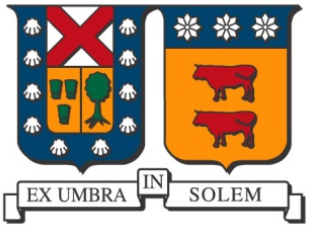
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“DISEÑO E IMPLEMENTACIÓN DE CRITERIOS DE ACEPTACIÓN PARA LA CONFIGURACIÓN AUTOMÁTICA DE ALGORITMOS UTILIZANDO IRACE”

DIANA PATRICIA GIL SILVESTRE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Nicolás Rojas Morales
Profesor Correferente: Leslie Pérez Cáceres

Mayo - 2024

DEDICATORIA

Dedico este trabajo a mi mamá, por apoyarme siempre y por el amor incondicional que me ha brindado. A mi papá, por todo el trabajo y sacrificio que haces todos los días y por guiarme para ser la persona que soy hoy. Les debo esto y más.

AGRADECIMIENTOS

Primero, quiero agradecer a mis padres, que no solo hicieron posible esta oportunidad de terminar mi carrera, sino que también me dieron en todo momento el soporte necesario para seguir adelante, aún estando en un lugar muy lejos de nuestro país. Siempre estuvieron para mí y siendo yo su prioridad número 1. Su amor y apoyo han sido fundamentales para mi éxito y felicidad, y no puedo expresar con palabras cuánto significan para mí.

Quiero agradecer también a mi profesor guía, Nicolás Rojas, por brindarme la oportunidad de trabajar en este tema, por su dedicación, orientación y apoyo continuo. Gracias a él, he aprendido muchísimo y pude adentrarme al mundo de la investigación. Siempre se preocupó por mi aprendizaje y se aseguró de que cada paso que diera en este proyecto me ofreciera numerosas oportunidades de crecimiento. También agradezco a la profesora Leslie Pérez de la PUCV, que me ayudó en la realización del trabajo y nos facilitó recursos para completar los experimentos.

Agradezco inmensamente a mi pololo, Fernando, por el ánimo, el amor y el apoyo incondicional durante todos estos años, que siempre vino en el momento adecuado y sin él, el camino hubiese sido casi imposible.

A toda mi familia que me acompaña día a día, mi hermana, mi tía, Sara, Rodrigo.

A mis compañeros, Dylan, Jeremy, Tomás, Gabi y Vania que fueron los mejores compañeros de U e hicieron de esta etapa la mejor. A los Kenai por siempre estar en las buenas y las malas, y que son una parte muy importante del trayecto.

Gracias a todos.

RESUMEN

Resumen— Las metaheurísticas requieren definir parámetros adecuados, ya que su rendimiento depende drásticamente de los valores asignados a los parámetros. El algoritmo irace es un método de configuración de algoritmos que utiliza eficazmente los recursos computacionales disponibles para buscar valores de parámetros adecuados. Sin embargo, irace puede sufrir de convergencia prematura, evaluando configuraciones de parámetros que son similares entre sí en términos de sus valores. Este trabajo propone una estrategia para aumentar la exploración realizada por irace utilizando dos algoritmos de clustering, grid-based y k-medoids. La idea es agrupar configuraciones élite según las áreas del espacio de parámetros y seleccionar configuraciones de estos grupos para influir en el muestreo de nuevas configuraciones. Para evaluar nuestra propuesta, configuramos el conocido marco de optimización de colonia de hormigas considerando tres escenarios: un escenario homogéneo (TSP con 2000 ciudades), un escenario ligeramente menos homogéneo (TSP con 1000-3000 ciudades) y un escenario heterogéneo (QAP con dos niveles de dispersión). Los resultados muestran que incluir nuestra estrategia permite aumentar el nivel de exploración de irace y en algunos casos, obtener mejores resultados, alcanzando configuraciones que son estructuralmente diferentes, pero con compensaciones de rendimiento dependientes del escenario.

Palabras Clave— Metaheurísticas, Configuración de Algoritmos, Carrera Iterada, irace, Clustering

ABSTRACT

***Abstract*—** Metaheuristics require defining appropriate parameters, as their performance drastically depends on the values assigned to the parameters. The irace algorithm is a well-known and powerful algorithm configuration method that effectively utilizes available computational resources to automatically search for suitable parameter values. However, irace can suffer from premature convergence, suggesting and evaluating parameter configurations that are similar to each other in terms of their parameter values. This work proposes a strategy to increase the exploration performed by irace using two clustering algorithms, grid-based and k-medoids. The idea is to group elite configurations according to the areas of the parameter space to which they belong and to select parent configurations from these groups, influencing the sampling of new configurations and extending the exploration

capabilities of irace. To evaluate our proposal, we configure the well-known Ant Colony Optimization framework considering three scenarios: a homogeneous scenario (TSP instances with 2000 cities), a slightly less homogeneous scenario (TSP instances with 1000-3000 cities), and a heterogeneous scenario (QAP instances with two levels of dispersion). The results show that including our strategy allows increasing the level of exploration of irace and in some cases obtaining better results, reaching configurations that are structurally different but have scenario-dependent performance trade-offs.

Keywords— Metaheuristics, Algorithm Configuration, Iterated Racing, irace, Clustering

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los algoritmos meta-heurísticos son técnicas ampliamente utilizadas en la resolución de problemas de optimización. Aplicar estos algoritmos requiere establecer los valores de sus parámetros; un conjunto de elecciones de diseño que permiten definir el comportamiento de búsqueda del algoritmo. Los valores de los parámetros son cruciales para el rendimiento de los algoritmos, y el proceso de determinar estos valores se conoce como el Problema de Configuración de Algoritmos.

Considerando que las meta-heurísticas son intrínsecamente estocásticas, el proceso de obtener valores de parámetros adecuados requiere realizar múltiples ejecuciones sobre un conjunto de instancias. Además, los valores de los parámetros adecuados dependen del problema, ya que no hay configuraciones universales que permitan resolver todo tipo de problemas y además, los parámetros pueden tener relaciones desconocidas y difíciles de entender. Por lo tanto, el problema de configuración del algoritmo es una tarea difícil de resolver y que consume mucho tiempo.

El sintonizador irace es un método de ajuste bien conocido y se ha aplicado con éxito en diferentes contextos. El proceso de búsqueda aplicado por irace implementa un procedimiento de carrera iterada. En cada iteración (carrera), un conjunto de configuraciones candidatas se evalúa progresivamente en un conjunto de instancias del problema. Las configuraciones candidatas se descartan de la carrera, deteniendo su evaluación, tan pronto como haya suficiente evidencia estadística de su bajo rendimiento. En cada carrera, las nuevas configuraciones candidatas se muestrean a partir de un conjunto de modelos de muestreo local. Estos modelos están asociados con configuraciones sobrevivientes de carreras anteriores y son heredados por las configuraciones recién generadas. En cada iteración, se seleccionan las mejores configuraciones entre las sobrevivientes (configuraciones élite) y se actualizan sus modelos, enfocando el muestreo en el área del espacio de parámetros definida por sus valores de parámetros. Esta estrategia permite a irace converger, enfocando progresivamente el presupuesto computacional disponible en áreas prometedoras del espacio de búsqueda de parámetros. Sin embargo, este proceso de selección e intensificación puede generar configuraciones demasiado similares, especialmente después de algunas iteraciones, reduciendo la exploración que irace puede realizar.

En este trabajo, proponemos incluir técnicas de clustering para agrupar configuraciones élite y evitar el estancamiento temprano, modificando el mecanismo de selección de modelos predeterminado de irace. El objetivo es ampliar las capacidades de exploración de irace, tratando de evitar converger a configuraciones similares demasiado pronto en la búsqueda, sin disminuir su potencial para alcanzar configuraciones de parámetros adecuadas. En particular, evaluamos la inclusión de un algoritmo de clustering grid-based y un algoritmo de k-medoids. Para evaluar nuestra estrategia, utilizamos como referencia conocidos marcos de optimización de colonias de hormigas para resolver: (a) ACOTSP, utilizando un escenario con instancias de referencia con 2000 ciudades, (b) ACOTSP con instancias considerando en-

tre 1000 y 3000 ciudades, y (c) el Problema de Asignación Cuadrática considerando diversas instancias de dispersión con 60 elementos.

Las contribuciones de este trabajo son:

- La propuesta de utilizar dos algoritmos de clustering en irace.
- La evaluación de la estrategia propuesta en la sintonización de algoritmos conocidos.

En esta memoria, se evalúa el agrupamiento solo en dos parámetros numéricos, eligiendo aquellos con el menor número posible de valores. El objetivo es analizar si esta decisión produce no solo una exploración de los parámetros seleccionados, sino también, afecta los niveles de exploración en todo el espacio de búsqueda de parámetros.

Este documento está organizado de la siguiente manera. Inicialmente, en el capítulo 1 se define el problema a abordar en la presente memoria y los objetivos tanto generales como específicos. Luego, en el capítulo 2 se realiza una revisión de la literatura asociada a la sintonización de parámetros, los principales algoritmos y una descripción del proceso del algoritmo irace. Además, se definen los algoritmos de clustering a utilizar. El capítulo 3 presenta la propuesta de solución, sus fundamentos y la metodología aplicada para resolver el problema. En el capítulo 4 se describen los escenarios experimentales empleados para validar la propuesta. Posteriormente, en la sección 4.2 se muestran los resultados obtenidos de los experimentos y se realiza un análisis de lo obtenido. Finalmente, en el capitulo 5 se presentan las conclusiones de trabajo realizado.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

Una meta-heurística es un marco algorítmico de alto nivel que utiliza técnicas heurísticas para abordar problemas combinatorios, ya sean de naturaleza continua o discreta. Estos problemas pueden ser tanto multi-objetivo como mono-objetivo. También son utilizadas para resolver problemas de satisfacción de restricciones. Entre las metaheurísticas más utilizadas en el contexto de la resolución de problemas de optimización están los *Genetic Algorithms* [Holland, 1975], [Goldberg, 1989], [Srinivas y Patnaik, 1994], *Simulated Annealing* [Kirkpatrick *et al.*, 1983], *Tabu Search* [Glover, 1989], *Particle swarm optimization* [Kennedy y Eberhart, 1995] y *Ant-colony optimization* [Dorigo *et al.*, 1991], [Dorigo *et al.*, 1996], [Dorigo *et al.*, 2006].

Las metaheurísticas suelen estar parametrizadas, lo que significa que definen un conjunto de parámetros que deben asignarse a valores concretos para su ejecución. La definición y valores de estos parámetros varían según el algoritmo en cuestión; por ejemplo, el tamaño de la población para los *Evolutionary Algorithms* o la longitud de la lista tabú en un algoritmo *Tabu Search*. Los valores asignados a estos parámetros desempeñan un papel crucial en la calidad de las soluciones obtenidas. Esto se traduce tanto en el valor de la función de evaluación del algoritmo objetivo como en el tiempo de ejecución. La asignación de estos valores se considera, en sí misma, un problema de optimización complejo y se reconoce como una tarea que consume mucho tiempo [Montero *et al.*, 2014]

A raíz de lo anterior, en la literatura se distinguen dos clases de métodos que permiten asignar valores a los parámetros de un algoritmo: los métodos de control de parámetros y los métodos de sintonización de parámetros. Los métodos de control de parámetros consisten en determinar los valores de los parámetros de forma dinámica durante la ejecución del algoritmo. Típicamente, se inicia con una asignación de parámetros y estos se van ajustando en tiempo de ejecución. Por otro lado, las estrategias de sintonización de parámetros o de configuración automática de algoritmos se encargan de automatizar la asignación de parámetros para un conjunto de instancias y un algoritmo en particular. Esto permite determinar aquel conjunto de valores que resulta en un mejor desempeño para la ejecución posterior del algoritmo objetivo con instancias no vistas para la métrica que se desee optimizar [Eiben *et al.*, 1999].

Actualmente, los métodos de sintonización de parámetros o configuración automática se han vuelto fundamentales para el rendimiento de los algoritmos que resuelven problemas de optimización combinatoria, ya que han demostrado significar una mejora en el desempeño de los mismos y facilita su diseño. Hasta la fecha, existen varios enfoques de sintonización de parámetros. Algunos ejemplos de sintonizadores ampliamente utilizados incluyen ParamILS [Hutter *et al.*, 2009], EVOCA [Riff y Montero, 2013], F-RACE [Birattari *et al.*, 2002], irace [López-Ibáñez *et al.*, 2016] y SMAC [Hutter *et al.*, 2011]. En este memoria, se trabajará con

el algoritmo de configuración irace.

El algoritmo de configuración automática irace se ha destacado como una herramienta ampliamente utilizada para configuraciones de buena calidad. Este método tiene como base los componentes principales de los algoritmos de *racing*, considerando la evaluación de un conjunto de instancias durante la sintonización para poder estimar con la menor varianza posible la mejor configuración. irace inicia el proceso de configuración seleccionando configuraciones iniciales de manera aleatoria y uniforme del espacio de búsqueda, para luego evaluarlas en un conjunto de instancias dado. Posteriormente, aquellas configuraciones de parámetros para las cuales se ha reunido suficiente evidencia estadística de que su desempeño es significativamente peor que otra configuración, son descartadas. Luego, las configuraciones sobrevivientes participan de la generación de nuevas configuraciones para repetir el proceso iterativamente hasta cumplir con un máximo de tiempo o límite de computación.

irace, junto a otros algoritmos de configuración, tienen limitaciones relacionadas con las regiones de espacios de parámetros que son visitadas durante la ejecución del algoritmo de configuración [Ye *et al.*, 2022]. Para definir estas limitaciones, es útil abordar los conceptos de 'exploración' y 'explotación'. La exploración también se define como diversificación y corresponde a la habilidad de visitar varias y distintas regiones del espacio de búsqueda. En cambio, explotación o intensificación generalmente se refiere a la habilidad de obtener soluciones de alta calidad dentro de dichas regiones [Lozano y García-Martínez, 2010].

Usualmente, los diseñadores de algoritmos o estrategias de configuración intentan balancear el potencial de exploración y explotación de sus algoritmos, junto con el tiempo de ejecución para poder determinar un conjunto de configuraciones de buen desempeño dado cierto límite de tiempo de ejecución total [Anastacio *et al.*, 2019]. Particularmente, se ha observado en algunos casos que irace puede converger prematuramente hacia ciertas regiones del espacio de parámetros [López-Ibáñez *et al.*, 2016]. Esta convergencia temprana puede limitar la capacidad de encontrar una mayor variedad de configuraciones con buen desempeño en otras regiones.

irace se ha mantenido al nivel de otros algoritmos de vanguardia en el área del problema de sintonización de parámetros, lo que lo convierte en una elección ideal para implementar criterios de aceptación que permitan aumentar su capacidad de exploración del espacio.

Es a partir de esta consideración que se centra la propuesta de esta memoria, que consiste en la introducción de criterios de aceptación para la selección y descarte de configuraciones, con el objetivo de prevenir la convergencia prematura y fomentar una exploración más robusta del espacio de parámetros, mejorando así el proceso de sintonización y la calidad de las soluciones obtenidas por el algoritmo objetivo usando las configuraciones encontradas por irace.

En relación a lo mencionado, se definen a continuación los objetivos y alcances de este trabajo. El objetivo general de este trabajo corresponde a diseñar e implementar un conjunto de criterios de aceptación para el algoritmo de sintonización de parámetros de Iterated Racing (irace), con el propósito de mejorar la eficiencia de la configuración automática de algoritmos. Los objetivos específicos son los siguientes:

- Estudiar el Estado del Arte para el problema de sintonización de parámetros hasta la fecha, junto con las variantes desarrolladas de irace.
- Definir y diseñar criterios de aceptación que puedan ser incorporados en el algoritmo irace, considerando la naturaleza del problema y los componentes más importantes del algoritmo.
- Analizar y comparar los resultados obtenidos en términos de la calidad de las configuraciones encontradas por la solución propuesta, con los resultados obtenidos por irace y sus variantes.

CAPÍTULO 2
MARCO CONCEPTUAL

Los algoritmos meta-heurísticos son ampliamente utilizados en la resolución de problemas de optimización. Estos algoritmos requieren la definición de un conjunto de valores para ciertos parámetros necesarios para su ejecución. Estos parámetros pueden llegar a ser determinantes para el propósito de los componentes del algoritmo, la calidad de las soluciones obtenidas y el tiempo de ejecución. Los parámetros de un algoritmo pueden clasificarse en tres categorías [López-Ibáñez *et al.*, 2016]:

- Numéricos: Corresponden a valores reales o enteros. Un ejemplo de un parámetro numérico es la tasa de mutación de un algoritmo evolutivo.
- Categóricos: Los parámetros categóricos son valores discretos sin un orden explicito. Definen procedimientos o funciones que son implementadas por el algoritmo, tales como el movimiento utilizado en un algoritmo de búsqueda local.
- Ordinales: Son parámetros categóricos que definen un orden entre sus posibles valores, e.g un parámetro con los valores alto, medio y bajo.

El proceso de determinar los valores de los parámetros es una tarea compleja y que consume mucho tiempo, tal que se requiere ejecutar el algoritmo objetivo en múltiples ocasiones. Naturalmente, estos valores a definir están fuertemente relacionados con el problema con el cual se está trabajando. Por ende, donde no existen configuraciones universales que nos permitan resolver toda clase de problemas con un buen desempeño. Además, parámetros pueden además tener relaciones desconocidas entre sí. Como consecuencia, estas dificultades dan lugar a la dificultad del problema de configuración de algoritmos o definición de parámetros [Bezerra *et al.*, 2017].

Se han desarrollado múltiples soluciones a este problema, siendo las estrategias de configuración automática aquellas con un amplio panorama de investigación. A pesar de sus diversas metodologías, comparten un único objetivo, el cual es buscar automáticamente la mejor configuración para los valores de parámetros de algoritmos de optimización. La utilidad de estas estrategias ha sido evidente en diversos contextos, tales como el diseño de algoritmos [Riff y Montero, 2013].

Antes de adentrarnos en la propuesta de solución de esta memoria, es esencial llevar a cabo una revisión exhaustiva de la literatura y definir el marco teórico. Es necesario contextualizar sobre el problema de configuración de algoritmos y sus componentes, así como abordar las dificultades inherentes al mismo. Además, se proporciona una definición detallada de las estrategias de control y sintonización de parámetros. En paralelo, resulta fundamental establecer una definición formal del algoritmo de sintonización irace, que será el foco de

este trabajo. Este capítulo detalla su funcionamiento general junto con sus características principales. Finalmente, se presenta un análisis del trabajo realizado con irace hasta la fecha disponible en la literatura, resaltando los aspectos clave de cada propuesta, los escenarios en los cuales fueron validadas y los resultados obtenidos.

