Universidad de Oviedo

Programa Oficial de Doctorado en Ingeniería de Producción, Minero-Ambiental y de Proyectos

TESIS DOCTORAL

Exploitation of Parallelism in Population-Based Metaheuristics and Application to a Galvanizing Line

Diego Díaz Fidalgo

Noviembre de 2021

Universidad de Oviedo

Programa Oficial de Doctorado en Ingeniería de Producción, Minero-Ambiental y de Proyectos

TESIS DOCTORAL

Exploitation of Parallelism in Population-Based Metaheuristics and Application to a Galvanizing Line

Memoria presentada para la obtención del grado de Doctor por la Universidad de Oviedo

Diego Díaz Fidalgo

Directores: César Menéndez Fernández, Profesor Titular del Departamento de Matemáticas de la Universidad de Oviedo; y Francisco Ortega Fernández, Catedrático del Departamento de Explotación y Prospección de Minas.

Oviedo, Noviembre de 2021

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

RESUMEN DEL CONTENIDO DE TESIS DOCTORAL

1.- Título de la Tesis	
Español/Otro Idioma: **Explotación del paralelismo en la mejora de algoritmos metaheurísticos evolutivos y aplicación a una línea de galvanizado**	Inglés: **Exploitation of Parallelism in Population-Based Metaheuristics and Application to a Galvanizing Line**

2.- Autor	
Nombre: **DIEGO DIAZ FIDALGO**	DNI/Pasaporte/NIE:
Programa de Doctorado: **INGENIERÍA DE PRODUCCIÓN, MINERO-AMBIENTAL Y DE PROYECTOS**	
Órgano responsable: **UNIVERSIDAD DE OVIEDO - CENTRO INTERNACIONAL DE POSTGRADO**	

FOR-MAT-VOA-010 (Reg.2018)

RESUMEN (en español)

El acero es ubicuo en el mundo moderno, y una parte importante de muchos sectores económicos. El segmento del Acero Galvanizado es el mayor entre los diferentes productos de acero, dada su utilización en un amplio abanico de industrias, tales como la construcción, el automóvil, los electrodomésticos, y muchos otros.

La programación, o secuenciación, de las bobinas que se producen en una Línea de Galvanizado es extremadamente importante, puesto que tiene un gran impacto en los costes y la factibilidad de la producción.

Este problema de secuenciación es muy parecido a un Problema del Viajante (Travelling Salesperson Problem, TSP), con la complicación añadida de que algunos segmentos no son factibles. Para ciertos productos existen restricciones adicionales; estas restricciones invalidan cualquier aproximación basada en TSP, al depender de subsecuencias más largas que una bobina y su sucesora inmediata.

Las soluciones típicas para este problema se basan en programación por restricciones que depende de prioridades de las restricciones diseñadas cuidadosamente para cada línea con sus condiciones particulares y su combinación de productos.

El problema de programación de tareas (Job Scheduling Problem, JSP) y sus variantes no encajan adecuadamente con la programación de una Línea de Galvanizado. Igualmente incapaces de representar todas las condiciones necesarias son las formulaciones como problema de programación lineal entera (Integer Linear Program, ILP).

Descartados los métodos exactos, la aproximación habitual son los metaheurísticos. Los metaheurísticos basados en población, en particular, son fácilmente paralelizables y su estructura común permite diseñar métodos de alto nivel aplicables a todos ellos.

La evolución de los ordenadores lleva a la paralelización de algoritmos para extraer toda la capacidad computacional de los sistemas modernos, y así poder atacar problemas de mayor tamaño y complejidad. Esta tendencia ya resulta evidente en la literatura.

Al atacar un problema de optimización complejo de gran escala utilizando metaheurísticos debe tenerse en cuenta cuál es la mejor manera de sacar el máximo rendimiento de los equipos y cómo encaja con las propiedades del problema en cuestión.

Los desarrollos en metaheurísticos llevan décadas explotando estas capacidades, empezando por implementaciones muy específicas y avanzando hacia aproximaciones más generales.

Los beneficios de hacer más eficiente el uso de los recursos debe equilibrarse con la

complejidad incurrida, tanto en la construcción del algoritmo como en la comunicación añadida, que en la práctica pueden perjudicar en mayor medida que los beneficios obtenidos. Este equilibrio se puede interpretar como un problema de optimización multi-objetivo. Los algoritmos existentes pueden situarse en el frente de Pareto representando diferentes compromisos entre las métricas. Aunque se pueden obtener mejores soluciones con métodos más complejos y computacionalmente más exigentes, hay un claro hueco en la región de baja complejidad.
Esta tesis trata de cubrir este hueco, proponiendo un método, denominado Multiverso, centrado en mejorar el desempeño en la región de la línea de referencia del frente de Pareto. No es un algoritmo concreto, sino una familia de algoritmos resultante de la aplicación de una transformación a cualquier metaheurístico basado en población.
El método Multiverso se prueba sobre el TSP, tanto con Algoritmos Genéticos como con Optimización por Colonia de Hormigas para validar las mejoras en calidad de la solución final y en la rapidez para alcanzar buenas soluciones. Finalmente, se aplica al problema de programación de la Línea de Galvanizado, para evaluar su capacidad de trasladar esas mejoras a este problema industrial bajo las condiciones que se dan en la práctica, incluida una exigente limitación del tiempo de ejecución debida a los requisitos operacionales del proceso.

RESUMEN (en Inglés)

Steel is pervasive in the modern world, an important part of many economic sectors. The Galvanized Steel market segment has the largest share among the different steel products, due to its wide application in varied industries such as construction, automotive, home appliances, and others.

Sequencing of the coils that will be produced is of utmost importance, as it will have a big impact on production costs and even feasibility.

This sequencing problem closely resembles a Travelling Salesperson Problem (TSP), with the additional complication of infeasible segments due to the constraints. For certain products, additional constraints apply, which completely invalidate any TSP-related approach. These constraints relate longer sub-sequences, rather than just one coil and its immediate neighbour.

The traditional approach to solving this problem is constraint programming based on constraint priorities painstakingly configured for each line to yield good results for its particular conditions and product mix.

The job scheduling problem and its many variations do not properly map to the scheduling of a galvanizing line. The same goes for efficient formulations as Integer Linear Programs (ILPs).

With exact optimization out of the question, metaheuristics are the usual approach under these circumstances. Population-based metaheuristics in particular are amenable to parallelization. Their common structure allows for the design of high-level methods that apply to all of them.

Recent trends in the evolution of computing power stress the importance of parallelization and distribution for improving the performance and scalability of algorithms.

The evolution of hardware leads towards parallelization of algorithms in order to extract the full performance of new systems and so tackle larger and more complex problems. This trend is already evident in the literature.

When tackling a large, complex optimization problem using metaheuristics one must consider how to best exploit the available hardware and how it fits with the properties of the problem at hand.

The benefits of enabling a more efficient use of the available resources must be balanced against the complexity involved, both in terms of building the algorithm and the communication required, which can in practice outweigh the gains. This balance can be interpreted as a multi-objective optimization problem. Existing algorithms can be placed on the Pareto front at different trade-offs of the metrics. While better solutions can be obtained by more complex and computationally expensive methods, there is an outstanding gap in the lower complexity region.

This thesis addresses this gap by proposing a method, named Multiverse, focusing on performance improvement at the baseline end of the Pareto front. The proposed method is not a concrete algorithm, but rather a whole family of them, resulting from the application of a transformation to any population-based metaheuristic.

The Multiverse method is tested on the TSP with both Genetic Algorithms and Ant Colony Optimization to validate the improvements in both solution quality and anytime performance. Finally, it is applied to the galvanizing line scheduling problem, to assess its capability to bring those improvements to this industrial problem under practical conditions, including stringent constraints on running time due to operational requirements of the process.

SR. PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO EN INGENIERÍA DE PRODUCCIÓN, MINERO-AMBIENTAL Y DE PROYECTOS

To those who supported me

You know who you are

Acknowledgements

I would like to thank my mother and my wife for putting up with my spending so much time on this work, their ongoing encouragement, and for (gently) pushing me to get it finished.

I also would like to recognize my advisors for their advice (as implied by the name) and guidance, and the department staff for their support in the administrative tasks.

I must also mention my colleagues at ArcelorMittal for their help in setting up and using the experimental environment, and for many insightful conversations.

Finally, I would like to thank Douglas Adams for the *Hitch-hiker's Guide to the Galaxy*, without which I could not have faced the University's bureaucracy.

Abstract

Steel is pervasive in the modern world, an important part of many economic sectors. The Galvanized Steel market segment has the largest market share among the different steel products, due to its high added value and its wide application in varied industries such as construction, automotive, home appliances, and others.

Sequencing of the coils that will be produced is of utmost importance, as it will have a big impact on production costs and even feasibility. These issues can be modelled as constraints and costs of an optimization problem.

This sequencing problem closely resembles a Travelling Salesperson Problem (TSP), with the additional complication of infeasible segments due to the constraints. This already makes it a different type of problem, as it is not guaranteed that for every problem instance a feasible solution exists. A way to bridge this gap is to consider constraints as a high-value cost that ensures that a tour that violates a single constraint in the tour is always considered worse than a feasible tour; beyond that, tours with fewer unmet constraints are naturally better. The values of the costs needed to represent the constraints in this way often result in numerical instability in the usual TSP solution methods.

For certain products, additional constraints related to quality apply, which completely invalidate any TSP-related approach. These constraints relate longer sub-sequences, rather than just one coil and its immediate neighbour. The result is a problem that adds complexity on top of the already NP-hard TSP problem, in a search space of size $n!$ for a sequence of n coils.

The traditional approach to solving this problem is, including in most commercial scheduling software, constraint programming based on constraint priorities painstakingly configured for each line to yield good results for its particular conditions and product mix.

The job scheduling problem and its many variations seem at first sight to be a good match, but do not properly map to the scheduling of a galvanizing line. The same goes for efficient formulations as Integer Linear Programs (ILPs).

With exact optimization out of the question, metaheuristics are the usual approach under these circumstances. Metaheuristics encompass a wide range of algorithms for optimization. Originally applied to heuristics or strategies that coordinate lower level search procedures, in time the concept extended to include methods that employ ways of escaping local optima in complex search spaces.

Population-based metaheuristics in particular are amenable to parallelization; at each step of the process, the same operations are carried out on several individuals of the population: all of them, a subset, or some pairwise combination, depending on the method. This common structure of population-based metaheuristics allows for the design of high level methods that apply to all of them.

Recent trends in the evolution of computing power stress the importance of parallelization and distribution for improving the performance and scalability of algorithms.

The evolution of hardware leads towards parallelization of algorithms in order to extract the full performance of new systems and so tackle larger and more complex problems. This trend is already evident in the literature, driven by a combination of stagnation of processor speeds and increased availability of hardware: consumer-grade hardware with multi-core CPUs, servers with dozens of threads, GPUs, high performance computing clusters, the cloud, etc.

When tackling a large, complex optimization problem using metaheuristics one must consider how to best exploit the available hardware and how it fits with the properties of the problem at hand.

GPUs are typically used for local, fine-grain parallelization: acceleration of function evaluation if the function is computationally intensive enough to actually gain from the overhead of using the GPU, or parallelization of solution evaluation at the population or neighbourhood level. The flip side to the potential acceleration, where applicable, is the need to develop the models at a very low level, managing data communication, memory usage, and the computations themselves. Due to the fine-grain approach, the methods are very algorithm- and/or problem-specific.

Multithreading is nowadays almost universally available, in that most computers sport multi-core processors[1]. From a practical perspective, most

[1]Multithreading as such can be used in a sequential processor, and it was used before the advent of multi-core processors; in that case, it does not provide any advantage in terms of computation for CPU-bound processes, in fact it is slower than a single-thread

if not all metaheuristic implementations can benefit from multithreading, especially population-based ones which are naturally parallelizable. Multithreading also affords the most flexibility, and can be applicable at all levels of the algorithm: from the evaluation of the fitness function to a coarse-grain, population-level parallelization. The main limit is imposed by the requirement to operate within a single machine, which restricts the amount of memory and computation power available.

The next step is multiprocessing, which is very similar to multithreading in its generality. It is slightly more complex to work with, as it requires isolation of the processes to a greater extent than multithreading requires it of threads; this entails some duplication of data and more communication. On the other hand, this very separation across processes avoids some of the pitfalls of concurrent programming, so to a certain extent it evens out. The main advantage of multiprocessing is that it allows to escape the limitation of a single machine, opening the opportunity of using clusters of workstations, high performance computing clusters, cloud environments, and other distributed systems. Because of the added overhead in data duplication and (potentially) network communication, this approach is usually reserved for very coarse-grain parallelization, normally at the population level.

Developments in metaheuristics have been exploiting these capabilities for decades, starting with very specific implementations addressing a single problem or method, but growing towards more general approaches. Islands models for Genetic Algorithms and Multicolony variants of Ant Colony Op-

approach because of the context switching between threads. Instead it was useful for I/O-bound processes, where the latency of *e.g.* accessing the network or a file on disk leaves idle cycles in the CPU.

timization are clear examples of this generality: they can be applied orthogonally to other problem-specific improvements such as the representation of individuals, or the definition of operators (crossover, mutation, solution construction).

Metaheuristics have followed this generalization trend to the point where several frameworks exist to build new algorithms piece-wise from individual blocks, mixing and matching from components of different methods. The original algorithms can be replicated in this fashion, but also new ones can be devised. These frameworks have also embraced the parallelization abilities, and typically support some level of multithreading and/or multiprocessing.

The benefits of enabling a more efficient use of the available resources must be balanced against the complexity involved, both in terms of building the algorithm and the communication required, which can in practice outweigh the gains. This balance can be interpreted as a multi-objective optimization problem. In this perspective, the efficient solutions are those lying on the Pareto front, also known as non-dominated solutions.

Existing algorithms can be placed on this Pareto front at different trade-offs of the metrics. While better solutions can be obtained by more complex and computationally expensive methods, there is an outstanding gap in the lower complexity region.

This thesis addresses this gap by proposing a method focusing on performance improvement at the baseline end of the Pareto front. The proposed method is not a concrete algorithm, but rather a whole family of them, resulting from the application of a transformation to any population-based metaheuristic: a meta-algorithm that takes an algorithm as an input and

produces a modified distributed algorithm as its output. The method interprets metaheuristics as a form of stochastic predictors of the optimal solution, making it akin to the use of ensemble methods.

The proposed method combines multiple instances of any population-based metaheuristic to improve its efficiency, while introducing as little additional overhead as possible. For easier reading, the method is referred to as the *Multiverse* method, as opposed to Multiple Independent Runs, which is called *Multistart*. The metaphor for the *Multiverse* being that each instance is a universe of its own, but within this method they all form a single entity, hence the Multiverse.

In the Multiverse method, one of the multiple instances has special status: the *collector*. It receives updates of the best solutions of the other instances. The application of this adaptation to the algorithms is straightforward, as all population-based metaheuristics already step through iterations (or generations), combine the solutions in their populations to create new, improved ones, and possess mechanisms to work with multiple solutions (the population). All other instances contribute their current best solution at each iteration to the population of the collector; the same process that takes place normally, applied to this extended population, is responsible for the mixing of solutions.

The added overhead is small: injection of external solutions into the population, and one-way communication of a single solution from each instance. Furthermore, this communication scheme fits a star-like topology, such as the one provided by a standard switch, rather than the more complex and costly mesh favoured by other configurations, such as the Islands model, and

more typical of super-computers than of clusters built from commodity-grade computers.

The *Multiverse* method is tested on the TSP with both Genetic Algorithms and Ant Colony Optimization to validate the improvements in both solution quality and anytime performance. Finally, it is applied to the galvanizing line scheduling problem, to assess its capability to bring those improvements to this industrial problem under practical conditions, including stringent constraints on running time due to operational requirements of the process.

Keywords: Optimization, Swarm Intelligence, Probabilistic, ACO, Ant Colony Optimization, Steel Industry, Galvanizing, Operational Research, Scheduling.

Resumen

El acero es ubicuo en el mundo moderno, y una parte importante de muchos sectores económicos. El segmento de mercado del Acero Galvanizado es el mayor entre los diferentes productos de acero, dado su alto valor añadido y su utilización en un amplio abanico de industrias, tales como la construcción, el automóvil, los electrodomésticos, y muchos otros.

La programación, o secuenciación, de las bobinas que se producen en una Línea de Galvanizado es extremadamente importante, puesto que tiene un gran impacto en los costes de producción e incluso en hacer la producción posible. Estas dificultades pueden modelarse como restricciones y costes de un problema de optimización.

Este problema de secuenciación es muy parecido a un Problema del Viajante (Travelling Salesperson Problem, TSP), con la complicación añadida de que algunos segmentos no son factibles debido a las restricciones. Esta diferencia lo transforma en una clase diferente de problema, al no haber garantías de que para todas las instancias exista alguna solución factible. Una manera de eludir este problema consiste en considerar estas restricciones como un coste de valor suficientemente alto como para asegurar que cualquier circuito que incluya al menos un segmento no factible resulte más costoso que

cualquier solución totalmente factible. A mayor número de segmentos no factibles, peor valor de la solución. Los valores de los costes necesarios para representar las restricciones de este modo a menudo provocan inestabilidad numérica en los métodos habituales de resolución de TSP.

Para ciertos productos existen restricciones adicionales relacionadas con la calidad; estas restricciones invalidan cualquier aproximación bassada en TSP, al depender de subsecuencias más largas que una bobina y su sucesora inmediata. El resultado es un problema que añade complejidad sobre el TSP, que de por sí es NP-hard, y con un espacio de búsqueda de tamaño $n!$ para secuencias de n bobinas.

Las soluciones típicas para este problema, incluso en programas de secuenciación comerciales, se basan en programación por restricciones que depende de prioridades de las restricciones diseñadas cuidadosamente para cada línea de modo que den buenos resultados para sus condiciones particulares y su combinación de productos.

El problema de programación de tareas (Job Scheduling Problem, JSP) y sus múltiples variantes pueden parecer a primera vista una buena opción, pero no encajan adecuadamente con la programación de una Línea de Galvanizado. La misma incapacidad para representar todas las condiciones necesarias afecta a las formulaciones eficientes como problema de programación lineal entera (Integer Linear Program, ILP).

Una vez descartada la posibilidad de métodos de optimización exactos, la aproximación habitual en estas circunstancias son los metaheurísticos. Los metaheurísticos engloban un amplio abanico de algoritmos de optimización. Aplicados al principio a heurísticos o estrategias que coordinan procedimien-

tos de búsqueda de bajo nivel, con el tiempo el concepto se ha extendido, incluyendo métodos que emplean diferentes maneras de escapar de mínimos locales en espacios de búsqueda complejos.

Los metaheurísticos basados en población, en particular, son fácilmente paralelizables; en cada paso del proceso se aplican las mismas operaciones sobre varios individuos de la población: todos, un subconjunto, o una combinación por pares, dependiendo del método. Esta estructura común de los metaheurísticos basados en población permite diseñar métodos de alto nivel aplicables a todos ellos.

La evolución de los equipos informáticos conduce hacia la paralelización de algoritmos para permitir extraer toda la capacidad computacional de los sistemas modernos, y así poder atacar problemas de mayor tamaño y complejidad. Esta tendencia ya resulta evidente en la literatura, impulsada por una combinación del estancamiento de la velocidad de los procesadores y la creciente disponibilidad de equipamiento: ordenadores personales con CPU multi-núcleo, servidores con docenas de hilos, tarjetas gráficas, clústeres de computación de alto rendimiento, la nube, etc.

Al atacar un problema de optimización complejo de gran escala utilizando metaheurísticos debe tenerse en cuenta cuál es la mejor manera de sacar el máximo rendimiento de los equipos disponibles y cómo encaja con las propiedades del problema en cuestión.

La computación con tarjetas gráficas se aplica normalmente para la paralelización de baja granularidad: acelerar la evaluación de la función objetivo o las restricciones, si tienen la complejidad computacional necesaria para compensar la carga adicional de usar la tarjeta gráfica, o paralelizar la eva-

luación de soluciones dentro de una población o una estructura de vecindad. La contrapartida a esta posible aceleración, en los casos en los que existe, es la necesidad de desarrollar los modelos a muy bajo nivel, gestionando las comunicaciones de datos, el uso de la memoria, y la programación de los propios cálculos. Debido a que es una aproximación de granularidad tan fina, los métodos desarrollados son muy específicos para cada algoritmo y problema.

La computación multi-hilo es prácticamente omnipresente hoy en día: casi todos los ordenadores tienen procesadores multi-núcleo. Desde un punto de vista práctico, casi todos —si no todos— los metaheurísticos pueden beneficiarse de esta forma de paralelización, es especial los basados en población, que admiten ser paralelizados de forma natural. La computación multi-hilo también resulta la forma de ejecución concurrente más flexible, y puede aplicarse a todos los niveles de los algoritmos: desde la evaluación de funciones objetivo hasta la paralelización a nivel de población. Su mayor limitación es estar restringida a operar en los confines de una única máquina, lo que pone una cota superior a la memoria y la capacidad de computación disponibles.

El siguiente paso es la computación multi-proceso, muy similar a la multi-hilo en cuanto a generalidad. Resulta ligeramente más complicado trabajar en multi-proceso, ya que los procesos deben estar más aislados unos de otros que los hilos. Esto provoca cierto nivel de duplicación de datos y de comunicación entre procesos. Por otro lado, esta misma separación entre procesos ayuda a prevenir algunos de los peligros de la programación concurrente, así que en cierta manera una cosa compensa la otra. La principal ventaja de la computación multi-proceso es que permite eliminar la limitación de utilizar

una única máquina, ofreciendo la oportunidad de utilizar clústeres de ordenadores (personales o de computación de alto rendimiento), entornos en la nube, y otros sistemas distribuidos. Debido a la necesidad de duplicación de datos y de comunicaciones, posiblemente a través de la red, esta variante se reserva para la paralelización a alto nivel, típicamente a nivel de población.

Los desarrollos en metaheurísticos llevan décadas explotando estas capacidades, empezando por implementaciones muy específicas dirigidas a un único problema o método, y avanzando en el tiempo hacia aproximaciones más generales. Los modelos de islas en algoritmos genéticos y los multi-colonia en optimización por colonia de hormigas son claros ejemplos de esta generalidad: su aplicación es ortogonal a otras mejoras específicas como pueden ser la representación de los individuos o la definición de los operadores (mutación, cruce, construcción de soluciones).

Los metaheurísticos han seguido esta tendencia hacia la generalizacion, hasta el punto de que existen varias plataformas para la creación de nuevos algoritmos combinando bloques básicos individuales tomados de componentes de diferentes métodos. Estas plataformas permiten reconstruir los algoritmos base originales, pero también crear otros nuevos. Muchas de ellas proporcionan cierto nivel de soporte para la paralelización multi-hilo o multi-proceso.

Las tendencias recientes en la evolución de la capacidad de computación ponen de relieve la importancia de la paralelización y distribución de los algoritmos para mejorar su eficiencia y escalabilidad.

Los beneficios de hacer más eficiente el uso de los recursos disponibles debe equilibrarse con la complejidad incurrida, tanto en la construcción del algoritmo como en la comunicación añadida, que en la práctica pueden perju-

dicar en mayor medida que los beneficios obtenidos. Este equilibrio se puede interpretar como un problema de optimización multi-objetivo. Desde este punto de vista, las soluciones eficientes son las que se encuentran en el frente de Pareto, o soluciones no dominadas.

Los algoritmos existentes pueden situarse en este frente de Pareto representando diferentes compromisos entre las métricas. Aunque se pueden obtener mejores soluciones con métodos más complejos y computacionalmente más exigentes, hay un claro hueco en la región de baja complejidad.

Esta tesis trata de cubrir este hueco, proponiendo un método centrado en mejorar el desempeño en la región de la línea de referencia del frente de Pareto. El método propuesto no es un algoritmo concreto, sino una familia de algoritmos resultante de la aplicacion de una transformacion a cualquier metaheurístico basado en población. El método interpreta los metaheurísticos como predictores estocásticos de la solución óptima, lo que lo asemeja a los métodos de *ensembles*.

El método propuesto combina múltiples instancias de cualquier metaheurístico basado en población para mejorar su eficiencia, introduciendo la mínima carga adicional posible. Para facilitar la lectura, en adelante este método se denomina *Multiverso*, en contraposición a múltiples instancias independientes, o *Multi-inicio*. La metáfora del *Multiverso* consiste en que cada instancia es un universo de por sí, pero en este método forman una entidad conjunta, o Multiverso.

En el método Multiverso, una de las instancias tiene un estatus especial: el *colector*. Recibe actualizaciones con las mejores soluciones de las demás instancias. Esto es sencillo de adaptar a los diferentes algoritmos, dado

que todos los metaheurísticos basados en población avanzan por iteraciones (o generaciones), en las que combinan las soluciones de su población para crear nuevas soluciones mejoradas y disponen de mecanismos para tratar con múltiples soluciones (la población). Las demás instancias contribuyen en cada iteración con su mejor solución, y el mismo mecanismo que se aplica normalmente en el algoritmo se encarga de integrar estas soluciones.

La carga adicional es reducida: la inyección de soluciones externas en la población, y la comunicación en un sentido de una única solución de cada instancia. Además, este esquema de comunicación encaja con una topología en estrella, como la que proporciona un switch estándar, en lugar de un formato de malla, más complejo y caro, típico de superordenadores, y que se usa por ejemplo en modelos de islas.

El método *Multiverso* se prueba sobre el TSP, tanto con Algoritmos Genéticos como con Optimización por Colonia de Hormigas para validar las mejoras en calidad de la solución final y en la rapidez para alcanzar buenas soluciones. Finalmente, se aplica al problema de programación de la Línea de Galvanizado, para evaluar su capacidad de trasladar esas mejoras a este problema industrial bajo las condiciones que se dan en la práctica, incluida una exigente limitación del tiempo de ejecución debida a los requisitos operacionales del proceso.

Palabras clave: Optimización, Inteligencia de enjambre, Probabilístico, ACO, Optimización por Colonia de Hormigas, Industria Siderúrgica, Galvanizado, Investigación Operativa, Secuenciación.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Figures

1.1 The Steel Industry

Steel is pervasive in the modern world, an important part of many economic sectors. If you look around you right now you are bound to find multiple objects that are made of or include steel in some form.

It is used in building construction in the form of beams, pillars, and rebar; but also in the form of panels, plates, and even as an aesthetic facade element. In transportation, it is the material of choice for infrastructure such as rails, guardrails, signposts, bridges, or pipes, not to mention its prominent use in building cars, trucks, trains, ships, and other vehicles and equipment. About half of all beverage cans produced in the world are made of steel, as are most of the food packaging cans. It is employed also in household and office items and appliances of every size and shape from paper clips and cutlery to furniture and refrigerators. Tooling, starting at hammers and screwdrivers and extending to industrial machinery and robots, the military, high-tension long-range electrical lines, containers, or wind turbines are some additional examples of fields of application that rely on steel.

To cover so many applications, steel comes in a broad variety of formats, each with specific properties to fit the job. A car provides a representative example of this micro-cosmos of steel types. The chassis requires tough, rigid steel that must withstand extreme stress without buckling, while the structural elements that protect the cabin in case of impact must deform predictably, absorbing a large amount of energy. Body parts must be extraordinarily malleable to be stamped into creative shapes with thickness in the order of tenths of a millimetre and be corrosion resistant for decades.

The steels used in engine blocks must suffer through mechanical and thermal fatigue without cracking, and the materials for the pistons and cylinders need high machinability and have very stringent tolerances on wear and thermal deformation. With the advent of electric vehicles, new steels with specific requirements in thermal and electrical properties have been developed for battery casings and motors. Inside the cabin, chromed steel requires strict surface quality and a very even coating. The wires embedded in the rubber, which provide structural stability to the tyres, combine high strength and enough ductility to be produced by extrusion.

The catalogue of steel types is still wider to account for all the different applications. By dipping its toes in all these sectors, steel is deeply intertwined with the global economy in general. Even with the recent dip in economic activity due to the global pandemic, steel production has stayed at the same level, with the worldwide production of crude steel in 2020 at 1,878 million metric tons, more than doubling in the last 20 years[1], in part driven by the rapid growth of both demand and production in China since the year 2000. Equivalently, this corresponds to an apparent per capita use of 227.5 kg of finished steel products globally.

1.2 Steel Production Process

Figure 1.1 summarizes the typical steel production processes currently used. In some specific applications, though, ingot casting is used instead of continuous casting.

[1]The source for this and further figures in this chapter is the World Steel Association's

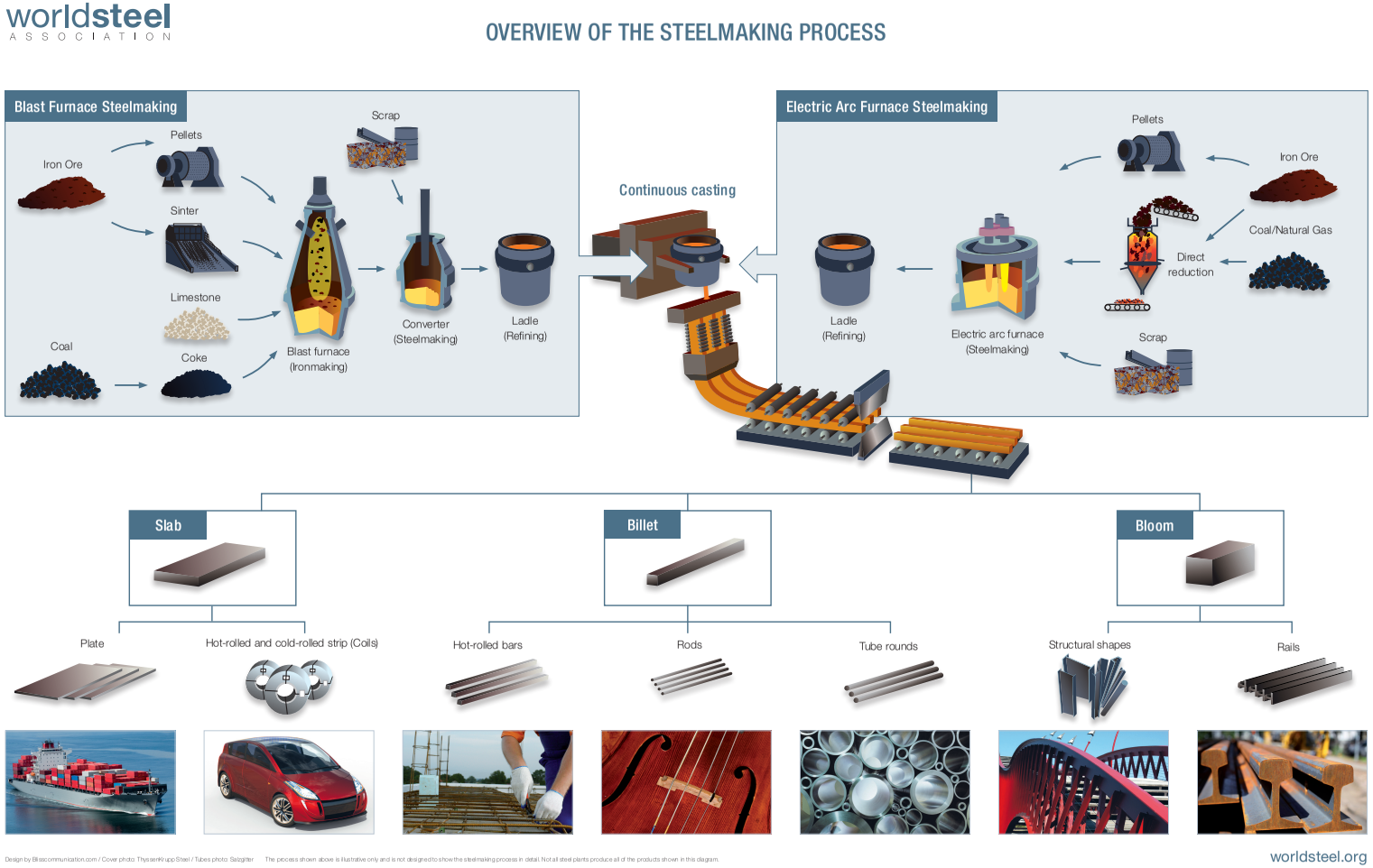

Figure 1.1: Overview of the Steelmaking Process. Source: World Steel Association.

