	8.001×10^{-3}	156 322,70
PSO X	50	4	8.053×10^{-3}	159 403,08
PSO X	50	5	7.755×10^{-3}	153 751,55
PSO X	50	6	7.716×10^{-3}	154 798,85
PSO X	50	7	7.771×10^{-3}	154 340,74
PSO X	50	8	7.893×10^{-3}	156 742,07
RS	50	1	7.912×10^{-3}	172 039,76
RS	50	2	8.202×10^{-3}	171 437,91
RS	50	3	8.262×10^{-3}	172 395,17
RS	50	4	8.285×10^{-3}	172 114,20
RS	50	5	8.234×10^{-3}	172 687,68
RS	50	6	8.380×10^{-3}	173 280,76
RS	50	7	8.250×10^{-3}	172 110,98
RS	50	8	8.193×10^{-3}	171 677,82

Tabla A.9: Resultados de la evaluación para 100 iteraciones y 50 ciclos.

Algoritmo	Ciclos	Iteraciones	Nro de ejec.	Fitness	Tiempo (s)
PSO W	25	100	1	8.954×10^{-3}	78 336,19
PSO W	25	100	2	9.099×10^{-3}	79 805,79
PSO W	25	100	3	8.809×10^{-3}	77 893,09
PSO W	25	100	4	9.112×10^{-3}	79 575,02
PSO W	25	100	5	9.141×10^{-3}	81 545,87
PSO W	25	100	6	9.252×10^{-3}	81 113,18
PSO W	25	100	7	9.020×10^{-3}	78 906,24
PSO W	25	100	8	9.198×10^{-3}	80 923,59
PSO X	25	100	1	9.311×10^{-3}	82 266,91
PSO X	25	100	2	9.021×10^{-3}	78 099,64
PSO X	25	100	3	8.749×10^{-3}	77 776,36
PSO X	25	100	4	9.215×10^{-3}	79 177,77
PSO X	25	100	5	9.287×10^{-3}	87 223,53
PSO X	25	100	6	9.157×10^{-3}	87 184,24
PSO X	25	100	7	9.259×10^{-3}	87 930,90
PSO X	25	100	8	8.945×10^{-3}	85 039,46
RS	25	100	1	9.249×10^{-3}	87 171,71
RS	25	100	2	9.681×10^{-3}	86 530,75
RS	25	100	3	9.798×10^{-3}	87 174,63
RS	25	100	4	9.676×10^{-3}	86 868,08
RS	25	100	5	9.515×10^{-3}	86 880,72
RS	25	100	6	9.516×10^{-3}	87 357,00
RS	25	100	7	9.484×10^{-3}	87 029,27
RS	25	100	8	9.591×10^{-3}	86 674,12

Tabla A.10: Resultados de la evaluación para 25 ciclos y 100 iteraciones.

Algoritmo	Ciclos	Iteraciones	Nro de ejec.	Fitness	Tiempo (s)
PSO W	25	200	1	9.167×10^{-3}	160 517,86
PSO W	25	200	2	9.291×10^{-3}	158 424,06
PSO W	25	200	3	9.065×10^{-3}	158 281,14
PSO W	25	200	4	9.105×10^{-3}	158 322,34
PSO W	25	200	5	9.261×10^{-3}	159 904,87
PSO W	25	200	6	8.875×10^{-3}	155 901,60
PSO W	25	200	7	8.806×10^{-3}	153 155,51
PSO W	25	200	8	9.133×10^{-3}	158 152,17
PSO X	25	200	1	9.027×10^{-3}	160 296,56
PSO X	25	200	2	9.121×10^{-3}	162 781,67
PSO X	25	200	3	8.843×10^{-3}	160 651,67
PSO X	25	200	4	8.737×10^{-3}	157 920,81
PSO X	25	200	5	8.806×10^{-3}	157 497,22
PSO X	25	200	6	8.766×10^{-3}	158 864,46
PSO X	25	200	7	9.056×10^{-3}	157 437,50
PSO X	25	200	8	8.875×10^{-3}	159 335,29
RS	25	200	1	9.076×10^{-3}	175 695,84
RS	25	200	2	9.577×10^{-3}	175 618,93
RS	25	200	3	9.550×10^{-3}	175 283,41
RS	25	200	4	9.487×10^{-3}	175 683,87
RS	25	200	5	9.140×10^{-3}	175 564,50
RS	25	200	6	9.690×10^{-3}	176 104,75
RS	25	200	7	9.019×10^{-3}	177 861,35
RS	25	200	8	9.048×10^{-3}	176 064,89

Tabla A.11: Resultados de la evaluación para 25 ciclos y 200 iteraciones.