2.1. El problema de configuración automática de algoritmos

Formalmente, se define el problema de configuración automática de algoritmos como:

- Un algoritmo objetivo A el cual está parametrizado, es decir, posee una serie de parámetros que pueden ajustarse para influir en su comportamiento y rendimiento.
- Un conjunto de configuraciones Θ, donde una configuración θ implica una asignación completa de valores a todos los parámetros del algoritmo A
- Un conjunto de instancias I, que se subdivide a su vez en instancias de entrenamiento y de prueba, en las cuales se evaluarán las configuraciones durante la búsqueda (instancias de entrenamiento) y posterior a la búsqueda (instancias de prueba).
- Una métrica de costo $c(\theta, i)$ que se utiliza para medir el rendimiento del algoritmo A bajo una configuración θ específica, frente a una instancia de problema i. La métrica de costo usualmente puede estar relacionada con el tiempo de ejecución (necesario para resolver una instancia de problema) o con la calidad de la mejor solución obtenida.
- Un presupuesto de computación B, que establece un límite en los recursos disponibles para sintonizar el algoritmo objetivo A (usualmente definido en torno al número de evaluaciones o tiempo total de ejecución).

El problema de configuración consiste en asignar valores únicos a los parámetros de A. El objetivo es encontrar $\theta \in \Theta$, tal que se optimice el valor de c para θ e I, bajo un presupuesto B.

Durante muchos años, la resolución de este problema era realizada eligiendo a mano los valores de los parámetros u optimizando los parámetros durante el tiempo de ejecución del algoritmo, proceso que sigue ocurriendo hasta el día de hoy [López-Ibáñez *et al.*, 2016]. Sin embargo, esto puede ser una tarea tediosa y que consume mucho tiempo, donde no existe una configuración universal que funcione para todas las clases de problemas y para todas las instancias, es decir, depende del problema en particular. Además, los parámetros pueden tener una relación no-líneal o incluso desconocida entre ellos [Riff y Montero, 2013]. Este problema dio origen a las estrategias de control de parámetros y las estrategias de sintonización de parámetros. A continuación, se definen las estrategias mencionadas.

2.2. Estrategias de control de parámetros

Las estrategias de control de parámetros se refieren a técnicas utilizadas para ajustar los parámetros de un algoritmo de forma dinámica durante la ejecución del algoritmo objetivo. Normalmente un componente del algoritmo de control selecciona un conjunto de valores para el algoritmo objetivo, de modo que este se ejecute durante un tiempo determinado y luego se retorne una medida de rendimiento. De esta manera, el método de control evalúa qué tan acertada fue la elección de parámetros. Estos pasos se repiten de manera iterativa, de forma que se optimice la medida de costo o rendimiento al final del proceso de resolución del problema [Gomes Pereira de Lacerda *et al.*, 2021].

Se pueden clasificar las estrategias de control en dos categorías: los métodos diseñados específicamente para una aplicación o algoritmo particular, que se aplican en un contexto específico y controlan parámetros específicos del algoritmo para el cual fueron diseñados; y los métodos de control general, que pueden aplicarse a cualquier algoritmo sin necesidad de una estructura de parámetros en particular [Karafotias *et al.*, 2015].

Algunas de las ventajas del control de parámetros definidas por [Karafotias *et al.*, 2015] corresponden a:

- Permite que el algoritmo ajuste los valores de parámetros a los más apropiados para diferentes etapas del proceso de búsqueda.
- Permite que el algoritmo se adapte a los cambios de los *fitness landscape*, que corresponde al paisaje de la región de posibles soluciones al problema.
- Permite que el algoritmo recolecte información sobre el *fitness landscape* durante la búsqueda y utilice la información acumulada para mejorar el rendimiento en etapas posteriores.

Un ejemplo clásico de una estrategia de control de parámetros corresponde al control de temperatura efectuado en el algoritmo de *Simmulated Annealing* para adaptar las capacidades del algoritmo de intensificar y diversificar en el espacio de soluciones [Kirkpatrick *et al.*, 1983].

2.3. Estrategias de sintonización de parámetros

Un algoritmo de sintonización de parámetros tiene como objetivo, dado un algoritmo parametrizado y un conjunto de instancias, encontrar los mejores valores de los parámetros para los cuales el algoritmo obtenga el mejor rendimiento para un conjunto de instancias no vistas por el mismo. Para ello, se deben definir dos conjuntos de instancias que serán utilizadas

para el proceso de sintonización y para el proceso de *testing* o prueba del algoritmo con los parámetros encontrados en el paso anterior [Huang *et al.*, 2020].

A la fecha, se han desarrollado múltiples estrategias para la sintonización de parámetros, las cuales se pueden clasificar de acuerdo a las siguientes definiciones [Hoos, 2011]:

- Métodos *Model-free*: Estos métodos se basan en su mayoría en heurísticas; elección de parámetros al azar o aplicando algún operador sobre vectores de parámetros existentes. Si bien, representaron los primeros avances para el problema, estos métodos tienen un potencial de exploración del espacio de parámetros limitado. Se enfocan principalmente en optimizar parámetros numéricos. Los más predominantes son ParamILS [Hutter *et al.*, 2009], Meta-EAs [Grefenstette, 1986], GGAs [Ansótegui *et al.*, 2009] y EVOCA [Riff y Montero, 2013].

- Métodos de *Racing*: Los métodos basados en *racing* buscan de manera iterativa el mejor conjunto de parámetros para un algoritmo objetivo, eliminando configuraciones con mal rendimiento en cada iteración, las cuales simulan una carrera entre configuraciones. Un test estadístico es utilizado para reunir información sobre las configuraciones prometedoras y aquellas que deben descartarse. Estos métodos trabajan con un criterio de término dependiente de las configuraciones restantes o un presupuesto de configuración B. Además, entre las variantes más conocidas están F-Race [Birattari *et al.*, 2002], I/F-Race [Balaprakash *et al.*, 2007] y irace [López-Ibáñez *et al.*, 2016].

- Métodos Basados en Modelos: Los métodos de configuración *Model-Based* utilizan modelos surrogados para predecir aquellas configuraciones que parecen prometedoras y así poder enfocar el esfuerzo computacional en evaluar esas configuraciones, la cual es usualmente la componente más costosa de los algoritmos de configuración. Estos modelos ofrecen los prospectos de interpolar información del desempeño entre configuraciones de parámetros y extrapolar a regiones del espacio de parámetros no visitadas. Algunos métodos de configuración de esta clasificación son SBMO, SMAC [Hutter *et al.*, 2011], SPO [Bartz-Beielstein *et al.*, 2005] y sus variantes.

- Métodos basado en diseño de experimentos (DOE): Los métodos basados en diseño de experimentos consisten en un enfoque estadístico que implica el diseño de experimentos para modelar de forma empírica ciertos procesos. Lo anterior con el fin de comprender cómo los cambios en los valores de los parámetros de un algoritmo afectan su rendimiento [Gunawan *et al.*, 2011]. Un ejemplo de sintonizador perteneciente a esta categoría es el método *CALIBRA* [Adenso-Díaz y Laguna, 2006], que hace uso de diseño experimental y búsqueda local para favorecer la explotación y la exploración del proceso de configuración respectivamente.

Los métodos de sintonización mencionados se caracterizan por ser procesos automáticos que permiten focalizar el proceso de configuración para un problema en particular, que no

solo conlleva a mejorar el rendimiento del algoritmo objetivo, si no que también permite obtener un mayor entendimiento de como se relacionan los valores de los parámetros del mismo con el desempeño obtenido [Eiben y Smit, 2011].

2.4. El algoritmo de Iterated Racing

Un algoritmo de racing para la configuración de algoritmos representa una estrategia dinámica donde múltiples configuraciones compiten simultáneamente, siendo evaluadas de manera paralela en número predefinido de instancias. Este enfoque se fundamenta en la idea de llevar a cabo una búsqueda eficiente en el espacio de configuración, identificando configuraciones de mala calidad y priorizando aquellas que muestran ser prometedoras durante el proceso de sintonización [Balaprakash *et al.*, 2007]. Este proceso se ilustra en la figura 1, donde vemos gráficamente el proceso de evaluación de las configuraciones (θ_n) con el conjunto de instancias I_m, representadas por los nodos de la figura. El ejemplo evalúa al menos cinco instancias antes de aplicar el proceso de eliminación y luego se realiza tras la evaluación de una instancia. 'X' significa que no se realiza el proceso de eliminación, '–' significa que se descartó al menos una configuración, '$=$' significa que no se descartó ninguna configuración.

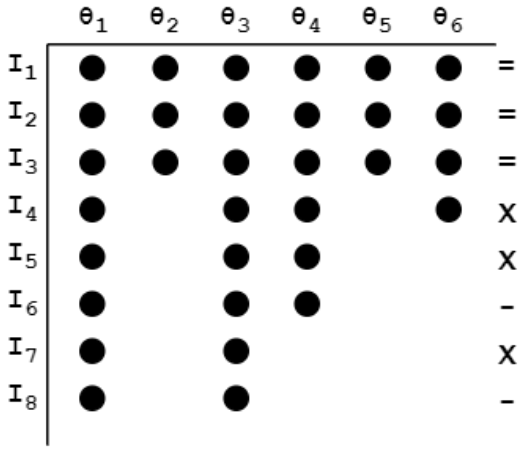


Figura 1: Diagrama del proceso de racing para la configuración automática de algoritmos. Fuente: Elaboración propia.

El primer algoritmo basado en racing fue propuesto por Birattari y otros, denominado F-race, por la naturaleza del test estadístico ejecutado (el análisis de varianza de dos vías de Friedman) [Birattari *et al.*, 2002]. Este método fue inspirado por la selección de modelos de Machine Learning usando cross-validation. F-race inicia su ejecución evaluando todas las configuraciones posibles del espacio de búsqueda, lo que es potencialmente costoso.

Luego, como mejora a la propuesta de Birattari, Balaprakash propuso F/I-RACE [Balaprakash *et al.*, 2007], método que implementa sampling e iterated racing, con el fin de adaptarse a problemas con grandes espacios de búsqueda y reducir las evaluaciones

del algoritmo objetivo.

Finalmente, Lopez-Ibánez y otros desarrollaron el irace package que se describe como una versión extendida del algoritmo de I/F-Race, que además utiliza el test denominado t-test como herramienta estadística durante el proceso de racing.[López-Ibáñez *et al.*, 2016]. Esta variante se caracteriza por implementar las ideas generales de racing, considerando el factor estocástico de la evaluación de los algoritmos en los procesos de búsqueda y evaluación del algoritmo. El algoritmo 1 muestra la estructura general de irace.

Algorithm 1 Algoritmo de irace

Require: $I = \{I_1, I_2, \ldots\} \subseteq I$, parameter space X, cost measure $C : X \times I \to \mathbb{R}$, tuning budget B

1: $\Theta_{\text{alive}} \leftarrow$ UniformSampling(X)
2: **while** $B_{\text{used}} < B_t$ **do**
3: $\quad \Theta_{\text{elite}} \leftarrow$ Race(Θ_{alive}, t)
4: $\quad \Theta_{\text{alive}} \leftarrow$ SelectElite(Θ_{elite})
5: $\quad t \leftarrow t + 1$
6: $\quad$ UpdateSamplingModel(Θ_{elite})
7: $\quad \Theta_{\text{new}} \leftarrow$ Sample()
8: $\quad \Theta_{\text{alive}} \leftarrow \Theta_{\text{new}} \cup \Theta_{\text{elite}}$
9: **end while**
10: **Output:** best configuration found from Θ_{elite}

El proceso general que describe el algoritmo consta de tres pasos que son detallados a continuación:

1. Generación de nuevas configuraciones: irace inicializa el set de configuraciones posibles muestreando uniformemente desde el espacio de búsqueda (línea 1, algoritmo 1). Para las iteraciones siguientes, cada parámetro de la configuración está asociado a una distribución de probabilidad (independientes entre sí). Estas distribuciones de muestreo, suelen ser una distribución normal en el caso de parámetros numéricos o una función discreta en el caso de parámetros categóricos. La actualización de estas distribuciones (línea 6, algoritmo 1) se realiza en base a las configuraciones de mejor desempeño, e implica ajustar la media y la desviación estándar (en el caso de distribuciones continuas) y/o modificar las probabilidades discretas en el caso de distribuciones discretas. Así, se genera una cantidad fija de configuraciones que se integran al proceso siguiente (línea 7, algoritmo 1).

2. Racing: Después de muestrear nuevas configuraciones de acuerdo con estas distribuciones, se utiliza un proceso de racing para seleccionar las configuraciones elite (línea 3, algoritmo 1). La carrera inicia con un número determinado de configuraciones. En cada paso de la carrera, estos candidatos se evalúan utilizando una sola instancia del problema. Luego de un cierto número de pasos, las configuraciones que estadísticamente tienen un peor rendimiento (que al menos una de las otras configuraciones)

se descartan, como se muestra en la figura 1. Esto se logra aplicando tests estadísticos (comúnmente el análisis de varianza de dos vías de Friedmann o el t-test), los cuales se usan como una heurística de selección por parte del algoritmo. La carrera continúa con las configuraciones que permanecen en la competencia. Este proceso de eliminación continúa hasta que se alcanza un número mínimo de configuraciones sobrevivientes, se ha agotado un máximo de instancias o se ha alcanzado un límite de recursos de cómputo, ya sea en términos de tiempo de ejecución o de experimentos realizados (línea 2, algoritmo 1).

3. Selección de las configuraciones sobrevivientes: Al final del proceso de racing, las configuraciones son ordenadas según el test realizado. En caso del F-test, se utiliza el *rank* como criterio. Para el caso del t-test, es el desempeño medio obtenido hasta el momento que determina el orden (línea 4, algoritmo 1). Así, se elige un número fijo de configuraciones utilizando *greedy-truncation selection* según el *rank* de cada configuración, las cuales continúan siendo evaluadas en la próxima iteración.

Los 3 pasos se iteran hasta que un criterio de término se satisface, el cual revisa si el presupuesto de computación B se ha agotado, o bien, si el número de configuraciones a evaluar al inicio de una carrera no es mayor que el número de configuraciones elite. El esquema de ejecución del algoritmo se ejemplifica mediante la figura 2. Esta figura describe los pasos principales del algoritmo. Primero, hay un proceso de racing que produce las configuraciones elite. Estas configuraciones influyen en el siguiente paso, que es la actualización de los modelos de muestreo. Finalmente, se generan nuevas configuraciones que se integran en el proceso de una nueva carrera.

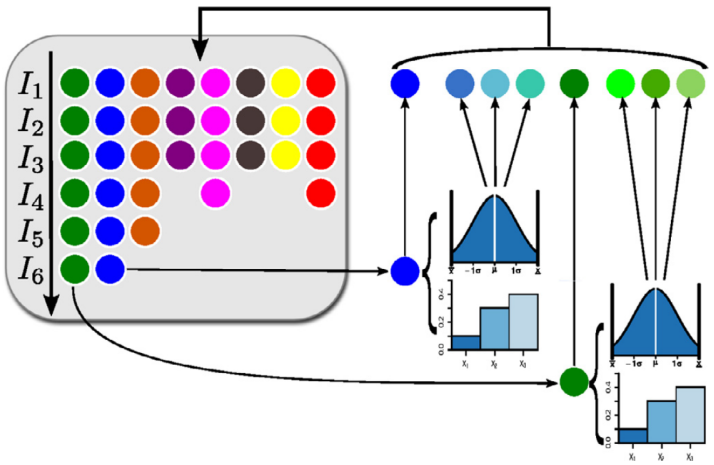


Figura 2: Esquema del algoritmo irace.
Fuente: The irace package [López-Ibáñez *et al.*, 2016]

La versión actual de irace implementa dos métodos cuyos objetivos son evitar la convergencia prematura del algoritmo y optimizar los recursos computacionales, respectivamente.

En primera instancia, se introduce un mecanismo de *soft-restart*. Un *restart* implica reiniciar el proceso de búsqueda y/o generación de nuevas soluciones para evitar el estancamiento y promover la exploración de una gama más amplia de soluciones. En contraste, el *soft-restart* no implica reiniciar la búsqueda completamente desde cero, si no que comienza desde un punto específico previamente alcanzado en el proceso. Concretamente, en irace con *soft-restart*, se ajustan las distribuciones de los parámetros de las configuraciones padres cuando la distancia con otra configuración generada es igual a cero. Este ajuste "retrocede" las distribuciones de muestreo de las configuraciones dos iteraciones atrás, seguido de la exclusión de aquellas configuraciones.

Por otra parte, se introduce una variante elitista de la carrera de irace. Esta variante, denominada *elitist iterated racing*, tiene como objetivo evitar que las configuraciones que hayan tenido un buen desempeño en carreras pasadas, sean descartadas por evaluaciones atípicas. Un ejemplo de lo anterior corresponde a obtener una configuración elite c en la primera iteración de irace, la cual para las primeras evaluaciones de la segunda iteración resulta en malas evaluaciones, las cuales podrían deberse a que la configuración c en general puede no ser la mejor para un subconjunto particular de instancias, considerando que el proceso de evaluación es un proceso estocástico. Así, se establece un mínimo de evaluaciones para las nuevas instancias en la carrera para poder eliminar configuraciones que hayan sido elite en carreras previas. Si una configuración elite fue evaluada en un numero n de instancias, esta no se puede eliminar hasta que las nuevas configuraciones hayan sido evaluadas con un total de $T + n$ instancias, siendo T el número de instancias fijas a evaluar previo a un test estadístico, cuyo valor por defecto es $T = 1$. Esta estrategia tiene la ventaja de hacer que el algoritmo converja más rápido hacia buenos valores de parámetros, pero a su vez representa una desventaja al reducir la exploración de nuevas configuraciones alternativas.

Aunque irace ha demostrado ser una herramienta flexible y poderosa para la configuración automática de algoritmos, con una amplia gama de características y personalización para sintonizar los parámetros de diferentes problemas de optimización, este algoritmo invierte mucho tiempo evaluando configuraciones de mala calidad y además converge rápidamente a conjuntos de parámetros de buen desempeño, sin potenciar la exploración del espacio de búsqueda. Los factores mencionados anteriormente son las causas principales que dan lugar a la problemática central bajo la cual se han desarrollado múltiples investigaciones de modificaciones del algoritmo.

2.5. Modificaciones propuestas de irace

Por su competitividad con otras implementaciones de algoritmos del estado del arte, se han trabajado diversas mejoras para el paquete de irace implementado por [López-Ibáñez *et al.*, 2016].

2.5.1. *Capping* adaptivo

El algoritmo irace es capaz de optimizar tanto la calidad de la solución, como el tiempo de ejecución. Este último también se utiliza como métrica de optimización en problemas de configuración automática por algoritmos destacados como ParamILS y SMAC. Con el fin de disminuir el tiempo de evaluación para configuraciones de mala calidad, Pérez-Cáceres et al. [Cáceres *et al.*, 2017b] implementa una estrategia para establecer un límite sobre el tiempo total de ejecución para las evaluaciones del proceso de configuración. Este tiempo es adaptado al mejor tiempo de ejecución encontrado hasta el momento. Esto se denomina como *capping* adaptativo. Como adición a lo anterior, los autores definen un criterio de eliminación dominante, donde para cada instancia nueva probada, se elimina una configuración si el tiempo de ejecución es mayor a cierto tiempo definido por el limite de tiempo estimado. Este enfoque permite hacer uso de la herramienta irace para problemas cuyo objetivo es minimizar el tiempo de ejecución. La validación de esta propuesta se realizó ejecutando 5 escenarios de configuración: 3 utilizan CPLEX para resolver los problemas de Regions100, Regions200 y Corlat (problemas de programación mixta de enteros), junto con 2 escenarios para los algoritmos Lingeling y Spear que resuelven problemas de satisfacción booleana (SAT).