There are two main routes: the Blast Furnace (BF) and the Electric Arc Furnace (EAF). The BF route takes iron ore, consisting mostly of iron oxides, as raw material, while the EAF route takes scrap or pre-reduced iron instead. There is some overlap, as the BF route can and does use some scrap, and the EAF route can admit some amount of hot metal from a Blast Furnace if one is available, but this configuration is exceedingly rare.

The BF route starts with iron ore, which is directly mined from the ground. at the mine, the ore is ground and potentially concentrated by separation to improve the iron content. This form of the ore is called *concentrate*, and must undergo further processes for use in the BF to prevent clogging.

One of these processes is pelletization. This is typically performed at the mine. The concentrate is further ground to a more uniform granulometry, shaped into small spheres around 1–2cm in diameter, and baked in an induration furnace to achieve sinterization. This results in *pellets*, solid but porous spheres of iron ore.

The other process is sinterization, and is usually a part of the ironmaking stage of a steel plant. Concentrate, mixed to the required quality and including some amount of pulverized coal to serve as fuel, is passed through a furnace triggering a sinterization process resulting in a porous cake usable in the Blast Furnace, called *sinter*.

The other main input to the Blast Furnace is coke, which provides carbon to serve as fuel and also structural properties required of the load of the furnace. For this reason, not all coals can be used to make coke. Coking coal (coal of the required quality) is heated in a closed oven in absence of oxygen

report *2021 World Steel in Figures*, which can be accessed here.

for hours. Without oxygen combustion does not take place, but volatiles present in the coal escape as they expand due to the heat, leaving a porous material of almost pure carbon.

The Blast Furnace itself is in essence a chemical reactor where the iron oxides are reduced by carbon, resulting in *hot metal* and BF gas. Hot metal is an alloy of iron an carbon (with around 4.5% carbon, which is too high for steel, and is called cast iron or pig iron), while the BF gas is a mixture of CO and CO_2.

The BF works in a continuous fashion. It is fed alternate layers of iron ore (pellets and/or sinter) and coke. As they descend, they meet an opposite stream of gas due to the blowing of hot air from the bottom of the furnace, usually with some added fuel such as pulverized coal, to regulate the temperature. The ascending gas first generates intense heat to facilitate the reaction of the carbon in the coke with the iron oxides, forming CO and elemental iron, which drips down to the *hearth* below the furnace. Further up the air is not so hot, but it preheats the input materials, and part of the CO still reduces the iron oxides becoming CO_2.

Some additional materials, fluxes, included with the ores ensure that the conditions in the furnace are adequate for the chemical reactions, and fall on top of the liquid iron in the hearth forming a protective layer of slag that prevents reoxidation.

The hearth is periodically tapped by drilling holes to pour the liquid hot metal into refractory-lined vessels that will take it to the steelshop. The holes are plugged between tappings to allow the hot metal to accumulate. Optionally, the hot metal is desulphurized before moving on. Specific additives are

mixed into the hot metal to form a sulphur-affine slag that traps a significant part of the sulphur, and is removed from the surface. Sulphur is an undesirable element in steel, as it forms inclusions that distort the crystallographic structure. Sulphur and other elements that have a similar effect are called tramp elements. While they all need to be controlled within certain limits to ensure the quality of the final product, sulphur in particular cannot be removed under the conditions in subsequent steps due to the chemical characteristics of the slags needed in the steelmaking processes, so this previous step is needed if the sulphur content in the hot metal is excessive.

After this point, we move from ironmaking to steelmaking. The hot metal needs to have most of its carbon removed to be in the right range for steel. This is accomplished in the Basic Oxygen Furnace (BOF). In the BOF, oxygen is blown through the liquid hot metal to remove carbon and obtain liquid steel. As in the BF, fluxes are added to generate a slag that protects the liquid metal and provides the right conditions: the reaction in the BF by which carbon took oxygen from the iron is to an extent reversed here and oxygen is used to remove carbon from the iron. Additionally, other alloys are added at this stage to adjust the chemical composition of the steel, and argon or nitrogen complement the oxygen blowing for a strong stream that mixes the liquid metal.

After the BOF, and still in liquid phase, the steel may undergo one or several processes, known as secondary metallurgy, to adjust it to the requirements of the specific steel product:

Ladle Furnace The liquid steel is reheated and receives more additives to refine its chemistry.

Vacuum Degasser The liquid steel is subjected to a vacuum to remove gas components, especially hydrogen or carbon.

The liquid steel is then solidified in the continuous caster. The first stage is the *tundish*, a vessel that functions as a buffer to allow continuous operation: the ladles pour each batch of liquid steel into the tundish and it, in turn feeds the caster *mould*; the next batch will arrive before the tundish is completely empty, concatenating anywhere from three to about a dozen batches before the tundish needs to be replaced.

The mould defines a rectangular section; liquid steel is poured from the top and comes solid on the surface from below, but the inside will require additional cooling to fully solidify. This allows the 'bar' of steel to bend into a horizontal position while it is cooled with water sprays. Once it is fully solidified, it is cut to the desired length. The cross-section varies depending on the final product:

Slab A rectangular cross-section aimed at producing flat products. Width is usually 400–2,500mm and thickness 75–600mm.

Billet A square or close-to-square cross-section aimed at producing long products like beams, rails, rods, wires, etc. Typical sizes are 80–200mm to a side.

The EAF route is more compact, and characteristic of the so-called *mini-mills*. The EAF takes in mainly scrap, but also possibly pre-reduced iron or hot metal. Pre-reduced iron consists of pellets or brickets of ore that has undergone a reduction process in the solid state traditionally treating it with CO from natural gas to remove the oxygen from the iron oxides.

The scrap and other materials are loaded into the EAF, and a strong electrical current is passed through it by means of graphite electrodes to melt it into liquid steel. Oxygen blowing, addition of ferroalloys, and oxygen blowing may be applied as described for the BOF above. The liquid steel then moves on to secondary metallurgy.

The main drawback of this route is that the quality of the steel will be very dependent on the cleanliness of the input materials. Scraps come from many origins, and some of the potential contaminants such as copper cannot be removed. These contaminants are called residuals.

These initial stages of ironmaking and steelmaking are the most carbon intensive in the production of steel. In recent years, the industry is moving towards improving its carbon footprint and efforts naturally concentrate on this area. The use of hydrogen as a reducing agent in the BF and in the production of pre-reduced iron is a clear trend in this direction, while research on electrolysis to fully substitute the BF shows a potential avenue for future evolution.

After iron- and steelmaking, slabs and billets are sometimes sold directly, but mainly to other steel manufacturers for further processing. Downstream of this point, the processes are different depending on the desired product.

Heavy plate, used in building ships, pressurized containers, or windmills, is hot rolled into its final shape: the slab is reheated to a red- or white-hot temperature that makes the material malleable and cylinders apply pressure from above and below to compress the steel in thickness while extending it in width and length. The rolling process not only shapes the product, but combined with the thermal evolution, determined by the heating and con-

trolled cooling by water spray nozzles, helps to achieve the final mechanical properties.

Long products have several specialized routes. Each starts with a hot rolling stage to adapt the bloom into a shape more amenable for the subsequent stages. Sections (e.g. beams) and rails are rolled into square section bars, before being cold rolled with especially shaped rolls and cut to length. Rebar and wire start with hot rolling the bloom into circular section bars, which are cold rolled and cut in the case of rebar, or extruded into shape in the case of wire.

Flat products also start with a hot rolling stage, using several stands of cylinders that press the slab into shape in multiple forward and back passes and apply controlled cooling at the exit. After this, the slab has been transformed into a relatively thin (up to a few centimetres) strip of a similar width to that of the slab, but much longer, which is wound into a coil for storage.

According to the final application the flat product may end its route at this step or after any of the following stages. Downstream of the hot rolling mill, the finishing lines all work in a continuous process that takes one coil after another; both coils are welded together head to tail at the entrance and cut at the exit.

Accumulators are buffers at the beginning and end of each line that enable decoupling the speed in different parts of the line in order to stop the strips for welding and cutting while the line as a whole can keep functioning. They are comprised of several rolls that guide the strip into a zig-zag pattern like that of queues at airports or events; the rolls on one side are movable, changing

the length of material in the accumulator. The entry accumulator gathers material while a coil is being processed, so that when the end of the coil is reached it can give it back to feed the line while it is welded to the next coil. Similarly, the exit accumulator takes material in during cutting and gives it back for coiling.

The next stage is pickling, where the strip is submerged in a bath of sulphuric or hydrochloric acid to clean it of potential rust that may form on hot-rolled coil if unprotected. This may be done to package it as a final product or as preparation for cold rolling.

In cold rolling, also known as the tandem mill, the strip is pressed between cylinders to widths as small as a tenth of a millimetre, without preheating. This changes not only the shape but also the mechanical properties of the steel. The high pressure hardens and strengthens the steel as the microstructural grains break into smaller bits and dislocations are stretched. This also results in a more brittle material.

In a continuous annealing line the strip is heated in a furnace to reach and keep a temperature that, while not red-hot, allows internal diffusion, which regrows the microstructural grains and releases tensions in the material. The cooling process after the furnace is controlled to yield different properties as needed: slower cooling retains softer, more malleable material (ferritic or pearlitic steel); while faster cooling rates result in different steel phases (martensitic or bainitic steel) that are harder and tougher.

In a skinpass line special rolls with imprinted surface roughness transfer the pattern onto the strip to achieve certain surface aspects. This is normally integrated with a tension leveller which pulls the strip to a point just beyond

its elastic limit to cause a small plastic deformation; this overcomes the initial stage of plastic deformation which can be non-smooth, and ensures good formability of the material for further processing, such as extrusion or stamping.

Several lines apply different coatings to the steel to provide protection and/or aesthetic properties.

A tin-plate line applies a tin coating to isolate the steel from food or beverages in cans.

A galvanizing line applies a zinc (or a mix of zinc and other metals) layer to provide corrosion resistance; the galvanizing line is described below in more detail, as it will be used for testing the algorithms.

An organic coating line applies a layer of organic pigment to yield a painted coil (on one or both sides).

Some of these coatings may be applied to the same product, for instance a painted galvanized coil, but not all; tin-plate is not galvanized, for example.

Some speciality products, such as military-grade plate, have custom processes, often starting from ingots rather than slabs, as the smaller volumes do not call for continuous processes.

1.3 Galvanizing Line

The Galvanized Steel market segment has the largest market share among the different steel products, due to its high added value and its wide application in varied industries such as construction, automotive, home appliances, and others. In 2020 this corresponds to a market value of over 162.8 billion dollars

Figure 1.2: Galvanized steel coils in a coil yard

—for more than 150 million metric tons of production—, with a projection to grow beyond 220 billion dollars by 2027[2].

This means that in a year the number of galvanized coils that are produced and scheduled is in the order of 10 million worldwide, spread across hundreds of galvanizing lines belonging to many different steel manufacturers. ArcelorMittal alone has dozens of these lines all over the world.

In the galvanizing line the coils are coated with a zinc (or zinc alloy with aluminium and/or silicon) layer as a means to prevent corrosion from ambient moisture. This is accompanied by an annealing phase which helps to achieve specific mechanical properties and takes the steel to the right temperature

[2]Source: *Global Galvanized Steel Market Report, History and Forecast 2016–2020, Breakdown Data by Manufacturers, Key Regions, Types and Application* by 360 Research Reports (a summary is publicly accessible here). The production volume is inferred from the market value and current prices. The World Steel Association provides the *Steel Statistical Yearbook 2018* (later editions are only freely accessible for summary information); there, the latest reliable data sets a lower bound on production of over 121 million metric tons for 2015; it lacks information from several key markets in 2016 and onwards, especially China which in previous years accounted for around 40% of the global production (see `https://www.worldsteel.org/en/dam/jcr:e5a8eda5-4b46-4892-856b-00908b5ab492/SSY_2018.pdf`).

for the molten zinc to meld onto it. Depending on the line and product, a skinpass and tension leveller stage may also be included for surface aspect and formability.

Like other finishing processes, the galvanizing line starts and ends with accumulators to allow a continuous process by welding consecutive coils together at the entrance and cutting them at the exit.

Next comes the annealing furnace. While in a hot rolling mill the whole slab is placed in the reheating furnace and kept there for a time until it reaches the right temperature and it is extracted to be rolled, in a continuous line the strip keeps moving within the furnace, a tunnel lined with radiant tubes. The line speed must adapt to ensure the right dwelling time and furnace temperature combination for the steel to follow the prescribed thermal profile. This is easy in steady state, but at the transitions between coils some length of a given coil and its successor will be inside the furnace; furthermore, both line speed and furnace temperature, especially the latter, are limited in the rate of change, so matching successive coils in the sequence is important to ensure smooth thermal cycle transitions.

The next stage is the molten zinc bath (for hot dip galvanizing; electrogalvanizig lines use an electric current to cause electrodeposition of the zinc). The strip passes through the zinc pot and a system of 'air knives' to spread the zinc layer evenly and to specification. As in the furnace, the transition between coils takes some time, which will be longer or shorter depending on how different the coating thickness is between coils; in this period a coil may be out of specification or overcoated; the latter is not usually grounds for a claim but is a waste of material and an added cost.

Finally, the strip reaches the skinpass and tension leveller stage if it exists, with similar considerations for the transitions with respect to the force applied in the skinpass cylinders and the tension used in the leveller.

All of these difficulties associated with the transitions mean that sequencing of the coils that will be produced is of utmost importance, as it will have a big impact on production costs and even feasibility. These issues can be modelled as constraints (e.g. maximum dimension difference between consecutive coils for a safe welding) and costs (e.g. value of the loss of material out of specification during a thermal cycle transition).

1.4 The Galvanizing Line Scheduling Problem

Taken like this, the sequencing problem closely resembles a Travelling Salesperson Problem (TSP), with the additional complication of infeasible segments due to the constraints. This already makes it a different type of problem, as it is not guaranteed that for every problem instance a feasible solution exists. A way to bridge this gap is to consider constraints as a high-value cost that ensures that a tour that violates a single constraint in the tour is always considered worse than a feasible tour; beyond that, tours with fewer unmet constraints are naturally better. This may be further complicated by having two different levels for the constraints (soft and hard constraints). In any case, the values of the costs needed to represent the constraints in this way often result in numerical instability in the usual TSP solution methods.

The high-value cost formulation also matches reality. When a feasible sequence cannot be found, the gaps are filled using coils from inventory that will be degraded and potentially scrapped later to make the production keep running, although with the high cost of the lost material. These are called warmer or stringer coils.

For certain products, additional constraints related to quality apply, which completely invalidate any TSP-related approach. These constraints relate longer sub-sequences, rather than just one coil and its immediate neighbour. One such constraint is, for example, that no more than a given length of strip should be rolled without a change in width. Doing so can wear an edge on the rolls at the rim of the strip, which can subsequently cause marks on the strip. Not only does this involve several coils, but the exact number depends on the sequence itself, and the length of the individual coils.

The result is a problem that adds complexity on top of the already NP-hard TSP problem, in a search space of size $n!$ for a sequence of n coils.

The traditional approach to solving this problem is, including in most commercial scheduling software, constraint programming based on constraint priorities painstakingly configured for each line to yield good results for its particular conditions and product mix.

The job scheduling problem and its many variations (multiple machines, time windows, set-up times, etc.) seem at first sight to be a good match. It consists of a number of jobs that need to be scheduled in one or more machines; the jobs have due dates or ranges of valid completion dates and the time they take to process in a machine; constraints define which machines can perform which jobs, precedence relationships between jobs, and other

complications. The solution methods typically aim at minimizing a metric such as tardiness (the delay of jobs with respect to their due dates) or machine idle time.

This problem, however, does not properly map to the scheduling of a galvanizing line. The functions that they minimize cannot represent the costs associated to subsequences longer than a single transition, and they do not consider the potential for infeasible sequences. The use of these methods is therefore discarded. Even if a heuristic-based approach might be attempted, it would likely require rebuilding the heuristic for each line, due to the different conditions. A more optimization-oriented approach should aim at minimizing constraint violation and costs, which would provide a more generic solution that can be adapted to each line just by modelling the cost functions and constraints. Such a solution has the added benefit of being easily extensible to other production stages.

A relatively direct formulation of this optimization approach involves Integer Linear Programs (ILPs), using binary variables representing the allocation of an item to a position in the sequence. Subsequence constraints and costs can be modelled in this framework as long as the length of the subsequence in terms of number of items is known beforehand; otherwise, conditional elements are needed which makes the problem non-linear. Even the linear version suffers from very poor scalability due to the combinatorial nature of the problem and the corresponding exponential complexity of the search space ($O(2^{n^2})$ for n items), and the non-linearity only makes it worse. With state-of-the-art solvers, a few tens of items are enough to require impractical running times.

More advanced ILP formulations use variables that represent succession in the sequence; although additional constraints are needed to avoid loops, this is offset by the internal structure of the problem which can be exploited by the branch-and-bound heuristics in the solvers. Combining this and decomposition strategies, it is possible to address somewhat larger problems, requiring additional work to design a decomposition strategy that works for the problem at hand. However, this formulation lacks the ability to express the subsequence costs and constraints, so it cannot be applied, even if practical sizes could be handled for the particular problem.

With exact optimization out of the question, metaheuristics are the usual approach under these circumstances. Metaheuristics, if properly used, can usually find good-enough solutions in reasonable times, and can handle non-linear and non-analytic cost functions and constraints, providing additional flexibility when modelling the problem.

As an example, this galvanizing line scheduling problem has been addressed by [Fer+14] using Ant Colony Optimization, and will be one of the test-beds for the algorithms presented in this work.

Despite their advantages, even metaheuristics are limited, and as the problem becomes larger (*e.g.* by considering a longer scheduling horizon), they start to suffer and solution quality drops within the stringent time constraints imposed in the industrial environment. One way to counter this to some extent is to increase the computational capacity through parallelization.

Therefore, this work looks at the application of parallel metaheuristics to tackle the galvanizing line scheduling problem, so it can be extended to address larger, more complex problems, and still provide answers in a timely

fashion.

1.5 Metaheuristics

Metaheuristics encompass a wide range of algorithms for optimization. Originally applied to heuristics or strategies that coordinate lower level search procedures, in time the concept extended to include methods that employ ways of escaping local optima in complex search spaces, typically several neighbourhood strategies or some extent of randomness. While these algorithms cannot compete with strict optimization methods where they are applicable, such as solving linear programs, they excel at attaining good enough solutions for problems that do not match the expression capabilities of strict methods or that span a large search space that cannot be exhaustively searched by exact methods [GP10].

Given their diversity, there are many different classifications of metaheuristics, according to different features. The classification most relevant to this work divides them into two classes:

1. Population-based, which at any time hold a number of solutions (the population) and create new ones by combining or modifying them; some examples of this class are Genetic Algorithms, Ant Colony Optimization, and Artificial Bee Colony.

2. Single-solution, which focus on iteratively improving just one solution; some examples of this class are Variable Neighbourhood Search, Simulated Annealing, and Iterated Local Search.

Population-based metaheuristics are more amenable to parallelization; at each step of the process, the same operations are carried out on several individuals of the population: all of them, a subset, or some pair-wise combination, depending on the method. Single-solution metaheuristics, on the other hand, are by nature sequential. This does not mean that parallelization is not possible, but it is tied to the particular algorithm, such as parallelizing the calculation or evaluation of neighbours for Local Search variants, or even the particular problem, such as exploiting the properties of the fitness function. This second parallelization approach can also benefit population-based metaheuristics, and it is currently a vibrant field of research; but the common structure of population-based metaheuristics allows for the design of high level methods that apply to all of them.

1.6 Parallel and Distributed Computation

Recent trends in the evolution of computing power stress the importance of parallelization and distribution for improving the performance and scalability of algorithms [Sut05]. Moore's Law states that the number of transistors that can be fitted into a chip roughly doubles every 18 months. For a while, this translated directly into raw power increase, and the same programs became faster just by running on newer hardware.

More recently, as the development hit certain physical limits, increases in clock speed stalled and timing and power constraints made it necessary to separate the processor into several cores in order to extract all its potential. Single-thread programs no longer reap the rewards of increased computing

power, as they only make use of a fraction of the processor.

To be able to take advantage of the new advances, programs need to parallelize operations. There are different ways to attain this, and they can be included in different layers of the design.

The paradigm for GPU acceleration follows SIMD (Single Instruction Multiple Data): the same sequence of operations are executed on many data instances. Its main advantages come from vector- or matrix-like calculations with very large numbers of items; actual GPU speeds are usually quite a bit slower than CPU speeds, but they make up for it by performing hundreds or thousands of simultaneous operations. This form of parallelization requires programming at very low level and taking care of interleaving the computations with data transfers to and from the GPU. Because of this, the use of GPU is very much problem-dependent and used in the innermost loops, such as fitness function evaluation.

When using the CPU, a distinction is needed between parallelization and distribution. The former merely implies that several operations can take place at the same time, whereas the latter also considers that the separate execution threads do not share the same context, but need to explicitly communicate with each other. In both cases, the instructions are independent for each instance, as opposed to the case of the GPU.

Parallelization would then correspond to a multi-threading approach, where several threads are run at the same time, while sharing the same memory space; likewise, distribution would correspond to separate processes running in the same or even different CPUs, and all coordination is carried out by means of inter-process communication (IPC) mechanisms, such as

message passing, mutexes, etc.

Multi-threading is therefore useful for exploiting multiple cores in the same CPU, while distribution can take advantage of multiple CPUs, in the same or different (networked) computers.

1.7 Parallel Metaheuristics

In the context of population-based metaheuristics, multi-threading is typically used to run the operations on the individuals of the population at the same time, rather than one after another; and distribution is typically used to run several populations across a cluster, either independently or with some level of communication, usually in an *Islands* model, where each population transfers good solutions to its neighbours. Here, neighbours in the sense of the network topology; the transference only takes place between nodes that are directly connected to avoid high communication overhead. The most common topology is a 2-dimensional mesh, where each node is connected to four other nodes.

The benefits of enabling a more efficient use of the available resources must be balanced against the complexity involved, both in terms of building the algorithm (coding and maintaining) and the communication required, which can in practice outweigh the gains. This balance can be interpreted as a multi-objective optimization problem. In this perspective, the efficient solutions are those lying on the Pareto front, also known as non-dominated solutions. A solution dominates another if it is at least as good in all objectives, and strictly better in at least one of them. Evidently, dominated

solutions are not efficient, as the solutions that dominate them are clearly better. However, comparing non-dominated solutions is not as easy: one will be better in some aspects and the other one in others, hence the Pareto front. For a specific application, a solution from the Pareto front may be chosen by selecting a trade-off of the objectives appropriate to the task.

For the assessment of parallel population-based metaheuristics, three metrics are relevant: complexity, running time (given the available resources), and solution quality. Complexity refers to the added difficulty of building the algorithm; running time corresponds to how long it takes to reach a solution of a given quality, while solution quality reflects how good a solution is reached in a given running time. Complexity is dealt with in a qualitative way, as defining a proper metric would be overly complex and to understand its influence on the Pareto front only a ranking is needed. The metric for solution quality will be the value of the objective function attained in optimization with a given time budget[3], while the metric for running time will be the hypervolume of the objective function attained versus iteration curve, a measure of anytime performance explained in more detail in the discussion on metrics in chapter 2 and when presenting the validation results in chapter 4.

The algorithms and approaches described above can be placed on this Pareto front at different trade-offs of the metrics. While better solutions can be obtained by more complex and computationally expensive methods, and there is vibrant research in that region of the front, there is an outstanding

[3]Actually, it will be a budget of function evaluations, but coupled with an analysis of its impact on running time.

gap in the lower complexity region. Most parallelization strategies require additional effort in establishing communication and coordination across the multiple threads of execution, whereas the usual baseline consisting of Multiple Independent Runs of a sequential algorithm is quite straightforward to implement.

This thesis addresses this gap by proposing a method focusing on performance improvement at the baseline end of the Pareto front, adding as little extra complexity and communication as possible, in a position between simple multiple parallel runs and fully connected systems. The proposed method is not a concrete algorithm, but rather a whole family of them, resulting from the application of a transformation to any population-based metaheuristic. The method interprets metaheuristics as a form of stochastic predictors of the optimal solution. They take a specific instance and calculate, through a combination of deterministic and random operations, a value that tries to approximate the solution to the problem.

Thus, running several instances of a metaheuristic is equivalent to using a group of stochastic predictors. Even if in the case of metaheuristics it is possible to know which of the instances yields the best solution by comparing their fitness, it is possible to draw some ideas from the statistician's toolbox to extract more value from running several instances.

The group of methods, employed in statistics and Machine Learning, collectively known as ensemble methods or ensemble learning deal with exactly this scenario. All of them combine multiple models or predictors to obtain a better result than possible with any one of the models individually [MO99].

The best known examples of ensemble methods are Bootstrap Aggregat-

ing (Bagging), which trains several models of the same type with random samples of the original data and then generates its prediction by aggregating the results of all models with equal weights (averaging for continuous variables, selecting the most voted class for categorical ones), and Boosting, which trains models successively so that each one places special emphasis on the instances misclassified by the previous one; but many others exist: Bayesian Parameter Averaging, Bucket of Models, Stacking, etc.

The overarching theme in all ensemble methods is that improvement can be obtained from the aggregation of multiple (weak) models to build a better one.

The same concept is present in the very procedure of many metaheuristics: from the partial reuse of solutions in Genetic Algorithms, to the preference for frequently travelled edges in Ant Colony Optimization, to name a few.

Path Relinking showcases this behaviour better than any other metaheuristic: its concept itself is exactly to combine good solutions to find better ones. Starting from a pool of solutions (generated randomly and/or using some heuristic), it selects the best ones and builds paths between pairs of them. Each step consists of a modification of the current point as in a local search, but always moving towards the other solution (guiding solution). As new solutions are generated, they may become new path sources or guiding solutions.

1.8 The Multiverse Method

Taking all of this into account, this thesis proposes a method that combines multiple instances of any population-based metaheuristic to improve its efficiency, while introducing as little additional overhead as possible. For easier reading, the method is referred to as the *Multiverse* method in the remainder of the thesis, as opposed to Multiple Independent Runs, which is called *Multistart*. The metaphor for the *Multiverse* being that each instance is a universe of its own, but within this method they all form a single entity, hence the Multiverse.

In the Multiverse method, one of the multiple instances has special status: the *collector*. It receives updates of the best solutions of the other instances. The application of this adaptation to the algorithms is straightforward, as all population-based metaheuristics already step through iterations (or generations), combine the solutions in their populations to create new, improved ones, and possess mechanisms to work with multiple solutions (the population). All other instances contribute their current best solution at each iteration to the population of the collector; the same process that takes place normally, applied to this extended population, is responsible for the mixing of solutions.

The added overhead is small: injection of external solutions into the population, and one-way communication of a single solution from each instance. Furthermore, this communication scheme fits a star-like topology, such as the one provided by a standard switch, rather than the more complex and costly mesh favoured by other configurations, such as the Islands model, and

more typical of super-computers than of clusters built from commodity-grade computers, usually dubbed clusters of workstations (COWs).

This *Multiverse* method, by extracting more performance from available hardware, is one way to tackle larger and more complex versions of the galvanizing line scheduling problem, which is the objective set at the start. Additionally, the approach is applicable in a more general fashion: any problem tackled through a population-based metaheuristic can benefit from enhancing the solution algorithm in this way.

Chapter 2

State of the Art

Contents

2.1 Metaheuristic Algorithms

Since the target of this work is population-based metaheuristics, the review focuses on Genetic Algorithms and Ant Colony Optimization as representative examples of the two main subclasses: combining and constructive metaheuristics, respectively.

2.1.1 Genetic Algorithms

John Holland introduced the concept of Genetic Algorithms (GAs) in [Hol75]. The field has since expanded well beyond the original idea, into a more general class of Evolutionary Algorithms that share the underlying idea of simulating neo-Darwinian evolution and natural selection through mutation and recombination of individuals.

In its initial form, GA encoded individuals (candidate solutions) as binary vectors, and used a function $f : \{0,1\}^n \mapsto \mathbb{R}$ mapping the individual to the corresponding fitness value[1]. Starting from a random population, it applied a selection method to determine which solutions in the population produce new individuals; this selection reinforces the fitness function by sampling with higher probability individuals with better fitness. Selected individuals go on to produce new offspring through recombination and mutation. In recombination, the features of the two parent solutions are combined by randomly setting the value at each position to the value of one of the parents. Mutation randomly flips a bit with a probability inversely proportional to

[1]Strictly speaking, the binary vector is the encoding of the solution, or *genotype*, and the actual solution variables, or *phenotype*, are the decoded version; the fitness function maps the *phenotype* onto the fitness.

the length of the solution.

A comprehensive review of the history and application of Genetic Algorithms is given in [Ree10] and the updated version in [Whi19]. The most notable milestones are described below.

The selection method described above, known as roulette-wheel selection, can be visualized as a roulette spinner where the angle of a sector corresponding to an individual is proportional to its fitness. It was extended by Baker's *stochastic universal selector* (SUS), which selects multiple parents with each 'spin' by considering multiple, evenly-spaced spinner arms. This speeds up the selection by avoiding the rescaling needed to perform sampling without replacement. Different scaling variations have been developed, as well as a ranking version which takes the individuals in order of fitness and assigns a predefined probability based on rank, typically an arithmetic progression. Another alternative that forgoes scaling is tournament selection, where to select one parent two individuals are sampled uniformly and the one with better fitness kept.