Bibliografía

[1] N. A. Ab Aziz and Z. Ibrahim. Asynchronous particle swarm optimization for swarm robotics. *Procedia Engineering*, 41:951–957, 2012.

[2] M. Abido. Optimal power flow using particle swarm optimization. *International Journal of Electrical Power & Energy Systems*, 24(7):563–571, 2002.

[3] V. M. Alberto Colorni, Marco Dorigo. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[4] G. D. Anthony Carlisle. An off-the-shelf pso. In *Proceedings of The Workshop On Particle Swarm Optimization*, pages 1–6. Indianapolis, USA, 2001.

[5] J. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham. Inertia weight strategies in particle swarm optimization. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, pages 633–640. IEEE, 2011.

[6] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[7] C. Blum and M. Manfrin. Metaheuristics network. `http://www.metaheuristics.net`. (último acceso Mayo 2016).

[8] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35, 2003.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[10] M. Clerc. Particle swarm central. `http://www.particleswarm.info/Programs.html`. (último acceso Mayo 2016).

[11] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[12] P. Curto-Risso, A. Medina, and A. C. Hernández. Optimizing the operation of a spark ignition engine: Simulation and theoretical tools. *Journal of Applied Physics*, 105(9):094904, 2009.

[13] P. L. Curto-Risso, A. Medina, and A. C. Hernández. Theoretical and simulated models for an irreversible otto cycle. *Journal of Applied Physics*, 104(9):094911, 2008.

[14] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley. Particle swarm optimization: basic concepts, variants and applications in power systems. *Evolutionary Computation, IEEE Transactions on*, 12(2):171–195, 2008.

[15] R. C. Eberhart and X. Hu. Human tremor analysis using particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[16] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 84–88. IEEE, 2000.

[17] A. Engelbrecht. Particle swarm optimization: Velocity initialization. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[18] M. L. Fred Glover. *Tabu search*, volume 22. Springer, 1997.

[19] J. García-Nieto and E. Alba. Why six informants is optimal in pso. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 25–32. ACM, 2012.

[20] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[21] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. *AAAS Publications, Washington, DC*, 1990.

[22] X. Hu. Particle swarm optimization. `http://www.swarmintelligence.org/index.php`, 2006. (último acceso Mayo 2016).

[23] T. Huang and A. S. Mohan. Significance of neighborhood topologies for the reconstruction of microwave images using particle swarm optimization. In *Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings*, volume 1, pages 4–pp. IEEE, 2005.

[24] J. Kennedy. The behavior of particles. In *Evolutionary programming VII*, pages 579–589. Springer, 1998.

[25] J. Kennedy. Stereotyping: improving particle swarm performance with cluster analysis stereotyping: improving particle swarm performance with cluster analysis. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 1507–1512, 2000.

[26] J. Kennedy and R. Eberhart. Particle swarm optimization. *In Proceedings of IEEE International Conference on Neural Networks*, 4:1942–1948, 1995.

[27] J. Kennedy and R. C. Eberhart. *Swarm Intelligence.* Morgan Kaufman, 2001.

[28] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1671–1676. IEEE, 2002.

[29] W. F. Leong. *Multiobjective particle swarm optimization: Integration of dynamic population and multiple-swarm concepts and constraint handling.* ProQuest, 2008.

[30] S. Y. Lim, M. Montakhab, and H. Nouri. A constriction factor based particle swarm optimization for economic dispatch. 2009.

[31] G. Litak, T. Kamiński, J. Czarnigowski, A. K. Sen, and M. Wendeker. Combustion process in a spark ignition engine: analysis of cyclic peak pressure and peak pressure angle oscillations. *Meccanica*, 44(1):1–11, 2009.

[32] G. Litak, R. Taccani, R. Radu, K. Urbanowicz, J. A. Hołyst, M. Wendeker, and A. Giadrossi. Estimation of a noise level using coarse-grained entropy of experimental time series of internal pressure in a combustion engine. *Chaos, Solitons & Fractals*, 23(5):1695–1701, 2005.