Los resultados se presentan mediante una comparación entre las estadísticas del tiempo de ejecución total de la configuración y porcentaje de configuraciones con *time-out*. Estos resultados indican que este método presenta resultados significativamente mejores que aquellos obtenidos por la versión original de irace para los escenarios de Regions 100-200, Corlat y Spear, mientras que para el escenario de Lingeling no se encontraron diferencias significativas. Además, la estrategia de *capping* y la eliminación dominante implementadas resultan en una búsqueda con mayor intensificación, según los gráficos presentados para el porcentaje medio de configuraciones seleccionadas para la eliminación. Esto se debe a que el *capping* permite seleccionar más configuraciones para su eliminación en todas las etapas de la carrera, a diferencia de la versión sin *capping*, que elimina principalmente en las fases tempranas de la búsqueda.

2.5.2. Uso de modelos surrogados

Pérez-Cáceres sugiere utilizar un modelo surrogado basado en Random Forest (RF) para identificar y priorizar las mejores configuraciones para el proceso de evaluación, con el fin de abordar el problema de las evaluaciones de configuraciones de bajo rendimiento [Cáceres *et al.*, 2017a]. El estudio implementó el uso del modelo para el muestreo inicial de configuraciones. Los modelos de Random Forest son entrenados con datos del rendimiento de configuraciones ejecutadas con el conjunto de instancias de entrenamiento. Luego, estos modelos predicen el desempeño p_c de una configuración no evaluada c calculando el promedio de los desempeños p_c^i, que corresponden al desempeño predicho de la configuración c para la instancia i, que a su vez es un promedio de las predicciones de los árboles del modelo

de Random Forest. En resumen, la predicción se hace para una configuración e instancia en particular, para todo el conjunto de instancias. Es mediante estas predicciones que se define el desempeño esperado de una configuración.

El método se evaluó en cuatro escenarios de algoritmos distintos. Los dos primeros escenarios definen el algoritmo objetivo como optimización de colonia de hormigas (ACO) para resolver el problema del vendedor viajero (TSP) para distintas clases de instancias. Los dos escenarios restantes corresponden a CPLEX para la resolución de Regions100, que corresponde a un problema de programación entera, junto a Spear para resolver instancias del problema de la clase SAT.

Los autores efectuaron dos análisis para validar la propuesta. El primero se enfoca en evaluar la robustez de los modelos de Random Forest utilizados para predecir el desempeño de una configuración. Se presentan los coeficientes de correlación del *ranking* promedio de cada configuración para los datos obtenidos por el modelo y los datos reales. Los resultados para ACOTSP muestran un buen coeficiente de correlación, sin embargo para los escenarios de Spear y Regions 100 es considerablemente más bajo. Los resultados sugieren que la definición del escenario de configuración determina en gran medida de que forma se deben diseñar los modelos de Random Forest.

Por otro lado, se evaluo el efecto de los modelos de Random Forest en la selección de configuraciones. Se recolectó información sobre la desviación porcentual relativa del óptimo, tiempo de ejecución y desempeño promedio para la selección original de irace y la selección guiada por Random Forest. Los resultados de este estudio concluyeron que no hay mejora significativa en los resultados comparados con la implementación original de irace.

Como conclusión, si bien un modelo surrogado como el implementado en la propuesta puede producir buenas predicciones, estas son muy dependientes del escenario de configuración. El uso de Random Forest puede justificarse en escenarios con un presupuesto de computación bajo o bien donde los parámetros tienen interacciones desconocidas que irace no puede detectar fácilmente. Se enfatiza la importancia del proceso de muestreo para irace, ya que al aplicar el modelo para la selección, disminuyó el desempeño de irace en la mayoría de los casos.

2.5.3. Transformaciones numéricas del espacio de parámetros

Para la definición de los parámetros numéricos, no sólo se tiene un amplio rango de valores posibles, si no que también influye la representación del parámetro, es decir, si este es sensible a variaciones en la escala numérica o escala multiplicativa. Un ejemplo de lo anterior sería para un algoritmo basado en la densidad de población, es fácilmente comprensible que la sensibilidad del tamaño de la población varíe en función de su valor. El impacto de aumentar el tamaño de la población en una cantidad constante, digamos, 10, difiere significativamente según si el aumento es de 1 a 11 o de 1000 a 1010. Es por ello que la elección de la escala

está relacionada con la representación del parámetro en cuestión. En base a esto, Franzin y otros trabajaron en proponer una estrategia para mejorar irace al aplicar transformaciones numéricas automáticas a los parámetros muestreados [Franzin *et al.*, 2018]. Se consideraron 2 transformaciones de un sólo parámetro como base para la propuesta, con el fin de explorar distintas regiones del espacio de parámetros al momento de definir las configuraciones. Si definimos $[a, b]$ como el rango de valores posibles para el parámetro x, las transformaciones posibles son:

- Transformación logarítmica, $Log(x)\colon f(x \in [\log a, \log b]) \longrightarrow 10^x + a \in [a, b]$, que favorece el muestreo de los valores de la parte inferior del intervalo.
- Reflexión de la transformación logarítmica, $RLog(x)\colon f(x \in [\log a, \log b]) \longrightarrow b - 10^x + a \in [a, b]$, que favorece el muestreo de los valores de la parte superior del intervalo.

A partir de estas transformaciones, se propone un método para adaptar automáticamente la transformación adecuada al muestrear parámetros los numéricos. La propuesta implica muestrear cada parámetro numérico inicialmente con muestreo uniforme. Luego de obtener los resultados de la primera iteración, se simula un muestreo considerando tres posibilidades para las carreras siguientes, calculando un estimado del rendimiento promedio para cada valor de parámetro, basado en la desviación promedio del mejor resultado entre las instancias de problemas ya evaluadas. Estos estimados se ordenan y se crean tres subconjuntos que simulan la generación de valores cercanos al extremo inferior (transformación $Log(x)$), al extremo superior $RLog(x)$, y sin transformación. En base al mejor subconjunto, se aplican o no las transformaciones correspondientes. Este proceso se repite para cada nueva configuración, permitiendo que un mismo parámetro pueda tener diferentes transformaciones según la configuración.

La validación de la propuesta consistió en configurar el algoritmo de Simulated Annealing para resolver el problema de asignación cuadrática (SA-QAP). Se compararon los resultados para la desviación porcentual de la solución óptima para SA-QAP. Este proceso de experimentación resultó en qué la aplicación de transformaciones fue de utilidad y representa una mejora en la mayoría de los escenarios. Se sugiere que la aplicación de una transformación debe realizarse sólo después de un análisis detallado de su eficacia, es decir, con un proceso similar al descrito en el trabajo, ya que aplicar una transformación sin analizar su efecto puede ser perjudicial para el proceso de configuración.

2.5.4. Muestreo inicial utilizando LHD

Wessing y otros en [Wessing y López-Ibáñez, 2019], investigaron una alternativa al muestreo uniforme que realiza irace para inicializar las configuraciones iniciales, utilizando el diseño

space-filling basado en *latin hypercube design*, que es una técnica de muestreo que distribuye de manera uniforme valores de los parámetros en un espacio de búsqueda para mejorar la exploración de dicho espacio. El objetivo del trabajo es determinar si optimizar el muestreo inicial puede conducir a una mejora en el rendimiento de la mejor configuración encontrada por irace. Se evalúan en total seis métodos de muestreo: en primer lugar, cuatro variantes de *Multilevel Optimization*, que difieren entre si en los criterios utilizados para optimizar el LHD. Estos criterios incluyen la energía, la correlación, la suma ponderada de ambos, y una selección basada en la relación de dominancia de Pareto considerando ambos criterios. Se acepta cualquier mejora en términos de esta relación de dominancia. Un quinto método corresponde al algoritmo mejorado de LHS (*Latin hypercube sampling*) y el último método es el muestreo uniforme aleatorio original de irace.

Los experimentos comprenden 3 escenarios, el primero con ACO como algoritmo objetivo para resolver el problema de asignación cuadrática (QAP) y otros dos escenarios que modelan enfoques de búsqueda local para resolver problemas clásicos de optimización continua (Ackley's y Rosenbrock's). Aquí, se comparan los valores promedios de la medida de costo de los algoritmos para las configuraciones obtenidas con cada metodología, concluyendo que las variantes de muestreo optimizado si pueden generar configuraciones de mejor rendimiento en comparación con el muestreo uniforme por defecto de irace. Sin embargo, en situaciones adversas, con espacios de parámetros extensos y un bajo número de configuraciones, el muestreo optimizado podría no ser considerablemente superior al muestreo uniforme aleatorio, pero en promedio, no debería ser peor.

2.5.5. Selección no-elitista

Como se explicó en la sección anterior, al final de cada carrera, irace realiza un proceso de selección entre las configuraciones sobrevivientes. Esta selección se ejecuta utilizando un procedimiento llamado *greedy trucation selection*, que selecciona configuraciones *elite*, es decir, aquellas con mejor *rank*. Furong Ye y otros [Ye *et al.*, 2022] proponen 2 alternativas a esta selección, enfocadas a evitar la convergencia prematura del algoritmo hacia determinadas regiones del espacio de parámetros. Ambas propuestas priorizan la configuración con mejor *rank* y completan el conjunto de configuraciones elite de distintas maneras.

La primera propuesta selecciona el resto de configuraciones de forma aleatoria entre las configuraciones restantes, con el fin de inducir cierta diversidad mediante el azar. Por otro lado, se estudió el uso de calcular la entropía utilizando la entropía de Shannon con el propósito de controlar la diversidad entre las configuraciones de élite. El proceso consiste en seleccionar un subconjunto de configuraciones que resultan en la mayor diversidad calculada según la entropía promedio del conjunto de parámetros. Para un conjunto j de configuraciones, la diversidad se calcula de la siguiente manera:

$$D(\Theta) = \frac{\sum_{j=1}^{p} H(\Theta^j)}{p}$$

Donde $H(X)$ es la entropía normalizada de la variable aleatoria X, con distribución de probabilidad $P(X)$, y se define de la siguiente manera:

$$H(X) = \sum_{i=1}^{n} P(X_i)\text{log}P(X_i)/\text{log}(n)$$

Las propuestas descritas fueron evaluadas en los siguientes escenario: ACOTSP, ACO para la resolución del problema de asignación cuadrática (QAP) y SPEAR, algoritmo que resuelve instancias de SAT.

En la evaluación de la estrategia, los autores comparan la desviación del óptimo de las configuraciones resultantes para irace con selección elitista y para las dos propuestas. En los resultados se observa que el uso de entropía representa una mejora en todos los escenarios. Alternativamente, la selección aleatoria resultó en mejores configuraciones para instancias específicas de Spear. Las conclusiones del estudio indican que, si bien se alcanzaron mejoras para casi la totalidad de los escenarios, este método significó un aumento en la varianza de los resultados obtenidos. Por lo tanto, aún existe espacio para realizar nuevos estudios sobre el uso de medidas de diversidad en los operadores de selección de irace.

2.6. Clustering

El clustering o agrupamiento es una técnica perteneciente a la minería de datos exploratoria. Corresponde a la acción de agrupar distintos puntos de datos u objetos por similitud, tal que, aquellos objetos pertenecientes a un mismo grupo, puedan asociarse a través de características similares. Una definición operativa de la tarea de clustering puede ser la siguiente: Dada una representación de n objetos, encontrar K grupos bajo una medida de similitud, tal que dicha medida de similitud entre objetos del mismo grupo sea alta, mientras que las similitudes entre objetos es grupos distintos sea baja [Han *et al.*, 2001].

El clustering es ampliamente utilizado para analizar datos con multiples variables en distintos campos de investigación. Principalmente, el propósito de un análisis de clustering puede clasificarse bajo los siguientes puntos:

- Para visualizar la estructura subyacente de los datos
- Clasificación natural

- Compresión y organización de los datos

En la actualidad, las aplicaciones del clustering abarcan múltiples campos y se han convertido en herramientas de gran utilidad para estos. Algunos ejemplos comprenden el clustering en análisis de datos financieros [Cai *et al.*, 2016], en análisis de redes sociales [Pham *et al.*, 2011] y en bio-informática [žurauskienė y Yau, 2016].

2.6.1. Clasificación de los algoritmos de clustering

A la fecha, se han desarrollado un gran numero de algoritmos de clustering. A modo general, podemos categorizarlos obteniendo cuatro grandes grupos [S. Das y Konar, 2009], en base al método que emplean para la división de los datos en los clusters.

En primer lugar, tenemos el clustering jerárquico, el cual se define como la división de los datos en base a una descomposición jerárquica de los datos. Este a su vez, se divide en dos categorías de acuerdo a cómo se realiza esta descomposición: aglomerativos, que proceden mediante una serie de agrupamientos de los datos, y divisivos, que separan sucesivamente los datos en agrupaciones más finas. Ejemplos de algunos algoritmos de clustering jerárquicos corresponden a BIRCH [Zhang *et al.*, 1996], DIANA [Patnaik *et al.*, 2016], CURE [Guha *et al.*, 1998] y los métodos algomerativos de *linkage* o enlazamiento [Müllner, 2011].

Por otro lado, el clustering particional consiste en construir particiones de datos, donde cada partición representa un cluster, sin imponer una estructura jerárquica entre ellos. En estos algoritmos, el número de clusters a conformar se debe específicar previo al proceso de agrupamiento o determinar una moda. Entre los algoritmos de este tipo más populares se tiene el algoritmo k-means y sus variaciones [MacQueen, 1967], *partitioning around medoids* o k-*medoids* [Kaufman y Rousseeuw, 1990].

También podemos mencionar la clasificación de algoritmos de clustering basados en densidad, el cual se basa en considerar los grupos como regiones densas de objetos en el espacio de datos, separadas por regiones de baja densidad [Kriegel *et al.*, 2011]. La premisa principal de este enfoque es identificar áreas de alta densidad y baja densidad, donde las regiones de alta densidad están claramente diferenciadas de aquellas de baja densidad de datos. El algoritmo de DBSCAN *(Density-Based Spatial Clustering of Applications with Noise)* [Ester *et al.*, 1996] es un algoritmo perteneciente a esta clasificación de amplio uso en la literatura.

La última clasificación abordada es la de clustering basado en cuadrícula (*grid-based* clustering), la cual se explorará en detalle en la siguiente sección de este capítulo.

2.6.2. Grid-based clustering

Definimos los métodos *grid-based* o basados en cuadrícula como aquellos algoritmos que trabajan con los datos de manera indirecta, mediante la construcción de resúmenes de los datos sobre subconjuntos del espacio de atributos. Estos algoritmos realizan una segmentación del espacio y luego agregan los segmentos apropiados para retornar los clusters. Nos referimos a todos los métodos de clustering que dividen el espacio como métodos *grid-based*, ya que generalmente se adopta algún tipo de agrupación para atributos numéricos. Una ventaja de estos métodos es que no considera términos como densidad y conectividad en sus procesos, sino que hereda la topología de del espacio de atributos de los datos, limitando así la cantidad de cálculos realizados. Los algoritmos *grid-based* consideran segmentos de varias dimensiones, que corresponden a productos cartesianos de subrangos o valores de atributos individuales. También es importante señalar que, si bien los métodos de particionamiento basados en densidad funcionan mejor con atributos numéricos, los métodos basados en cuadrículas trabajan con atributos de varios tipos [Berkhin, 2006]. En la figura 3 observamos cómo se grafican los clusters para un caso de 2 dimensiones, donde el espacio de datos X_1 se divide en bloques a lo largo de cada dimensión, y los límites de cada bloque se definen según los valores mínimos y máximos de los puntos de datos en esa dimensión, creando así las particiones de clusters.

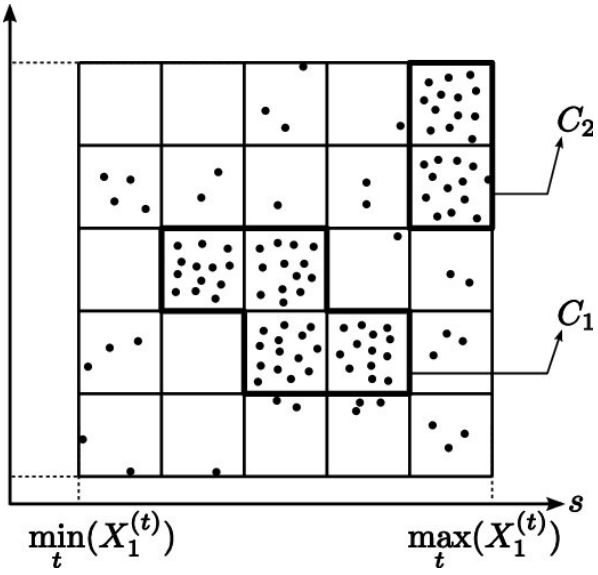


Figura 3: Ilustración de un algoritmo de *Grid-based* clustering en 2 dimensiones con 6 clusters.

Fuente: [Bandaru y Deb, 2010]

Entre los algoritmos de grid-based clustering más relevantes en la actualidad tenemos CLIQUE [Ma *et al.*, 2024] y STING [Wang *et al.*, 1997], los cuales utilizan una cuadrícula multidimensional para encontrar regiones densas en el espacio de atributos y WaveCluster [Sheikholeslami *et al.*, 1998], que emplea una cuadrícula adaptativa que se ajusta a la distribución de los datos para identificar clusters de diferentes formas.

2.6.3. El algoritmo de k-medoids

El algoritmo de Partitioning around medoids (PAM) [Kaufman y Rousseeuw, 1990] o mas conocido como k-*medoids* actualmente, es un algoritmo de clustering que mantiene la idea de los algoritmos de clustering particional, la cual corresponde a dividir un conjunto de datos en un numero de clusters k dado, de tal forma que se minimice la diferencia entre objetos del mismo cluster y a la vez, se maximice la diferencia entre objetos de clusters distintos. El proceso general de construcción de clusters se realiza de la siguiente manera: (1) Para conformar K clusters, se eligen K posibles *medoids* u objetos representativos para cada cluster, (2) dividir los objetos restantes en torno a los *medoids*, (3) Calcular la disimilitud promedio entre objetos, (4) repetir los pasos anteriores hasta encontrar la asignacion de clusters con menor disimilitud promedio. La diferencia de este algoritmo con otros del mismo tipo, como k-means, es la forma en la que los puntos representativos de cada cluster es elegido desde el conjunto de datos a clusterizar, es decir, es un punto real de los datos y no un punto tomado al azar del espacio que podría no ser parte del conjunto.