The original crossover operator is known as uniform crossover (UX). The main additional operators are one- and two-point crossover (1X and 2X, respectively), although the general m-point crossover has been used with greater values of m. In 1X, a single split point is chosen uniformly in the individual, and the genes after that point are exchanged. In 2X, 2 split points are selected and the genes between the two points are exchanged. Further crossover operators have been defined for other encodings, such as permutations (widely used in TSP and scheduling problems), for which the usual ones would yield invalid offspring. Two alternative operators for permuta-

tions are PMX (partially mapped crossover) and a reinterpretation of the usual crossover. In PMX, a selection of genes is made as for UX, but instead of exchanging the genes between parents, the selected value in one parent is swapped in that parent with the value that appears in that position in the other parent. The reinterpretation of the usual crossover keeps the values in the selected genes fixed for one parent, and then fills in the gaps with the rest of the values in the order in which they appear in the other parent.

Originally, the new population of the next generation was composed entirely of the offspring resulting from crossover and mutation. Certain population overlaps may be kept, replacing only a fraction of the original population or keeping a small number of the best solutions from one generation to the next (elitism). Several methods apply also to the selection of the individuals to be removed from the population, from directly eliminating those with the worst fitness to selecting from the worst $p\%$ (*e.g.* below the median for $p = 50$) or from the oldest ones.

The generation of the initial population also developed several variants other than pure random generation. A more even and thorough coverage of the search space can be obtained using a generalization of the Latin hypercube. Yet another usual approach is to seed the population with known good solutions, if available or if some other heuristic can be used to generate them, although this method involves some risk of inducing premature convergence.

These improvements or variations apply to the more standard flavours of GAs, but often they are adapted to address specific problems, using representations and operators that better fit the use case, *e.g.* by incorporating domain knowledge. The use of permutations for scheduling problems and

TSP is one example. There are many others, and the performance of the algorithm may be completely different using one representation and set of operators or another. GAs adapted in this form are considered *grey box optimizers*.

One final improvement is to refine each generated individual using local search rather than mutation. These are known as hybrid GAs or memetic algorithms.

Beyond the developments in GAs proper, the field has expanded into related variations of the theme, like Genetic Programming (GP), Estimation of Distribution Algorithms (EDA), Evolution Strategies (ES), co-evolution, etc.

2.1.2 Ant Colony Optimization

Ant Colony Optimization (ACO) actually encompasses a number of similar algorithms for solving discrete optimization problems and which are inspired by the behaviour of real ant colonies in nature. Some species of ants deposit pheromone trails when travelling between the nest and a food source; shorter paths accumulate more pheromone as they are travelled more frequently, which attracts more ants, which in turn deposit more pheromone in a positive feedback loop.

The first ACO system, known as Ant System (AS), was introduced in [Dor92]. It sets the framework common to all the variations:

1. Initialization of the pheromone

2. Iteration loop until ending condition is met

(a) Construction of ant solutions

(b) Optionally, application of local search

(c) Global pheromone update

All ACO methods represent solutions as sequences of building blocks or components, so that they can be constructed step-wise. For instance, in the case of TSP, which was the first application, an ant starts at one of the cities, and in each step adds a new city to the tour. The pheromone, combined with heuristic information if available, determines the probability of selecting a particular component for the next step.

In AS, all pheromone is initialized to a value τ_0, which is a parameter of the algorithm. The construction of the solution for each ant progresses by taking the next step with probability

$$p(b_{ij}|s_i) = \frac{\tau_{ij}^{\alpha} \cdot [\eta(b_{ij})]^{\beta}}{\sum_{b_{ik} \in \mathcal{N}(s_i)} \tau_{ik}^{\alpha} \cdot [\eta(b_{ik})]^{\beta}} \quad \forall b_{ij} \in \mathcal{N}(s_i), \tag{2.1}$$

where $p(b_{ij}|s_i)$ is the probability of adding block ij given the current state of solution s_i; τ_{ij} is the pheromone associated to block b_{ij}; $\eta(b_{ij})$is the heuristic associated to block b_{ij}; $\mathcal{N}(s_i)$ is the neighbourhood of the current state of solution s_i, that is, the blocks that can be added at given the current state at the moment; and α and β are algorithm parameters that regulate the relative weight of pheromone and heuristic. In the case of the TSP, b_{ij} is the arc from i to j, and the neighbourhood of s_i is the set of arcs going out of city i and into yet unvisited cities, *i.e.* those not in s_i, which is the (partial) tour built up to the current stage.

After every ant has built its solution, and evaluated the evaluation function $f(s)$, the pheromone is updated as:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{s|b_{ij}\in s} \frac{1}{f(s)} \quad \forall ij, \tag{2.2}$$

where $\rho \in (0,1)$ is an algorithm parameter.

Ant Colony Optimization as a general framework is described in [DDG99].

In time additional variants have surfaced. Following mainly [DS19], the most relevant ones are discussed below.

The elitist strategy is the first variation to appear. At each iteration, the best global solution adds pheromone even if it was not present in the current iteration, and it is given a strong weight, compared to the rest of solutions.

Rank-based Ant System (AS_{rank}) takes this elitist strategy one step further by having only the best solutions update the pheromone. The number of updating solutions is a parameter n; the best solution is assigned rank $r = 1$, the next best $r = 2$, and so on. Each of them performs the pheromone update with weight $(n + 1 - r)$.

$\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) bounds the pheromone values above and below, $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max} \forall \tau_{ij}$, and initializes all pheromone to the upper bound, to encourage exploration in the initial phase. The lower bound helps prevent stagnation. Pheromone update follows a strong elitist strategy as described above, using either the global best or the iteration best, or alternating them. $\mathcal{MM}$AS also introduced two tools that are applied in other variants: occasional pheromone reinitialization to increase exploratory behaviour and the application of local search to each solution before the

pheromone update.

Ant Colony System (ACS) focuses on the exploitation of information collected by previous ants. ACS uses strong elitism, only adding pheromone from the global best solution (or, less frequently, the iteration best solution) after each iteration, in amount of $\frac{\rho}{f(s)}$. The selection of the next component during solution construction follows the pseudo-random proportional rule: with probability q_0, $0 \leq q_0 < 1$, it takes a greedy step, *i.e.* it selects the component with the highest probability as given by equation (2.1); alternatively, with probability $1 - q_0$ it applies the usual selection method. Additionally, ants update the pheromone trails during construction of the solutions, which results in actually removing pheromone from the components used in building a solution, which makes it less likely that all ants converge to the same solution.

The Ant Colony Optimization approach has been adapted to additional scenarios beyond the original discrete optimization field, such as multi-objective optimization ([GCH07; LS12a]), optimization of dynamic problems which change over time ([MLY17]), or stochastic optimization ([Bia+09]). Since these fall outside of the scope of the current work, the interested reader is directed to the relevant references.

2.2 Parallel Metaheuristics

The execution of metaheuristics can become very resource-intensive, depending on the problem. This may arise from the computation of the objective function, the constraints, or solution repairing procedures; or from the need

to explore a huge space of potential solutions. This is true of every kind of metaheuristic, whether single-solution-based, or population-based.

One way to mitigate this effect is to exploit the available computational resources to perform several calculations simultaneously, thus reducing the wall time[2] needed. Since the algorithms may need to be adapted or modified to take advantage of parallelization, other benefits besides a speed-up can also be obtained, such as improving the quality of solutions or the robustness of the algorithm, or tackling larger scale problems.

2.2.1 GPU-based Parallelization

GPUs follow a very different paradigm from that of CPUs. The CPU has a limited number of processing units, limiting its ability for simultaneous execution, but it can handle diverse tasks that require large amounts of data, managed by a complex control unit and enabled by a relatively ample cache to accelerate memory access. GPUs, on the other hand, sport a very high number of computation units, potentially in the thousands, with more limited caching and flow control. Moreover, the GPU is a co-processor that depends on the CPU for exchanging data with the main memory and defining the operations via kernels.

The typical architecture of a GPU consists of multiple streaming multiprocessors (SMs), each containing several streaming processors (SPs), also

[2]In computation systems different times can be measured, mainly wall time (also known as wall-clock time, real time, or real elapsed time), the time as measured outside of the computer from beginning to end, that is what we would experience as users of the system, as opposed to CPU time (or process time), which is the amount of time the processing unit was used to do the work. This can exceed wall time in multi-threaded, multi-core, or distributed systems where operations can happen in parallel. CPU time can be further divided into user, system, idle, and steal times.

known as processor cores. Each core can execute a limited set of operations, usually simple precision arithmetic and some special functions such as square roots or trigonometric functions depending on the model, in a SIMD (single instruction multiple data) fashion, that is, the operation is the same for all processes, though each one applies it on its own data instance. The prototypical example is element-wise vector addition, where each core takes as data the corresponding elements in two vectors (say, the i-th element of each vector), calculates their sum, and stores the result as the i-th element of the output vector; all cores do this at the same time, performing the addition in parallel for the whole vector. There are 32 execution threads in a SM working in lockstep as described; each of this groups is called a *warp*.

SMs also include shared memory for its SPs, and are grouped into thread processing clusters (TPCs) for additional caches shared among the SMs. The interconnection network allows communication among these elements and also with the global memory (DRAM).

One relevant offshoot of the SIMD execution paradigm of GPUs is the effect of branching. When alternative execution flows are needed, such as in the case of an if-else clause which evaluates to true for some threads and to false for others, the SM will execute both sequentially, applying the effects only on the appropriate threads. With additional nested branching, the execution paths increase exponentially, diminishing the parallelization advantage, to the point where it can be counter-productive.

Modern GPUs can simultaneously perform the execution of its threads and move data both to and from main memory. However, this relies on the CPU providing the instructions in a manner conducive to it. The instructions

are received asynchronously, and launched as soon as the GPU has the ability. For large loads or iterative operations, this becomes very relevant to ensure avoiding idle times and maximizing utilization.

An in-depth description of GPU architectures is available in [ND10] and [Ryo+08].

With the advent of general libraries that provide a C-like environment for programming GPUs, especially CUDA and OpenCL, and more recently Vulkan, there has been intense research on the parallelization of metaheuristics using GPUs.

The main difficulties of adapting metaheuristics to GPUs are explored in, among others, [Van11], which addresses metaheuristics in general, including both single-solution and population-based metaheuristics. These challenges can be summarized as:

CPU and GPU coordination Which parts of the algorithm should be run on each, considering the effects that partition will have on the level of parallelization, the utilization of the GPU capabilities, and the need for data transference between CPU and GPU.

GPU computation Making the most of the parallelization capabilities of the GPU within the relatively limited amount of memory, mapping the tasks into memory structures that can be appropriately addressed by the threads.

Memory management Optimization of data access is needed for performance, by efficiently using the multiple GPU memory spaces considering their sizes and latencies. Additionally, data exchange with the

CPU must be adequately interleaved with the operations.

These are considered using several standard problems as use cases: the Permuted Perceptron Problem, the Quadratic Assignment Problem, the Weierstrass Continuous Function, the Travelling Salesperson Problem, and the Golomb Rulers. The solution method relies on single-solution metaheuristics that work with a neighbourhood structure such as hill climbing, by parallelizing the generation and evaluation of the neighbourhood in the GPU for each iteration. While this allows for some level of generalization, the calculations in the GPU, as well as the definition and management of data structures are problem dependent. The tests show important acceleration of the algorithms, and its integration in the ParasdisEO[3] framework is showcased.

More specifically for GAs, [Krö+11] implements GPU accelerated Genetic Algorithm and Differential Evolution methods for task scheduling by offloading problem-specific computation to the GPU. They also show speed-ups of several orders of magnitude, but once again the implementation is problem-specific and relies on the fitness function structure being amenable to efficient implementation in the SIMD paradigm.

As an example for ACO, [Del+13] proposes several GPU implementations of $\mathcal{MMAS}$ for the TSP, ensuring equivalent execution, and therefore results, to the sequential version, and analysing the resulting speed-ups in several instances of varying sizes:

- A single colony is used in one GPU, with one ant in each iteration assigned to a single GPU execution thread for tour construction and

[3] `http://paradiseo.gforge.inria.fr/`

local search. Pheromone information, distance matrix, and probability arrays for each ant need to be stored in the global memory, as they soon grow too large for the limited local memory available to each thread. For different numbers of blocks and threads used, speed-ups range between 0.17 and 0.82. This means that the overhead of memory management and CPU-GPU communication actually makes the algorithm slower than running sequentially on the CPU.

- A single colony is used in one GPU, with one ant in each iteration assigned to a GPU block, parallelizing tour construction and local search across the threads in the block. The increased available local memory —now the whole block— means that the data structures needed to calculate the state transition can be kept in local or global memory, while pheromone information and the distance matrix still can only fit in global memory. The speed-ups in this case range from 2.85 to 12.48, depending on the size of the problem and the number of blocks used. Also, the variant using local memory shows better results for smaller instances, where the faster local memory is fully exploited; for larger instances, the higher volume of data in shared memory limits the number of concurrently active blocks, penalizing performance to the point of making the variant using global memory faster.

- Multiple colonies are used, each one assigned to one execution block; within the block, one ant is assigned to each thread as above. For the Multicolony approach the whole algorithm is moved into the GPU, rather than just the tour construction and local search stages to avoid

the need of continuously exchanging multiple pheromone matrices between the CPU and GPU. Speed-ups result in the range 0.06–1.77. Only for a large number of colonies (256) the GPU version is faster than the sequential one, at the expense of solution quality. Similar effects to those described above, together with the synchronization of threads within a block for tour construction and local search, undo all potential benefits of GPU parallelization.

- Multiple colonies are used, each assigned to one GPU, with each ant assigned to a block. In this case each colony is an independent algorithm, run as a CPU thread or process and using the GPU as a coprocessor. This results in the highest speed-ups: 9.38–12.70 for 256 iterations with one colony, and 16.24–23.60 for two colonies, 128 iterations each.

While relatively important speed-ups are attained, it is by trying out multiple strategies, all tailor-made for the specific algorithm and problem combination, from the GPU kernels to memory and CPU-GPU communication management.

The general conclusion on GPU acceleration of metaheuristics is that there is no generality. Instead, each algorithm and problem combination needs to be analysed and the right approach designed to extract the most from the parallelization ability of the GPU.

The literature focuses mostly on academic problems, with a straightforward definition of the fitness and constraint functions. For other, more practical, types of problems this may not be the case. The scheduling problem defined in chapter 1 includes several cost terms and constraints that

require multiple branching control flow, which clashes with the SIMD execution paradigm in GPUs.

2.2.2 Multiprocessing

The potential for parallelization of metaheuristics has long been recognized. For instance, [PLG87; JSV91] already discuss parallel versions of Genetic Algorithms and [Stü98] does the same for Ant Colony Optimization only a few years after the first publication of the algorithm.

Numerous works analyse and propose various approaches. Some of them focus on fine-grained parallelization. This was addressed in part in the previous section on GPUs; in the case of multiprocessing, as before, the fine granularity means that the parallelization strategies must be problem dependent, rather than generic. Therefore, the focus will be on the more coarse-grained variants. Among all the possible references, [Alb05; Tal09; Cra19] provide broad surveys (including fine-grained approaches).

Three main alternatives present themselves for coarse-grained parallelization of metaheuristics: Multiple Independent Runs, domain decomposition, and population-level parallelization.

Multiple Independent Runs has already been mentioned as the baseline parallelization strategy. Each processor runs one instance of the problem, possibly with different hyperparameter configurations, and the best solution overall is considered the final result.

Domain decomposition partitions the problem in some form, and each processor runs the search algorithm within its own sub-domain. As with

Multiple Independent Runs, the best solution overall is the final result, but unlike it, this approach allows a deeper search by concentrating the computing power of a processor on a restricted region instead of the whole problem.

The partitioning is usually based on the input space, the possible values of the decision variables, but other dimensions can be applied depending on the problem. If several classes of solutions are possible, each subdomain could correspond to one class and optimize the parametrization of the class; the parameters or variables for each class could be different in this case.

This reliance on the specifics of the problem limit the general applicability of this strategy.

The third strategy, population-level parallelization is the most widespread. It is also generic, like Multiple Independent Runs and unlike domain decomposition. In this strategy, each processor runs one population for the problem, possibly with different hyperparameters as in the case of Multi-start; only the populations are not fully independent, instead there is some exchange of information among them to guide the optimization process in a different way.

There are several design levers that apply:

Communication structure and neighbourhood topology This specifies which populations communicate with which others; populations that are in contact with one another are considered neighbours:

All-to-all Every population is a neighbour of every other population.

Ring topology Population i neighbours populations $i-1$ and $i+1$ (wrapping around).

Mesh topology Populations exist on a 2-dimensional mesh, and population (i, j) has four neighbours: $(i, j \pm 1)$ and $(i \pm 1, j)$. This topology can be generalized to higher dimensions, but 2 is the most usual number as it maps directly to the network topology of some high performance computing clusters.

Hypercube topology There are 2^k populations, and populations i and j are neighbours when the binary representations of i and j differ by exactly one bit.

Random topology The connectivity is determined randomly at each communication step, and different methods can be used for the determination, *e.g.* each population randomly selects one other population to communicate with.

Information exchanged What is communicated between populations.

Solutions This is the most used option: each population communicates one or several solutions, typically its global or local best.

Other information Other forms of transmitting information are possible, but algorithm-dependent. For instance, in Ant Colony Optimization pheromone matrix or pheromone update information can be used instead of solutions.

Communication frequency When does the communication take place:

Every iteration Populations exchange information after each iteration.

Every k iterations This can be synchronous, with all populations exchanging information at the same time, or asynchronous, with $\frac{m}{k}$ populations communicating on any given iteration, for m populations.

Solution quality dependent A population sends out information only under certain conditions, such as improvement in best solution or after significant variation of the population.

Heterogeneous or homogeneous The populations can use the same or different parameters; these include not only algorithm hyperparameters, but also foundational building blocks such as selection strategies for Genetic Algorithms or solution construction methods for Ant Colony Optimization. An extreme form of heterogeneity is to run completely different algorithms on each population, as a form of hybridization.

This type of parallelization strategy improves on the sequential approach in the following aspects [JSV91]:

- The selection of individuals is local to the subpopulation, which requires less computation compared to selecting from the whole population.
- Each subpopulation can progress asynchronously, reducing the synchronization overhead.
- The algorithm is more robust, as performance of each processor is independent from the others.

Genetic Algorithms

Population-level strategies are the most used ones in Genetic Algorithms, from fully independent subpopulations to fully-connected topologies; although the predominant topology uses a limited number of neighbours for each population in a mesh or mesh-like configuration: the Islands model proposed in [PLG87].

The potential for acceleration was already shown in [MSB91], evaluating an Islands model on different numbers of processors, and obtaining superlinear speed-ups for some configurations, which suggest that there is an inherent benefit in the exchange of information to more efficiently guide the search. Additional variations and applications shown in [RPE99] corroborate the trend.

While these keep the Islands model as the base parallelization strategy, they explore the parameters of the populations: population sizes, selection methods, crossover and mutation operators, etc. A later review in [Alb05] shows the same prevalence of population-level (coarse-grained) strategies, and the first generic frameworks for parallel algorithms like DREAM and ParadisEO.

In general, the parallelization strategies for Genetic Algorithms closely follow the generic ones described above, as they were mostly originally devised for them and then generalized to other population-based metaheuristics.

ACO

Parallelization strategies for ACOs can be classified in the following categories [PNC11]:

Leader-follower[4] A leader process keeps track of global information (pheromone matrix, best solution so far, etc.) and offloads some operations on the follower processes. Three sub-categories are distinguished by the granularity of the offloaded operations:

Coarse-grain leader-follower Interaction with the followers is based on full solutions, with each follower running one or more ants and returning the solutions to the leader.

Medium-grain leader-follower Each follower solves a subproblem, derived through decomposition of the problem, and the leader pieces the partial solutions together to construct a complete solution.

Fine-grain leader-follower The followers process minimum granularity operations, such as component evaluation, requiring very frequent communication.

Cellular model The colony is structured according to some neighbourhood scheme, keeping several pheromone matrices updated by ants belonging to each neighbourhood; neighbourhoods overlap so that high-quality solution spread by diffusion.

[4]Historically called *master-slave*, including in the indicated references. Leader-follower is one of several contenders to replace the term, as it has lately become controversial. However, the original name has to be pointed out, since that is the form in which it is found in the literature.

Parallel independent runs The Multistart method discussed above.

Multicolony models Several colonies, each with its own pheromone matrix, explore the search space, and periodically exchange information.

Hybrid models These models combine properties from several of the categories above.

Others propose similar classifications. For instance, [Sal09], analysing parallel strategies for metaheuristics in general, not just Ant Colony Optimization, reviews multiple parallel metaheuristics applied to cutting, packing, and related problems, finding two main classes: leader-follower, where the leader manages the population and the followers perform operations on the individuals or subpopulation assigned to them, such as fitness evaluation; and structured population, a generalization of the use of independent subpopulations with varying degrees of communication.

Likewise, [CT10] introduces a similar classification into three types, two of which match the ones described above (parallelization of low level computation within an iteration, and multiple subpopulations with varying degrees of granularity and communication); the third type involves partitioning the solution space across the components of the solution variables, with each processor working on a given subset, assuming all other components constant. This usually involves multiple iterations for a suitable exploration of the solution space, and is reminiscent of coevolution methods.

Although different authors slice and dice the categories slightly differently, the main concepts are the same, and it is easy to find the correspondences in the different taxonomies. The Multicolony models most closely correspond to

the multiple populations, though some forms of coarse-grain leader-follower can also be used (the main difference being that multiple processors compute for a single colony, which can be inefficient for distributed processors, but not necessarily so for multiple cores in a same CPU).

Another specificity of Ant Colony Optimization is how to integrate the new information. Equivalently to the migration of individuals, a solution (as built by an ant) can be exchanged, then the pheromone update may apply or not (*e.g.* for elitist strategies the new solution will deposit pheromone only if it improves on the incumbent best solution). The use of solutions in this way is specially straightforward in population-based ACOs. Alternatively, pheromone information can be exchanged directly, which has no direct analogy in GAs and other metaheuristics.

Pheromone exchange can happen by communicating the whole or part of the pheromone matrix, which then must be integrated with the existing one in a way consistent with the interpretation of the pheromone for the problem at hand. It can also happen by communicating the pheromone update triggered by the solution(s), which can then be directly applied in the receiving colony.

2.2.3 Evaluation of Parallel Metaheuristics

For the evaluation and comparison of parallel metaheuristics, [AL06] analyses meaningful metrics and common pitfalls. The following are specially relevant to the present analyses:

- The use of wall time when comparing the time performance of algorithms, as this includes all the overheads and other effects of the ap-

proaches being compared, and best reflects the overall running time. This is opposed to process time or predefined effort (number of evaluations).

- Ensuring that the algorithms are comparable, both from a design and implementation point of view, and from a solution quality perspective. The predefined effort is used for this purpose when comparing solution quality.

- Since metaheuristics are stochastic, a single value or a summary aggregation such as the mean value is not enough for fair comparison. Instead, the probability distributions must be compared, checking for statistical significance in the difference. Typically a combination of normality test and Student t-test or ANOVA is used, or some non-parametric test is applied if normality cannot be ascertained.

The metrics used for evaluation are also important to afford meaningful and relevant results. This work focuses on three metrics:

Wall time to run the algorithm until exhausting a predefined number of function evaluations.

Solution quality the best objective function value achieved within a predefined number of function evaluations. This is a standard metric in the field.

Hypervolume as described in [PLS14], as a metric of anytime performance.

The hypervolume metric provides a more nuanced view of the trade-off between running time and solution quality. This is related to how fast the

algorithm reaches better quality solutions, as more demanding running time constraints might require ending the algorithm early, and we would want the solution at that stage to be as good as possible. This is measured as the (properly normalized) area under the curve of time (iteration) versus solution quality (assuming minimization), and it's better the smaller it is[5].

The term hypervolume for this metric comes from the interpretation of this curve as the Pareto front of the bi-objective optimization that minimizes the objective function and the running time. Considered in this way, it is evident that the best solution so far each iteration are the non-dominated solutions.

[5]The original definition of hypervolume actually uses a reference point x_0, and considers the space between the hyperplanes $x^{(i)} = x_0^{(i)}$ and the Pareto front. With this definition, and the typical choice of a nadir-like reference point, hypervolume improves as it increases. For the specific case in this work, this alternative definition corresponds to choosing the origin $(0, 0)$ as the reference point, which will result in negative hypervolume as both the number of iterations and the cost functions are always non-negative. Increasing hypervolume under those conditions is equivalent to decreasing absolute value of the hypervolume, which in turn is equivalent to the area below the curve.

Chapter 3

Methods

Contents

3.1 Introduction

Parallelization and distribution have become the main way of achieving performance and scalability of algorithms, following the recent dynamics of computing power increase. Where the evolution used to consist of higher speeds, now it comes in the form of additional cores or processors. Programs no longer run faster just by updating the hardware; instead, the operations must be parallelized to exploit the new advances. This can be attained in diverse ways and in different layers of the design, as described in chapter 2.

The choice of the parallelization method depends on the available hardware and the details of the problem, and also the chosen algorithm.

If GPU is an option, computationally intensive evaluation of the fitness function or constraints can be parallelized if the functions have the right properties; large-scale vectorized or matrix operations are the textbook example. For simpler functions, multiple solutions can be evaluated in parallel, whether it is a population in a population-based method, or the current neighbourhood in single-solution methods. It may be even possible to combine both to extract as much performance as possible from the GPU and compensate for the overhead of memory management.

If the functions are not suitable for the GPU (large data structures that don't fit nicely in the right memory sections, deep branching, or different operations depending on the individual) or the GPU just isn't an option, a similar approach can be taken by using multithreading in multi-core processors. The number of concurrent execution threads is much smaller, but with more flexibility in terms of memory and execution flow.

For smaller problems, this parallelization approach can also be applied at a more coarse-grain level: start multiple instances or populations.

Although it is possible to extend this when moving into multiprocessing, the network latencies introduce a new trade-off that makes frequent and/or verbose communication less desirable. Therefore, in this conditions more independent processes are preferred; they may be fully isolated, but some level of communication is possible without too much of an impact, and in fact the diversity induced by information exchange tends to enable improved solutions over fully independent approaches.

Some algorithms are specifically designed for the parallelized scenario, and may be applicable for a given problem; however, this needs to be assessed each time.

This reasoning focuses on the ability to accelerate the computation by taking advantage of the available hardware to the greatest extent possible. Some additional considerations apply in practice.

One of those is the ability to apply these methods. Concurrent algorithms in general are harder than sequential ones and require additional knowledge. GPUs are specially relevant in this respect, as even with the latest advances they require very low level development, with careful management of memory, communication, and execution flow. Even if in the right circumstances they can provide speed-ups of several orders of magnitude, those circumstances are severely limited and the programming expertise is not usually widely available within the metaheuristics community.

Multithreading is the least difficult to apply. While the usual caveats related to concurrency must be considered (race conditions, consistency, etc.),

the memory model and the fact that communication is effectively instantaneous provide some advantages. On the flip side, this is the least scalable option, as the number of simultaneous threads of execution in a single processor is limited to a few tens[1] (at the moment).

Multiprocessing is very similar to multithreading conceptually, but some additional practical concerns arise. By not sharing the memory space, the communication methods and thus the parallelization primitives are different. Also, the data must be copied and moved, incurring network latency, which must be accounted for when considering the usefulness of the additional development effort.

Multithreading and multiprocessing need not be so different, especially if one takes into account that the multiprocessing methods are perfectly viable within a single processor, if incurring some penalties due to the need to maintain and communicate multiple copies of the data. Conversely, by enforcing the communication approach and not sharing the memory space, they remove a whole class of potential bugs.

Both newer and older libraries used for multiprocessing allow for decoupling of the hardware architecture: the algorithm is developed using the communication primitives, and then the parallelization framework can be deployed locally or across a network.

In order to evaluate the improvement brought about by a new parallel algorithm, two components need to be differentiated:

[1]This discussion assumes a standard computing architecture. A memristor-based in-memory computing platform, for instance, can change the dynamics, but those are still experimental and would require a redesign of the algorithms to exploit the new architecture.

1. the contribution of parallelization, and

2. the contribution of the novelty of the algorithm itself.

Since most parallel algorithms tend to be problem-specific, it is not known *a priori* whether any potential gain comes just from the additional computation budget or from new features. This is usually addressed by using Multiple Independent Runs as a baseline for the performance of the algorithm.

Multiple independent runs is the most basic parallelization strategy for metaheuristics. It consists of running several independent instances, and keeping the best one. If we assume that the main constraint is wall time, this multiplies the computation budget according to the number of concurrent instances, and is equivalent to a (sequential) Multistart, applicable to most metaheuristics, whether single-solution or population-based.

Even if it can be simulated sequentially, normally the Multiple Independent Runs algorithm will be implemented in parallel as a first step, after all it will be the baseline for the comparison of a distributed algorithm, so the parallelization framework will be deployed anyway; the parallel implementation also allows to directly compare wall times.

3.2 Objective

Looking at the spectrum of parallel metaheuristics from the point of view of the trade-off of efficiency, both in terms of acceleration and solution quality, and algorithm complexity, both from the conceptual and practical perspectives, Multiple Independent Runs lies at one end, that of the simplest imple-

mentation. Other methods span the efficient frontier from general methods, like the Islands model in Genetic Algorithms or Multicolony Ant Colony Optimization, that add limited complexity for some additional gains in efficiency to customized, problem-specific approaches that require more work but allow tackling extreme problems.

While attacking the further end of the spectrum is an attractive prospect, and indeed an active field of current research, doing so means abandoning the generality of algorithm improvement and delving into customization and adaptation to specific problems.

There remains, however, a gap between the Multiple Independent Runs approach and full-fledged distributed algorithms that may be exploited for general gains across whole families of algorithms. In particular, the best possible outcome in this area is to improve the efficiency of the algorithms while keeping the complexity constant. Looking at the trade-off as a Pareto front of efficient solutions, this would mean finding an approach that dominates Multiple Independent Runs, and therefore takes its place in the Pareto front. At the very least, if the incurred added complexity is negligible, the new method will be part of the efficient frontier, and an interesting option as a baseline.

This thesis posits that the Multiverse method fits into that gap. Like Multiple Independent Runs, it can be considered a meta-algorithm that applies to other algorithms to create parallel versions. As such, it is applicable to all population-based metaheuristics. This is a narrower scope than that of Multiple Independent Runs, but is still a very broad range of algorithms. If it can substitute Multiple Independent Runs as the baseline for population-

based individuals, it will claim a large swath of applications.

On the complexity side, Multiverse adds a single, one-directional communication step that is hardly noticeable. In the implementation used for the tests in this work, it amounts to three extra lines of code. Once the parallelization framework is set up, this is negligible. And, like Multiple Independent Runs, it is trivial to emulate in sequential programming by running the collector last in each iteration.

Only the effects on running time and solution quality remain to be tested. If the method can be proven to improve on these, the conditions described above are met.

3.3 The Multiverse Method

The Multiverse method aims to provide the same generality and ease of use and implementation that Multiple Independent Runs does, while improving solution quality without affecting running time, potentially becoming the new baseline for multiprocessing parallelization of population-based metaheuristics.