[33] W. Lu, H. Fan, A. Leung, and J. Wong. Analysis of pollutant levels in central hong kong applying neural network method with particle swarm optimization. *Environmental monitoring and assessment*, 79(3):217–230, 2002.

[34] A. Medina, P. L. Curto-Risso, A. C. Hernández, L. Guzmán-Vargas, F. Angulo-Brown, and A. K. Sen. *Quasi-Dimensional Simulation of Spark Ignition Engines: From Thermodynamic Optimization to Cyclic Variability.* Springer Publishing Company, Incorporated, 2013.

[35] M. M. Millonas. Swarms, phase transitions, and collective intelligence; and a nonequilibrium statistical field theory of swarms and other spatially extended complex systems. Technical report, 1993.

[36] L. Mussi, S. Cagnoni, and F. Daolio. Empirical assessment of the effects of update synchronization in particle swarm optimization. In *Proceeding of the AIIA Workshop on Complexity, Evolution and Emergent Intelligence*, pages 1–10, 2009.

[37] A. Nowak, J. Szamrej, and B. Latané. From private attitude to public opinion: A dynamic theory of social impact. *Psychological Review*, 97(3):362, 1990.

[38] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[39] M. Pedemonte. Ant colony optimization para la resolución del problema de steiner generalizado. Master's thesis, Instituto de Computación - Facultad de Ingeniería Universidad de la República, 2009.

[40] J. Peng, Y. Chen, and R. Eberhart. Battery pack state of charge estimator design using computational intelligence approaches. In *Battery Conference on Applications and Advances, 2000. The Fifteenth Annual*, pages 173–177. IEEE, 2000.

[41] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[42] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel problem solving from nature-PPSN III*, pages 249–257. Springer, 1994.

[43] J. Rada-Vilela, M. Zhang, and W. Seah. A performance study on synchronous and asynchronous updates in particle swarm optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 21–28. ACM, 2011.

[44] C. W. Raynolds. Flocks, herds, and schools: A distributed behavioral model. *ACM Computers Graphics*, 21(4):25–34, 1987.

[45] W. T. Reeves. Particle systems–a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, Volume 17,N°3, 1983.

[46] J. Riget and J. Vesterstrøm. A diversity-guided particle swarm optimizer - the arpso. Technical report, 2002.

[47] J. J. Romero and P. Machado. *The art of artificial evolution: a handbook on evolutionary art and music.* Springer Science & Business Media, 2007.

[48] F. Rothlauf. *Design of Modern Heuristics: Principles and Application.* Springer, 2011.

[49] A. C. Rovira. Particle swarm optimization (pso). `http://tracer.uc3m.es/tws/pso/neighborhood.html`, 2005. (último acceso Mayo 2016).

[50] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

[51] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary programming VII*, pages 591–600. Springer, 1998.

[52] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[53] D. Srinivasan, W. H. Loo, and R. L. Cheu. Traffic incident detection using particle swarm optimization. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 144–151. IEEE, 2003.

[54] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[55] E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[56] G. S. Tewolde, D. M. Hanna, and R. E. Haskell. Enhancing performance of pso with automatic parameter tuning technique. In *Swarm Intelligence Symposium, 2009. SIS'09. IEEE*, pages 67–73. IEEE, 2009.

[57] F. Van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):225–239, 2004.

[58] F. Van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information sciences*, 176(8):937–971, 2006.

[59] S. Voss, I. H. Osman, and C. Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization.* Kluwer Academic Publishers, 1999.

[60] S. Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1), 1984.

[61] S. Xue, J. Zhang, and J. Zeng. Parallel asynchronous control strategy for target search with swarm robots. *International Journal of Bio-Inspired Computation*, 1(3):151–163, 2009.

[62] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian. On the convergence analysis and parameter selection in particle swarm optimization. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 3, pages 1802–1807. IEEE, 2003.

Lista de acrónimos

AC ..... Autómata Celular

ACO ..... Ant Colony Optimization

ARPSO ..... Attractive and Repulsive Particle Swarm Optimization

BC ..... Bee Colony

BLS ..... Basic Local Search

CCGA ..... Cooperative Coevolutionary Genetic Algorithm

CPSO ..... Cooperative Particle Swarm Optimization

GA ..... Genetic Algorithms

GRASP ..... Greedy Randomized Adaptative Search Procedure

PSO ..... Particle Swarm Optimization

RS ..... Random Search

SI ..... Swarm Intelligence

TS ..... Tabu Search