Otro punto importante a considerar es cómo calcular la disimilitud entre objetos. Una opción usualmente utilizada es calcular la distancia euclidiana entre los puntos. La fórmula para calcular la distancia Euclidiana entre dos objetos P y Q es la siguiente:

$$d(P,Q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2}$$

donde q_i y p_i representan la coordenada i-ésima de la posición de los objetos Q y P, respectivamente.

Otra opción utilizada para este análisis corresponde a la distancia Manhattan, la cual se diferencia de la distancia Euclidiana al utilizar el valor absoluto de las diferencias por sobre el uso de la raíz cuadrada, como podemos ver en la siguiente ecuación:

$$d(P,Q) = |q_1 - p_1| + |q_2 - p_2| + \ldots + |q_n - p_n|$$

Esta distancia es útil para casos donde las diferencias discrepan en gran magnitud en alguna de las variables de los objetos, tal que la distancia Euclidiana otorga más peso a dicha discrepancia, mientras que la distancia de Manhattan considera la diferencia absoluta.

2.7. Resumen

En esta sección, se presenta un análisis en profundidad del problema de la configuración automática de algoritmos, fundamental en el ámbito de las metaheurísticas, las cuales son ampliamente utilizadas en la resolución de problemas de optimización. Se describen las clases de parámetros (numéricos, categóricos y ordinales), subrayando la dificultad y el desafío de hallar sus valores óptimos. Se presentan estrategias de control y ajuste de parámetros, enfocándose en la sintonización automática que busca determinar los mejores valores de los

parámetros mediante procesos iterativos de evaluación. Además, se introduce el algoritmo Iterated Racing (irace) como una herramienta crucial para la configuración de algoritmos, resaltando sus etapas e características principales en la búsqueda de configuraciones óptimas mediante un proceso de racing. También se debaten las modificaciones sugeridas para irace con el fin de mejorar su rendimiento. Finalmente, se abordan conceptos clave de clustering y se presenta el algoritmo k-medoids como una técnica importante en el proceso de clustering de datos.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

Como se mencionó en el capítulo 3, irace es una algoritmo de sintonización con mucho potencial. Mejorar el proceso de sintonización para encontrar nuevas configuraciones es importante para poder innovar en el área. A raíz de lo anterior, la propuesta de solución de este trabajo consiste en modificar el criterio de aceptación empleado por irace para descartar configuraciones posterior al proceso de racing. Esto mediante el uso de algoritmos de clustering definidos en la sección anterior. En el presente capitulo, se define y describe la propuesta de solución de este trabajo, que corresponde a implementar un procedimiento de clustering que permita agrupar de forma ideal las configuraciones y así poder influenciar la búsqueda de mejor manera durante la ejecución de irace. Se introduce la motivación de la propuesta y las principales características de la misma, asimismo explicando cómo esta se integra en el proceso actual de configuración de irace.

3.1. Introducción y Contexto del Enfoque Propuesto

El algoritmo de irace define las configuraciones élite como a) aquellas que obtuvieron el mejor rendimiento iterativamente durante el proceso de configuración, y b) que no muestran diferencias significativas en su rendimiento entre todas las configuraciones de dicha iteración. Estas se seleccionan del conjunto de configuraciones supervivientes y corresponden a las que guían la generación de nuevas configuraciones. Esto se hace modificando la distribución de muestreo de configuraciones hacia las regiones del espacio de parámetros definidas por las configuraciones élite. Esta estrategia de muestreo permite que irace muestree configuraciones cada vez más parecidas a las que tienen el mejor rendimiento a medida que avanza el proceso. Sin embargo, este proceso puede causar una convergencia prematura, limitando la exploración del espacio de parámetros y, en consecuencia, generando configuraciones con valores y/o características similares.

Para resolver este problema, la propuesta se basa en modificar el criterio de aceptación de las configuraciones elite. Esto involucra la inclusión de técnicas de clustering para seleccionar configuraciones que amplíen la cobertura del espacio de parámetros. Después de que el proceso de racing elimine las configuraciones para las cuáles se ha encontrado evidencia estadística significativa de su mal rendimiento, se planea agrupar las configuraciones supervivientes en clusters. Estos se construirán a partir de las particiones de los valores de los parámetros de configuración. Por lo tanto, se propone que los modelos de muestreo para generar nuevas configuraciones se actualicen en base a estos clusters. Cada cluster tendrá un modelo representativo, definido a partir de una de las configuraciones del mismo. Esto resultará en una configuración elite por cluster, representando una sección única del espacio de parámetros (en términos de sus valores). La idea es mejorar la capacidad exploratoria de

irace, asegurando que el muestreo se realice desde un conjunto más diverso de configuraciones elite, basado en los valores de los parámetros. Al mismo tiempo, proporcionará un punto de referencia para medir la exploración del algoritmo. La figura 4 ilustra el proceso de configuración de irace propuesto. Al final de cada iteración, las configuraciones sobrevivientes se agruparán en clusters (1). A continuación, se seleccionará una configuración representativa por cada cluster (2). Estas configuraciones representativas actualizarán sus distribuciones de muestreo para sesgar la generación de nuevas configuraciones (3). Finalmente, una vez que la nueva iteración de la carrera haya concluido, los clusters se actualizarán asignando las nuevas configuraciones sobrevivientes a los mismos (4).

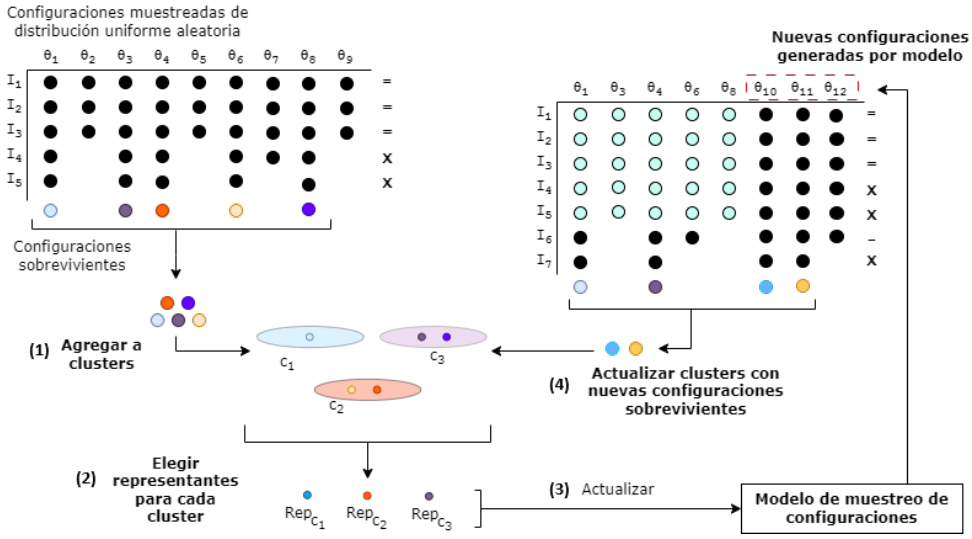


Figura 4: Propuesta de criterios de aceptación elite basados en clustering
Fuente: Elaboración propia.

3.2. Metodología de la propuesta

A continuación se describen los fundamentos de la estrategia de clustering de la propuesta y cómo se introduce en irace.

3.2.1. Grid-based clustering

En este trabajo, se evalúa el uso de la técnica de *grid-based clustering* definida en el capítulo 2.6.2. El criterio para agrupar en clusters las configuraciones se basa en particiones del dominio del espacio de parámetros. La idea es construir un número fijo de particiones y ubicar cada configuración en una sola de ellas. En particular, la propuesta considera la definición de clusters basados en parámetros numéricos.

El procedimiento para definir los clusters considera la combinación de todos los valores posibles, resultando en una cuadrícula de parámetros donde cada entrada define un cluster. También se consideran los valores condicionales de los parámetros en la cuadrícula. Para esta memoria, nos enfocaremos exclusivamente en parámetros numéricos. En este contexto, los dominios se extienden hacia intervalos de valores infinitos (en el caso continuo) y múltiples valores posibles (para enteros). Por lo tanto, llevamos a cabo una partición del dominio para cada parámetro numérico, donde el número de particiones constituye un hiper-parámetro de la propuesta. La inclusión de parámetros categóricos se deja para trabajo futuro.

Una vez calculadas estas particiones para todos los parámetros, se consideran sistemáticamente las combinaciones de estas particiones. Cada combinación resultante representa de manera única un cluster. Por ejemplo, consideremos dos parámetros numéricos, c y d, cada uno con dominios {0, 10} y {0.5, 3.5}, respectivamente. La Figura 5 ilustra visualmente el proceso de particionamiento para cada dominio, mostrando particiones con longitudes de $2{,}5$ para el parámetro c y 1 para el parámetro d. En este escenario, calculamos un total de tres particiones por dominio. Las combinaciones resultantes de estos intervalos de partición definen clusters distintos dentro del conjunto de datos. Cabe destacar que los intervalos incluyen el punto final a la izquierda pero no a la derecha, excepto para el último intervalo.

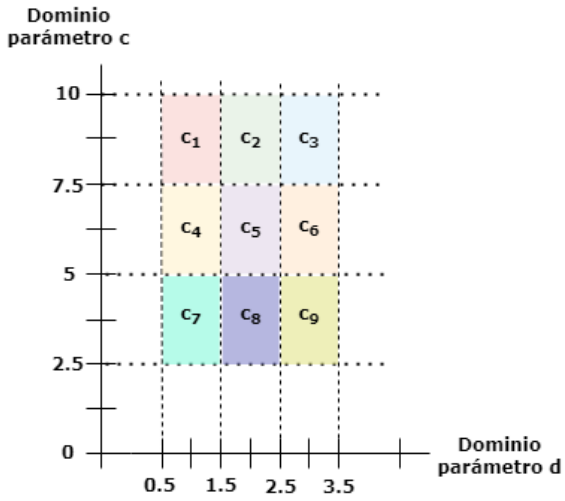


Figura 5: Ilustración de conformación de clusters basado en particiones del espacio de parámetros.

Fuente: Elaboración propia.

El proceso de clustering comienza ubicando las configuraciones según los valores de los parámetros numéricos. Para esto, consideramos dos posibilidades: (a) si no se crearon clusters previamente, se realiza un proceso de agrupamiento con todas las configuraciones existentes. A cada cluster se le asigna un número identificador de cluster, considerando la coincidencia de combinaciones de intervalos de parámetros numéricos, o (b) si existen clusters de iteraciones anteriores, las configuraciones agregadas a los clusters en el paso anterior se

integran en los clusters existentes. Si no se encuentra ninguna coincidencia de cluster según los valores de parámetros numéricos para una configuración, se crea un nuevo cluster.

El proceso de búsqueda para encontrar una coincidencia de clusters numéricos puede ser costoso, ya que combinar particiones puede resultar en una cuadrícula muy grande, para la cual se necesita verificar si cada valor numérico está dentro de los intervalos correspondientes o no, por lo tanto, la coincidencia de una configuración dentro de un cluster se realiza aprovechando la estructura de la cuadrícula de parámetros. Primero, se encuentra una coincidencia para el último parámetro, que es el que se fija para todas las combinaciones de intervalos de los demás parámetros. Cuando se encuentra una coincidencia, la búsqueda se estrecha a las combinaciones que coinciden con el intervalo para el último parámetro que ya se buscó, eliminando el resto de las opciones de la búsqueda. Esto se ejecuta de manera recursiva hasta que solo quede una combinación, cuyo índice en la cuadrícula es la coincidencia de cluster para la configuración actual.

La estrategia propuesta requiere la definición de qué parámetros se utilizarán para realizar el agrupamiento y el número de particiones de dominio por parámetro numérico (p). Estas decisiones son muy relevantes ya que tienen un impacto en cómo se mejora la exploración en el proceso de búsqueda realizado por irace. Elegir qué parámetros se agruparán es una decisión difícil y puede requerir algún conocimiento sobre el algoritmo objetivo. Algunos tipos de parámetros que podrían considerarse como candidatos para esta tarea son: 1) parámetros cuyo ajuste adecuado es altamente relevante para obtener un alto rendimiento del algoritmo objetivo, 2) parámetros que son difíciles de ajustar debido a interacciones con otros parámetros, y 3) parámetros que definen grandes dominios. Notamos que el número de parámetros utilizados para realizar la tarea de agrupamiento dependerá de la técnica aplicada.

Respecto al número de particiones a realizar por cada dominio de los parámetros, al considerar un gran número de particiones, el número de clusters a considerar aumenta considerablemente. Además, debido al comportamiento naturalmente explotador de irace, un número importante de clusters puede volverse rápidamente 'inactivo'. Nos referimos a los clusters inactivos como clusters que no tienen configuraciones sobrevivientes asignadas a ellos. Un número adecuado de particiones debe permitir que la exploración tenga suficientes clusters activos para poder guíar la búsqueda de manera eficiente.

3.2.2. K-medoids

Adicional a la propuesta de implementación, evaluamos el uso de un algoritmo de clustering clásico para crear conjuntos de configuraciones élite al final de cada iteración. Este proceso se realiza para contrastarlo con el enfoque básico e intuitivo descrito en la sección anterior. Para ello, se considera el algoritmo K-Medoids, ya que es un algoritmo de clustering conocido en la literatura y que puede agrupar exitosamente parámetros categóricos, numéricos y ordinales.

Con este algoritmo, es necesario definir al menos qué parámetros se agruparán y cuántos clusters (K) serán considerados para la ejecución del algoritmo. El procedimiento de agrupación se ejecuta después de cada iteración con las configuraciones sobrevivientes. Posteriormente, se aplica el algoritmo K-Medoids a los parámetros seleccionados.

Como se mencionó anteriormente, este algoritmo ofrece la opción de elegir una medida de disimilitud, que puede ser Manhattan o Euclidiana, y se define como un parámetro, siendo la última la opción predeterminada. Esta medida se utiliza para calcular una matriz de disimilitud entre cada elemento del conjunto de configuraciones.

3.2.3. Representantes por cluster

En esta propuesta de memoria, los clusters juegan un papel importante en la generación de nuevas configuraciones para las iteraciones posteriores. Definimos una configuración representativa en un cluster k como una configuración que permanece viva después de un procedimiento de racing y cuyo valor de rendimiento (calidad media o rank) es el mejor entre todas las configuraciones en el cluster k. Las configuraciones representativas se utilizarán para generar nuevas configuraciones para la próxima carrera. Al final de cada iteración de irace, se realizan dos procesos de selección: (1) selección de elite: seleccionando un subconjunto de configuraciones sobrevivientes que influirán en el proceso de generación de nuevas configuraciones, y (2) selección de padres: seleccionando probabilísticamente un padre para muestrear cada nueva configuración de su modelo. En el procedimiento de selección de elite, irace clasifica las configuraciones elite sobrevivientes considerando su rendimiento, y las L mejores se consideran para la selección de padres. La idea es distribuir la selección de padres entre los distintos clusters para así aumentar la diversidad de las configuraciones generadas y así promover la exploración del algoritmo, pero además considerando la mejor configuración por cluster, manteniendo el componente de explotación introducido por la selección elite de irace.

El algoritmo 2 muestra el procedimiento para reemplazar la selección de elites basada en los clusters previamente definidos. Primero, creamos un conjunto (Θ_K) con todas las configuraciones representativas (líneas 3-5). De igual manera que en irace, un número de $nbRep$ configuraciones formarán parte de la selección de padres (línea 6). El valor $nbRep$ es el mínimo entre el número de configuraciones sobrevivientes ($NbAlive$) y un número predefinido de configuraciones ($minSurvival$). Se realiza una selección con una ruleta clásica con reemplazo con todas las configuraciones en Θ_K (considerando su valor de rango de rendimiento) para seleccionar $nbRep$ configuraciones (líneas 8-11).

Para el proceso de la ruleta, los pesos o ponderaciones para cada representante pueden calcularse según tres opciones: (a) Inverso normalizado del rank, (b) Suma normalizada de calidades y (c) Rank inverso de ranks, el cual es el peso utilizado por irace para la elección de padres en el muestreo de configuraciones. La selección del método de cálculo del peso se replica tanto para la elección de representantes como para la elección de padres.

Algorithm 2 Parent configuration selection

Require: K clusters
$\Theta_K, \Theta_{Rep} \leftarrow \emptyset$
for each cluster k in K **do**
    $\Theta_K \leftarrow \Theta_K \cup GetBestConfigInCluster(k)$
end for
$nbRep \leftarrow$ mín(NbAlive, minSurvival)
weights ←CalculateWeights(Θ_K)
while $|\Theta_{Rep}| < nbRep$ **do**
    $\theta \leftarrow$ RouletteWheel(weights)
    $\Theta_{Rep} \leftarrow \Theta_{Rep} \cup \theta$
end while
Output: Θ_{Rep}

Este proceso de elección de representantes se ejecuta independientemente del proceso de clustering aplicado al final de cada carrera donde se deba llevar a cabo el procedimiento de *sampling* de nuevas configuraciones.

3.3. Conclusiones

En conclusión, la solución propuesta en este capítulo tiene como objetivo mejorar el proceso de sintonización de irace al modificar el criterio de aceptación de las configuraciones élite. Esta modificación implica la utilización de técnicas de agrupamiento para seleccionar configuraciones que amplíen la cobertura del espacio de parámetros. La metodología propuesta se basa en agrupar las configuraciones supervivientes en clusters, donde cada uno representa una sección única del espacio de parámetros y está definido por divisiones de los valores de los parámetros numéricos de cada configuración. A partir de estos clusters, se actualizan los modelos de muestreo para generar nuevas configuraciones, mejorando así la capacidad exploratoria de irace. La estrategia también incluye la selección de un representante por cluster, crucial para la generación de nuevas configuraciones en las siguientes iteraciones. Estos representantes se eligen entre las configuraciones supervivientes y se utilizan para influir en el proceso de generación de nuevas configuraciones. Se evalúa el uso de dos algoritmos de agrupamiento clásico, Grid-based clustering y K-Medoids, como alternativa para crear conjuntos de configuraciones élite al final de cada iteración.

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se proporciona una descripción detallada del proceso de validación que se ha llevado a cabo para la solución propuesta anteriormente. En particular, se exponen las principales características del escenario experimental que se ha utilizado para la evaluación de las estrategias de clustering implementadas para modificar los criterios de aceptación de configuraciones de irace. Este proceso de evaluación es crucial para determinar la eficacia de nuestro enfoque y sus características principales. La metodología se desarrolla para configurar dos algoritmos diferentes de la familia de ACOTSP, utilizando distintos parámetros para el presupuesto de sintonización y conjunto de instancias a evaluar.

Primero, se detallan los escenarios experimentales empleados, abarcando los algoritmos objetivo, los parámetros a sintonizar y sus rangos de valores, los conjuntos de instancias utilizados y los recursos computacionales asignados. Posteriormente, se describe el procedimiento de ejecución de los algoritmos y se especifican los valores de los parámetros seleccionados para llevar a cabo el clustering. Finalmente, se exponen los resultados acompañados de un análisis fundamentado en el rendimiento obtenido y en la capacidad de exploración de irace dentro dela propuesta descrita.