In order to attain this level of generality, it cannot rely on particulars of the algorithm it is applied to or the specific problem that is being solved. This makes it a *meta-algorithm*, an algorithm that takes one algorithm as input and returns a modified algorithm.

It also means that it is orthogonal to other parallelization, acceleration, or improvement approaches. For instance, it is trivial to combine Multiverse at the node level in a cluster with intra-node acceleration using GPUs for

e.g. solution evaluation. The Multiverse method falls in the population-level parallelization class, corresponding to an Islands model for GAs and a Multicolony model for ACO; however, the neighbourhood structure, rather than following a ring, hypercube, or mesh configuration as usual, adopts a star-like structure that mirrors the connectivity of a cluster of workstations. The more usual patterns derive from the hardware configuration of high performance computing clusters.

Although the Multiverse method is compatible with other approaches such as local acceleration, different hyperparameter settings —or even different algorithms— for each population, etc., its greatest advantage is its ability to generate an effective parallel version of an algorithm without complex algorithm design and implementation.

The choice of exchanging information in the form of solutions (individuals) stems from this need of generality. All population-based methods have the concept of individuals and mechanisms to integrate them in the solution procedures, while other representations would be algorithm-dependent, or even problem-dependent. As long as all populations use the same algorithm, this communication should result transparent to the local algorithms; if multiple algorithms are mixed, some transformations may be needed: different algorithms may benefit from different representations that better fit the operations performed. This transformations can however be performed at the time of injection of the migrant solutions, without a significant impact. Since this work focuses on the baseline use of the method, the variant using the same algorithm throughout is assumed without loss of generality (except if the transformations are computationally intensive).

See the discussion above and [Cra19; Alb05; JSV91; PNC11; Kaw+00; MRS02; JMM05] for in-depth surveys of existing models.

As a note on implementation, the chosen communication framework, MPI, is one of the most (if not *the* most) usual ones in the literature (*cf.* the references in 2.2.2 and the previous paragraph). At the time of development it also provided the needed features with the minimum fuss for implementation, and it was (and still is) widely available in distributed computing facilities. However, in the intervening time and with the developments following the increased importance of distributed computation new and improved frameworks have appeared.

One cannot fail to mention Hadoop and Spark, the *de facto* standard Big Data platform. While it could be possible to implement these methods on top of such a platform, the underlying map-reduce structure constrains the development and goes against the simplicity and generality principles that are the main objectives.

Generic DAG-based[2] parallelization libraries offer a more modern approach without sacrificing flexibility. If developed today, considering the use of Python as a programming language, the main candidates would be Dask[3], probably the most widespread such library in Python, and PyCOMPSs [Tej+17], developed by the Barcelona Supercomputing Centre following the same philosophy as a wrapper around their COMPSs library[4] for high performance

[2]DAG: Directed Acyclic Graph; it refers to the graph of operations defined by the code, and the constraint that it should be directed and not contain cycles to allow for efficient execution scheduling methods.

[3]`https://dask.org/`

[4]`https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar`

Listing 3.1: Pseudocode for population-based metaheuristics

```
1 Population ← InitialPopulation
2 While not EndCondition:
3   Contributors ← Select(Population)
4   NewIndividuals ← Process(Contributors)
5   Population ← Trim(Population ∪ NewIndividuals)
6 Return Best(Population)
```

computing. Both are free open source projects that, had they been available at the time, would have simplified both the implementation of the algorithm and the overall set-up.

The proposed Multiverse method extends population-based metaheuristics in general. Population-based metaheuristics follow the common procedure described in listing 3.1, repeating the loop until a certain condition is met: a number of iterations, a given fitness level, some number of generations without improvement, etc.

Different algorithms define alternative strategies for each of the steps in the procedure, and may in implementation blur the distinctions between them.

Genetic algorithms and Ant Colony Optimization are examples of the specialization of the general procedure. In Genetic Algorithms the selection is *e.g.* through a tournament or round-robin, and the generation of the new population, the `Process` function, uses the crossover and mutation operators; while in ACO the generation of new individuals occurs when the ants build their solutions using pheromone information (which is the way that the 'population' is carried over from one generation to the next) and the selection corresponds to determining which individuals will deposit pheromone.

Listing 3.2: Pseudocode for Multiverse metaheuristics

```
1  Population ← InitialPopulation
2  While not EndCondition:
3    Contributors ← Select(Population)
4    NewIndividuals ← Process(Contributors)
5    Population ← Trim(Population ∪ NewIndividuals)
6    If not Collector:
7      Send(BestIndividual)
8    If Collector:
9      Population ← Population ∪ ReceivedIndividuals
10 Return Best(Population)
```

They can be considered the archetypes for two broad classes of population-based metaheuristics: those which build solutions atomically from previous individuals (like GAs) and those that construct new solutions by combining elements or components of previous solutions one at a time (like ACOs). As such archetypes, they are the obvious choice to test the generality of the Multiverse method.

The Multiverse method injects a new step just before closing the loop. In this step, every instance of the metaheuristic sends its current best solution to the controlling node, and it relays them to the collector instance, which adds them to its own population. Due to the very design of this type of metaheuristics, everything else just needs to go on as usual, giving the new procedure in listing 3.2.

The meta-algorithm cannot be tested directly. Instead, the evaluation must be performed on algorithms derived using it. Two are Multiverse variations of GA and ACO applied to the Asymmetric Travelling Salesperson Problem are developed to validate that the method brings about improve-

ments. These are reviewed in chapter 4.

Finally, to evaluate performance on the practical, real-world problem that is the main target of this work, the third implementation is a Multiverse variant of the Ant Colony Optimization method applied to the scheduling of a galvanizing line. The results are reviewed in chapter 5.

As the aim of this thesis is to assess the improvements brought by the Multiverse approach, the base algorithms are far from state-of-the-art. Instead, the focus is placed on creating a level playing field for both versions, with as little influence from external factors as possible. For this reason, the GA relies on relatively simple crossover and mutation operators among those available in the literature and the ACO uses the standard parametrization, and neither of them performs a local search step. Similarly, there is no comparison between the results and the best known solutions in the literature; doing this would require a much greater development effort in the design and programming of the algorithms, but is irrelevant to the objective.

3.4 Set-up and Methodology

The hardware set-up for the tests described below consisted of a cluster of 18 virtual machines with 512 MB of RAM memory running Arch Linux on top of an Intel Xeon E5-2695 at 2.40 GHz.

The logs containing all the information needed for the analysis are provided as supplementary material to this thesis. The program for the tests was written in Python, using MPI for communication across nodes, and is available in GitHub at the URL `https://github.com/valthalion/endof`.

The repository also includes the code used to process the results and automatically generate the tables and charts used in the thesis.

The overall procedure starts from the 18 instances of Asymmetric Travelling Salesperson Problem in the benchmark collection TSPLIB. For each of them, four sets of runs are executed: GA Multistart, GA Multiverse, ACO Multistart, and ACO Multiverse. Each set consists of 25 runs with random seeds. Each run logs to a text file the running time, random seed, final solution, and best solution at each iteration. The logs may be recreated by re-running the experiments using the recorded seeds.

The logs are parsed and loaded into a MySQL database for easier manipulation. The repository contains the Python script for populating the database, as well as the SQL code to generate the database schema.

Another script processes the database to generate the aggregated values reported in the tables throughout this work, including generating LaTeX code for some of them, the statistical tests comparing corresponding Multistart and Multiverse instances, and the charts summarizing the results: boxplots of best solution and hypervolume, and the graphs of evolution of the objective function with the iterations.

The scheduling problem follows the same procedure, using a Multistart and a Multiverse version of the model described in [Fer+14] and 46 scheduling problem samples taken from historical data of the galvanizing line. The code and benchmark files cannot be made available due to confidentiality reasons, but the resulting log files are provided along with the results.

For each Multiverse-Multistart pair, the following comparisons are performed:

- Average difference in running time
- Average and minimum values of the best solution found
- Average and minimum values of the hypervolume
- Statistical test for significance of the difference in the average solution value
- Statistical test for significance of the difference in the average hypervolume

Following the concerns on metrics discussed in chapter 2, these metrics and the procedure for applying them were defined before running the tests, and there is no selection of the instances used: the whole set from TSPLIB is used for the TSP problem, and all available instances of the scheduling problem at the time were included in the scheduling part. In fact, the analysis code was written before running the tests. The only changes made after the fact remove one or two of the simplest instances in the TSP problem from some of the analysis because of numerical problems; these are discussed in chapter 4 while reviewing the results.

All comparisons are performed at the probability distribution level. Although aggregated values, such as average and minimum, are used to showcase the results and charts of individual runs are shown to discus the evolution of the algorithms, boxplot charts are also included for comparison of the corresponding probability distributions, and conclusions are supported by statistical significance tests.

The selected test is Mann Whitney U. There is no reason to expect the distributions to be normal (and looking at the boxplots in chapter 5 they clearly are not), so a non-parametric test is preferable. The directed Mann Whitney U test is such a non-parametric test geared towards establishing statistical significance of the difference in the means of two empirical distributions, which perfectly matches the needs of this assessment.

The choice of stopping criterion is, accordingly, a predefined effort. Additionally, by evaluating anytime performance the potential bias introduced by the selection of the effort level is mitigated.

3.5 Algorithm Configuration

The algorithms use standard parameters, except for the termination condition which was set to a number of iterations equal to 10 times the number of cities, a usual heuristic for the Travelling Salesperson Problem.

3.5.1 Genetic Algorithm

The configuration for GA is:

- A population of 50 solutions is used
- No elitism
- 50% and 5% crossover and mutation probabilities, respectively
- The crossover operator is the permutation interpretation of 2X
- The mutation operator randomly exchanges two cities in the loop

3.5.2 Ant Colony Optimization

The configuration for ACO is:

- Evaporation is $\rho = 0.95$
- Three ants deposit pheromones out of 50 ants per iteration
- A single elitist solution is kept
- $\alpha = \beta = 1$ for the TSP and $\alpha = 1$, $\beta = 2$ for the scheduling problem.

In both cases, all the choices correspond to standard recommendation for default parameters, global to the algorithm in general and TSP-specific in the case of the operators, to avoid having to tune the parallel algorithms individually.

Chapter 4

Validation on TSP

Contents

Figures

4.1 Problem Description

The Travelling Salesperson Problem (TSP)[1] is a classical, well-studied, NP-Complete combinatorial optimization problem that is frequently used to benchmark optimization algorithms, introduced in [GP78]. In its basic form, it consists on finding an optimal (*i.e.* shortest length) tour of a given number of cities, passing each exactly once.

In time, several variations were developed requiring an open or closed loop, introducing asymmetric distances (going from city i to city j is a different distance than going from city j to city i), time windows, etc.

More formally, the TSP corresponds to finding a minimum-weight Hamiltonian path in a fully-connected weighted graph where the nodes represent cities and the edges' weights represent the distances. The search space therefore is the set of permutations of cities, leading to an $O(n!)$ complexity in the number of cities. This clearly prevents any brute-force enumeration approach for any but the smallest instances.

Multiple approaches have been proposed, mainly heuristics, such as *k-opt*, metaheuristics, and, more recently, spectral methods. The problem remains open as there is no single method that consistently outperforms the others.

TSPLIB is a repository of TSP instances and additional information such as upper and lower solution bounds encompassing a wide range of problem sizes and difficulties [Rei91]. This work uses the set of Asymmetric TSP (ATSP) instances to evaluate the algorithms.

[1] Originally named *Travelling Salesman Problem*, it has shifted towards the new name in the last years.

4.2 Procedure

Listing 4.1 shows, in pseudocode, the operation of the Multiverse Genetic Algorithm for TSP. The condition `Collector` is true for the collector node in Multiverse mode; therefore, the code for Multistart is exactly the same, except for the two lines under this condition, as it never happens. This pseudocode corresponds to the nodes that run the GA instances; there is an additional node, the controller, that launches the execution of the instances, keeps the synchronization between them, tracks and consolidates the best solutions, and in the Multiverse case communicates the best solutions of the other instances to the collector.

Lines 1–6 initialize the population with random tours and calculate the costs associated to them, informing the controller. Inside the loop, lines 8–14 generate the new population by including directly the elite individuals, selecting the ones that will act as parents, and applying the crossover and mutation operators on them. Lines 15–17 add the solutions from other instances in the collector. Finally, lines 18–23 trim the solutions down to the population size discarding the ones with worse fitness, update the best solution if needed, and send the current best solution to the controller.

The `SelectParents` function selects a fraction of individuals from the population; this fraction is given by the crossover probability parameter. The selection is random, but the probability for each individual to be selected is inversely proportional to its cost. It deterministically includes the best individual as an elitist strategy. The crossover operator is a variation of order crossover: it randomly selects a subtour of one parent and places it

Listing 4.1: Pseudocode for Multiverse and Multistart Genetic Algorithm for TSP

```
Population = EmptyList
for i = 1 to PopSize:
    Population ← RadomPermutation(Cities)
Costs = EvaluateSolutions(Population)
BestSol, BestCost = min(Population, Costs)
SendToController(BestSol, BestCost)
while not TerminationCondition:
    Parents = SelectParents(Population, Costs)
    Offspring = EmptyList
    Offspring ← ApplyCrossover(Parents)
    Offspring ← ApplyMutation(Parents)
    NewPopulation = EmptyList
    NewPopulation ← EliteSolutions(Population, Costs)
    NewPopulation ← Offspring
    if Collector:
        ReceiveFromController(MoreSolutions)
        NewPopulation ← MoreSolutions
    Population = CutToPopSize(NewPopulation, Costs)
    Costs = EvaluateSolutions(Population)
    BestSolIter, BestCostIter = min(Population, Costs)
    if BestCostIter < BestCost:
        BestSol, BestCost = BestSolIter, BestCostIter
    SendToController(BestSol, BestCost)
```

in the same position in the offspring; the rest of the tour is filled up with the remaining cities in the order that they appear in the other parent. The `ApplyCrossover` function repeatedly draws two individuals from the parents group and generates two offspring with them (inverting which is used as first parent). The selection at this stage is again biased by the inverse of the cost, so that better solutions are selected more often. The mutation operator switches the position of two randomly selected cities. The `ApplyMutation` function goes through all the individuals selected as parents and applies the mutation operator with probability given by the mutation probability parameter. Each time it does apply it, it adds a new individual to the population. `EliteSolutions` includes the best individuals from the original population in the new one; the number of individuals to transfer in this way is given by the elitism parameter.

Listing 4.2 shows the pseudocode for the Multiverse ACO for solving a TSP, which closely mirrors the procedure outlined above for GA. The code is the same for the Multistart version, as the only difference is the two lines under the condition `Collector`, which is true only for the collector process in the Multiverse version. As in the case for GA, this pseudocode corresponds to the nodes that run the ACO instances and there is an additional controller node.

Lines 1–3 initialize the algorithm by setting the initial pheromone value and creating the heuristic matrix from the distances between nodes. The main loop, lines 4–20, runs until the stopping criterion is met; each pass of the loop is one iteration in the algorithm, where it generates a new solution for each ant (lines 5–10; the evaluation of solutions is actually performed

Listing 4.2: Pseudocode for Multiverse and Multistart Ant Colony Optimization for TSP

```
PheromMatrix ← InitialValue
HeurMatrix ← DistanceMatrix
Population = EmptyPopulation
while not TerminationCondition:
  for i = 1 to NumAnts:
    Solution ← EmptySolution
    while CitiesLeft:
      Solution ← GetNext(CitiesLeft, PheromMatrix, HeurMatrix)
    Population ← Solution
  Costs = EvaluateSolutions(Population)
  BestSolIter, BestCostIter = min(Population, Costs)
  if BestCostIter < BestCost:
    BestSol, BestCost = BestSolIter, BestCostIter
  PheromMatrix ← Evaporate(PheromMatrix)
  PheromMatrix ← PheromUpdate(BestSol, BestCost)
  SendToController(BestSol, BestCost)
  Population = EmptyPopulation
  if Collector:
    ReceiveFromController(MoreSolutions)
    Population ← MoreSolutions
```

at the time of construction), updates the current best solution if needed (lines 11–13), updates the pheromone matrix (lines 14–15), and resets the population (line 17). Lines 16, 19 and 20 manage communication with the control node.

In each iteration, each of the `NumAnts` ants builds a tour by iteratively selecting the next city to go to. The function `GetNext` does this selection by assigning a probability to choose each of the cities not yet visited calculated as:

$$\Pr(c, c') = \frac{\tau_{cc'}^{\alpha} \eta_{cc'}^{\beta}}{\sum_{k \in C} \tau_{ck}^{\alpha} \eta_{ck}^{\beta}}, \tag{4.1}$$

where c is the current city in the tour, c' is the candidate city for the next step, C is the set of all potential candidate cities for the next step, including c'; τ_{ij} is the pheromone level associated to including going from city i to city j in the tour; η_{ij} is the heuristic associated to including going from city i to city j in the tour, namely the inverse of the distance from i to j; and α and β are algorithm parameters.

The update of the pheromone matrix consists of the evaporation step, calculated as $\tau_{ij} = \rho \tau_{ij}$, with the non-negative evaporation parameter $\rho < 1$, and the pheromone addition by the best ant along the tour it built, T:

$$\tau_{ij} = \tau_{ij} + \frac{1}{\sum_{i,j \in T} d_{ij}} \quad \forall i, j \in T, \tag{4.2}$$

where d_{ij} is the distance from i to j.

4.3 Results

The following analysis takes all 18 Asymmetric Travelling Salesperson Problem instances in the TSPLIB benchmark library, and runs both the Multiverse and Multistart variants of each GA and ACO on them. Due to the probabilistic nature of the algorithms, it repeats each run 25 times, tracking running time and the best solution found at each iteration. It also records the random seed generated in each run so that the exact same results can be reproduced.

Tables 4.1 and 4.2 show the outcome of the analysis for the Genetic Algorithm and the Ant Colony Optimization, respectively. The fields are:

Instance The file containing the problem data.

ΔAvg The percent difference between the average across repetitions of final solution fitness for Multistart and Multiverse.

ΔMin The percent difference between the best solution fitness across repetitions for Multistart and Multiverse.

RTime The percent difference between the averages across repetitions of CPU time elapsed for Multistart and Multiverse.

WTime The percent difference between the averages across repetitions of wall time elapsed for Multistart and Multiverse.

ΔHV The percent difference between the averages across repetitions of the hypervolume Multistart and Multiverse.

Table 4.1: Results from running Multistart and Multiverse Genetic Algorithm on TSPLIB instances.

Instance	**Δ Avg**	**Δ Min**	**RTime**	**WTime**	**Δ HV**
ftv33.atsp	−4.66	−2.94	−12.3	−10.1	−3.75
rbg403.atsp	−0.67	−0.844	+4.48	−5.54	−0.628
p43.atsp	−0.173	−0.283	−10.9	−9.15	−0.207
rbg443.atsp	−0.668	−0.184	+6.3	+6.19	−0.638
ftv47.atsp	−1.49	+3.34	−10.7	−9.27	−1.95
ft70.atsp	−0.723	−0.143	−7.69	−7.06	−1.16
br17.atsp	−0.102	+0.0	−12.1	−8.51	−0.292
ftv170.atsp	−0.563	−0.304	−3.96	−3.79	−0.593
rbg358.atsp	−1.42	−1.51	+3.27	+3.27	−0.934
ftv44.atsp	−0.988	+0.274	−9.76	−8.28	−1.57
ftv55.atsp	−0.497	−4.71	−9.11	−8.04	−2.25
ftv38.atsp	−3.88	−5.44	−10.9	−9.15	−3.12
ftv35.atsp	−3.67	−3.54	−11.0	−8.68	−3.28
kro124p.atsp	−1.01	−1.78	−5.91	−5.68	−0.969
ftv64.atsp	−0.672	−2.53	−8.17	−7.48	−1.8
ftv70.atsp	−0.399	−3.82	−8.35	−7.67	−0.11
ft53.atsp	−0.733	−1.43	−10.3	−9.08	−1.69
ry48p.atsp	−2.87	−5.96	−10.1	−8.81	−2.82
rbg323.atsp	−1.02	−0.808	+3.14	+3.13	−0.671

Table 4.2: Results from running Multistart and Multiverse Ant Colony optimization on TSPLIB instances.

Instance	**Δ Avg**	**Δ Min**	**RTime**	**WTime**	**Δ HV**
ftv33.atsp	+0.0	+0.0	−14.3	−5.92	−0.66
rbg403.atsp	−0.0565	−0.121	+1.48	+1.46	−0.237
p43.atsp	−0.037	−0.0178	−0.572	−0.252	−0.00103
rbg443.atsp	−0.109	−0.183	−0.995	−0.954	−0.3
ftv47.atsp	−0.156	+0.0	−0.649	−0.345	−0.119
ft70.atsp	−0.211	−0.234	−2.48	−1.92	−0.0868
br17.atsp	+0.0	+0.0	−1.42	−0.157	+0.00483
ftv170.atsp	−0.932	−0.525	−2.71	−2.67	−0.394
rbg358.atsp	−0.0201	−0.252	+0.596	+0.661	−0.309
ftv44.atsp	−0.052	+0.0	−1.11	−0.564	−0.282
ftv55.atsp	−0.354	+0.0	−1.88	−1.2	−0.447
ftv38.atsp	−0.0366	+0.0	−0.558	−0.225	−0.0138
ftv35.atsp	−0.0895	+0.0	−11.3	−4.89	−0.383
kro124p.atsp	−0.373	−0.966	−2.0	−1.84	−0.244
ftv64.atsp	−0.11	+0.0	−1.14	−0.821	−0.611
ftv70.atsp	−0.364	−0.102	−4.9	−3.97	−0.367
ft53.atsp	−0.642	−0.882	+1.8	+1.13	−0.0366
ry48p.atsp	−0.255	−0.504	−5.56	−6.84	−0.306
rbg323.atsp	+0.0237	−0.0742	−3.21	−3.24	−0.125

All differences are calculated as the value for Multiverse minus value for Multistart; since all the measured items are better the lower they become, this makes negative values favourable to Multiverse. The base for the percentage is the value for Multistart for the instance.

Hypervolume is a measure used to compare solutions to multi-objective problems. In such problems, the outcome is given as a Pareto front of non-dominated solutions, that is, solutions such that there is no other solution that improves on one of the objectives without getting worse on another objective. These solutions are also known as efficient solutions. Pérez Cáceres et al. [PLS14] apply the hypervolume measure to assess anytime performance of ACO algorithms, by considering the evolution of best solution found so far at each iteration as a bi-objective problem which aims at minimizing both the cost and the time to reach a given solution quality. This work includes the hypervolume in the same way as a measure of anytime performance.

Tables 4.1 and 4.2 show that the final solution quality is better in average using Multiverse, and that the best solution across runs for most of the instances also is attained by Multiverse. Elapsed times (CPU time and wall time) alternate between positive and negative, which is consistent with negligible overhead of Multiverse with respect to Multistart, as the random effect of synchronization is dominant; otherwise the time differences should be more consistently positive if the additional communication had a significant impact. Finally, hypervolume is in average also consistently better for Multiverse.

Instance `br17` is rather small (17 cities) and always yielded the globally optimal solution except in one run of Multistart. The analyses below do

not include this instance, as it behaves as an outlier, posing problems for normalization and application of statistical tests. At this size, there is no practical difference between the two methods, but also little incentive for parallelization. For the ACO, instance `ftv33` was also problematic for the same reason in the analysis of solution quality, but had enough variability to be included in the analysis of hypervolume.

Figure 4.1 shows a boxplot of the best costs achieved in the 25 runs of each instance for Multistart and Multiverse GA, providing a deeper view of the ΔAvg and ΔMin columns in Table 4.1. Since the values for each instance are very different from each other, the values are normalized as

$$x_{\text{norm}} = \frac{x - \bar{x}}{\bar{x} - x_{\text{min}}}, \tag{4.3}$$

where x_{norm} is the normalized value for x, $\bar{x}$ is the average value for Multistart, and x_{min} is the minimum value for Multistart. This performs a translation and scaling of both the Multistart and Multiverse values so that the resulting boxes are comparable and fit well in the graph, while maintaining the relative shapes and positions unmodified within each instance. This figure corroborates the typically improved performance of the Multiverse method (with labels ending in `.mv`) over the Multistart method (with labels ending in `.ms`). There are some instances, such as `ftv55` and `ftv64` where the high variability of solutions using Multiverse makes it difficult to reach a conclusion. Running a directed Mann Whitney U test at $\alpha = 0.05$ for each instance, supports the hypothesis that there is a statistically significant difference in solution quality for 10 out of 17 instances in favour of Multiverse, and no

Table 4.3: Instances for which the Mann Whitney U test supports that there is a significant difference between Multiverse and Multistart Genetic Algorithm in favour of Multiverse (MV) or no difference (EQ). There is no instance so that there is a difference in favour of Multistart.

MV	ftv33, rbg403, p43, rbg443, ftv47, ft70, rbg358, ftv38, ftv35, ry48p.
EQ	ftv170, ftv44, ftv55, kro124p, ftv64, ftv70, ft53.

Table 4.4: Instances for which the Mann Whitney U test supports that there is a significant difference in solution quality between Multiverse and Multistart Ant Colony Optimization in favour of Multiverse (MV) or no difference (EQ). There is no instance so that there is a difference in favour of Multistart.

MV	rbg403, p43, rbg443, ft70, ftv170, ftv44, ftv55, ftv38, ftv70, ft53, ry48p.
EQ	ftv47, rbg358, ftv35, kro124p, ftv64, rbg323.

significant difference for the other 7. Table 4.3 shows which instances pass the test.

Figure 4.2 is the equivalent of Figure 4.1 for ACO. The findings are similar, with 11 out of 17 significantly improving according to the directed Mann Whitney U test, as shown in Table 4.4.

Figures 4.3 and 4.4 are the equivalent of Figures 4.1 and 4.2, respectively, but for hypervolume instead of cost achieved. The analysis is similar, but for hypervolume the improvement obtained by Multiverse is more evident. The application of a directed Mann Whitney U test at $\alpha = 0.05$ for each instance, supports the hypothesis that there is a statistical significant difference in solution quality for Genetic Algorithm in 15 out of 17 instances in favour of Multiverse, and no significant difference for the other two: ftv170 and

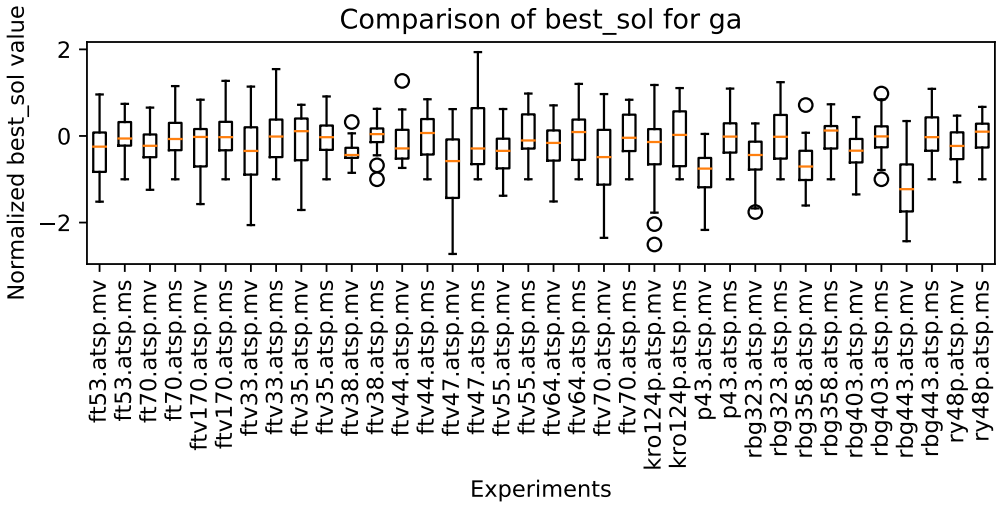

Figure 4.1: Boxplot of best cost achieved across the 25 runs for each instance problem using Multistart and Multiverse Genetic Algorithm. Costs are normalized to the corresponding average using Multistart so that they are comparable.

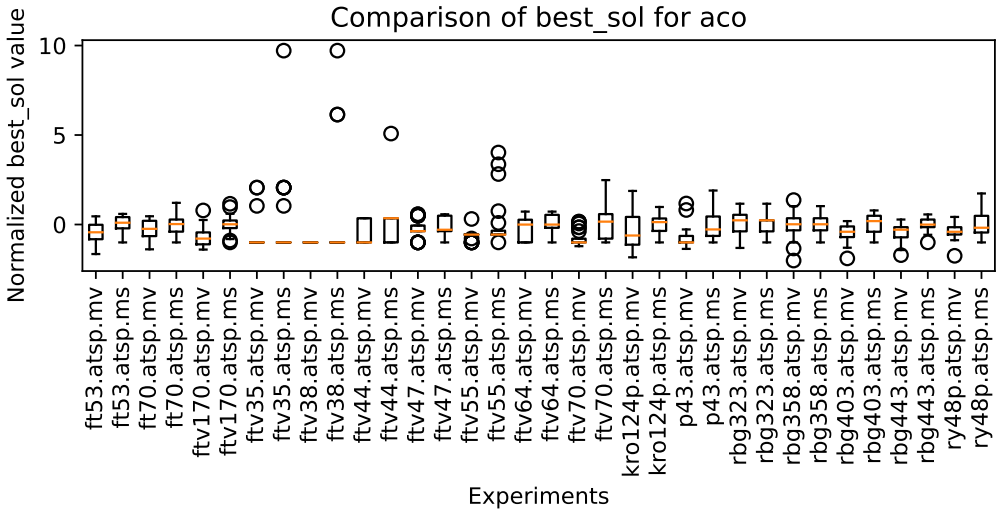

Figure 4.2: Boxplot of best cost achieved across the 25 runs for each instance problem using Multistart and Multiverse Ant Colony Optimization. Costs are normalized to the corresponding average using Multistart so that they are comparable.

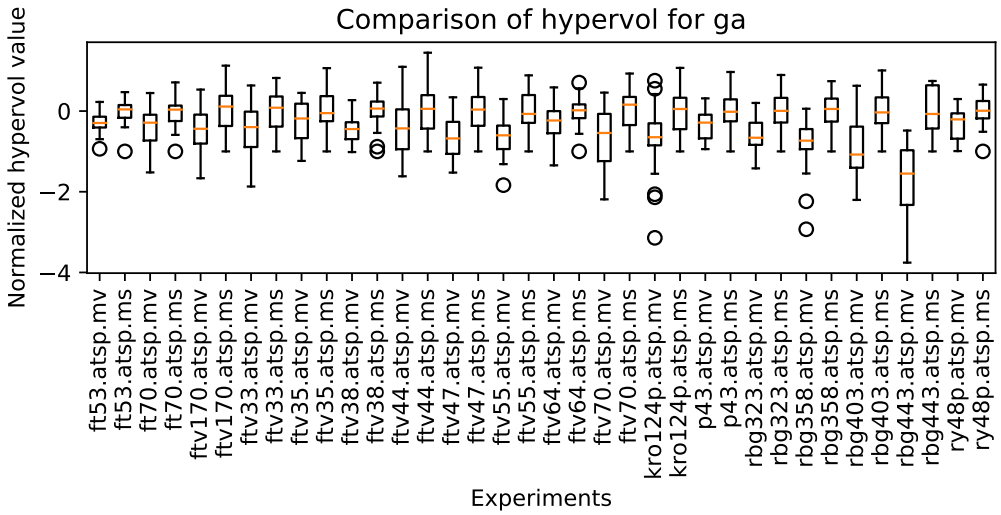

Figure 4.3: Boxplot of hypervolume across the 25 runs for each instance problem using Multistart and Multiverse Genetic Algorithm methods. Hypervolumes are normalized to the corresponding average using Multistart so that they are comparable.