4.1. Escenario experimental

Para describir el escenario experimental en su completitud, en las siguientes subsecciones describimos los algoritmos a sintonizar, el conjunto de instancias, budget de sintonización utilizado, los hiper-parámetros seleccionados y la metodología de experimentación.

4.1.1. Algoritmos Objetivo

El algoritmo base utilizado para evaluar nuestra propuesta corresponden a variaciones de Ant Colony Optimization [Dorigo *et al.*, 2006] y se definen a continuación:

- **ACOTSP** [Stützle, 2002]: un marco de optimización por colonias de hormigas que implementa diferentes algoritmos ACO para resolver el Problema del Viajante de Comercio (TSP). ACOTSP define 11 parámetros, detallados en la tabla 1. En esta tabla, el tipo indica si el parámetro es (i)nteger, (r)eal o (c)ategórico, y la última columna presenta la condición requerida para que un parámetro se utilice. El presupuesto para cada ejecución de ACOTSP es de 5000 evaluaciones.

Parámetro	Tipo	Dominio	Condición
$algorithm$	c	$\{as, mmas, eas, ras, acs\}$	-
$localsearch$	c	$\{0, 1, 2, 3\}$	-
$alpha$	r	$[0{,}00, 5{,}00]$	-
$beta$	r	$[0{,}00, 10{,}00]$	-
rho	r	$[0{,}01, 1{,}00]$	-
$ants$	i	$[5, 100]$	-
$q0$	r	$[0{,}00, 1{,}00]$	$algorithm$== 'acs'
$rasrank$	i	$[1, 100]$	$algorithm$== 'ras'
$elitistants$	i	$[1, 750]$	$algorithm$== 'eas'
$nnls$	i	$[5, 50]$	$localsearch \in \{1, 2, 3\}$
dlb	c	$\{0, 1\}$	$localsearch \in \{1, 2, 3\}$

Tabla 1: Parámetros de ACOTSP: dominio, tipo y condición (si aplica).

Parámetro	Tipo	Dominio
$localsearch$	c	$\{0, 1, 2, 3, 4\}$
rho	r	$[0{,}01, 1{,}00]$
$ants$	i	$[1, 100]$

Tabla 2: Parámetros de MMASQAP: dominios y tipo.

- **MMASQAP** [Stützle y Hoos, 1998]: un algoritmo de optimización por colonias de hormigas que implementa la estrategia Max-Min para resolver el Problema de Asignación Cuadrática (QAP). MMASQAP define 3 parámetros, detallados en la tabla 2. El presupuesto para cada ejecución de MMASQAP es de 10 segundos.

4.1.2. Instancias

Para evaluar nuestra propuesta, utilizamos tres conjuntos de instancias diferentes:

- *TSP-2000*: compuesto por 400 instancias aleatorias uniformes del *Traveling Salesman Problem* (TSP), cada una con 2000 ciudades. Tanto el conjunto de instancias de entrenamiento como el de prueba están compuestos por 200 instancias. Este conjunto es el más homogéneo de todos los *sets* de instancias utilizados, ya que solo considera instancias con el mismo número de ciudades.
- *TSP-1000/3000*: compuesto por 500 instancias aleatorias uniformes del *Traveling Salesman Problem* (TSP) con diferentes números de ciudades (que van desde 1000 hasta 3000 ciudades). Para el conjunto de entrenamiento, se consideran 250 instancias,

mientras que el conjunto de prueba incluye 250 instancias. Este conjunto es ligeramente más heterogéneo que *TSP-2000*.

- *QAP-60*: compuesto por 60 instancias con 60 elementos y diferentes valores de esparcimiento: 30 instancias estructuradas euclidianas con un valor de esparcimiento de $0{,}90$ y 30 instancias aleatorias con un valor de esparcimiento de $0{,}00$. Las instancias se describen en [Saifullah Hussin y Stützle, 2014]. Tanto los conjuntos de instancias de entrenamiento como los de prueba están compuestos por 30 instancias.

4.1.3. Budget de sintonización

Para analizar el efecto de la selección basada en clustering en la convergencia de irace, se define diferentes presupuestos de configuración. Para ACOTSP, consideramos 1,000 (1M) y 10,000 (10M) evaluaciones como presupuestos de configuración para irace. Para MMASQAP, consideramos 1,000 (1M) y 5,000 (5M) evaluaciones como presupuestos de configuración. El presupuesto más alto (10M y 5M) se seleccionó considerando el tamaño del espacio de búsqueda de parámetros definido por cada algoritmo objetivo.

4.1.4. Metodología de Validación de la solución

En este trabajo, las técnicas de clustering fueron aplicadas utilizando los parámetros $ants$ (número de hormigas) y rho (tasa de evaporación) como parámetros utilizados para el clustering en todos los escenarios. En ambos casos, estos parámetros fueron elegidos dado que su configuración debe considerar los valores de otros parámetros (es decir, rho debe configurarse en coordinación con otros parámetros que definen el comportamiento del algoritmo). Se presume que estos parámetros probablemente sean candidatos a requerir mantener un alto nivel de exploración. Para la construcción de las particiones para el *grid-based* clustering, el número de particiones p es igual a 4 en todos los escenarios evaluados. Este valor fue elegido considerando la elección de dos parámetros numéricos para realizar el clustering, ya que en este escenario se trabaja con una cantidad de clusters que permitiría realizar suficientes pruebas para un correcto análisis.

Para la presentación de los experimentos, se menciona clirace para hacer referencia a irace usando la técnica de propuesta de grid-based clustering. Por otro lado, para la implementación con k-medoids haremos referencia a k-clirace. Se realizaron 20 ejecuciones independientes de los irace y clirace algoritmos para cada presupuesto de evaluaciones, para los conjuntos de instancias *TSP-1000/3000*, *TSP-2000* y *QAP-60*. Para el caso de k-clirace, se consideraron distintos valores para el parámetro K, que denota el número de clusters. En concreto, para cada valor del conjunto $K = \{3, 5, 7, 10\}$, se realizaron 5 ejecuciones independientes del algoritmo para el conjunto de instancias de *TSP-2000*.

Para comparar el rendimiento entre algoritmos, se evalúan las mejores configuraciones ob-

tenidas al final de cada ejecución del sintonizador sobre el conjunto de instancias de prueba. El número de configuraciones evaluadas (élites) varía entre una y cinco configuraciones por ejecución del sintonizador, según el número de configuraciones sobrevivientes al final de las ejecuciones. Además, se realizaron 20 ejecuciones independientes para cada instancia de ACOTSP y 20 ejecuciones independientes por instancia de QAP.

4.2. Resultados

Esta sección presenta los resultados de la evaluación de la propuesta y los compara con los resultados de irace. Para cada conjunto de instancias, presentamos un conjunto de estadísticas (media, desviación estándar, valores mínimo y máximo, y kurtosis) del rendimiento según la distancia porcentual al óptimo de cada enfoque para cada presupuesto de configuración (B) de evaluaciones. Adicionalmente, para los resultados de la propuesta basada en k-medoids, se presenta una tabla para presentar los resultados sobre el conjunto de valores de K definidos anteriormente. Finalmente, presentamos un análisis del comportamiento producido por la inclusión del algoritmo de clustering en irace y sobre el tiempo de ejecución de las técnicas.

Los resultados de esta sección se presentan con respecto al rendimiento medio d_{med}, el cual se mide como la distancia media d_{ks} a cada óptimo $best$, calculado según la siguiente fórmula:

$$d_{med} = (\frac{d_{ks} - best}{d_{ks}}) \cdot 100$$

Además, se considera el subconjunto final de configuraciones sugeridas por irace, clirace y k-clirace, como se describe en la sección 4.1.4.

En primer lugar, la tabla 3 proporciona el rendimiento medio de las configuraciones obtenidas por los enfoques al configurar ACOTSP para resolver el conjunto de instancias *TSP-2000*.

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	**0.48**	0.12	0.03	1.25	0.24
clirace		0.63	0.31	0.05	6.20	35.83
irace	10M	0.49	0.12	0.03	1.29	0.09
clirace		0.50	0.13	0.06	1.89	1.28

Tabla 3: Desempeño promedio (distancia al óptimo) ACOTSP - *TSP-2000*.

Los resultados muestran que cuando $B = 1M$, las configuraciones sugeridas por clirace obtienen un peor rendimiento en comparación con irace. Esto se puede observar en la media, y valores máximos. Sin embargo, cuando $B = 10M$, el desempeño promedio de ambos algoritmos es muy similar a aquel obtenido por irace, según podemos ver para el valor promedio, con diferencias bajas en la desviación estándar y valores máximos y mínimos. Como las

configuraciones prometedoras pueden asignarse en el mismo cluster, clirace se ve obligado a explorar otras configuraciones en comparación con irace. En condiciones de bajo presupuesto de configuración ($B = 1M$), esta exploración no permite que las propuestas basadas en clustering puedan converger, mientras que al tener un presupuesto más grande ($B = 10M$) se observa tanto una alta exploración como convergencia.

La Tabla 4 muestra el rendimiento de la mejor configuración global obtenida por irace y clirace, para el conjunto de instancias *TSP-2000*.

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	**0.41**	0.10	0.09	0.95	0.16
clirace		0.44	0.11	0.11	0.97	0.15
irace	10M	0.46	0.11	0.09	0.99	0.32
clirace		**0.42**	0.10	0.12	0.82	-0.05

Tabla 4: Mejor desempeño (distancia al óptimo) ACOTSP - *TSP-2000*.

Considerando $B = 1M$, irace obtiene una configuración de mejor rendimiento en comparación con clirace y k-clirace, cuyos resultados de desempeño promedio son similares. Al configurar con un presupuesto más grande ($B = 10M$), la configuración obtenida por k-clirace tiene un rendimiento ligeramente mejor en comparación con la sugerida por clirace y irace, siendo la última la que obtiene el peor rendimiento promedio entre las tres. Esto se puede observar en el rendimiento medio y los valores mínimos alcanzados.

A continuación, comparamos los resultados de las ejecuciones de la propuesta basada en k-medoids (k-clirace) con los resultados de irace y clirace, para el cual se consideran solo 5 ejecuciones independientes de los algoritmos. De esa manera, podemos comparar los resultados para cada valor de K con la misma cantidad de ejecuciones para todos los algoritmos.

La tabla 5 muestra el desempeño promedio obtenido por las configuraciones resultantes de irace, clirace y k-clirace tras 5 ejecuciones independientes. En general, los resultados muestran un mejor rendimiento para irace. Sin embargo, para el escenario de mayor presupuesto ($B = 10M$), podemos ver que los valores de k-clirace para los valores de K propuestos, superan a irace en el desempeño promedio de la mejor configuración. Notamos igual que clirace tiene el peor rendimiento entre los 6 escenarios. Para el presupuesto de $B = 1M$, irace sigue siendo superior al resto con un margen notable de diferencia.

Por otro lado, la tabla 6 muestra el mejor desempeño obtenido por las configuraciones resultantes de irace, clirace y k-clirace tras 5 ejecuciones independientes. Estos resultados favorecen a los resultados obtenidos por k-clirace para ambos presupuestos ($B = 1M$ y $B = 10M$), donde para el menor presupuesto, k-clirace logra igualar el rendimiento de la mejor configuración encontrada por irace. Cuando $B = 10M$, los resultados de k-clirace son superiores a los resultados de irace y clirace.

Detallamos en particular los resultados de k-clirace para distintos valores de K. Observamos

Escenario	B	K	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	-	**0.47**	0.12	0.07	1.05	0.08
clirace		-	0.59	0.30	0.05	4.05	59.77
k-clirace		3	0.59	0.28	0.05	4.05	65.59
k-clirace		5	0.55	0.29	0.05	4.05	66.45
k-clirace		7	0.57	0.28	0.05	4.05	67.02
k-clirace		10	0.56	0.30	0.05	4.05	60.25
irace	10M	-	0.49	0.11	0.06	1.13	0.07
clirace		-	0.50	0.13	0.06	1.89	1.28
k-clirace		3	0.45	0.12	0.04	1.76	0.29
k-clirace		5	0.45	0.12	0.04	1.59	0.52
k-clirace		7	0.46	0.12	0.03	1.78	1.82
k-clirace		10	**0.44**	0.11	0.04	1.15	0.10

Tabla 5: Desempeño promedio (distancia al óptimo) ACOTSP - *TSP-2000* para 5 ejecuciones independientes de irace, clirace y k-clirace.

que, en general, el rendimiento promedio y mejor rendimiento tiende a ser similar entre ejecuciones para ambos presupuestos, independientemente del valor de K utilizado, tal que no existe relación directa con la cantidad de clusters K y el rendimiento promedio. Podemos notar que cuando $B = 10M$ los resultados son mejores en promedio en comparación al escenario de $B = 1M$, lo cual se alinea con los resultados obtenidos anteriormente, donde un mayor presupuesto de configuración resulta en un mejor rendimiento de las configuraciones encontradas. Esto se puede observar tanto en el promedio y desviación estándar, como en los valores máximos y kurtosis.

Escenario	B	K	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	-	**0.42**	0.10	0.10	0.87	-0.01
clirace		-	0.45	0.11	0.12	0.87	0.01
k-clirace		3	0.45	0.12	0.11	1.06	0.25
k-clirace		5	0.44	0.11	0.10	0.91	0.04
k-clirace		7	0.44	0.11	0.10	0.91	0.08
k-clirace		10	**0.42**	0.11	0.08	0.96	0.06
irace	10M	-	0.47	0.11	0.09	0.96	-0.001
clirace		-	0.50	0.13	0.06	1.89	1.28
k-clirace		3	**0.37**	0.10	0.05	0.73	-0.08
k-clirace		5	0.38	0.10	0.07	0.87	0.17
k-clirace		7	0.38	0.10	0.04	0.86	0.19
k-clirace		10	0.38	0.10	0.07	0.82	0.08

Tabla 6: Mejor desempeño (distancia al óptimo) ACOTSP - *TSP-2000* irace, clirace y k-clirace.

Las tablas 7 y 8 proporcionan el rendimiento promedio y el mejor rendimiento de las configuraciones obtenidas por irace y clirace al configurar ACOTSP para resolver el conjunto de instancias *TSP-1000/3000*. En ambas comparaciones, se puede observar que los niveles de exploración más altos de clirace producen configuraciones de menor rendimiento. Estos resultados nos permiten entender que, en algunos escenarios, estamos perdiendo las capacidades de explotación de irace al modificar su balance y aumentar la exploración. Tales resultados señalan que un control adaptativo del comportamiento (fusionando o dividiendo) de los clusters podría ser utilizado. Los resultados de la prueba de Wilcoxon pareada comparando el rendimiento de las mejores configuraciones muestran que hay una diferencia estadística entre su rendimiento.

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	**0.48**	0.21	0.00	4.06	48.09
clirace		0.57	0.32	0.00	4.33	40.34
irace	10M	**0.42**	0.17	0.00	1.38	-0.49
clirace		0.46	0.18	0.00	3.36	-0.21

Tabla 7: Desempeño promedio (distancia al óptimo) ACOTSP - *TSP/1000-3000*. Valores de prueba pareada de Wilcoxon: $1M : < 0{,}001$, $10M : < 0{,}001$

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	**0.41**	0.16	0.0	0.97	-0.47
clirace		0.44	0.18	0.00	0.97	-0.53
irace	10M	**0.40**	0.16	0.00	0.90	-0.52
clirace		0.46	0.18	0.00	1.76	-0.21

Tabla 8: Mejor desempeño (distancia al óptimo) ACOTSP - *TSP/1000-3000*.

La Tabla 9 muestra el rendimiento promedio (medido como la distancia a la mejor solución conocida) obtenido por irace y clirace al configurar MMASQAP.

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	**1.10**	1.31	0.00	10.59	6.69
clirace		1.19	1.44	0.00	14.06	6.34
irace	5M	**1.05**	1.32	0.00	10.39	6.18
clirace		1.12	1.41	0.00	12.80	6.07

Tabla 9: Desempeño promedio (distancia al óptimo) MMASQAP

En general, la inclusión del algoritmo de clustering hace que clirace alcance configuraciones que tienen un rendimiento ligeramente peor para ambos presupuestos de configuración. Esto se puede observar en la desviación estándar y los valores máximos. Esta alta variabilidad

en los resultados se explica principalmente por el aumento de la exploración en clirace, lo que permite que diferentes conjuntos de configuraciones realicen el muestreo de nuevas configuraciones. Esto se puede observar, tanto para los valores de presupuesto como para un ligero incremento de la desviación estándar en clirace.

Escenario	B	Promedio	Dev Std.	Min	Max	Kurtosis
irace	1M	0.78	0.54	0.0	3.63	3.83
clirace		**0.76**	0.57	0.0	3.87	3.13
irace	5M	0.76	0.59	0.0	4.98	8.25
clirace		**0.73**	0.56	0.0	3.49	3.38

Tabla 10: Mejor desempeño (distancia al óptimo) MMASQAP. Valores de prueba pareada de Wilcoxon $1M : < 0{,}248$, $10M : < 0{,}476$.

La Tabla 10 muestra el rendimiento de la mejor configuración obtenida por irace y clirace al sintonizar MMASQAP. En general, los valores de máximos, mínimos, desviación estándar y de kurtosis no muestran una diferencia significativa entre ambos algoritmos. Los resultados indican que clirace logró obtener una configuración de mejor rendimiento para ambos presupuestos de configuración. A pesar de esto, los resultados de la prueba de Wilcoxon pareada, que compara el rendimiento de las mejores configuraciones, indican que no hay una diferencia estadística entre su rendimiento.

4.2.1. Tiempo de ejecución

Respecto al tiempo de ejecución de clirace, resulta interesante comparar cuantitativamente en cuanto porcentaje aumenta el tiempo de ejecución con respecto a irace. Sin embargo, en algunos casos no fue posible recuperar el tiempo de ejecución del algoritmo clirace ya que la ejecución fue interrumpida y tuvo que ser reanudada. En cuanto a los casos de los cuales si pudo obtener un tiempo de ejecución completo, observamos que la inclusión del algoritmo de agrupamiento no produce un incremento del tiempo substancial de ejecución de irace. La implementación algorítmica establece eficazmente el clustering basado en cuadrícula y actualiza los clusters mientras mantiene o incluso disminuye los tiempos de ejecución de irace en algunos casos. No obstante, considerando que el número de clústeres generados y activos puede aumentar si se agrupan más parámetros que los considerados para la propuesta, la inclusión de un algoritmo de clustering si podría impactar en el tiempo de ejecución de irace en diferentes escenarios.

4.2.2. Análisis sobre el comportamiento de exploración

Esta sección presenta un análisis de las capacidades de exploración de clirace. El objetivo es observar el efecto de incluir una técnica de clustering, en términos de cómo se modi-

fica el proceso de muestreo durante la ejecución de irace. También analizamos el número de clusters activos (clusters que contienen configuraciones sobrevivientes) a lo largo de las iteraciones de clirace para visualizar el nivel de exploración durante la ejecución.

Las figuras 6 y 7 presentan coordenadas paralelas que ilustran las configuraciones sobrevivientes al final de las 20 ejecuciones de irace (líneas rojas) y clirace (líneas azules), considerando un presupuesto de configuración mayor ($10M$) para el conjunto de instancias *TSP-2000* (ACOTSP). En cada gráfico, se pueden observar los parámetros ajustados de cada configuración resultante y sus valores elegidos.

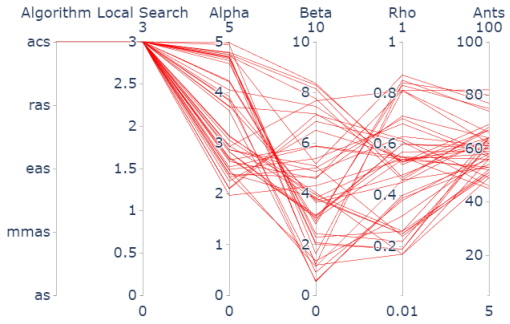


Figura 6: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de irace (líneas en rojo) sintonizando el algoritmo ACOTSP con el conjunto de instancias *TSP-2000* con 10.000 evaluaciones

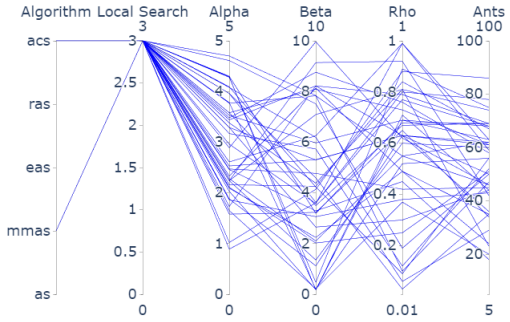


Figura 7: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el conjunto de instancias *TSP-2000* con 10.000 evaluaciones

En general, se puede notar que en ambos casos, los valores que forman parte de las configuraciones sobrevivientes en clirace cubren una porción más grande (o más valores) de los dominios de los parámetros en comparación con los de irace. Además, considerando que los parámetros $ants$ y rho se clusterizan en la ejecución de clirace, se puede observar un comportamiento de cobertura más alto en otros parámetros, tal como lo es el caso de $beta$. Se

pueden observar algunas situaciones interesantes al sintonizar ACOTSP, particularmente al considerar el algoritmo seleccionado y el procedimiento de búsqueda local. En la selección de algoritmos, se seleccionan los enfoques acs y $mmas$ por clirace, mientras que irace solo selecciona acs. Por otro lado, en ambos conjuntos de instancias y ajustadores, solo el operador 3-opt se selecciona como vecindario para el procedimiento de búsqueda local. Estas situaciones reflejan que la inclusión de la técnica de clustering permite a irace explorar más posibilidades, mientras aún mantiene cierta intensificación.

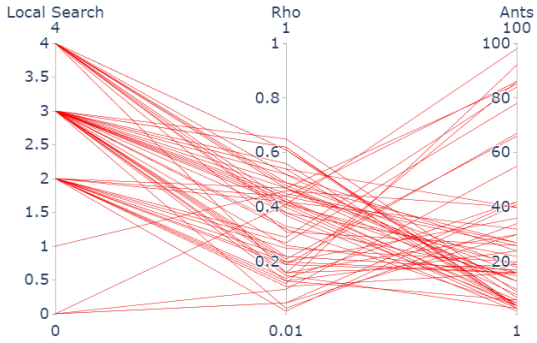


Figura 8: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de irace (líneas en rojo) sintonizando el algoritmo MMASQAP con 1.000 evaluaciones

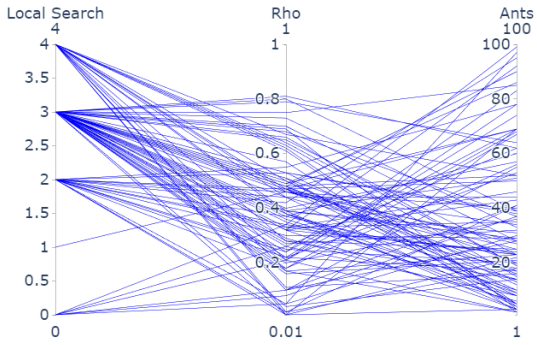


Figura 9: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de clirace (líneas en azul) sintonizando el algoritmo MMASQAP con 1.000 evaluaciones

Las figuras 8 y 9 muestran el gráfico de coordenadas paralelas para el caso del conjunto de instancias *QAP-60* con un presupuesto de configuración mucho menor ($1M$) para el algoritmo de MMASQAP.

En este caso, podemos ver que aún para un presupuesto de configuración más reducido, los

valores de los parámetros tienen a cubrir mayor parte del dominio, particularmente para los valores de rho y $ants$, los cuales son los parámetros utilizados para clusterizar. En particular, los valores de $localsearch$ se mantienen en el mismo rango. Sin embargo, se puede ver que se incluyen más configuraciones que no consideran el procedimiento de búsqueda local.

Para el siguiente análisis, utilizamos el paquete `iraceplot`[1], el cual nos permite hacer un análisis más detallado del comportamiento de irace y clirace durante el proceso de configuración.

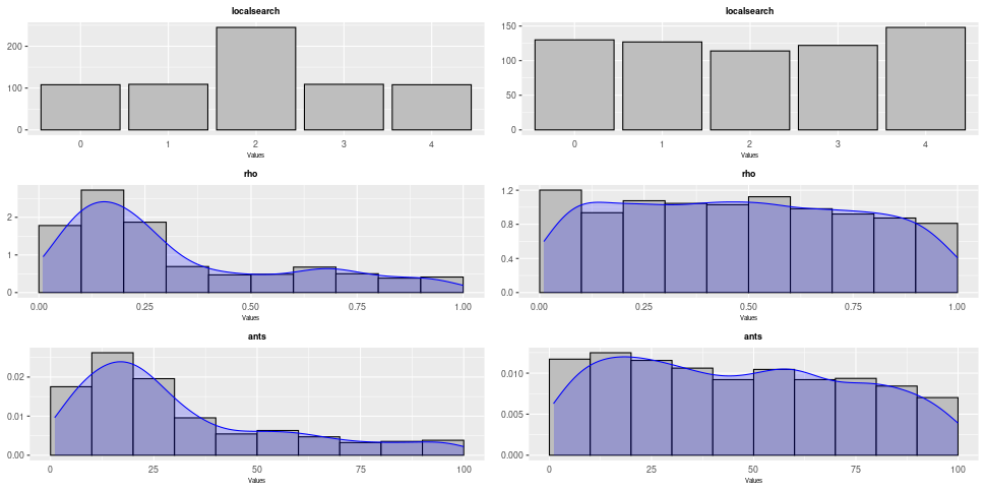


Figura 10: Gráfico de frecuencia de muestreo para irace (izquierda) y clirace (derecha) al sintonizar MMASQAP con 5.000 evaluaciones.

La figura 10 muestra los gráficos de frecuencia de muestreo de irace (izquierda) y clirace (derecha) para cada parámetro sintonizado de MMASQAP, el cual describe con cuanta frecuencia se muestrearon los valores del dominio de cada parametro. Se puede observar que irace se enfoca en subregiones del dominio de cada parámetro. La distribución de muestras de clirace es más dispersa para los 3 parámetros (independientemente de cuáles parámetros se eligieron para ser clusterizados), con picos suaves en algunos valores particulares que se asimilan a los puntos más altos de los gráficos de frecuencia de irace.

En la figura 11 podemos observar los gráficos de frecuencia de muestreo de irace (izquierda) y clirace (derecha) para el escenario de ACOTSP-1M3M. En este caso, notamos que con un presupuesto de configuración mayor, la diferencia entre las frecuencias de muestreo es mucho más notable que para escenarios con menor cantidad de evaluaciones.

[1]https://auto-optimization.github.io/iraceplot/

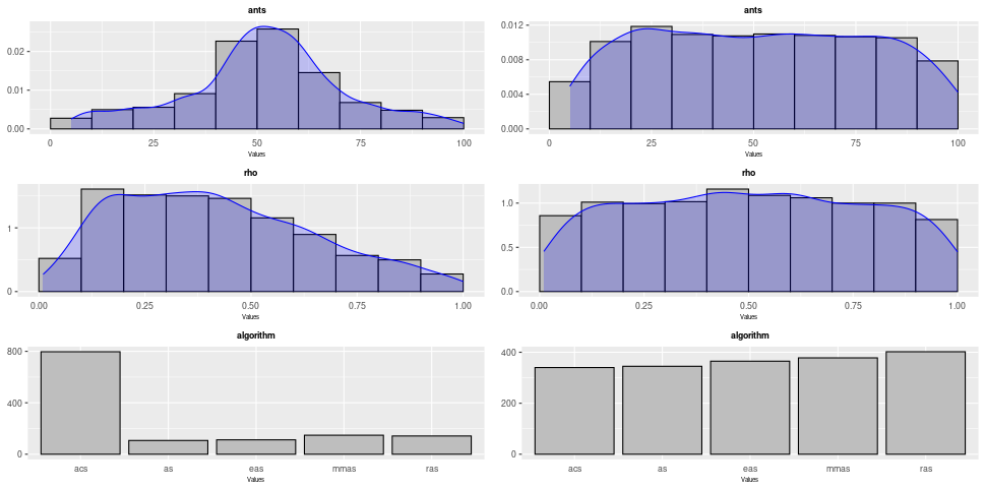


Figura 11: Gráfico de frecuencia de muestreo para irace (izquierda) y clirace (derecha) al sintonizar ACOTSP con el set de instancias 1M-3M con 10.000 evaluaciones.

Ahora, analizamos el comportamiento de exploración mediante la diferencia entre las configuraciones de cada proceso de sintonización.

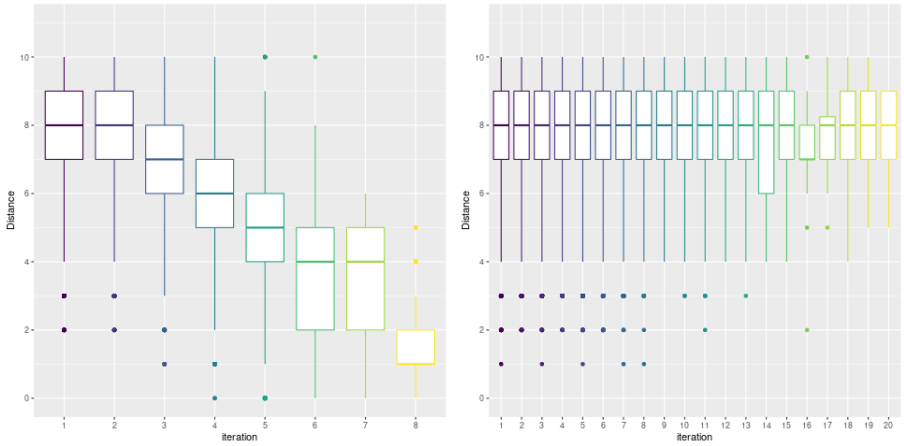


Figura 12: Gráfico de distancia de muestreo para irace (izquierda) y clirace (derecha) al sintonizar ACOTSP con el set de instancias 2M con 10.000 evaluaciones.

La figura 12 ilustran la distancia entre las configuraciones muestreadas durante una ejecución de irace y clirace, respectivamente. En el caso de irace, se puede observar un patrón de convergencia hacia configuraciones similares (distancias menores) a lo largo de las iteraciones. Por otro lado, en clirace, la distancia entre las configuraciones se mantiene constante durante la primera mitad de las iteraciones y luego tiende a disminuir ligeramente en las últimas iteraciones. Es importante destacar la diferencia en la cantidad de iteraciones realizadas por irace y clirace, con clirace llevando a cabo más del doble de iteraciones que irace. Esto podría deberse a que el muestreo de configuraciones genera configuraciones muy distintas entre sí, lo que provoca un retraso en la convergencia del algoritmo hacia configuraciones

similares en valor y de buen desempeño.

Finalmente, revisamos las características de exploración de clirace con respecto a la cantidad de clusters activos durante una ejecución de clirace. Los clusters activos representan cuantas regiones del espacio de parámetros contienen configuraciones que siguen en la carrera de irace, por lo tanto, no han sido eliminadas por el test estadístico.

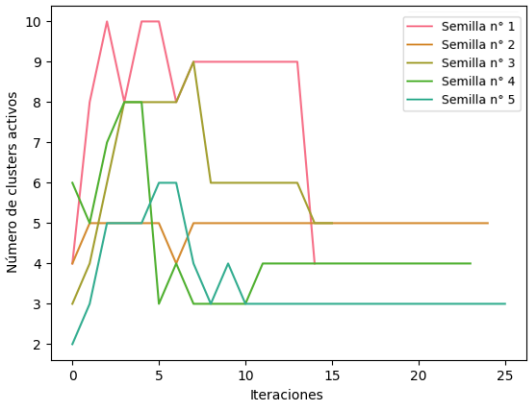


Figura 13: Número de clusters activo durante 5 ejecuciones de clirace sintonizando el algoritmo ACOTSP con el conjutno de instancias*TSP-1000/3000* con 10.000 evaluaciones

La figura 13 muestra el número de clusters activos por iteración de clirace para el escenario *TSP-1000/3000*. En general, el número de clusters activos tiende a aumentar en las primeras cinco iteraciones de clirace. Luego, el número de clusters activos tiende a disminuir en las siguientes cinco iteraciones. Se observa una convergencia del número de clusters activos en todas las ejecuciones, alrededor de la mitad del número total de iteraciones. Este comportamiento nos permite entender que el número de clusters podría adaptarse dinámicamente durante la ejecución de clirace.

4.2.3. Conclusiones

En esta sección, se ha llevado a cabo una exhaustiva validación de la propuesta de criterios de aceptación de irace basados en clustering, evaluando su rendimiento en distintos escenarios experimentales diseñados para realizar un correcto análisis de la solución implementada. A través de la realización de experimentos con 2 algoritmos distintos, ACOTSP Y MMASQAP, junto a distintos presupuestos de configuración y conjuntos de instancias, permiten presentar los resultados y análisis de los mismos.

Los resultados obtenidos indican que, en general, clirace tiene un rendimiento inferior al de irace. El potencial de clirace se aprecia principalmente en escenarios con un mayor presupuesto de configuración y conjuntos de instancias más heterogéneos, donde los resultados

pueden ser muy similares e incluso mejores que los de irace en algunos casos. Además, la solución basada en k-medoids ofrece resultados competitivos en comparación con irace en ciertas situaciones.

Los gráficos presentados en esta sección ofrecen una representación visual de la capacidad exploratoria de clirace, destacando características tales como gráficos de coordenadas paralelas, distribución de frecuencia de muestreo de cada parámetro, distancia entre configuraciones muestreadas y cantidad de clusters activos por iteración. Estos gráficos demuestran que clirace mejora la capacidad exploratoria del proceso de configuración al generar configuraciones a partir de diversos valores en el espacio de parámetros.

CAPÍTULO 5
CONCLUSIONES

El problema de la configuración automática de algoritmos es muy importante en el campo de la optimización meta-heurística. Las meta-heurísticas a menudo definen una gran cantidad de parámetros que influyen en gran medida en su comportamiento, incluyendo características como el tiempo de ejecución, los recursos utilizados y la calidad de la solución obtenida. Por lo tanto, encontrar los mejores valores para estos parámetros puede ser un proceso bastante complicado, y varios algoritmos de configuración se han mencionado en la literatura para abordar este problema.

Entre estos algoritmos, el algoritmo irace destaca por su capacidad para configurar varios algoritmos del estado del arte a través de un proceso de racing iterativo, utilizando test estadísticos para descartar configuraciones de bajo rendimiento. Al final de la carrera, se selecciona un conjunto de configuraciones élite de las configuraciones supervivientes que no fueron descartadas, a través de un proceso de selección con truncamiento *greedy*.

Este trabajo propone la incorporación de un algoritmo de clustering en los criterios de aceptación de configuraciones de irace. El objetivo es mejorar las capacidades exploratorias al seleccionar configuraciones de élite basándose en su ubicación en el espacio de parámetros, en lugar de centrarse en su rendimiento. La propuesta promueve la selección de configuraciones más diversas para el proceso de muestreo de irace, lo que permite generar configuraciones distribuidas en todo el espacio y evita la convergencia prematura del algoritmo.

Evaluamos la implementación de un algoritmo de clustering basado en cuadrícula o grid-based, con el objetivo de comparar su rendimiento con la versión tradicional de irace. Además, evaluamos una variante que utiliza un método tradicional de clustering, K-medoids.

El diseño de la metodología experimental incluye ACOTSP y MMASQAP como algoritmos objetivo, junto con distintos escenarios de presupuestos de computación, semillas y conjuntos de instancias. Esto nos permite obtener resultados confiables para llevar a cabo un análisis exhaustivo.

Los resultados muestran que los niveles de exploración de clirace aumentan al usar el algoritmo de clustering, alcanzando configuraciones estructuralmente diferentes a diferencia de aquellos niveles obtenidos por irace. Esto podemos verlo reflejado en los valores resultantes de las configuraciones finales, donde los valores de los parámetros difieren entre sí de manera más notable que para el caso de las configuraciones finales de irace.

Es importante destacar que los resultados indican que al aumentar el presupuesto para evaluaciones, la exploración permite a clirace alcanzar configuraciones de alta calidad. Esto podría deberse a que, con presupuestos más reducidos, el efecto de la convergencia prematura de irace no es tan imponente como en escenarios con mayor presupuesto. En escenarios de

sintonización más grandes, es probable que el algoritmo alcance la convergencia (hacia configuraciones de valor y desempeño similar), haciendo más visible el efecto de los algoritmos de clustering. Además, considerando la homogeneidad de algunos conjuntos de instancias, niveles más bajos de exploración podrían ser beneficiosos para llegar rápidamente a configuraciones de mayor calidad.

También, el análisis del comportamiento de exploración clirace muestra que la diversidad entre las configuraciones muestreadas durante su ejecución tiende a mantenerse a lo largo de un mayor número de iteraciones que para el caso de irace. Sin embargo, en un escenario (*TSP-1000/3000*), la falta de balance entre exploración y explotación resulta en que clirace no converja hacia configuraciones de buena calidad.

5.1. Trabajo futuro

Para futuros trabajos, se considerará la adaptación del número de clusters activos durante la ejecución de clirace. Los clusters podrían fusionarse o subdividirse según el estado del proceso de búsqueda, ya que diferentes grados de las características de exploración y explotación del algoritmo deben aplicarse en distintos puntos de la ejecución.

En la propuesta actual, no existe una heurística de selección de parámetros a clusterizar, lo que hace que esta decisión recaiga sobre el usuario. Agregar heurísticas de selección basadas en el dominio de los parámetros podría ser una mejora considerable para el algoritmo. También se considera la inclusión de parámetros categóricos en el proceso de agrupamiento, ya que para muchos casos, esta clase de parámetros es determinante para el desempeño del algoritmo objetivo y pueden llegar a determinar los valores de los parámetros numéricos.