`ftv70`. For Ant Colony Optimization, the advantage is found in 11 out of 18 instances (see Table 4.5); larger instances benefit the most from the Multiverse approach.

Figures 4.5 and 4.6 show the typical graph of best cost so far versus iteration number for GA and ACO, respectively. The lines represent the average of best solution achieved in each run by the corresponding iteration

Table 4.5: Instances for which the Mann Whitney U test supports that there is a significant difference in hypervolume between Multiverse and Multistart Ant Colony Optimization in favour of Multiverse (MV) or no difference (EQ). There is no instance so that there is a difference in favour of Multistart.

MV	`ftv33`, `rbg403`, `rbg443`, `ft70`, `ftv170`, `rbg358`, `ftv44`, `ftv55`, `ftv64`, `ftv70`, `rbg323`.
EQ	`p43`, `ftv47`, `ftv38`, `ftv35`, `kro124p`, `ft53`, `ry48p`.

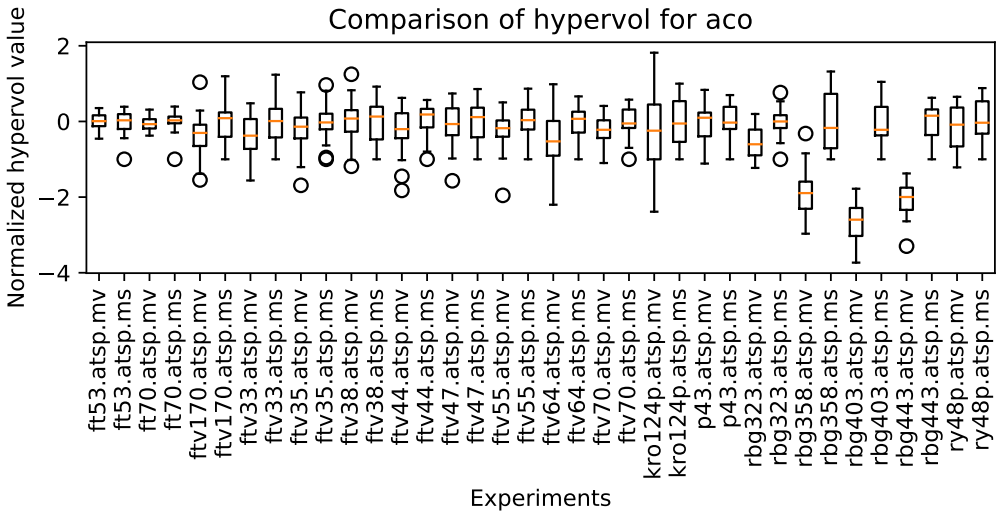

Figure 4.4: Boxplot of hypervolume across the 25 runs for each instance problem using Multistart and Multiverse Ant Colony Optimization methods. Hypervolumes are normalized to the corresponding average using Multistart so that they are comparable.

for Multiverse and Multistart respectively. The examples selected show a clear gap, and others are similar with varying gap sizes.

The evolution graphs for all TSP GA and ACO instances are shown in Appendix A and Appendix B, respectively.

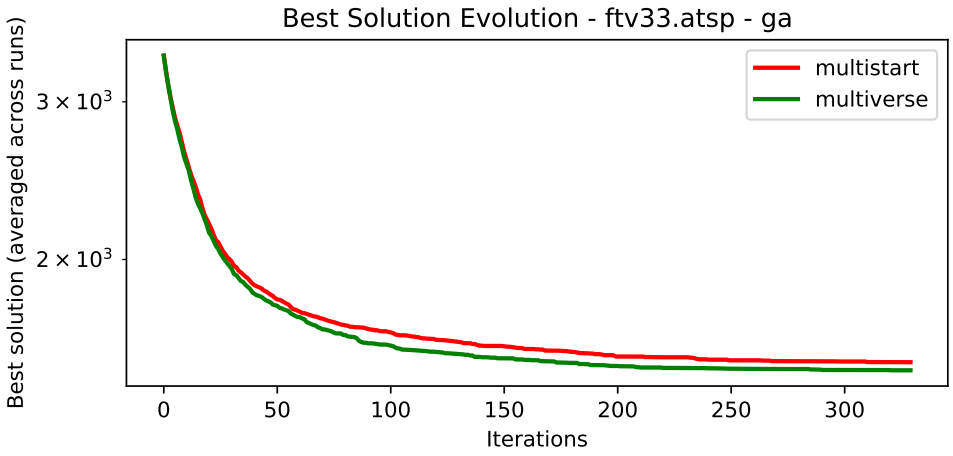

Figure 4.5: Typical evolution of best Genetic Algorithm solution versus iteration for Multistart and Multiverse. Each line follows the average of the best solutions across the 25 runs at the corresponding iteration for their respective method. The Y axis is shown in logarithmic scale.

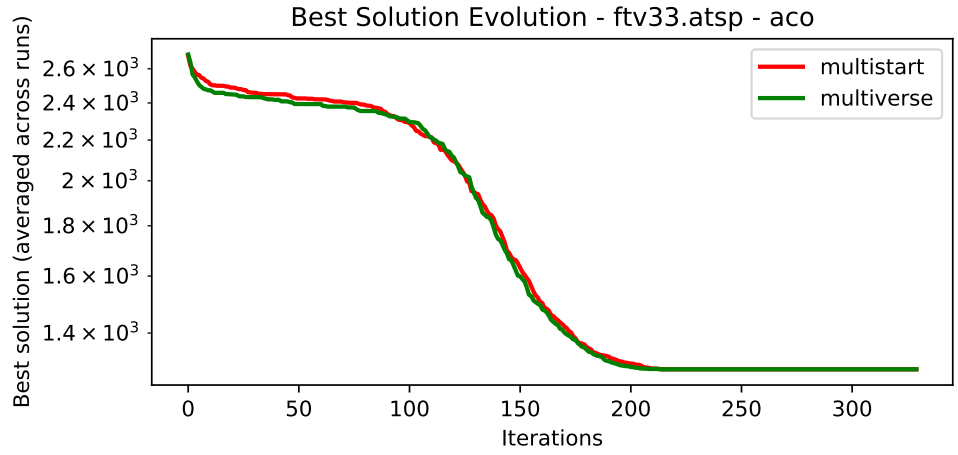

Figure 4.6: Typical evolution of best Ant Colony Optimization solution versus iteration for Multistart and Multiverse. Each line follows the average of the best solutions across the 25 runs at the corresponding iteration for their respective method. The Y axis is shown in logarithmic scale.

Chapter 5

Analysis of Results

Contents

Figures

Tables

5.1 Problem Description

The production of steel is a very complex process, with several functions involved in its transformation from coal and iron ore: iron making (conversion of iron ore into liquid iron), steelmaking (conversion of liquid iron into liquid steel), casting (solidification of liquid steel into semi-products: billets or slabs) and finally rolling, aimed at transforming the intermediate products into the format accepted by the client (normally coils of steel, bars, heavy plates, wire rod, etc.) to continue the transformation into cars, bridges, beverage cans, and many other products.

One of the most important products manufactured by steelmakers is galvanized steel. The main use of this steel is as a raw material in other industries, and especially in the automotive industry for building car bodies. The process of galvanizing consists in covering the steel with a zinc layer to protect it against corrosion.

The facility in charge of galvanizing steel is the galvanizing line. In this process, steel is coated with a zinc layer, which has the purpose of protecting the steel against air and moisture. In fact, the zinc layer is considered to be the most effective and low-cost means to achieve this goal. Galvanization is applied to steel coils, which are a finished steel product which has been wound or coiled after rolling slabs. Slabs are semi-finished steel products, obtained by processing through a continuous caster and cut into various lengths. The slab has a rectangular cross section and is used as a starting material in the production process of flat products, *i.e.* hot rolled coils or plates.

The process of galvanizing is continuous, that is, there is no separation

between the coils and the line never stops. The head of the next coil is welded to the tail of the coil in process. Thus, for the line, the input is an infinite strip which has points where there is a change in some of its characteristics: dimensions (width, thickness), steel grade, section, thickness of the zinc layer, etc.

After the welding area, the strip advances towards the accumulator, a kind of buffer that makes it possible to change the speed of the line depending on the needs of the furnace without running out of material. The furnace is the next stage; here the strip has to reach a target temperature depending on its chemical composition (steel grade), and for this line speed and furnace temperature have to be adjusted according to the thickness and width of the coil. Thicker and wider coils require lower speed or higher furnace temperature to reach the same target temperature. Due to the inertia of the system, changing these parameters takes some time, during which part of the coil may not be processed properly, depending on how different the coils and their targets are.

After the furnace, the strip is entered into the zinc pot, a bath of molten zinc at a temperature of around 460° Celsius, and later by passing through an 'air knives' system, the zinc layer is spread evenly and to the thickness specified by the client; this is critical in order to avoid coil rejections.

All this process is dramatically affected by the sequencing. Depending on it, it is possible to lose many of meters of strip due to a lack of quality, or even worse, to have a breakage that would halt the facility for several hours up to a full day. Every time the strip breaks in the furnace where they are heated before the zinc bath, the line must be stopped until the furnace cools

down, in order to remove the steel strip of the furnace; then the furnace must be heated up again before resuming production.

Sequencing is critical in production lines in the steel industry (as it is in the industry in general). Depending on the sorting, the sequence could have different impact in the process, and it can cause quality issues in the final product, lower productivity, higher energy consumption or even an incident in the facility.

Due to the technical limitations of the lines and the critical efficiency necessary to offer nowadays for each productive plant, the necessity of having a scheduling model able to solve the limitations of current scheduling algorithms has arisen. The objective is to maximize productivity, making the facility more competitive.

Like the TSP, this problem is a search in an $O(n!)$ complexity space, as it needs to explore the permutations of coils. The difference stems from the existence of constraints that exclude some of the permutations, and the fact that the cost function and some of the constraints depend on longer subsequences: whereas in the TSP the cost is equal to the sum of weights of the edges that form the Hamiltonian path, in this scheduling problem the cost or constraints may be different for the same transition from one coil to another depending on the previous coils in the sequence.

Traditionally, techniques based on constraint programming have been used for these purposes. These algorithms are focused on finding a solution with some properties, mainly fulfilling some relations among variables and respecting a set of constraints. Their main problem is their limitation for using complex cost functions, which prevents the exploration of the whole

search space. A more optimization-oriented approach is presented in [Fer+14] using ACO, which is the base for the parallelized algorithm presented below.

The benchmark instances consist of batches produced in a galvanizing line, and run the gamut in terms of size and complexity. Line experts made the selection to be representative of typical operating conditions, in order to ensure that the results are meaningful.

See chapter 1 for a more detailed description of the steelmaking process in general, and the galvanizing line and the sequencing problem in particular.

5.2 Procedure

In the TSP, the objective is to generate the shortest tour of a number of cities, given the distance matrix between each pair. The sequencing problem shows similarities with this specification, substituting cities for steel coils and distances for costs. Confidentiality limits the level of detail given below.

Since some of the orderings are not possible, some of the costs will be infinite (or, viewing the problem as a graph, the corresponding arcs do not exist). This is one of the two main differences with a TSP. One main goal is the reduction of these infeasible transitions, because every time there is one in a sequence, it is necessary to include a transition coil (a coil with no client) to solve the issue. This transition coil has a high cost (compared to a normal production cost) in terms of productivity (it requires producing something unneeded instead of the outstanding orderbook) and in terms of cost (yield), because maybe the coil will be sold as second quality or even used as scrap (lower value).

For over a year, the technical experts of the lines defined the losses associated to the transitions, analysing all the parameters and characteristics of the coils that have some kind of impact on the resulting meters of strip lost for a given sequence. These cost functions depend on the parameters of the coils and their differences in the transitions. The losses are mainly generated by width changes between consecutive coils, thickness changes, thermal losses, etc.

The general idea is to try and minimize the transitions for each parameter relevant to sequencing, looking for a smoother evolution and therefore a more stable production.

The input to the problem consists on the campaign type, and the features of the coils to be sequenced. The campaign type is relevant because some of the rules only apply to specific campaigns, and the values of the parameters in constraints and cost functions may be different per campaign type.

The coil features include:

ID A coil identification code, used for tracking

Width (w_i) The width of the coil in millimetres

Thickness (e_i) The thickness of the coil in millimetres

Weight (m_i) The weight of the coil in tons

Product type The steel grade of the coil

Target zinc coating (z_i) The amount of coating to apply, in grams per square metre

Line speed (v_i) The speed of the line, in metres per minute, needed to properly process the coil

Target temperature (t_i) The ideal temperature for the coil in the thermal cycle, in degrees Celsius

Temperature range (t_i^-, t_i^+) The minimum and maximum temperature admissible for the thermal cycle, in degrees Celsius. Obviously $t \in [t_i^-, t_i^+]$.

Most constraints are transition constraints, they depend only on two adjacent coils in the sequence: i and $i+1$. These are:

$$w_{i+1} - w_i \leq \overline{\Delta w^+}, \tag{5.1}$$

$$w_i - w_{i+1} \leq \overline{\Delta w^-}, \tag{5.2}$$

$$e_{i+1} - e_i \leq \overline{\Delta e^+}, \tag{5.3}$$

$$e_i - e_{i+1} \leq \overline{\Delta e^-}, \tag{5.4}$$

$$|v_{i+1} - v_i| \leq \overline{\Delta v}, \tag{5.5}$$

$$[t_i^-, t_i^+] \cap [t_{i+1}^-, t_{i+1}^+] \neq \emptyset. \tag{5.6}$$

In words, the width and thickness can only change so much from one coil to the next, and the limit may be different going up or down; these limits stem from the ability to safely weld the coils together. The change in line speed is also limited; this is related to the operational limitations of the line. And the temperature ranges of the coils must overlap, so that there is a valid operating point while the furnace contains part of both strips.

Yet another transition constraint is expressed as a compatibility table which forbids certain product type combinations to be adjacent in the se-

quence. This summarizes multiple issues such as weldability difficulties derived from the materials chemistry or the need to make sudden changes in operation setpoints of machines in the line.

In specific campaigns, a 'wide-to-narrow' constraint forbids any increasing in width along the sequence, although there is some little wiggle room and isolated increases below a certain threshold are allowed. This is a sequence constraint rather than a transition constraint.

Also in specific campaigns, a certain product type must be included in the sequence at regular time intervals, to allow certain maintenance operations needed to ensure the quality required in those campaigns.

One final sequence constraint prevents width increases after more that a given volume of material, in tons, has been processed without a change in width. There is a threshold, in millimetres, determining the required width difference to consider that the width has changed. This requirement comes from quality considerations: prolonged rolling at the same width may cause the edges to carve a groove on the cylinders, which will transfer it to a wider strip or affect the edges of subsequent strips of the same width.

The cost function adds up several components. Most of them are transition costs: the cost associated to changes in thickness (C_e), the cost associated to changes in zinc coating (C_z), the cost associated to changes in line speed (C_v), and the cost associated to changes in target temperature (C_t); the last one is calculated based on the target temperature of the coils t_i, which should not be confused with the bounds of the admissible temperature range, as there is already a constraint that forces the ranges to overlap.

These costs are formulated as follows:

$$C_e = K_e \left| e_{i+1} - e_i \right|, \tag{5.7}$$

$$C_z = K_z \left| z_{i+1} - z_i \right|, \tag{5.8}$$

$$C_v = \begin{cases} K_v \left| v_{i+1} - v_i \right| & \text{if } \left| v_{i+1} - v_i \right| \geq \overline{\Delta v} \\ 0 & \text{otherwise,} \end{cases} \tag{5.9}$$

$$C_t = \begin{cases} K_t \left| t_{i+1} - t_i \right| & \text{if } \left| t_{i+1} - t_i \right| \geq \overline{\Delta t} \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

Besides these transition costs, there is a sequence cost, C_w, associated to the risk of quality defects with consecutive coils of the same width. This cost only applies when widening out, *i.e.* when the width is increasing for that part of the sequence, and uses a custom proprietary function to calculate the probability of having a defect for a given number of consecutive coils of the same width. This probability is then multiplied by a constant representing the typical cost of a defect.

The total cost is simply the sum of all these components:

$$C = C_e + C_z + C_v + C_t + C_w. \tag{5.11}$$

Initially, before any pheromone has been laid out, the transition costs, *i.e.* the costs that only depend on two consecutive coils, function as a heuristic for solution construction. Before launching the algorithm, the transition cost for each combination of nodes $\sigma(n_1, n_2)$ is precalculated; this cost can be

classified as: infinite (if the transition does not respect some of the constraints of the line), zero or finite (there are losses associated to the transition, but it respects all the constraints of the line).

Given the existence of zero and infinite costs it is not possible to apply directly the usual selection criterion of selecting the next node with probability inversely proportional to the arc cost. Instead, a two-step selection chooses first among the three categories (if present in the node), biasing against infinite costs; and in the second step, if zero or infinite cost arcs have been selected, all of them are equiprobable, whereas for finite cost arcs the usual approach (probability inversely proportional to cost) applies.

Infeasible transitions have to be accepted because not doing so would require each ant to build a Hamiltonian path. This is a hard problem in itself, and in the problem there is often no such path. A second option would be to accept them only if no feasible transition is left; this leads to much worse sequences, because it tends to leave out the least connected nodes, accumulating large numbers of infeasible transitions at the end of the sequence. As a mitigation strategy, infinite cost transitions are allowed, with low probability, at any time to enable the model to discover infeasible transitions that save more of them later.

These infeasible solutions are "solved" by the operators of the line by means of inserting transition coils between two coils that are not compatible from the production point of view. This *transition coil* is especially expensive, as described above, and this is the reason why it is considered as an infinite cost. Therefore, the priority of the algorithm will always be to minimize first these infeasible transitions and only then minimizing the total cost.

The function to deposit the pheromone in the path is the standard one, shown in (5.12), where τ_{ij} is the amount of pheromone between nodes i and j, *BestCost* the cost of the best solution found up to the moment of the update and ρ the evaporation factor:

$$\tau_{ij}^{k} \leftarrow \begin{cases} (1-\rho)\tau_{ij}^{k} + \frac{BestCost}{CostAnt_k} & \text{if arc } ij \in Ant_k \\ (1-\rho)\tau_{ij}^{k} & \text{otherwise} \end{cases} \tag{5.12}$$

With the advance of the iterations, the heuristic based on transition cost loses weight in favour of the pheromone. Heuristic and pheromone information are combined following the usual formula:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \quad \forall j \in \mathcal{N}_i^k, \tag{5.13}$$

where:

p_{ij}^{k}**:** is the probability of selecting node j after selecting i for ant k,

τ_{ij}**:** is the amount of pheromone between nodes i and j,

η_{ij}**:** is the heuristic value calculated as $\frac{1}{Cij}$,

C_{ij}**:** represents the meters of strip lost in transition between nodes i and j,

α**:** is a parameter to control the influence of the pheromone,

β**:** is a parameter to control the influence of heuristic based on losses, and

$\mathcal{N}_i^k$**:** is the set of accessible nodes for ant k from node i that have not been already selected.

The combination of the parameters α and β controls the influence of the pheromone and transition costs over the decision of choosing a path. In the tests,the parameters are set to $\alpha = 1$ and $\beta = 2$, with the objective of having a good balance between the heuristic and experience (pheromone).

Elitism allows only the top 10% ants in each iteration to deposit pheromone in the matrix.

5.3 Results

The following analysis replicates the analysis performed for the Asymmetric Travelling Salesperson Problem above to the 46 instances of the scheduling problem, running both the Multiverse and Multistart variants on them. As in the previous case, there are 25 runs of each instance and method, tracking running time and the best solution found at each iteration. It also records the random seed generated in each run so that the exact same results can be reproduced.

Table 5.1 shows the outcome of the analysis for the scheduling problem. The fields are once again:

Instance The file containing the problem data.

ΔAvg The percent difference between the average across repetitions of final solution fitness for Multistart and Multiverse.

ΔMin The percent difference between the best solution fitness across repetitions for Multistart and Multiverse.

RTime The percent difference between the averages across repetitions of CPU time elapsed for Multistart and Multiverse.

WTime The percent difference between the averages across repetitions of wall time elapsed for Multistart and Multiverse.

ΔHV The percent difference between the averages across repetitions of the hypervolume Multistart and Multiverse.

Table 5.1: Results from running Multistart and Multiverse Scheduling Algorithm on TSPLIB instances.

Instance	Δ Avg	Δ Min	RTime	WTime	Δ HV
10_coils_30	-0.00125	-0.00427	+8.97	+2.22	-0.00213
10_coils_60	-3.98	-7.8	+4.47	+2.12	-0.433
10_coils_90	-5.74	+22.0	-1.31	-0.801	-3.51
11_coils_30	+0.000166	-0.00217	-7.49	-2.09	+0.286
11_coils_60	-1.32	-1.13	+4.22	+2.15	-0.454
11_coils_90	-0.298	-0.322	+0.283	+0.26	-0.0929
12_coils_30	-0.871	-0.0397	-3.97	-1.1	-0.53
12_coils_60	-1.24	-30.5	+0.598	-1.97	-0.448
12_coils_90	+0.227	-0.077	-0.626	-0.386	+0.324
13_coils_30	+0.145	-0.129	+0.855	+0.242	+0.0792
13_coils_60	-0.0624	-0.0554	-1.9	-2.44	-0.135
13_coils_90	-0.532	-4.85	-0.601	-0.368	-0.865

Continued on next page

Table 5.1 – *Continued from previous page*

Instance	Δ Avg	Δ Min	RTime	WTime	Δ HV
14_coils_30	-0.000866	-0.0397	+8.75	+2.79	-0.0577
14_coils_60	-0.042	-0.0088	-1.69	-0.894	+6.29e-05
14_coils_90	+0.641	+1.17	-1.63	-1.13	-0.66
15_coils_30	-1.97e-05	+0.000493	+1.09	-0.446	-0.00146
15_coils_60	-0.014	-0.00714	+3.5	+1.98	+65.4
15_coils_90	-0.985	-1.16	-2.41	-1.76	-1.06
1_coils_27	-0.0559	-0.000287	+2.06	+0.791	-0.0705
1_coils_30	-5.41e-05	+0.0	+0.502	+0.251	-0.0178
1_coils_60	-0.000582	-0.00637	-0.561	+0.674	-0.00874
1_coils_90	-3.33	-1.53	-0.0756	-0.062	-2.87
2_coils_30	-1.55	-0.0266	-1.7	-1.66	-1.95
2_coils_60	-9.96	-11.4	+1.62	+1.36	-8.93
2_coils_90	-0.565	+0.0579	+0.874	+1.29	-0.288
3_coils_30	-2.48	-1.39	-5.98	-2.79	-2.17
3_coils_60	-0.00288	-0.0028	+1.28	+1.09	-0.01
3_coils_90	-0.000589	-0.000982	-0.458	-0.413	-0.000639
4_coils_30	-0.0118	+0.0	-3.17	-1.44	-0.0649
4_coils_60	-0.0587	-0.124	-1.47	-1.21	-0.039
4_coils_90	-0.203	-0.68	+0.019	+0.0269	-0.243
5_coils_30	-4.68	-2.22	-1.68	-0.789	-3.74
5_coils_60	-0.603	-1.21	-1.79	-1.48	-0.489
5_coils_90	-0.0524	-0.156	+1.41	+1.34	-0.108

Continued on next page

Table 5.1 – *Continued from previous page*

Instance	Δ Avg	Δ Min	RTime	WTime	Δ HV
6_coils_30	-0.000164	+0.0	-0.84	-0.377	-0.0349
6_coils_60	+5.91e-05	-0.000246	+0.111	+0.77	-6.27e-05
6_coils_90	-3.32	-2.99	+0.0151	+0.0263	-2.5
7_coils_30	-0.00699	-0.00176	+0.612	+0.3	-0.0095
7_coils_60	-8.14	-12.8	+1.69	+1.42	-6.48
7_coils_90	-0.0507	-0.0389	+0.185	+0.185	-0.0593
8_coils_30	-0.000124	+0.0	-3.19	-2.71	-0.0439
8_coils_60	-5.08	-2.28	+1.65	+1.4	-4.7
8_coils_90	-0.272	-0.598	-0.343	-0.31	-0.257
9_coils_30	-0.000568	+0.0	+2.65	+1.21	-0.00377
9_coils_60	-0.00377	-0.00868	+0.874	+0.742	+1.88
9_coils_90	-0.262	-1.27	+0.489	+0.856	-0.246

All differences are calculated as the value for Multiverse minus value for Multistart; since all the measured items are better the lower they become, this makes negative values favourable to Multiverse. The base for the percentage is the value for Multistart for the instance.

Hypervolume is a measure used to compare solutions to multi-objective problems, used here as a measure of anytime performance. See section 4.3 for the rationale behind this metric.

Table 5.1 shows that the final solution quality is most frequently better in average using Multiverse, and that the best solution across runs for most of the instances also is attained almost always by Multiverse; the instances

where this does not hold have very small differences. Elapsed times (CPU time and wall time) alternate between positive and negative, which is consistent with negligible overhead of Multiverse with respect to Multistart, as the random effect of synchronization is dominant; otherwise the time differences should be more consistently positive if the additional communication had a significant impact. Finally, hypervolume is in average also consistently better for Multiverse, with the exception of `15_coils_60`, which can be shown to be an artifact due to an outlying run (see Table 5.1 and Figure C.17 in Appendix C).

Figure 5.1 shows a boxplot of the best costs achieved in the 25 runs of each instance for Multistart and Multiverse GA, providing a deeper view of the ΔAvg and ΔMin columns in Table 5.1. Since the values for each instance are very different from each other, the values are normalized as

$$x_{\text{norm}} = \frac{x - \overline{x}}{\overline{x} - x_{\min}}, \tag{5.14}$$

where x_{norm} is the normalized value for x, $\overline{x}$ is the average value for Multistart, and $x_{\min}$ is the minimum value for Multistart. This performs a translation and scaling of both the Multistart and Multiverse values so that the resulting boxes are comparable and fit well in the graph, while maintaining the relative shapes and positions unmodified within each instance. This figure corroborates the typically improved performance of the Multiverse method (with labels ending in `.mv`) over the Multistart method (with labels ending in `.ms`).

The results for this case are more variable than for the TSP. This is

related to the characteristics of the problem, and specifically to the increased difficulty in finding feasible solutions besides improving sequence cost. This translates into wider ranges in the boxplot and fewer instances where the advantage is as clear as it was for the TSP.

Another issue related to this is the appearance of outliers due to very high costs used to represent infeasible solutions. These have been filtered out (only for the figure, not for the results table and the statistical analysis) to afford a better view of the boxplots, which would otherwise be compressed to a few pixels. Values outside the interval $[-10, 10]$ after normalization were removed from the plots.

Running a directed Mann Whitney U test at $\alpha = 0.05$ for each instance, supports the hypothesis that there is a statistically significant difference in solution quality for roughly two-thirds of the instances in favour of Multiverse, and no significant difference for the other one-third. Table 5.2 shows which instances pass the test.

Figure 5.2 is the equivalent of Figure 5.1, but for hypervolume instead of cost achieved. The analysis is similar, and the same outlier filtering is applied to obtain a clear view of the bulk of the data. The application of a directed Mann Whitney U test at $\alpha = 0.05$ for each instance, supports the hypothesis that there is a statistical significant difference in solution quality for the Scheduling Algorithm again for about two-thirds of the instances, and no significant difference for the other one-third (see Table 5.3).

Figure 5.3 shows the typical graph of best cost so far versus iteration number. The lines represent the average of best solution achieved in each run by the corresponding iteration for Multiverse and Multistart respectively.

Table 5.2: Instances for which the Mann Whitney U test supports that there is a significant difference between Multiverse and Multistart Scheduling Algorithm in favour of Multiverse (MV) or no difference (EQ). There is no instance so that there is a difference in favour of Multistart.

MV	10_coils_30, 10_coils_60, 10_coils_90, 11_coils_60, 11_coils_90, 12_coils_30, 13_coils_60, 14_coils_30, 14_coils_60, 15_coils_90, 1_coils_27, 1_coils_30, 1_coils_90, 2_coils_30, 2_coils_60, 3_coils_30, 3_coils_60, 3_coils_90, 4_coils_30, 5_coils_30, 5_coils_60, 6_coils_30, 6_coils_90, 7_coils_30, 7_coils_60, 7_coils_90, 8_coils_60, 9_coils_30, 9_coils_60.
EQ	11_coils_30, 12_coils_60, 12_coils_90, 13_coils_30, 13_coils_90, 14_coils_90, 15_coils_30, 15_coils_60, 1_coils_60, 2_coils_90, 4_coils_60, 4_coils_90, 5_coils_90, 6_coils_60, 8_coils_30, 8_coils_90, 9_coils_90.

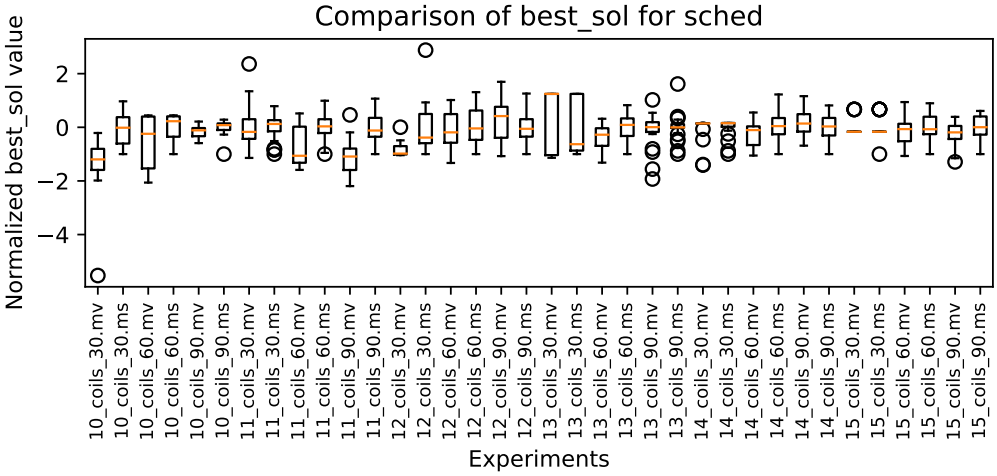

Figure 5.1: Boxplot of best cost achieved across the 25 runs for each instance problem using Multistart and Multiverse Scheduling Algorithm. Costs are normalized to the corresponding average using Multistart so that they are comparable.