Por otro lado, es crucial considerar las limitaciones de la propuesta respecto a la dimensionalidad de los algoritmos a sintonizar. A medida que aumentan los parámetros a considerar, la complejidad del problema crece. Usar una técnica con una implementación sencilla, como el agrupamiento basado en cuadrículas, puede hacer que el uso de clirace sea inviable.

Por lo tanto, es esencial estudiar otras técnicas de agrupamiento que permitan manejar grandes cantidades de parámetros, ya que estos casos son comunes en la práctica. Asimismo, es vital analizar los parámetros que se utilizarán para el algoritmo de agrupamiento y el número de divisiones por parámetro.

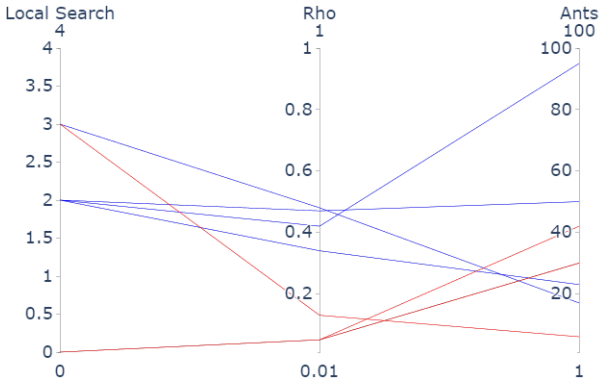


Figura 14: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

ANEXOS

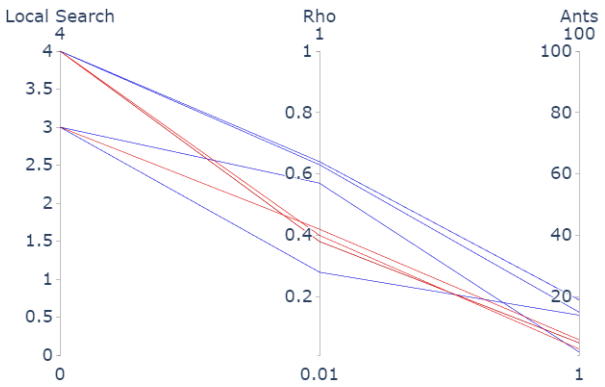


Figura 15: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

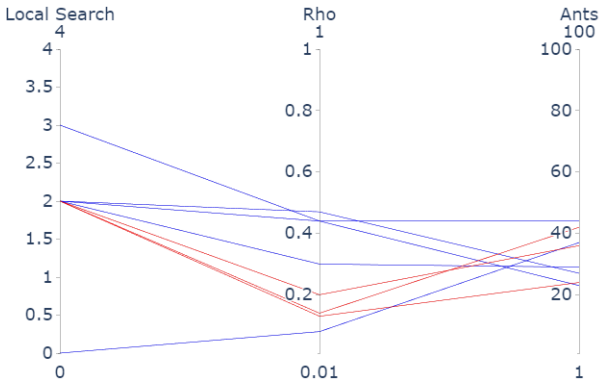


Figura 16: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

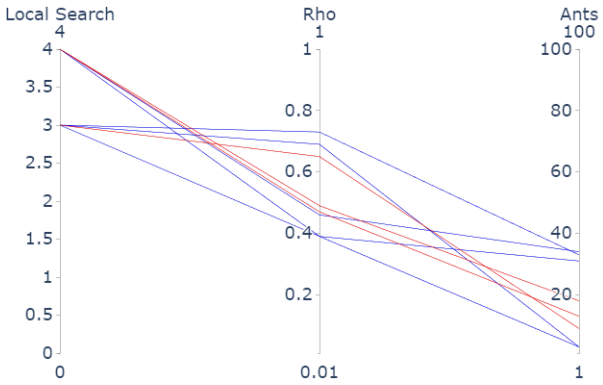


Figura 17: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

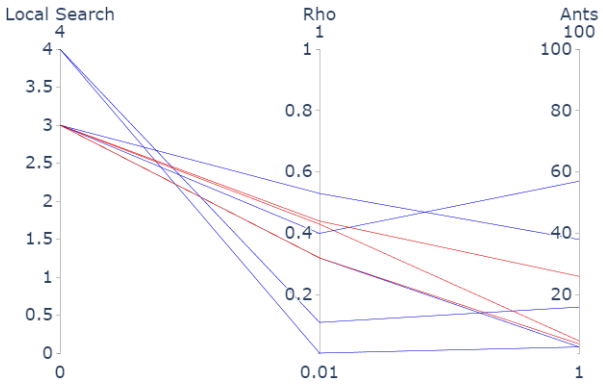


Figura 18: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

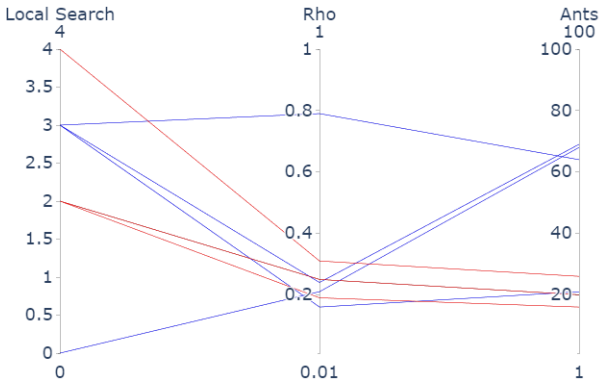


Figura 19: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

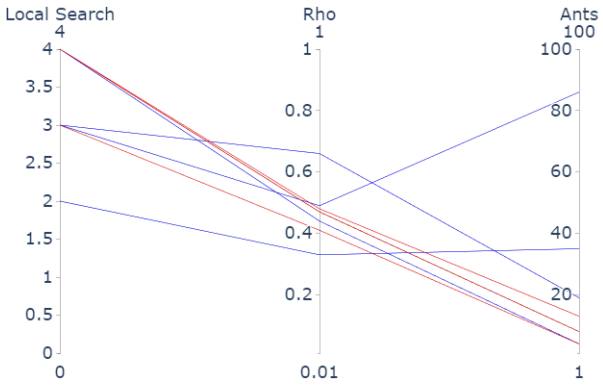


Figura 20: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

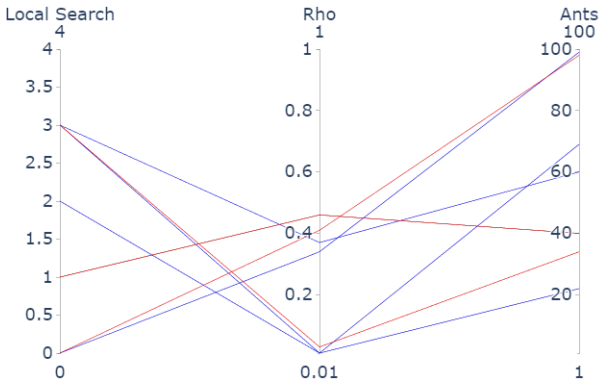


Figura 21: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 1.000 evaluaciones

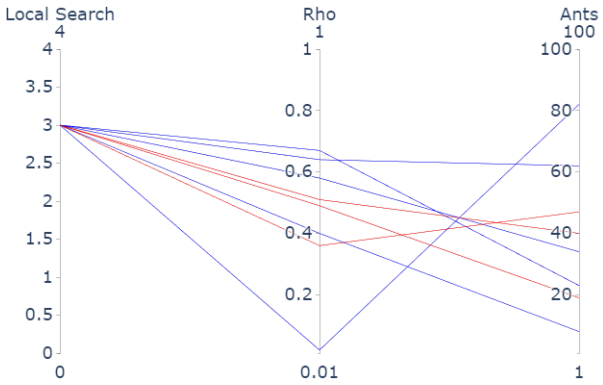


Figura 22: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 5.000 evaluaciones

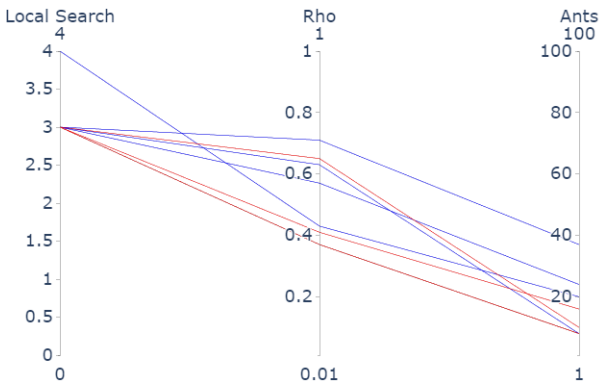


Figura 23: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 5.000 evaluaciones

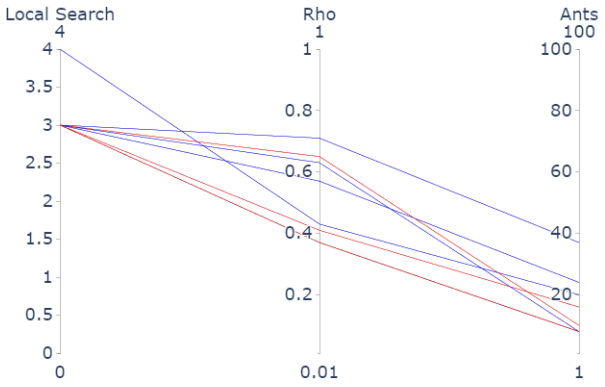


Figura 24: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 5.000 evaluaciones

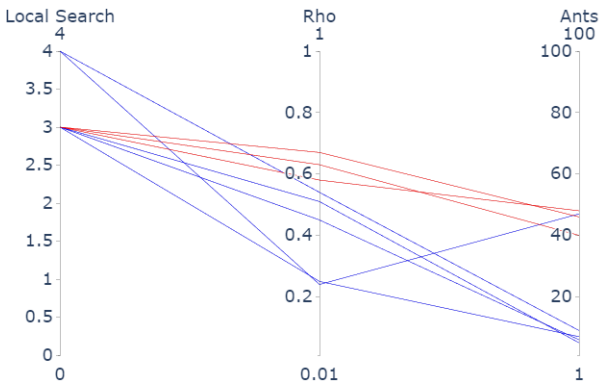


Figura 25: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 5.000 evaluaciones

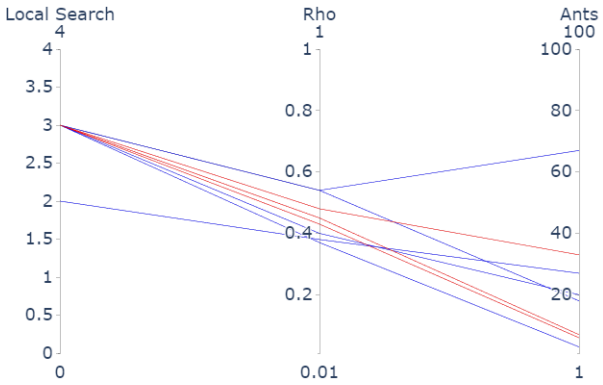


Figura 26: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMAS-QAP con 5.000 evaluaciones

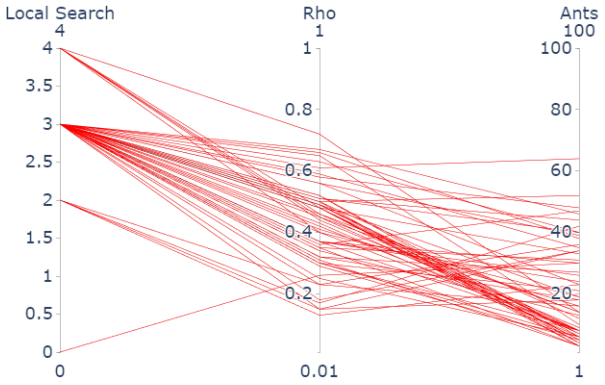


Figura 27: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de irace sintonizando el algoritmo MMASQAP con 5.000 evaluaciones

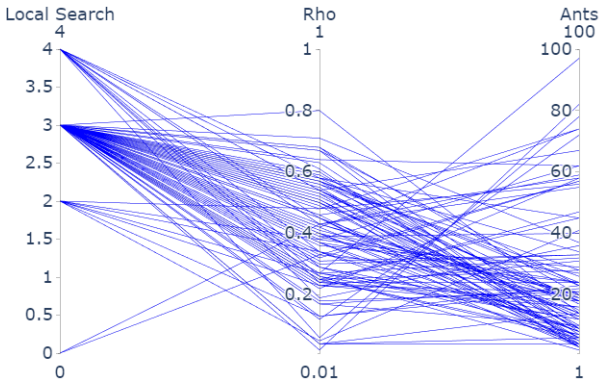


Figura 28: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de clirace sintonizando el algoritmo MMASQAP con 5.000 evaluaciones

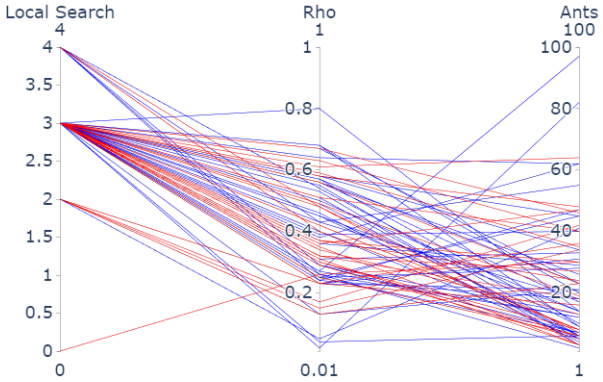


Figura 29: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 10 ejecuciones de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo MMASQAP con 5.000 evaluaciones

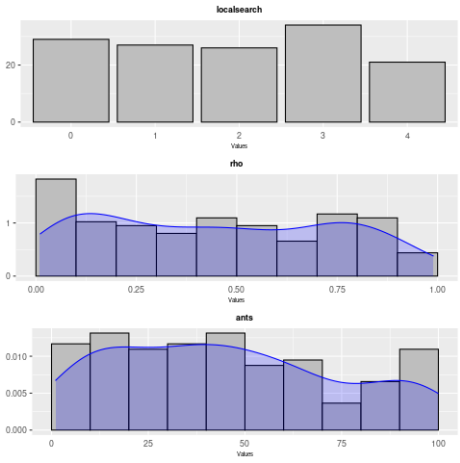


Figura 30: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 1.000 evaluaciones.

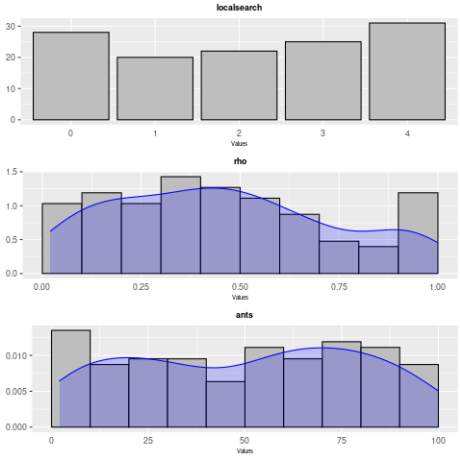


Figura 31: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 1.000 evaluaciones.

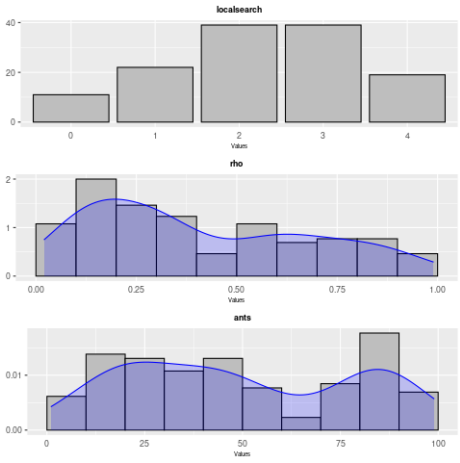


Figura 32: Gráfico de frecuencia de muestreo para irace al sintonizar MMASQAP con 1.000 evaluaciones.

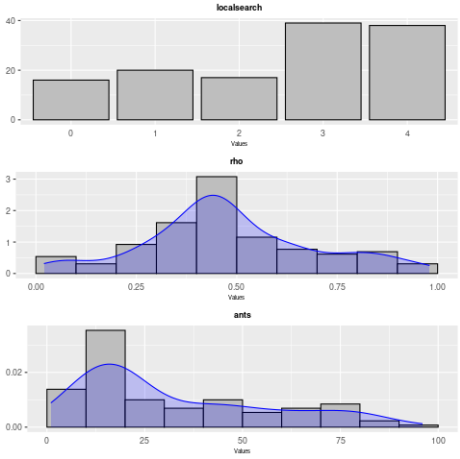


Figura 33: Gráfico de frecuencia de muestreo para irace al sintonizar MMASQAP con 1.000 evaluaciones.

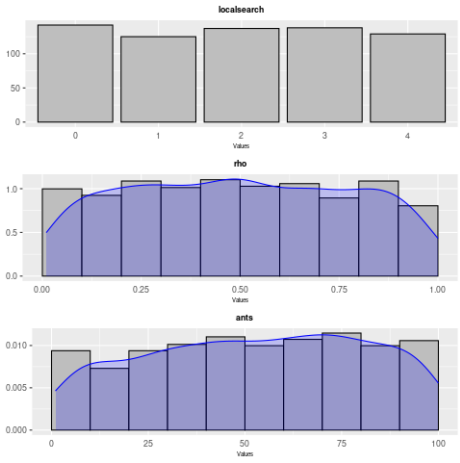


Figura 34: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 5.000 evaluaciones.

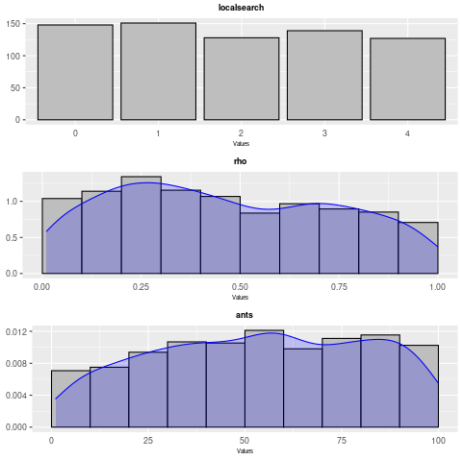


Figura 35: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 5.000 evaluaciones.

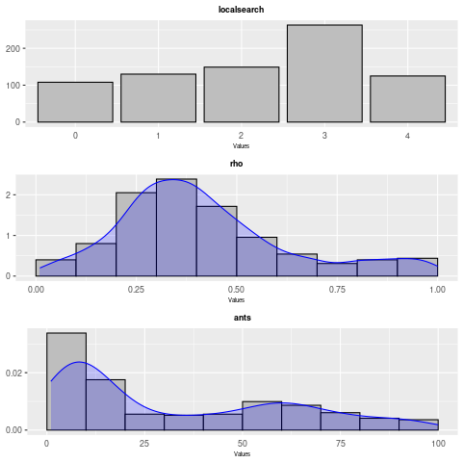


Figura 36: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 5.000 evaluaciones.