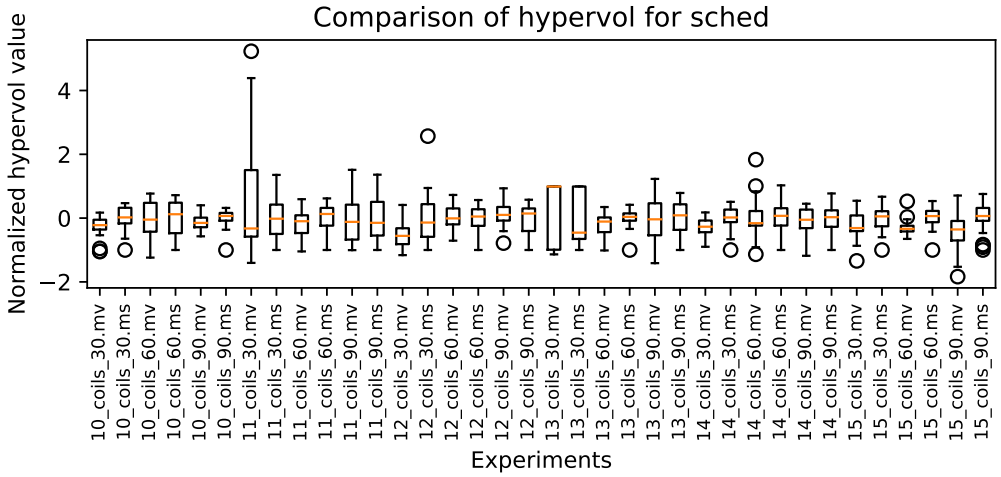

Figure 5.2: Boxplot of hypervolume across the 25 runs for each instance problem using Multistart and Multiverse Scheduling Algorithm methods. Hypervolumes are normalized to the corresponding average using Multistart so that they are comparable.

Table 5.3: Instances for which the Mann Whitney U test supports that there is a significant difference in hypervolume between Multiverse and Multistart Scheduling Algorithm in favour of Multiverse (MV) or no difference (EQ). There is no instance so that there is a difference in favour of Multistart.

MV	10_coils_30, 10_coils_90, 11_coils_60, 12_coils_30, 13_coils_60, 14_coils_30, 15_coils_30, 15_coils_60, 15_coils_90, 1_coils_27, 1_coils_30, 1_coils_90, 2_coils_30, 2_coils_60, 3_coils_30, 3_coils_60, 3_coils_90, 4_coils_30, 5_coils_30, 5_coils_60, 6_coils_30, 6_coils_90, 7_coils_30, 7_coils_60, 7_coils_90, 8_coils_30, 8_coils_60.
EQ	10_coils_60, 11_coils_30, 11_coils_90, 12_coils_60, 12_coils_90, 13_coils_30, 13_coils_90, 14_coils_60, 14_coils_90, 1_coils_60, 2_coils_90, 4_coils_60, 4_coils_90, 5_coils_90, 6_coils_60, 8_coils_90, 9_coils_30, 9_coils_60, 9_coils_90.

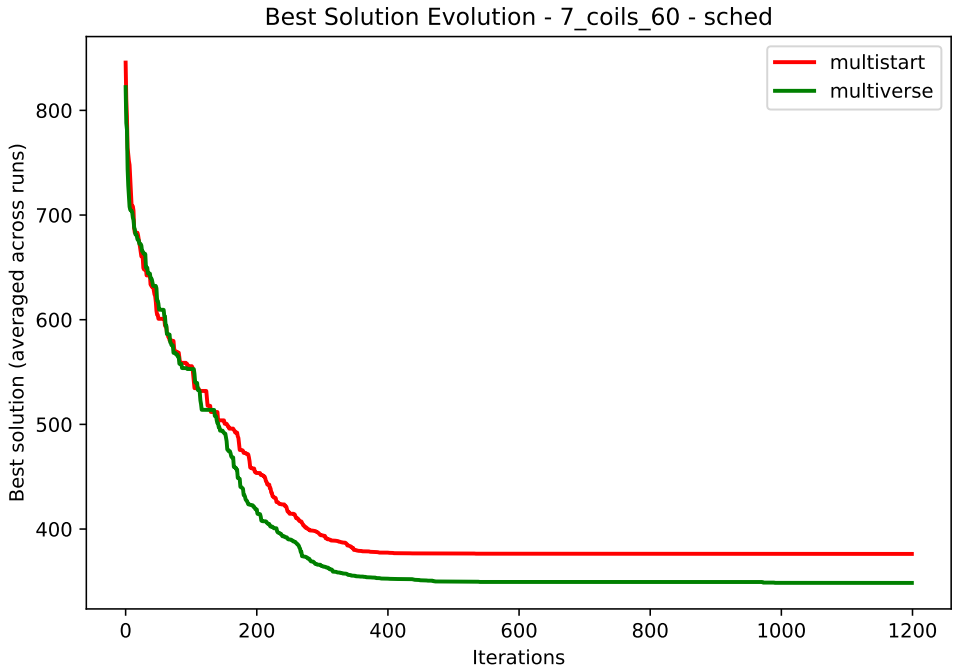

Figure 5.3: Typical evolution of best Scheduling Algorithm solution versus iteration for Multistart and Multiverse. Each line follows the average of the best solutions across the 25 runs at the corresponding iteration for their respective method.

The example selected shows a clear gap, and others are similar with varying gap sizes, and given the higher variability in the solutions to this problem the lines cross over one another in some instances. In particular instance `13_coils_30` goes the opposite direction with the Multistart average outperforming the Multiverse average throughout, but the range of results is similar for both approaches in terms of best solution and hypervolume.

The evolution graphs for all Scheduling Algorithm instances are shown in Appendix C.

Chapter 6

Conclusions and Future Work

Contents

6.1 Conclusions

The multiple approaches to the parallelization of metaheuristics span different trade-offs of solution quality, algorithm complexity, communication overhead, and generalization capability.

Multiple Independent Runs, or Multistart, sits at one end of the trade-off spectrum: fully generic, with minimum complexity and overhead, and is typically considered the baseline against which to compare other parallel methods. This work introduces the Multiverse method, first published in [Día+20], as an alternative to multiple independent runs.

Both the Multistart and Multiverse methods are fully generic. Indeed, they can be considered meta-algorithms, as they can be applied to a metaheuristic to generate a new, parallel algorithm. Multistart has a wider range of application, as it can take both single-solution and population-based metaheuristics; Multiverse still is very general, being applicable to any population-based metaheuristic. Compared to Multistart, the Multiverse method incurs negligible overhead both in terms of added complexity and communication.

In order to test these claims, and to evaluate its performance in solution quality with respect to Multistart, this thesis describes concrete instantiations of the method for solving the Travelling Salesperson Problem using a Genetic Algorithm and Ant Colony Optimization as representatives of the two broad classes of population-based metaheuristics. The tests are run on the asymmetric TSP instances in the benchmark library TSPLIB.

The Multiverse variant is superior or equal in solution quality at the end of the run than the Multistart approach. The statistical tests support this

superiority in 10 out of 18 instances in the case of Genetic Algorithms and in 11 out of 18 in the case of Ant Colony Optimization. In the remaining 8 or 7 cases, respectively, the solution quality is equivalent for Multiverse and Multistart. The largest instances, especially those with over 400 cities, tend to benefit more.

Looking at anytime behaviour, or the ability to reach better solutions faster, in terms of hypervolume, the situation is similar for Ant Colony Optimization, where the Multiverse variant shows statistically significant improvement in 11 out of 18 problem instances; for Genetic Algorithms, the benefits of the Multiverse method are much greater, extending to 16 out of the 18 instances tested.

The added overhead to reap these benefits is lost within the intrinsic variability in running time of the problems, with both Multiverse and Multistart being slightly faster on some of the instances. The averages differ by just tens of seconds to a few seconds for total execution times ranging from around 30 seconds to 4–5 minutes (or more in some special cases), or a couple of percentage points at the most, depending on the instance.

Looking at the galvanizing line scheduling problem, which originated the need for the algorithm, similarly positive results are obtained. The industrial scheduling problem is solved for 46 representative instances using a specially adapted ACO.

As in the case of the TSP, the results using Multiverse for the scheduling problem show significant improvements both in solution quality and anytime behaviour. Out of the 46 available instances tested, 29 yield a statistically significant improvement in solution quality and 27 succeed likewise in terms of

hypervolume. For the remaining 17 or 19 respective instances, the differences are not statistically significant.

This in spite of a much higher variability in solution quality across runs for this problem due to the difficulties associated with finding feasible solutions in a complex search space with such a limited number o function evaluations.

As before, the differences in running times alternate in favour of one or the other method, always with absolute values that represent a small fraction of the total time, as is expected of a negligible added complexity and overhead.

These results support the Multiverse method as a potential candidate to replace multiple independent runs as the baseline for distributed algorithm comparison, and a promising initial approach to extending the galvanizing line scheduling problem to larger, more complex problems; this in turn enables addressing longer scheduling horizons or more complex cases, where feasible solutions are harder to find, all while respecting the stringent constraints on running time set by the operational needs of the industrial application.

Even though the improvements are relatively small, reaching 2%–3% in the best cases, and typically below 1%, the volumes involved result in meaningful returns. As a back-of-the-envelope calculation, assume a galvanizing line that produces 500 Kt/year. A 0.5% improvement in productivity adds 2.5 Kt, which translates to approximately $2.5M additional revenue at current prices at the time of writing[1]. That's a potential for over $750M increased revenue across the worldwide production if extrapolated to the more

[1]Taking China galvanized sheet spot price as a representative index: on 2021-11-03 the price per ton is 6,495 RMB or 1,015.51 USD.

than 150 Mt/year of galvanized steel produced globally.

While confidentiality prevents giving more details on the evaluation of the gains using the approach, the example above shows how apparently tiny improvements explode when the huge volumes of steel production are taken into account.

Furthermore, the same scheduling optimization approach can be extended to other production lines by adapting the cost and constraint functions. This should map quite directly to all finishing lines, albeit with some effort in modelling specifics of each line. For primary lines, the steelshop and the hot strip mill, the additional constraints and coupled behaviour would require new developments to tackle their scheduling. The increased scope, and therefore problem size, makes the parallelization abilities developed in this work even more relevant for that case.

6.2 Future Work

The field of metaheuristics is very broad, has a near-infinite range of applications, and intersects with other fields like multiobjective optimization or Machine Learning[2].

This opens up multiple options to advance from this point of parallelization of metaheuristics towards enabling solving larger and/or more complex problems. Below are some interesting avenues to extend the research presented in this thesis.

[2]Both in the use of metaheuristics as optimization tools for hyperparametrization of Machine Learning Models and in the use of Machine Learning to build surrogate models for use as an integral part of metaheuristic algorithms.

6.2.1 Algorithm Frameworks

One possibility is to move from the ad-hoc implementations of the algorithms used for the tests detailed above towards generic ones, ready to use by the research community. The most straightforward way to attain this is to integrate the algorithm into widely used frameworks for the development of metaheuristics.

A more ambitious goal to set would be developing a new framework focusing on the parallelization aspect as the central idea, instead of one of the supporting features. This would provide a tool to develop novel algorithms designed for parallelization from the ground up, rather than taking an existing sequential algorithm and adapting it to run concurrently.

There are already some algorithms designed following this principle, but the literature shows no general conceptual framework for it, like there is for the basic metaheuristic algorithms. Undoubtedly, such a conceptual framework would go hand-in-hand with the more practical development framework.

One such parallel-centric feature could be the use of asynchronous computation. This is not completely novel, but it is a very underrepresented option. Usually, the different execution threads advance more or less in lockstep, and explicit synchronization points are set, *e.g.* at the end of an iteration. This can, with some adaptations of the algorithms, be avoided, and instead allow a more dynamic environment that more closely resembles the real-life inspiration for most metaheuristics.

For instance, in Ant Colony Optimization, ants would not synchronize in iterations for a global pheromone update; each ant would build its solution,

determine whether pheromone should be deposited, and do the update, which can change the pheromone while another ant is building it own solution.

The effects of this type of features on solution quality and convergence have not yet been studied.

6.2.2 Non-Traditional Computing Architectures

With a few exceptions[3], metaheuristics are implemented using imperative or object-oriented programming languages on 'classical' computers.

Other options could be explored to assess potential benefits that could be gained at the moment, or to understand the potential improvements and new capabilities of future architectures.

Functional programming languages, due to their very design, enforce data immutability and encapsulation of behaviour, which matches perfectly with the needs of parallelization. Metaheuristics implemented using functional languages would parallelize very naturally, and options like Erlang, which runs on a distributed virtual machine, would make the distinction of whether the algorithm is run locally or distributed irrelevant.

Similarly, some languages are designed specifically for distributed systems, and they establish a crisp separation of the program logic and the distributed hardware mapping.

These options could be tried out to see if the new perspective on the algorithms provides advantages. Potentially they could improve upon current implementations because of their dedicated features, or at least make it easier

[3] *E.g.* there are some hardware implementations of Ant Colony Optimization using FPGAs.

to build distributed algorithms. But also they force a different way to look at the algorithms, which may lead to novel ideas.

Although the classical Von Neumann architecture has dominated computing for over half a century, new architectures are on the horizon.

Quantum Computing is the obvious one. Even if practical applications (and the exact technologies) are still years to decades away, the main principles and proof-of-concept hardware are available now. Quantum Computing will be very relevant to the field of metaheuristics, as important acceleration and scalability is expected in combinatorial optimization, where metaheuristics now dominate; but it is still to be seen whether Quantum Computing can be harnessed to run accelerated, improved, or completely new metaheuristic algorithms.

In-memory computing, while less known than Quantum Computing is also much closer to practical equipment that can be applied for real-life problems. The main difference between classical and in-memory computing is that the latter tries to break the communication bottleneck created by the need to move data to the CPU for computation by adopting a memory-centric architecture with processors distributed across the 'memory fabric'. Oversimplifying, it takes the computation to the data instead of taking the data to the computation.

Like Quantum Computing, being a new architecture, the design of algorithms has to be reworked. In-memory computing is closer to traditional algorithm design, and many of the skills translate quite directly; Quantum Computing, on the other hand, requires a completely different skill set, more related to quantum mechanics than computer science. Understanding the

implications of developing metaheuristic algorithms in an in-memory architecture would be another potential research avenue.

6.2.3 Parallel Multiobjective Metaheuristics

This work has touched lightly on multiobjective optimization as an interpretation of the several conflicting desirable properties in an algorithm, and used the concept of Pareto front to identify an existing gap to address.

When dealing with multiple objectives, the move from one to several dimensions means that the absolute ordering of solution quality disappears, and solution quality needs to rely on the concept of dominance.

This automatically rules out analytical optimization approaches, and the field of multiobjective optimization is almost monopolized by metaheuristics, especially Evolutionary Algorithms. Although we can take some of the lessons learned and apply them to multiobjective metaheuristics, significant adaptations or new developments are needed. For instance, when selecting the 'best solution' to migrate, it is no longer possible to have an absolute best solution.

There is already some research on parallelization of multiobjective metaheuristics. The most common approach is using decomposition strategies that have each population focus on a limited region of the Pareto front.

Much more can be explored in this area. Additionally, there are practical problems that drive the need for such systems. One example, if we move upstream from the scheduling problem introduced in this work, is the joint scheduling of the steelshop and hot strip mill. These two lines, as described in

chapter 1, need to be coordinated to have a fluid operation of the whole plant, and their requirements are different enough that their respective objectives can be considered as antagonistic. Add to that the need to consider lead times to ensure proper service levels, and the problem is not only much larger and complex than the scheduling of the galvanizing line, but it also becomes multiobjective.

Having such a problem as a practical, real-world test-bed for parallel multiobjective metaheuristics is icing on the cake, making this option possibly the most enticing for further research.

Chapter 7

Conclusiones y trabajo futuro

Contents

7.1 Conclusiones

Esta gran variedad de aproximaciones a la paralelización de metaheurísticos proporciona diferentes opciones de combinación de calidad de solución, complejidad de los algoritmos, volumen añadido de comunicación, y capacidad de generalización.

Las múltiples ejecuciones independientes (Multi-inicio), se sitúa en uno de los extremos del espectro de equilibrio de factores, con complejidad y carga de comunicaciones mínimas, y se utiliza habitualmente como método base de referencia para comparar otros métodos. Este trabajo presenta el método Multiverso, publicado por primera vez en [Día+20], como alternativa a las múltiples ejecuciones independientes.

Tanto el método Multiverso como el Multi-inicio son completamente genéricos. De hecho, pueden considerarse meta-algoritmos, que se aplican a un algoritmo para obtener un nuevo algoritmo, paralelizado. El método Multi-inicio tiene un ámbito más amplio, ya que puede aplicarse a cualquier metaheurístico. El método Multiverso, aplicable directamente a cualquier metaheurístico basado en población, sigue teniendo un ámbito de aplicación importante. Comparado con Multi-inicio, el incremento tanto de carga de comunicaciones como de complejidad de Multiverso es despreciable.

Para comprobar estas afirmaciones, y para evaluar su comportamiento en términos de calidad de soluciones comparado con Multi-inicio, esta tesis describe instanciaciones concretas del método para resolver el problema del viajante (*Travelling Salesperson Problem*, TSP) utilizando algoritmos genéticos y optimización por colonia de hormigas como representantes de las dos

clases generales de metaheurísticos basados en población. Las pruebas se realizan sobre las instancias de problema del viajante asimétrico de la librería de pruebas estándar TSPLIB.

La variante Multiverso supera o iguala a la aproximacion Multi-inicio en calidad de solución final. Los tests estadísticos demuestran que esta mejora es estadísticamente significativa en 10 de las 18 instancias en el caso de los Algoritmos Genéticos y en 11 de las 18 en el caso de la Optimización por Colonia de Hormigas. En los 8 ó 7 casos restantes respectivamente, no existe diferencia significativa, es decir, la calidad es equivalente en Multiverso y Multi-inicio. Las instancias de mayor tamaño, especialmente las que tienen más de 400 ciudades, tienden a benficiarse más de la mejora.

En cuanto a la rápidez para alcanzar soluciones de mayor calidad, según la métrica de hipervolumen, la situación es similar para la Optimización por Colonia de Hormigas, donde la variante Multiverso mejora de forma estadísticamente significativa a Multi-inicio en 11 de 18 instancias; para los Algoritmos Genéticos, los beneficios del método Multiverso son aún mayores, mejorando en 16 de las 18 instancias probadas.

El incremento de tiempo de ejecución necesario para obtener estos beneficios es indistinguible de la variabilidad intrínseca del tiempo de ejecución de cada problema. Tanto Multi-inicio como Multiverso resultan la opción más rápida, por márgenes muy reducidos, en varias de las instancias. Las diferencias en las medias de tiempos de ejecución difieren por décimas de segundo, o unos pocos segundos en los problemas de mayor tamaño, y los tiempos totales de ejecución varían entre unos 30 segundos y 4–5 minutos (o más en algunos casos particulares); esto supone diferencias de uno o dos

puntos percentuales a lo sumo, según la instancia.

En el problema de programación de la línea de galvanizado, que origina la necesidad de desarrollar este algoritmo, se obtienen resultados similarmente positivos. El problema industrial de programación se resuleve para 46 instancias representativas, utilizando un modelo de Optimización por Colonia de Hormigas especialmente adaptado.

Como en el caso del TSP, los resultados de la variante Multiverso para el problema de programación muestran mejoras significativas tanto en calidad de solución final como en rapidez para alcanzar mejores soluciones. De las 46 instancias disponibles para las pruebas, 29 resultaron en una mejora estadísticamente significativa de la calidad de la solución final y 27 mejoran similarmente en la métrica de hipervolumen. En las 17 ó 19 instancias restantes respectivamente, las diferencias no son significativas.

Estos resultados se obtienen a pesar de la variabilidad mucho mayor en calidad de solución entre ejecuciones de la misma instancia, debida a las dificultades asociadas a encontrar soluciones factibles en un espacio de búsqueda complejo con un número de evaluaciones de las funciones tan limitado.

Como en el caso anterior, las diferencias en tiempos de ejecución alternan a favor de uno u otro método, siempre con valores absolutos que representan una pequeña fracción del tiempo de ejecución total, como es de esperar si la complejidad y comunicación añadidas son despreciables.

Estos resultados apoyan la hipótesis de que el método Multiverso es un buen candidato para sustituir a Multi-inicio como modelo base para la comparación de métodos distribuidos, y una aproximación inicial prometedora para extender el problema de programación de la Línea de Galvanizado a

instancias de mayor tamaño y complejidad, en las que resulta más difícil encontrar soluciones factibles. Todo ello, respetando las duras limitaciones de tiempo de ejecución impuestas por las necesidades operativas de la aplicación industrial.

Aunque las mejoras son relativamente pequeñas, llegando al 2%–3% en los mejores casos, y manteniendose típicamente por debajo del 1%, los retornos totales son significativos, dado el volumen de material procesado. En términos aproximados, suponiendo una línea que produzca 500 Kt/año, con una mejora del 0,5% en la productividad añade 2,5 Kt, lo que se traduce en aproximadamente $2,5M en ingresos adicionales con los precios actuales en el momento de escribir estas conclusiones[1]. Esto supone un potencial de más de $750M de incremento de ingresos a nivel de la producción mundial si se extrapola este cálculo a las más de 150 Mt/año de acero galvanizado producidas globalmente.

Si bien por motivos de confidencialidad no es posible proporcionar más detalles sobre la evaluación de las ganacias derivadas de esta aproximación, este ejemplo muestra claramente cómo mejoras aparentemente pequeñas explotan al tener en cuenta los ingentes volúmenes de producción mundial de acero.

Adicionalmente, la misma aproximación de optimización de la programación se puede extender a otras líneas de producción adaptando las funciones de coste y restricciones. En el caso de otras líneas acabadoras, la adaptación debería ser relativamente directa, por las similitudes en el fun-

[1]Considerando el precio *spot* en China de bobina galvanizada como índice representativo, a 2021-11-03 el precio por tonelada es 6.495 RMB o 1.015,51 USD.

cionamiento, aunque requerirían esfuerzos para modelar los aspectos específicos de cada línea. Para las líneas de cabecera, acería y tren de bandas en caliente, las restricciones adicionales y su funcionamiento acoplado harían necesario desarrollar nuevos modelos para optimizar su programación. El ámbito de mayor amplitud, y por lo tanto la mayor complejidad y tamaño del problema, hacen que las capacidades de paralelización desarrolladas en este trabajo resulten aún más relevantes en ese caso.

7.2 Trabajo futuro

El campo de los metaheurísticos es muy amplio, con una variedad casi infinita de aplicaciones, y se solapa con otros campos como la optimización multiobjetivo o el aprendizaje automático.

Esto abre la puerta a múltiples opciones para continuar avanzando desde la paralelización de metaheurísticos hacia hacer posible resolver problemas más complejos y/o a mayor escala.

7.2.1 Plataformas de algoritmos

Una posibilidad es abandonar las implementaciones ad-hoc de los algoritmos utilizados en las pruebas realizadas en esta tesis y moverse hacia versiones más genéricas, listas para su uso por parte de la comunidad investigadora. La manera más directa de lograr esto es integrando el algoritmo en las plataformas de desarrollo de metaheurísticos de uso más común.

Una meta más ambiciosa sería la creación de una nueva plataforma, con la paralelización de algoritmos como idea central, en lugar de ser una fun-

cionalidad secundaria. Sería una herramienta para el desarrollo de nuevos algoritmos diseñados desde el inicio para la paralelización, en lugar de adaptar un modelo secuencial existente para ejecutarse de forma distribuida.

Ya existen algunos algoritmos diseñados siguiendo esta filosofía, pero no se ha identificado en la literatura ningún marco conceptual genérico para ello, como lo hay para los metaheurísticos 'clásicos'. Sin duda un marco conceptual así debería ir aparejado a una plataforma de este tipo.

Una característica específicamente asociada a la paralelización podría ser el uso de computación asíncrona. Esto no es completamente nuevo, pero es una opción extremadamente poco frecuente. Normalmente, los diferentes flujos de ejecución avanzan conjuntamente, y se establecen puntos de sincronización específicos, por ejemplo al final de cada iteración. Con ciertas adaptaciones en los algoritmos, se podría evitar esto y en su lugar permitir una evolución más dinámica, que además representaría más fielmente la inspiración biológica de muchos metaheurísticos.

Por ejemplo, en la optimización por colonia de hormigas, las hormigas no necesitarían sincronizarse por iteraciones para llevara a cabo una operación global de depósito de feromonas. En su lugar, cada hormiga construiría su solución, determinaría si debe depositar feromona, y en su caso haría el depósito. Esto podría cambiar los valores de las feromonas mientras otras hormigas están construyendo sus soluciones.

Los efectos de este tipo de características sobre la calidad de las soluciones y la convergencia de los algoritmos aún no han sido estudiados.

7.2.2 Arquitecturas de computación no convencionales

Salvo contadas excepciones, los metaheurísticos se implementan en ordenadores 'clásicos' utilizando lenguajes de programación imperativos u orientados a objetos.

Podrían explorarse otras opciones para evaluar posibles beneficios que podrían obtenerse en estos momentos, o para entender las potenciales mejoras y nuevas posibilidades de arquitecturas futuras.

Los lenguajes de programación funcionales, por diseño, obligan a utilizar datos inmutables y a aislar comportamientos, lo que encaja perfectamente con las necesidades de los algoritmos paralelos. Los metaheurísticos implementados con lenguajes de programción funcionales se adaptarían muy naturalmente a entornos distribuidos, y opciones como Erlang, que se ejecuta sobre una máquina virtual distribuida, pueden volver irrelevante la cuestión de si la ejecución es local o paralelizada.

Igualmente, algunos lenguajes se han diseñado específicamente para sistemas distribuidos, y separan claramente la lógica de programa y el mapeo a los equipos distribuidos.

Estas opciones podrían probarse para determinar si la nueva perspectiva sobre los algoritmos aporta ventajas. Potencialmente, podrían mejorarse las implementaciones actuales gracias a sus características específicas, o al menos facilitar el desarrollo de algoritmos distribuidos. Pero además, al forzar una manera diferente de considerar los algoritmos, podrían fomentar la aparición de nuevas ideas.

Aunque la arquitectura Von Neumann lleva más de medio siglo domi-

nando la computación, empiezan a vislumbrarse alternativas.

La más obvia es la computación cuántica. Si bien las aplicaciones prácticas (e incluso las tecnologías definitivas) aún tardarán años o décadas en llegar, los principios básicos y equipos a nivel de prueba de concepto ya existen. La computación cuántica será muy relevante para el mundo de los metaheurísticos, dadas la aceleración y la escalabilidad que se espera en el ámbito de la optimización combinatoria, que los metaheurísticos dominan hoy en día. Aún está por ver si la computación cuántica puede aprovecharse para ejecutar versiones aceleradas, mejoradas, o completamente nuevas de algoritmos metaheurísticos.

La computación en memoria, aunque es menos célebre que la computación cuántica, está más próxima a producir equipos y soluciones de aplicación práctica. La principal diferencia entre la computación clásica y la computación en memoria es que esta intenta superar el cuello de botella que supone la necesidad de mover datos a la CPU para realizar los cálculos; para ello, adoptan una arquitectura centrada en la memoria, con procesadores distribuidos por el 'tejido de memoria' (*memory fabric*). De forma muy simple, lleva la computación a los datos en lugar de los datos a la computación.

Al igual que en la computación cuántica, al ser una arquitectura nueva, la computación en memoria requiere rediseñar los algoritmos para sacarle el máximo provecho. La computación en memoria es más similar a la computación clásica, lo que hace más fácil transferir las habilidades de una a la otra. La computación cuantica, por el contrario, requiere unas competencias muy diferentes, más alineadas con la mecánica cuántica que con las ciencias de la computación. Entender las implicaciones de desarrollar algo-

ritmos metaheurísticos para estas arquitecturas sería otra posible línea de investigación.

7.2.3 Metaheurísticos multi-objetivo paralelos

Este trabajo ha mencionado la optimización multi-objetivo como la manera de interpretar múltiples requisitos opuestos en las propiedades de los algoritmos, usando el concepto de frente de Pareto para indentificar un área de interés sobre la que investigar.

Al tratar con varios objetivos, el cambio de una a varias dimensiones hace desaparecer la ordenación absoluta de las soluciones. Para comparar soluciones hay que recurrir al concepto de dominancia.

En estas condiciones, hay que descartar las aproximaciones analíticas, y la optimización multi-objetivo está prácticamente monopolizada por metaheurísticos, especialmente algoritmos evolutivos. Aunque se podrían trasladar algunas de las lecciones aprendidas para paralelizar estos métodos, harían falta importantes revisiones o incluso métodos completamente nuevos. Por ejemplo, al seleccionar la 'mejor solución' para migrar, ya no es posible tener una 'mejor solución' absoluta.

Ya se ha realizado algo de investigación en la paralelización de metaheurísticas multiobjetivo. La aproximación más habitual consiste en utilizar estrategias de descomposición por las que cada población se concentra en una región limitada del frente de Pareto.

Aún hay mucho más por explorar. Además, hay problemas prácticos que impulsan el desarrollo de este tipo de sistemas. Un ejemplo se encuentra

aguas arriba del problema de secuenciación que se presenta en este trabajo: la secuenciación conjunta de la acería y el tren de bandas en caliente. Estas dos líneas deben coordinarse para garantizar una operación fluida de toda la planta, y sus requisitos individuales son tan distintos que pueden considerarse antagónicos. Añadiendo a la mezcla la necesidad de tener en cuenta los tiempos de entrega para asegurar un nivel de servicio adecuado, el problema resulta mucho mayor y más complejo que la secuenciación de una línea de galvanizado, y además es multi-objetivo.

Tener un problema industrial realista como este como banco de pruebas para metaheurísticos multiobjetivo paralelos es el colofón para hacer que esta opción sea probablemente la más atractiva para investigación futura.

Appendices

Appendix A

Evolution Graphs for TSP GA

Figures

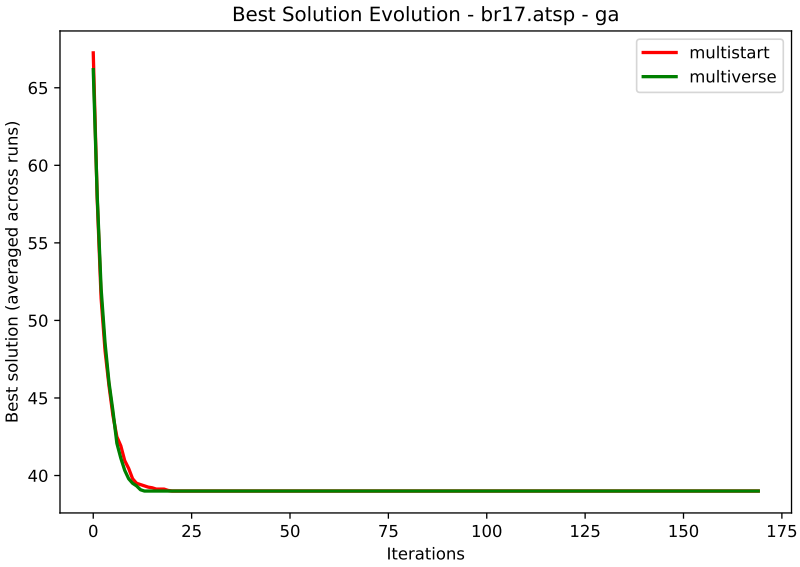

Figure A.1: Evolution of GA average best solution versus iteration for Multistart and Multiverse for br17

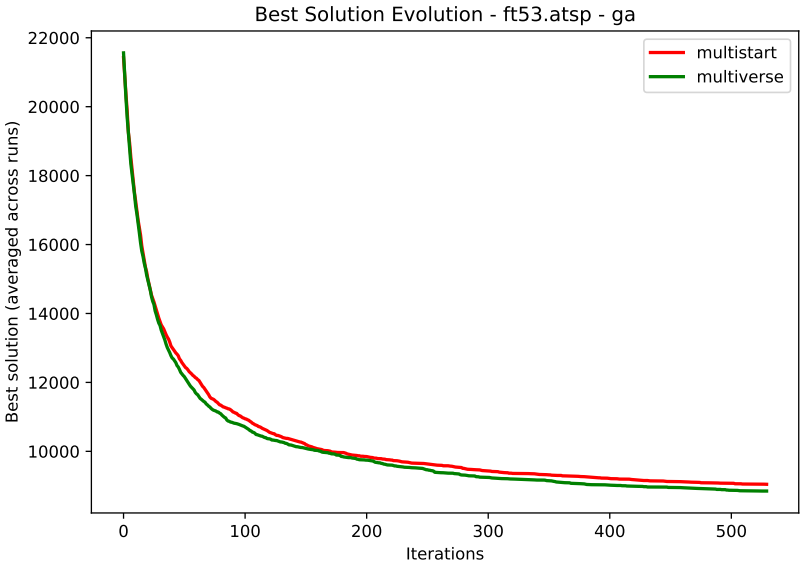

Figure A.2: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ft53

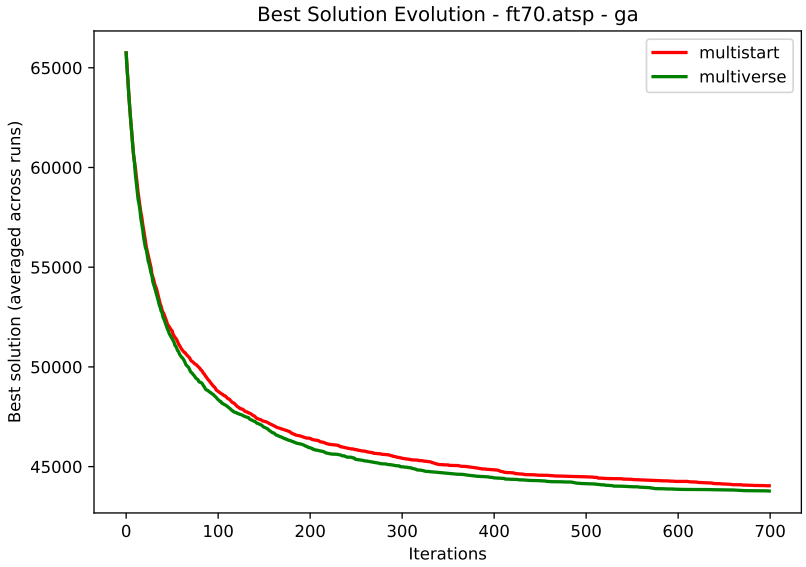

Figure A.3: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ft70

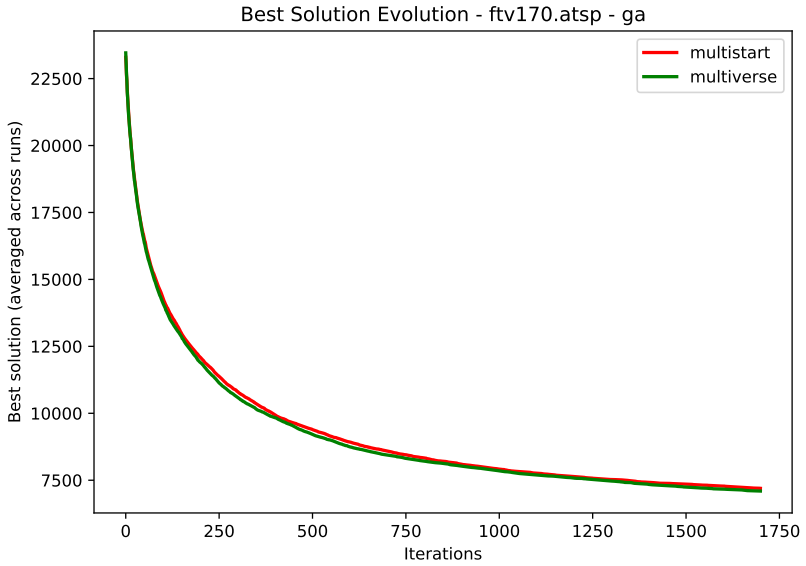

Figure A.4: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv170

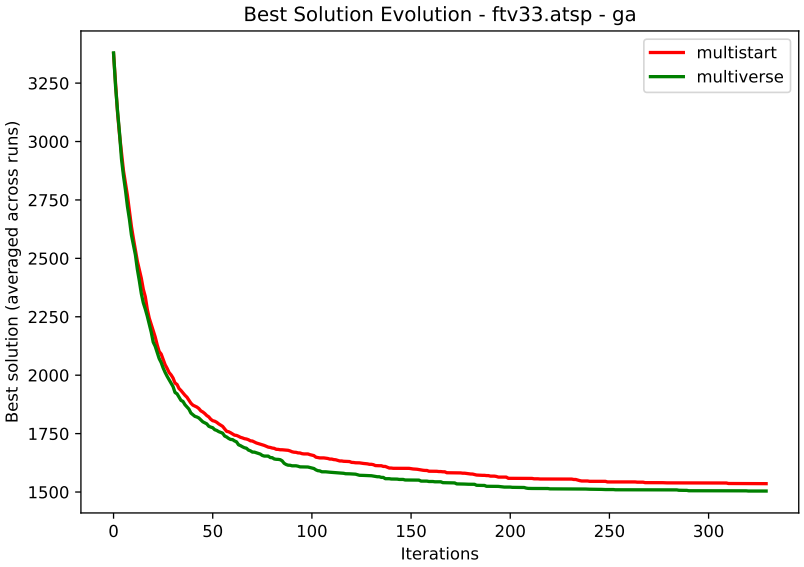

Figure A.5: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv33

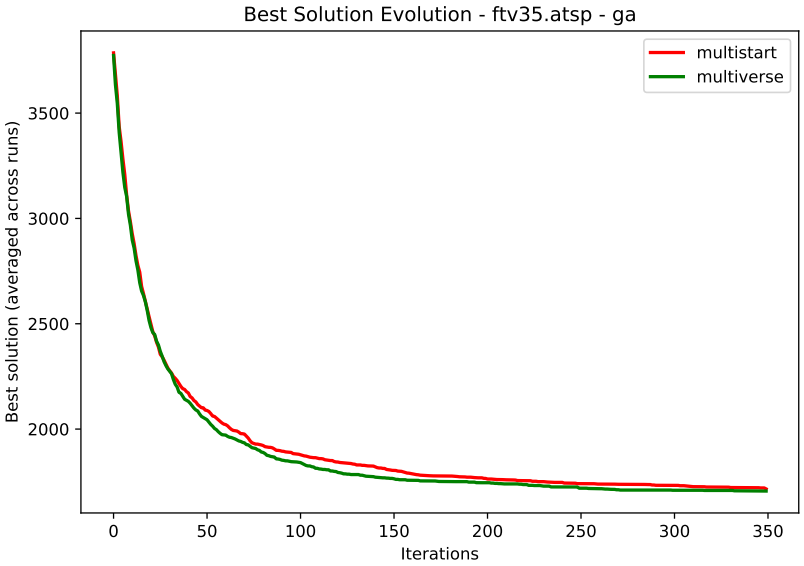

Figure A.6: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv35

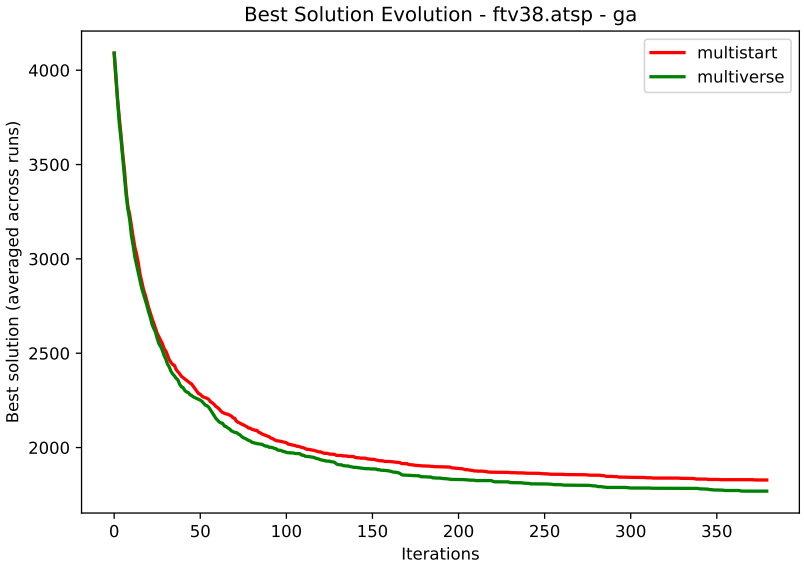

Figure A.7: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv38

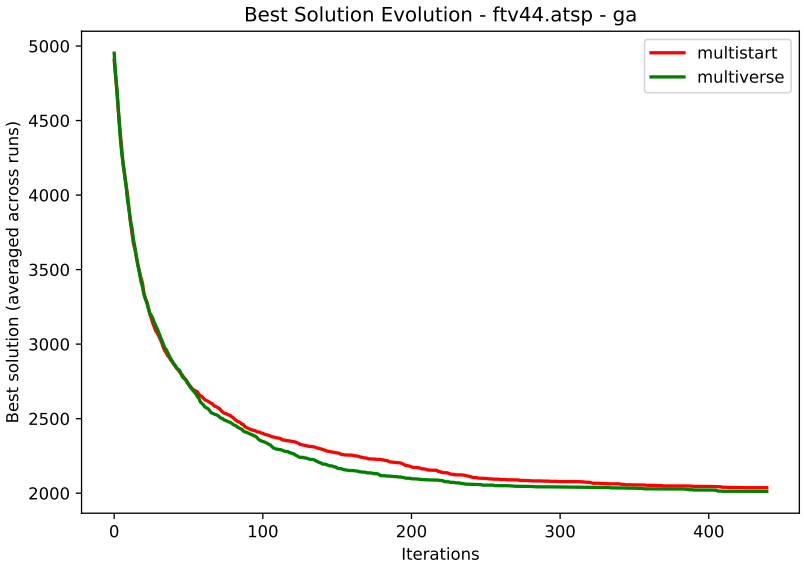

Figure A.8: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv44

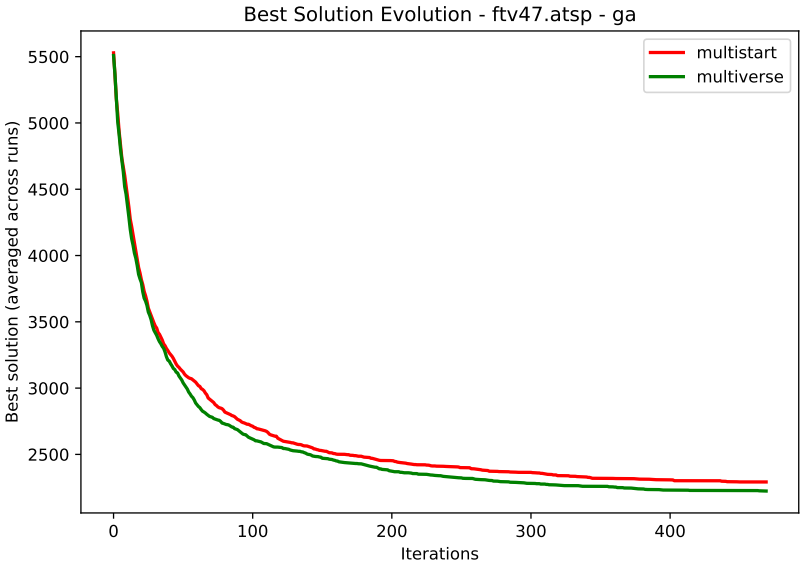

Figure A.9: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv47

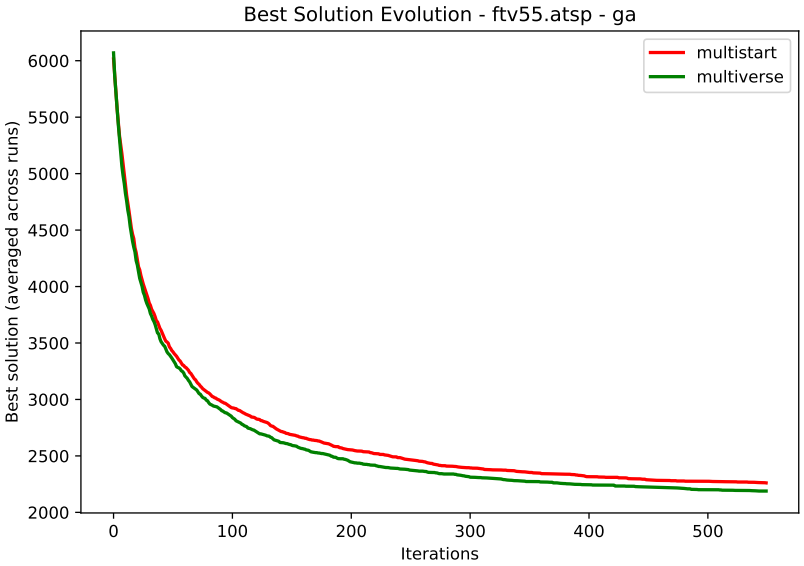

Figure A.10: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv55

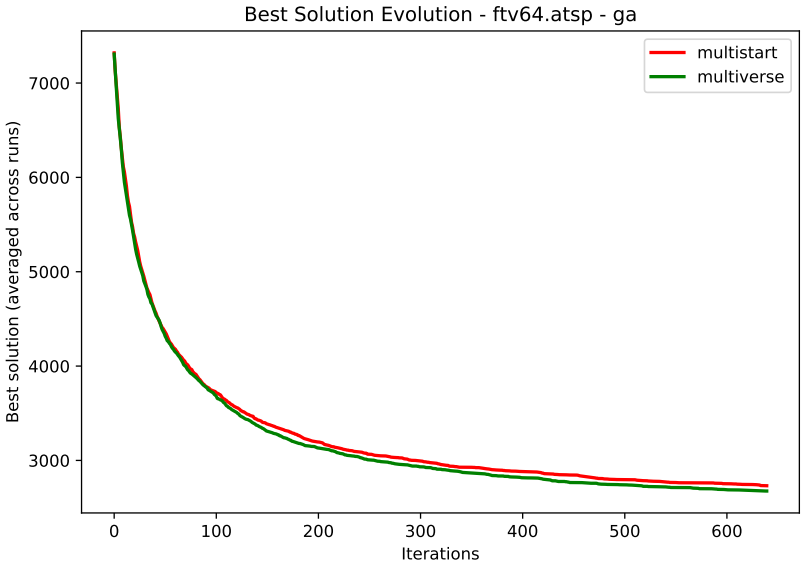

Figure A.11: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv64

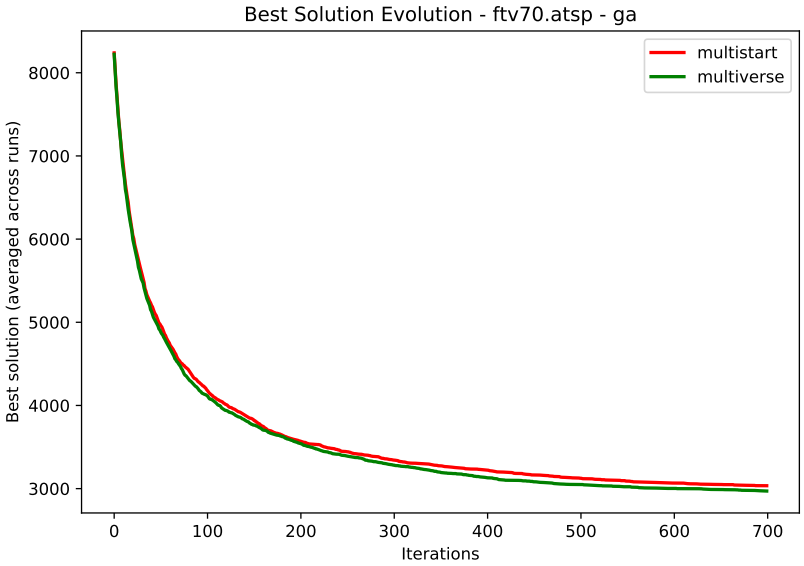

Figure A.12: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ftv70

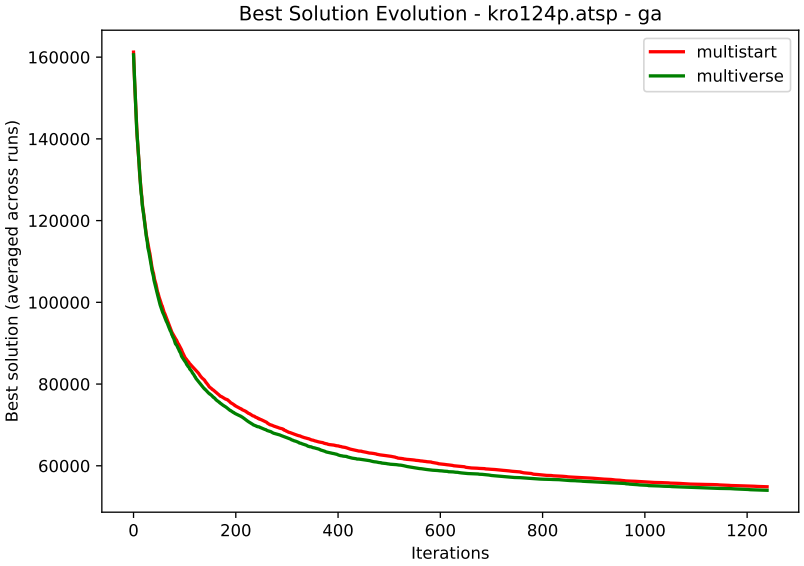

Figure A.13: Evolution of GA average best solution versus iteration for Multistart and Multiverse for kro124p

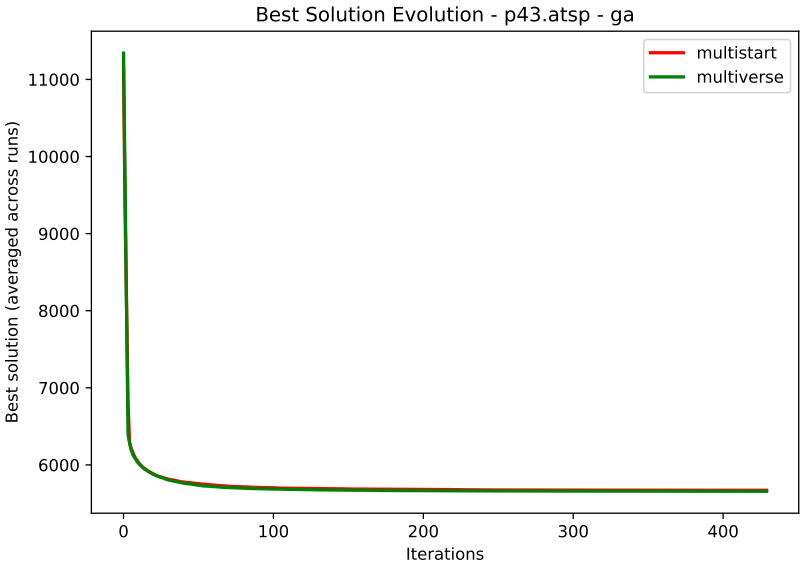

Figure A.14: Evolution of GA average best solution versus iteration for Multistart and Multiverse for p43

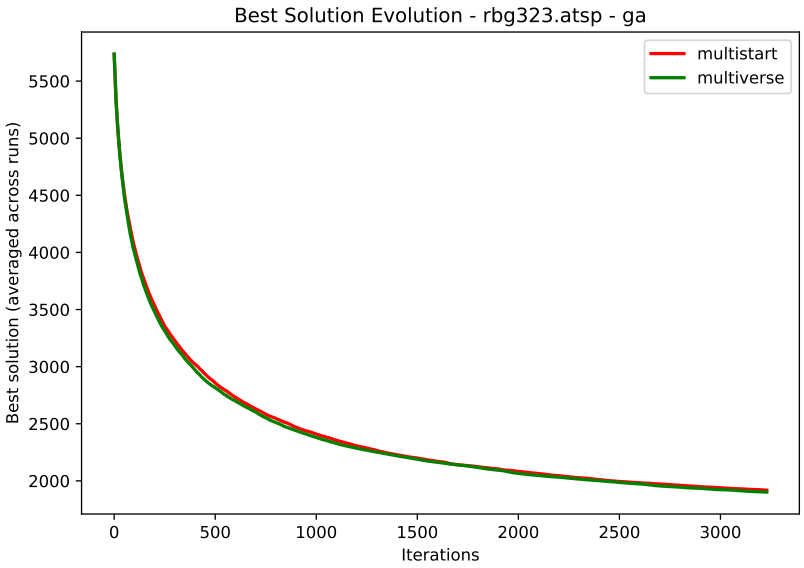

Figure A.15: Evolution of GA average best solution versus iteration for Multistart and Multiverse for rbg323

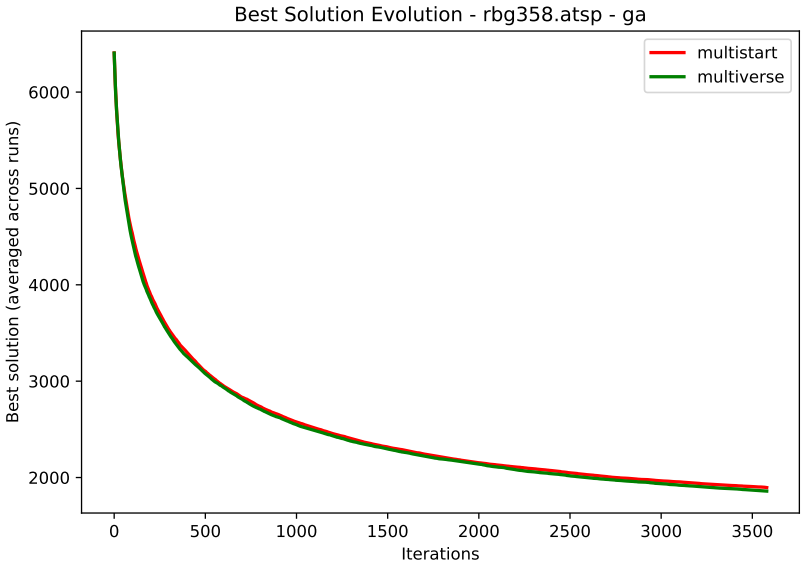

Figure A.16: Evolution of GA average best solution versus iteration for Multistart and Multiverse for rbg358

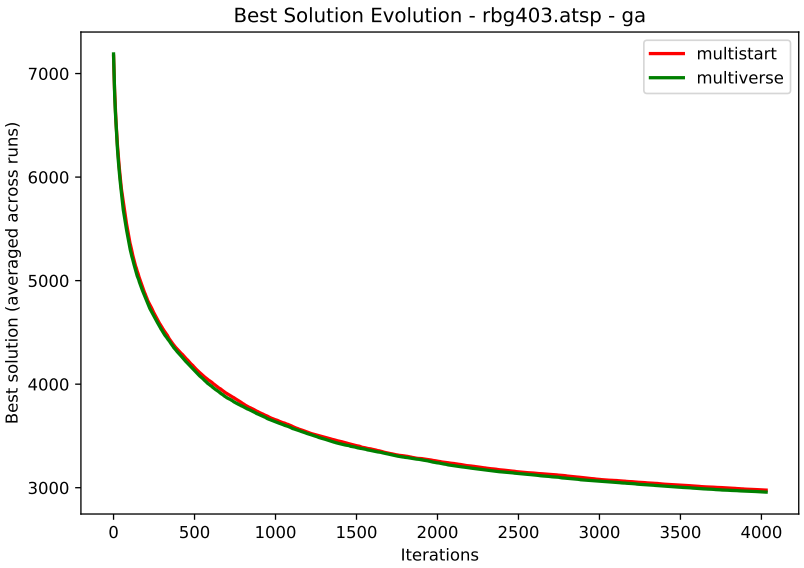

Figure A.17: Evolution of GA average best solution versus iteration for Multistart and Multiverse for rbg403

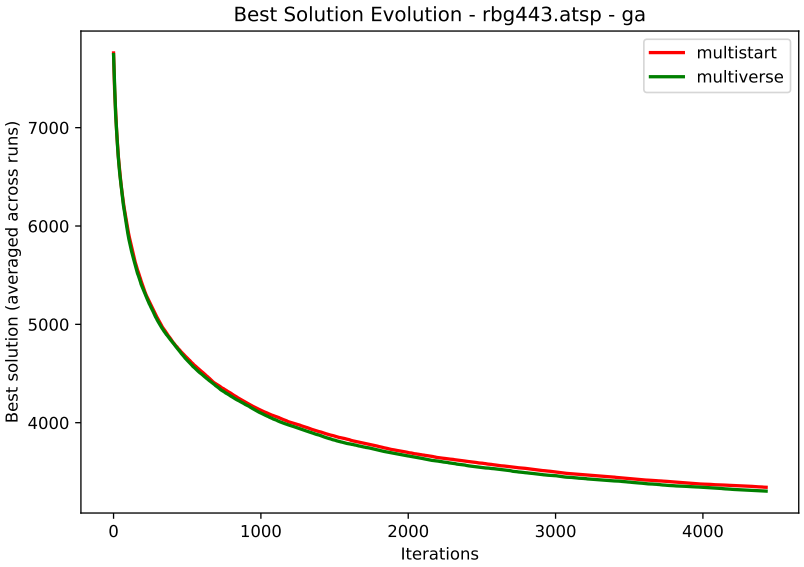

Figure A.18: Evolution of GA average best solution versus iteration for Multistart and Multiverse for rbg443

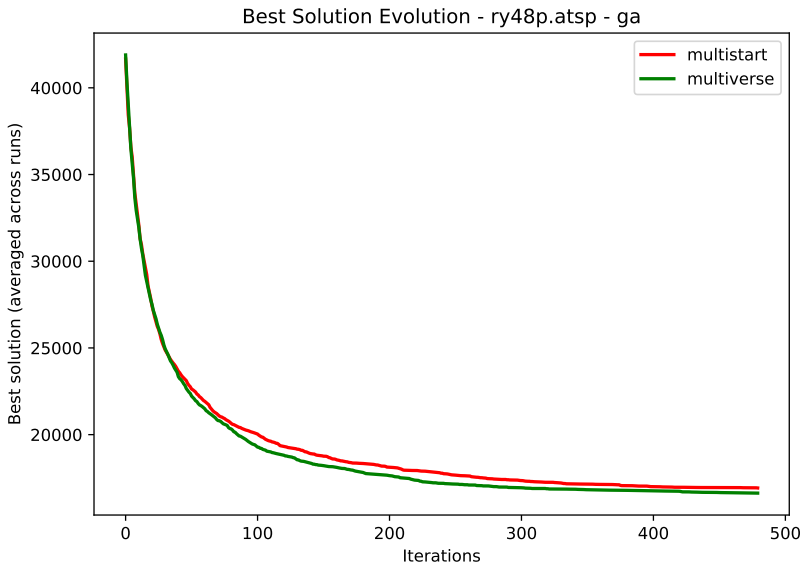

Figure A.19: Evolution of GA average best solution versus iteration for Multistart and Multiverse for ry48p