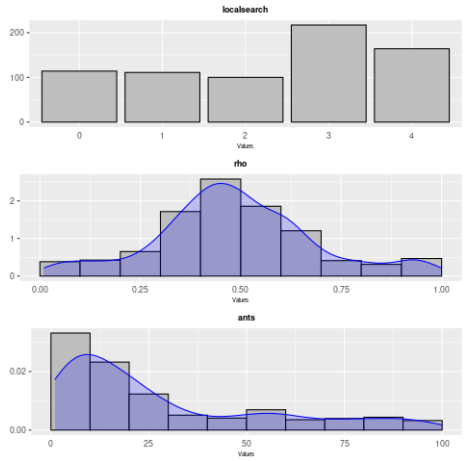


Figura 37: Gráfico de frecuencia de muestreo para clirace al sintonizar MMASQAP con 5.000 evaluaciones.

Figura 38: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 1.000 evaluaciones

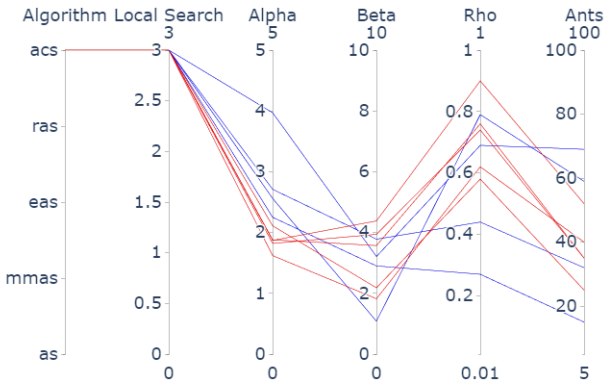


Figura 39: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 1.000 evaluaciones

Figura 40: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 1.000 evaluaciones

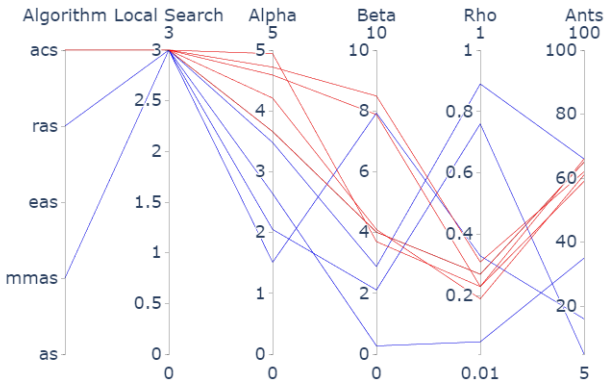


Figura 41: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 1.000 evaluaciones

Figura 42: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

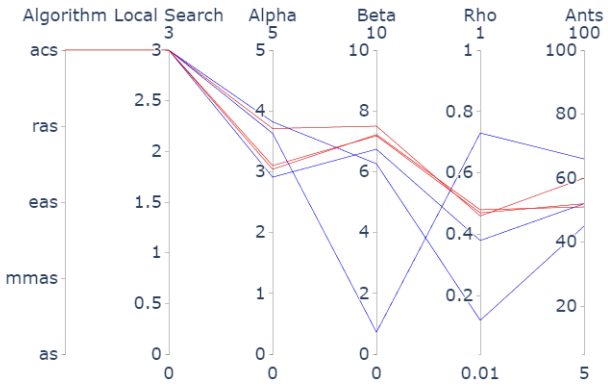


Figura 43: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

Figura 44: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

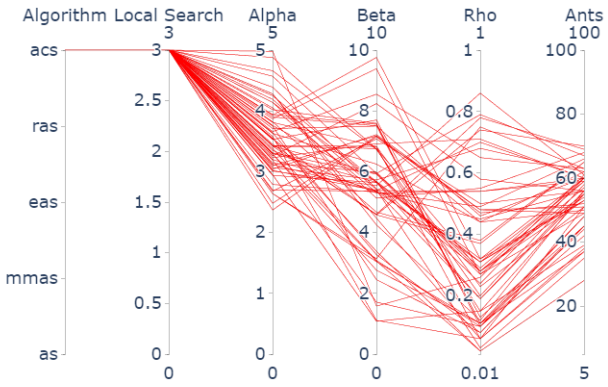


Figura 45: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de irace sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

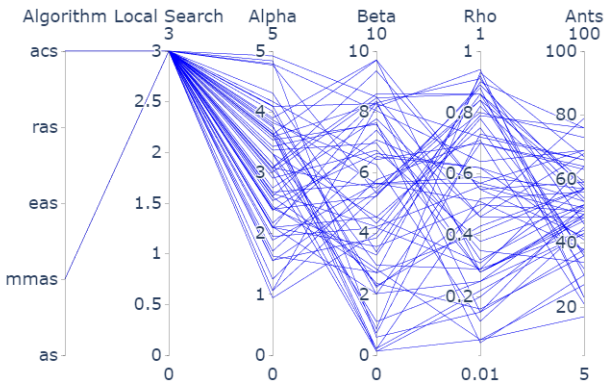


Figura 46: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de clirace sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

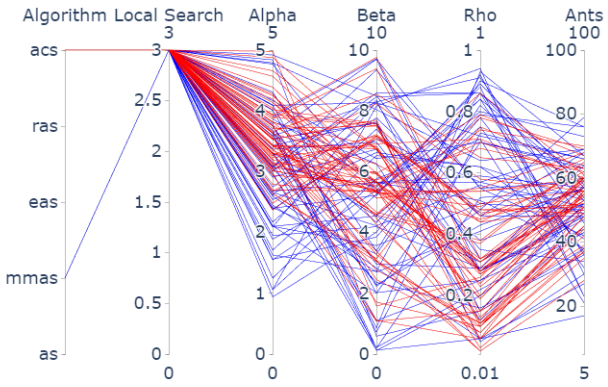


Figura 47: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 20 ejecuciones de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-1000/3000* con 10.000 evaluaciones

Figura 48: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 1.000 evaluaciones

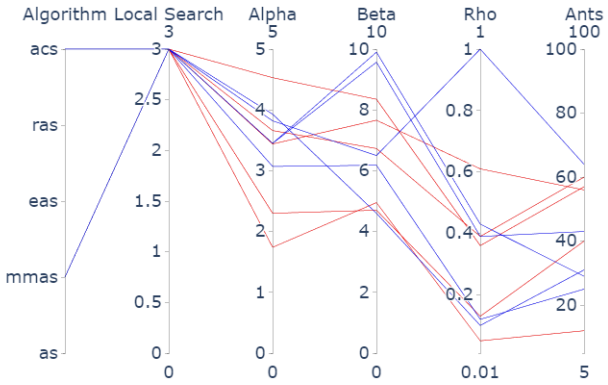


Figura 49: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 1.000 evaluaciones

Figura 50: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 1.000 evaluaciones

Figura 51: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de k-clirace para $K = 10$ sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 1.000 evaluaciones

Figura 52: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

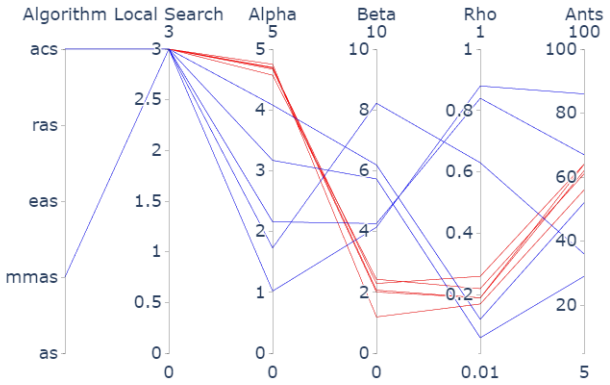


Figura 53: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

Figura 54: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

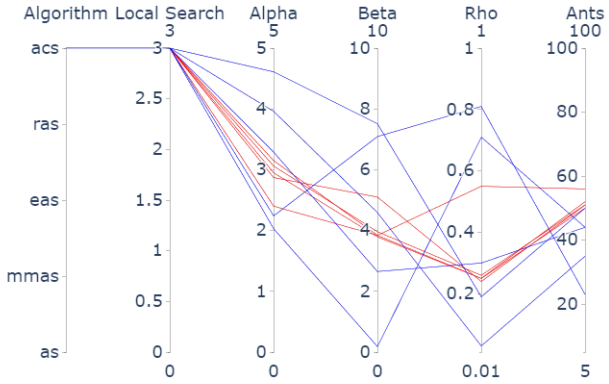


Figura 55: Gráfico de coordenadas paralelas de configuraciones sobrevivientes al final de 1 ejecución de irace (lineas en rojo) y clirace (líneas en azul) sintonizando el algoritmo ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

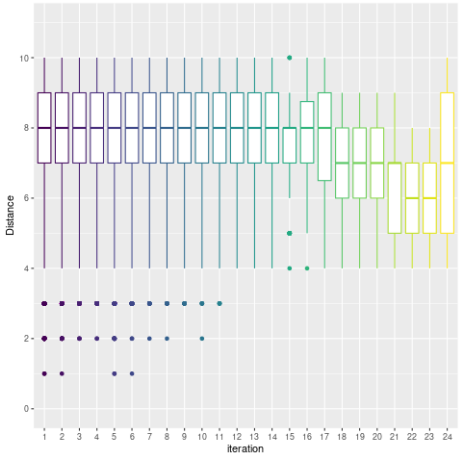


Figura 56: Gráfico de distancia de muestreo para irace (izquierda) y clirace (derecha) al sintonizar ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

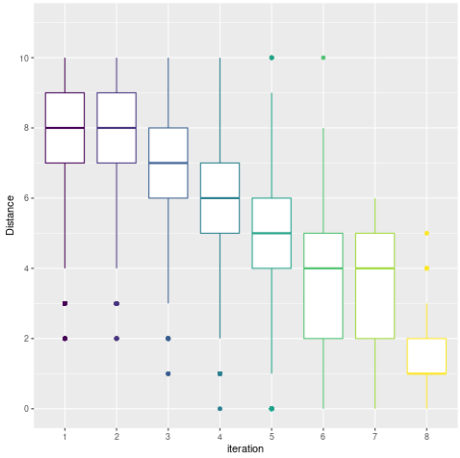


Figura 57: Gráfico de distancia de muestreo para irace (izquierda) y clirace (derecha) al sintonizar ACOTSP con el set de instancias *TSP-2000* con 10.000 evaluaciones

REFERENCIAS BIBLIOGRÁFICAS

[Adenso-Díaz y Laguna, 2006] Adenso-Díaz, B. y Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114.

[Anastacio *et al.*, 2019] Anastacio, M., Luo, C., y Hoos, H. (2019). Exploitation of default parameter values in automated algorithm configuration. *Proceedings of the Workshop Data Science Meets Optimization*.

[Ansótegui *et al.*, 2009] Ansótegui, C., Sellmann, M., y Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. En *Principles and Practice of Constraint Programming*, pp. 142–157.

[Balaprakash *et al.*, 2007] Balaprakash, P., Birattari, M., y Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. En *Hybrid Metaheuristics*, volumen 4771, pp. 108–122.

[Bandaru y Deb, 2010] Bandaru, S. y Deb, K. (2010). Automating discovery of innovative design principles through optimization.

[Bartz-Beielstein *et al.*, 2005] Bartz-Beielstein, T., Lasarczyk, C. G., y Preuss, M. (2005). Sequential parameter optimization. En *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 773–780.

[Berkhin, 2006] Berkhin, P. (2006). *A Survey of Clustering Data Mining Techniques*, pp. 25–71. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bezerra *et al.*, 2017] Bezerra, L. C. T., López-Ibá nez, M., y Stützle, T. (2017). Automatic configuration of multi-objective optimizers and multi-objective configuration. Technical Report TR/IRIDIA/2017-011, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

[Birattari *et al.*, 2002] Birattari, M., Stützle, T., Paquete, L., y Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. En *Experimental Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 11–18.

[Cai *et al.*, 2016] Cai, F., Le-Khac, N.-A., y Kechadi, T. (2016). Clustering approaches for financial data analysis: a survey.

[Cáceres *et al.*, 2017a] Cáceres, L. P., Bischl, B., y Stützle, T. (2017a). Evaluating random forest models for irace. En *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1146–1153.

[Cáceres *et al.*, 2017b] Cáceres, L. P., López-Ibáñez, M., Hoos, H., y Stützle, T. (2017b). An experimental study of adaptive capping in irace. En *Lecture Notes in Computer Science*, pp. 235–250.

[Dorigo *et al.*, 2006] Dorigo, M., Birattari, M., y Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

[Dorigo *et al.*, 1991] Dorigo, M., Maniezzo, V., y Colorni, A. (1991). Ant system: An autocatalytic optimizing process.

[Dorigo *et al.*, 1996] Dorigo, M., Maniezzo, V., y Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

[Eiben *et al.*, 1999] Eiben, A., Hinterding, R., y Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141.

[Eiben y Smit, 2011] Eiben, A. y Smit, S. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

[Ester *et al.*, 1996] Ester, M., Kriegel, H.-P., Sander, J., y Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. En *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231. AAAI Press.

[Franzin *et al.*, 2018] Franzin, A., Cáceres, L. P., y Stützle, T. (2018). Effect of transformations of numerical parameters in automatic algorithm configuration. *Optimization Letters*, 12(8):1741–1753.

[Glover, 1989] Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edición.

[Gomes Pereira de Lacerda *et al.*, 2021] Gomes Pereira de Lacerda, M., de Araujo Pessoa, L. F., Buarque de Lima Neto, F., Ludermir, T. B., y Kuchen, H. (2021). A systematic literature review on general parameter control for evolutionary and swarm-based algorithms. *Swarm and Evolutionary Computation*, 60:100777.

[Grefenstette, 1986] Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128.

[Guha *et al.*, 1998] Guha, S., Rastogi, R., y Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. En *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, p. 73–84, New York, NY, USA. Association for Computing Machinery.

[Gunawan *et al.*, 2011] Gunawan, A., Lau, H. C., y Lindawati (2011). Fine-tuning algorithm parameters using the design of experiments approach. En *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pp. 278–292. Springer.

[Han *et al.*, 2001] Han, J., Kamber, M., y Tung, A. (2001). Spatial clustering methods in data mining: a survey. *Data Mining and Knowledge Discovery - DATAMINE*, pp. 1–2.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: University of Michigan Press.

[Hoos, 2011] Hoos, H. (2011). *Automated Algorithm Configuration and Parameter Tuning*.

[Huang *et al.*, 2020] Huang, C., Li, Y., y Yao, X. (2020). A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216.

[Hutter *et al.*, 2011] Hutter, F., Hoos, H. H., y Leyton□Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. En *Learning and Intelligent Optimization*, volumen 6683, pp. 507–523.

[Hutter *et al.*, 2009] Hutter, F., Stützle, T., Leyton-Brown, K., y Hoos, H. H. (2009). Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

[Karafotias *et al.*, 2015] Karafotias, G., Hoogendoorn, M., y Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187.

[Kaufman y Rousseeuw, 1990] Kaufman, L. y Rousseeuw, P. (1990). *Finding Groups in Data: An Introduction To Cluster Analysis*.

[Kennedy y Eberhart, 1995] Kennedy, J. y Eberhart, R. (1995). Particle swarm optimization. En *Proceedings of ICNN'95 - International Conference on Neural Networks*, volumen 4, pp. 1942–1948 vol.4.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt, C. D., y Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[Kriegel *et al.*, 2011] Kriegel, H.-P., Kröger, P., Sander, J., y Zimek, A. (2011). Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, 1(3):231–240.

[Lozano y García-Martínez, 2010] Lozano, M. y García-Martínez, C. (2010). Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Computers Operations Research*, 37:481–497.

[López-Ibáñez *et al.*, 2016] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., y Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

[Ma *et al.*, 2024] Ma, F., Wang, C., Huang, J., Zhong, Q., y Zhang, T. (2024). Key grids based batch-incremental clique clustering algorithm considering cluster structure changes. *Information Sciences*, 660.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. En *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volumen 1, pp. 281–297. Oakland, CA, USA.

[Montero *et al.*, 2014] Montero, E., Riff, M. C., y Neveu, B. (2014). A beginner's guide to tuning methods. *Applied Soft Computing*, 17:39–51.

[Müllner, 2011] Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms.

[Patnaik *et al.*, 2016] Patnaik, A. K., Bhuyan, P. K., y Krishna Rao, K. (2016). Divisive analysis (diana) of hierarchical clustering and gps data for level of service criteria of urban streets. *Alexandria Engineering Journal*, 55(1):407–418.

[Pham *et al.*, 2011] Pham, M. C., Cao, Y., Klamma, R., y Jarke, M. (2011). A clustering approach for collaborative filtering recommendation using social network analysis. *J.UCS Journal of Universal Computer Science*, 17(4):583–604.

[Riff y Montero, 2013] Riff, M. C. y Montero, E. (2013). A new algorithm for reducing metaheuristic design effort. En *IEEE Congress on Evolutionary Computation*.

[S. Das y Konar, 2009] S. Das, A. A. y Konar, A. (2009). *Metaheuristic Clustering*, capítulo 1, pp. 14–15. "Metaheuristic Clustering.

[Saifullah Hussin y Stützle, 2014] Saifullah Hussin, M. y Stützle, T. (2014). Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Comput. Oper. Res.*, 43:286–291.

[Sheikholeslami *et al.*, 1998] Sheikholeslami, G., Chatterjee, S., y Zhang, A. (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. En *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, p. 428–439, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Srinivas y Patnaik, 1994] Srinivas, M. y Patnaik, L. (1994). Genetic algorithms: a survey. *Computer*, 27(6):17–26.

[Stützle, 2002] Stützle, T. (2002). Acotsp: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem.

[Stützle y Hoos, 1998] Stützle, T. y Hoos, H. (1998). Improvements on the ant-system: Introducing the max-min ant system. En *Artificial Neural Nets and Genetic Algorithms*, pp. 245–249, Vienna.

[Wang *et al.*, 1997] Wang, Wei and Yang, Jiong and Muntz, Richard and others (1997). Sting: A statistical information grid approach to spatial data mining. En *Vldb*, volumen 97, pp. 186–195.

[Wessing y López-Ibáñez, 2019] Wessing, S. y López-Ibáñez, M. (2019). Latin hypercube designs with branching and nested factors for initialization of automatic algorithm configuration. *Evolutionary Computation*, 27(1):129–145.

[Ye *et al.*, 2022] Ye, F., Vermetten, D., Doerr, C., y Bäck, T. (2022). Non-elitist selection can improve the performance of irace. En *International Conference on Parallel Problem Solving from Nature*, pp. 32–45.

[Zhang *et al.*, 1996] Zhang, T., Ramakrishnan, R., y Livny, M. (1996). Birch: an efficient data clustering method for very large databases. En *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, p. 103–114, New York, NY, USA. Association for Computing Machinery.

[žurauskienė y Yau, 2016] žurauskienė, J. y Yau, C. (2016). pcareduce: hierarchical clustering of single cell transcriptional profiles. *BMC Bioinformatics*, 17(1):140.