Appendix B

Evolution Graphs for TSP ACO

Figures

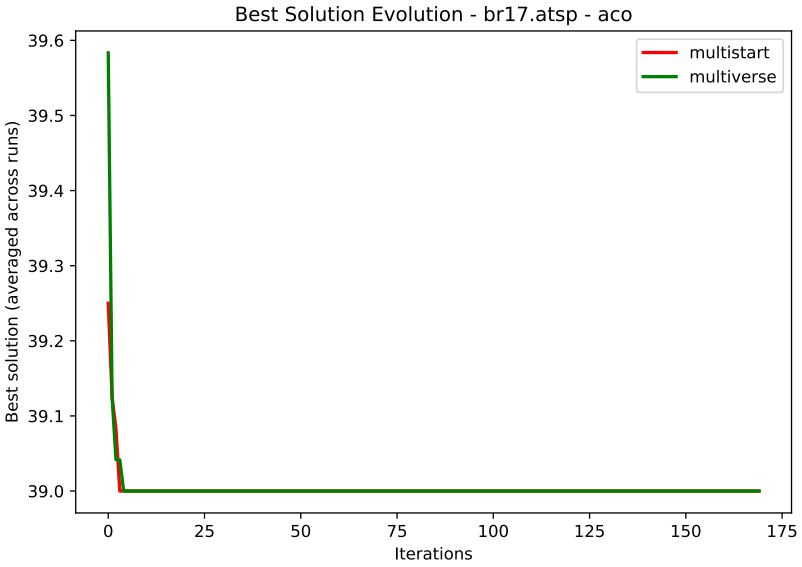

Figure B.1: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for br17

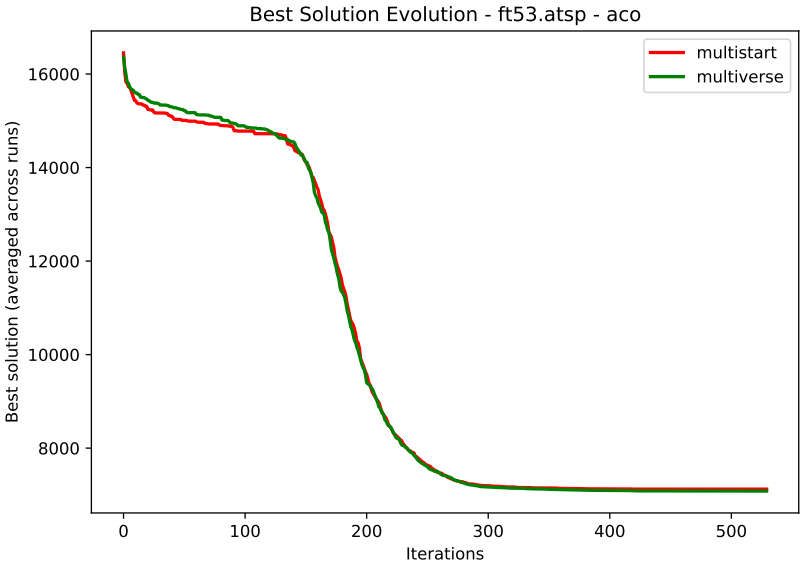

Figure B.2: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ft53

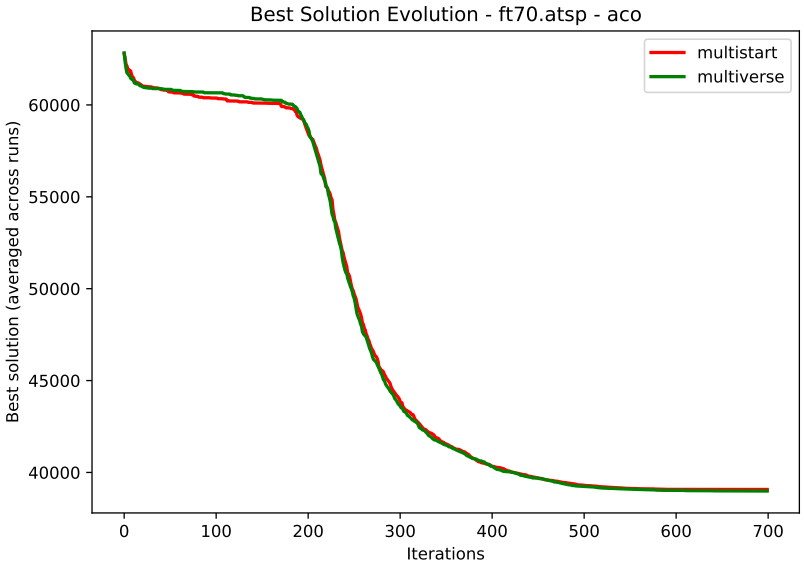

Figure B.3: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ft70

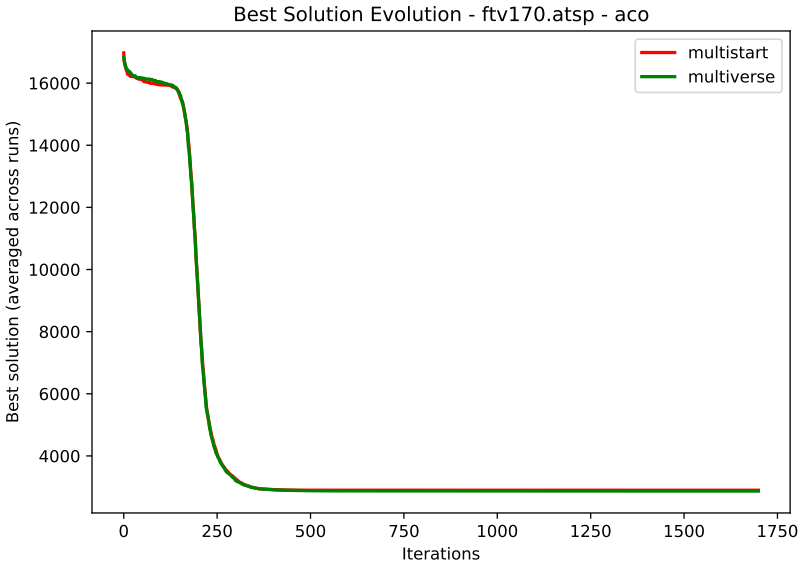

Figure B.4: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv170

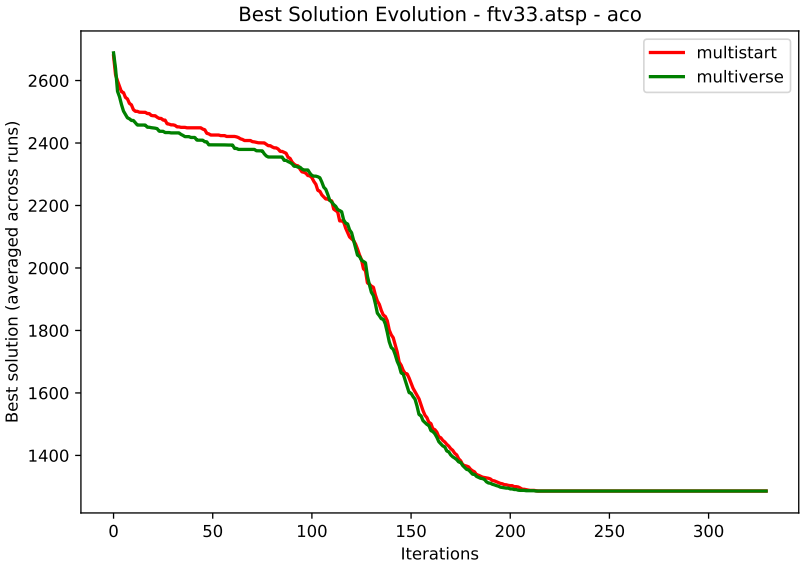

Figure B.5: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv33

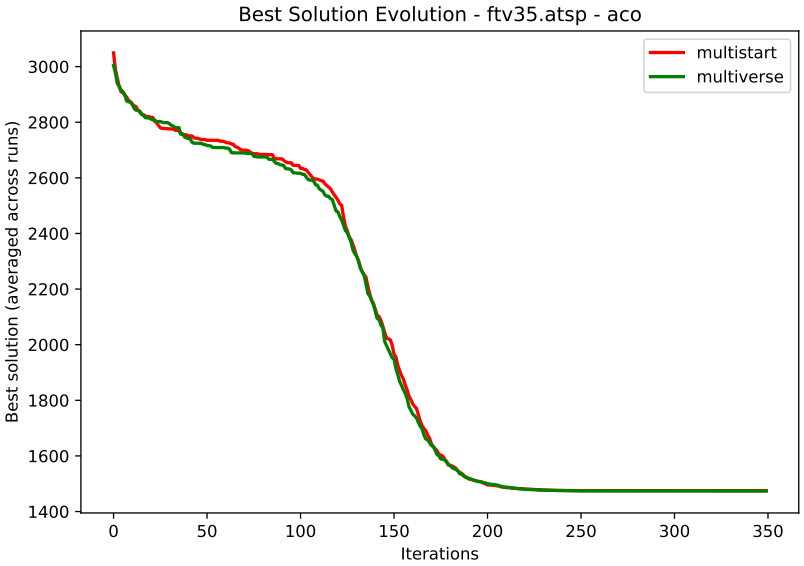

Figure B.6: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv35

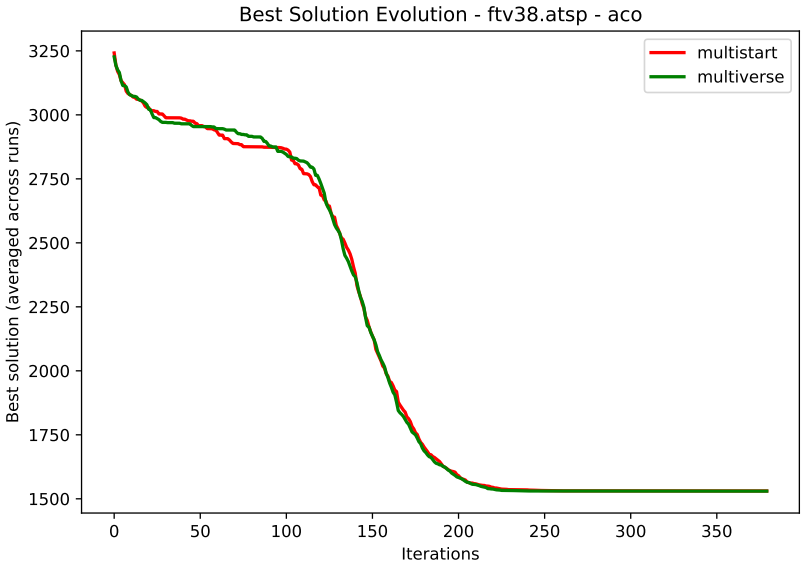

Figure B.7: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv38

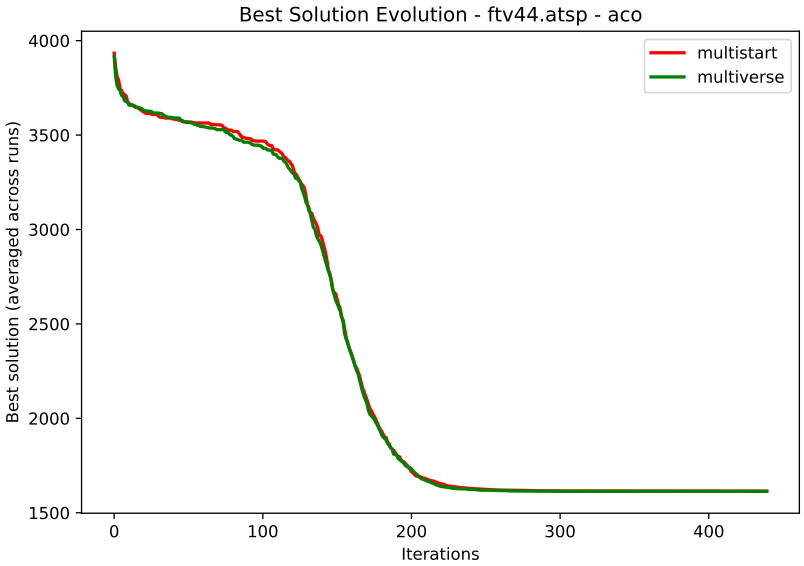

Figure B.8: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv44

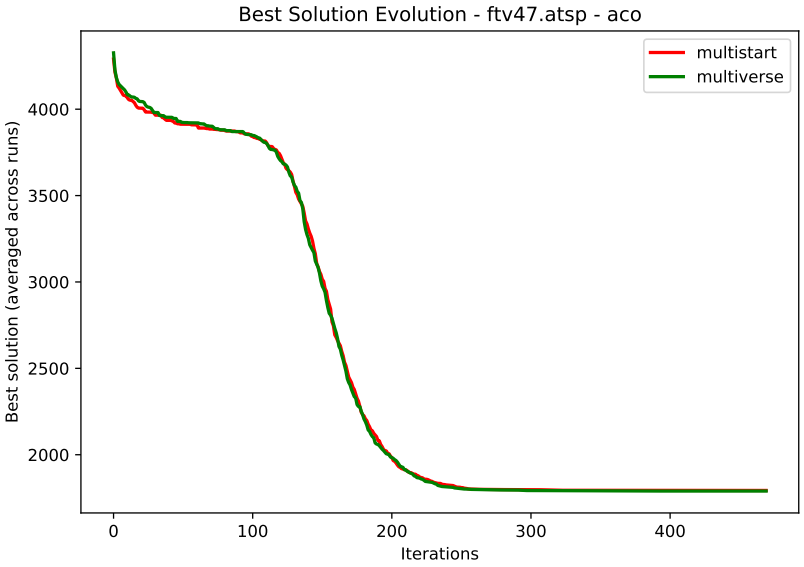

Figure B.9: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv47

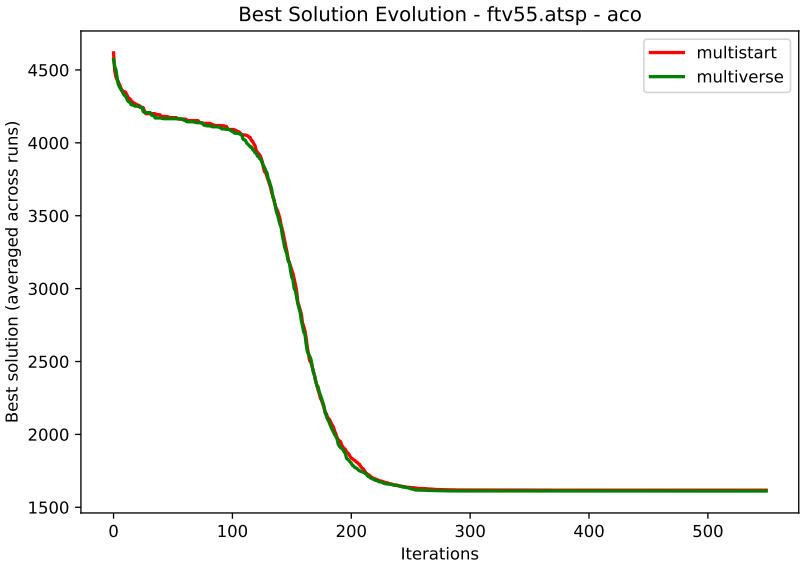

Figure B.10: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv55

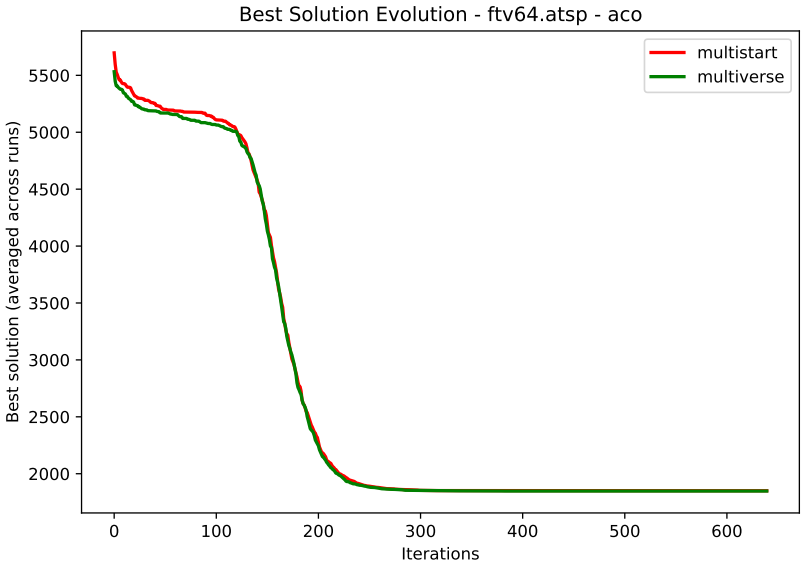

Figure B.11: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv64

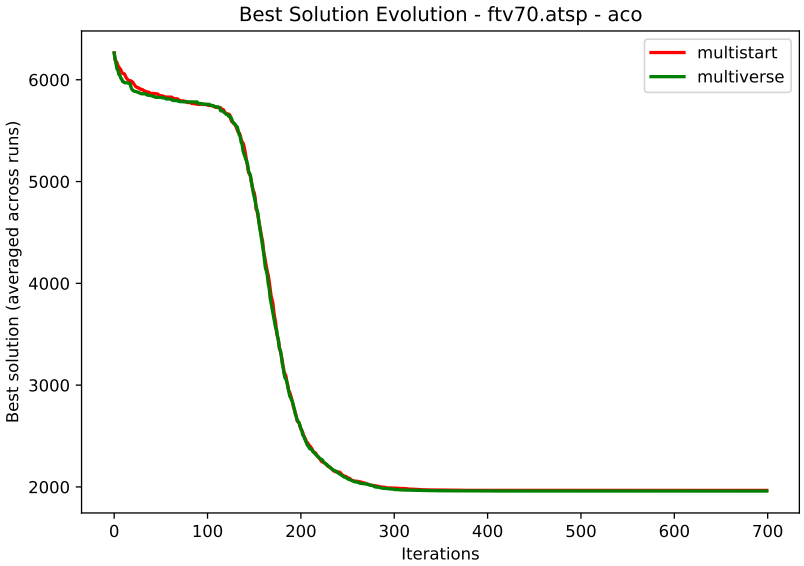

Figure B.12: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ftv70

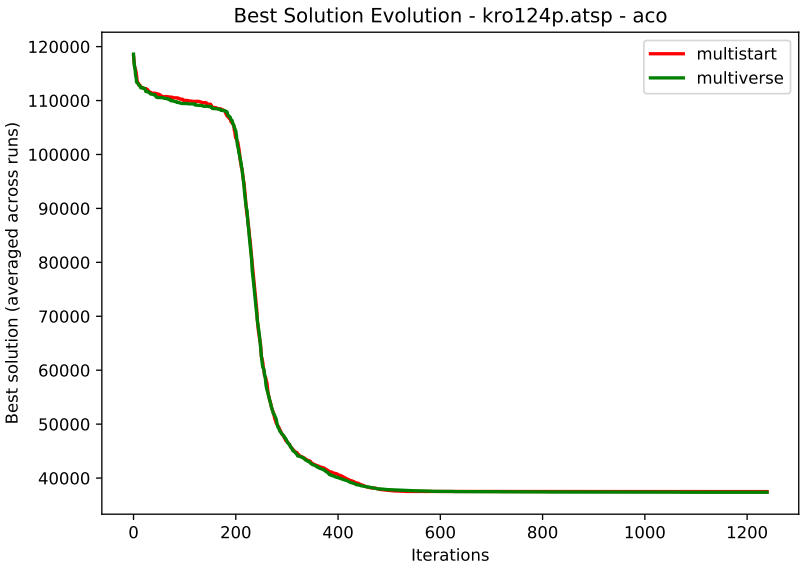

Figure B.13: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for kro124p

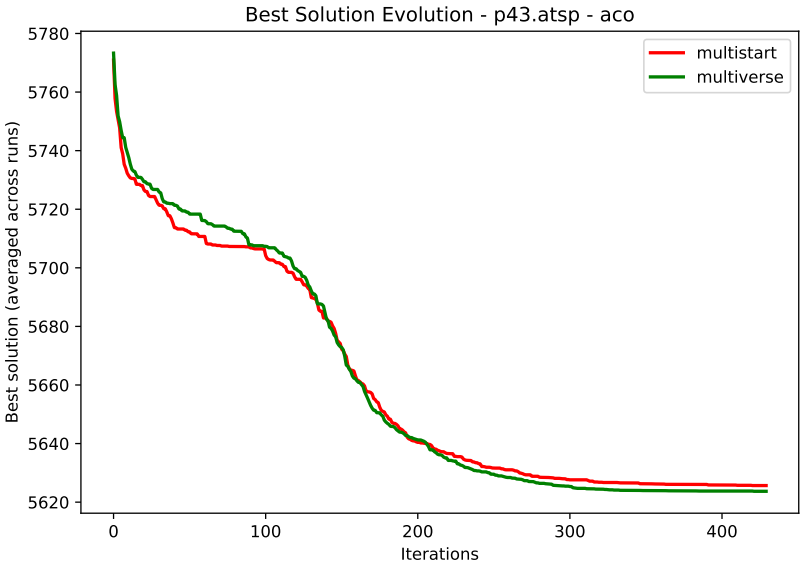

Figure B.14: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for p43

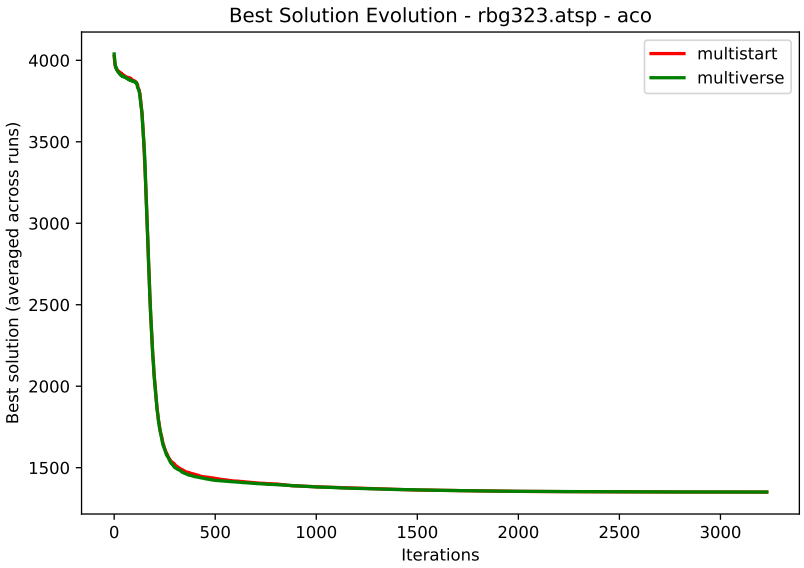

Figure B.15: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for rbg323

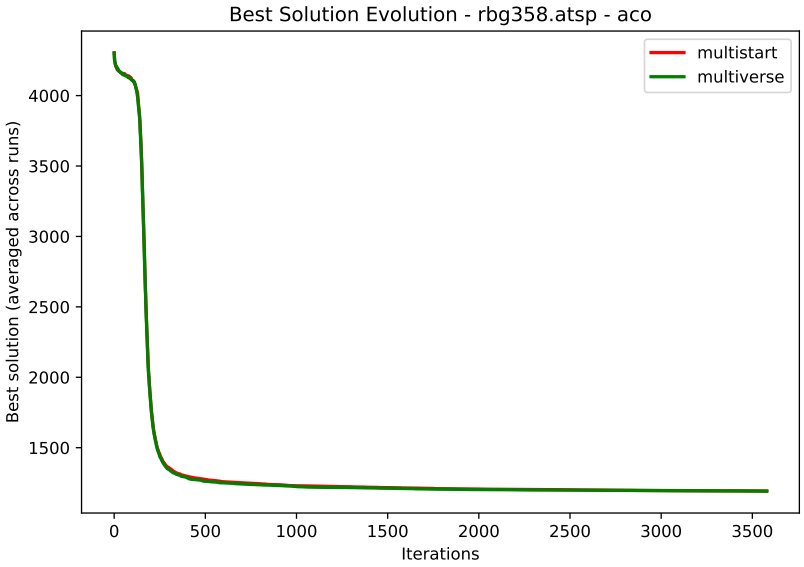

Figure B.16: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for rbg358

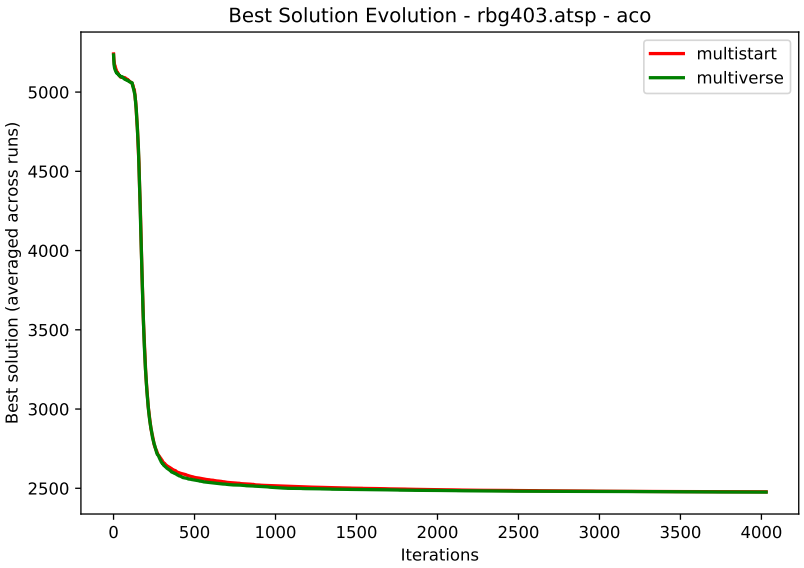

Figure B.17: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for rbg403

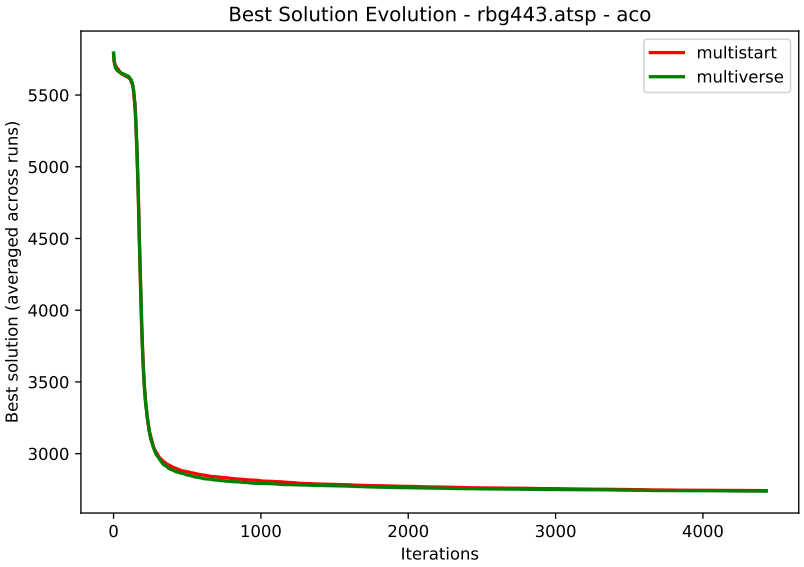

Figure B.18: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for rbg443

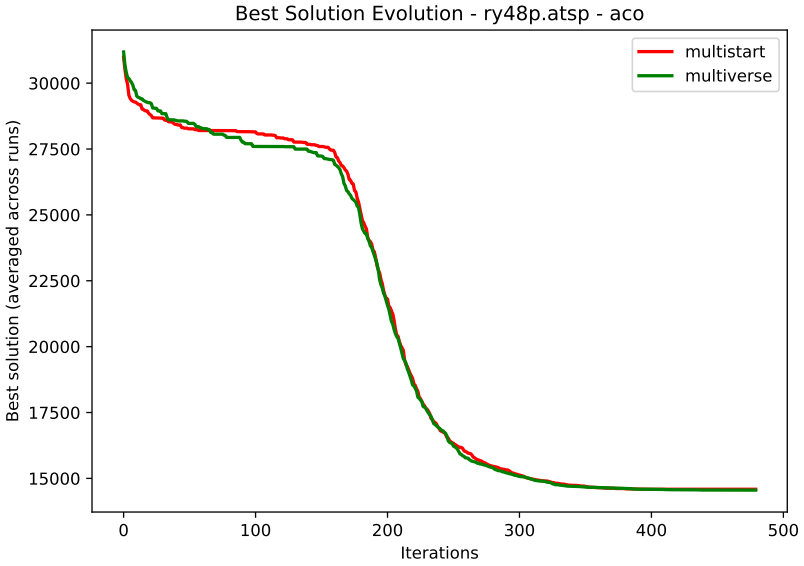

Figure B.19: Evolution of ACO average best solution versus iteration for Multistart and Multiverse for ry48p

Appendix C

Evolution Graphs for the Scheduling Problem

Figures

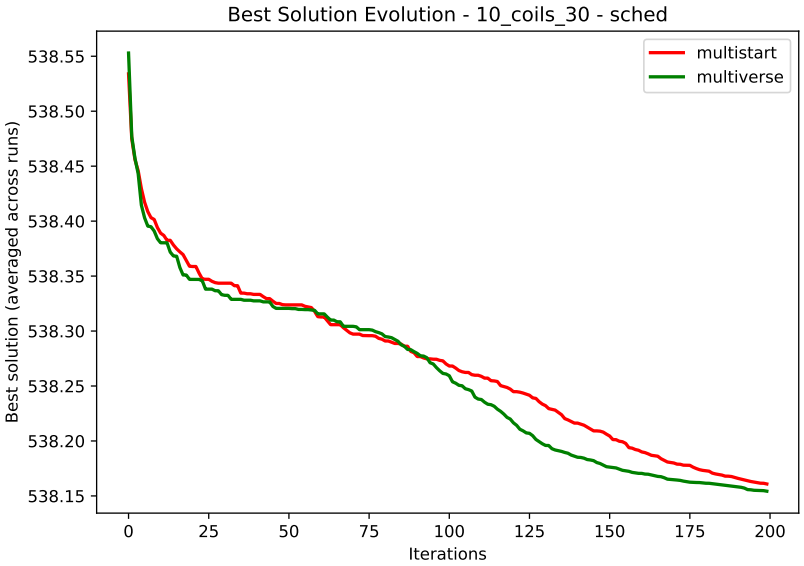

Figure C.1: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 10_coils_30

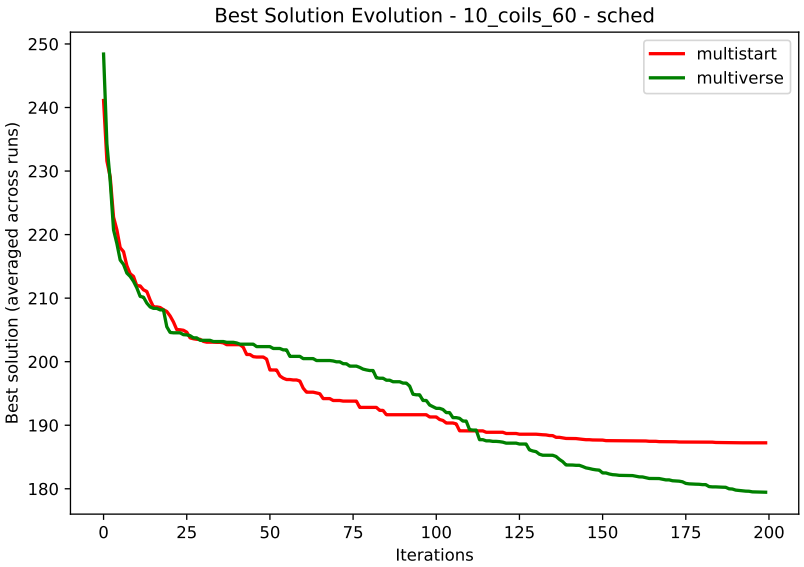

Figure C.2: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 10_coils_60

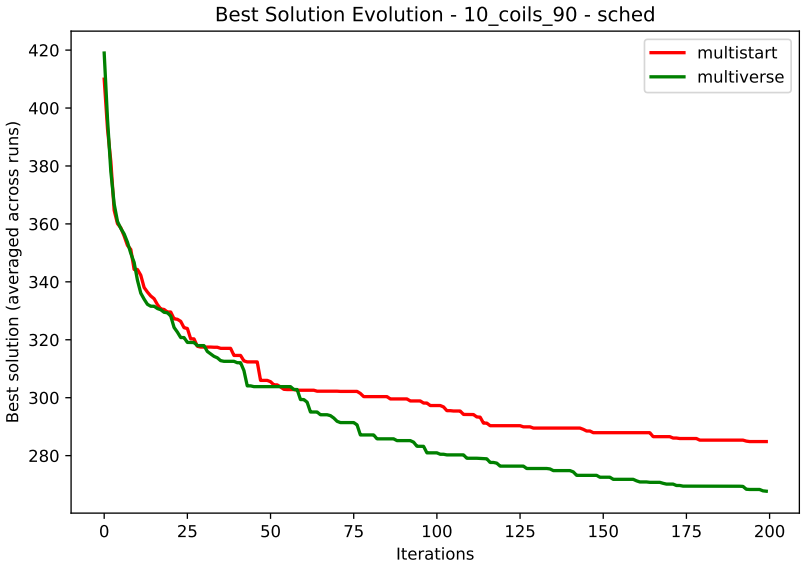

Figure C.3: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 10_coils_90

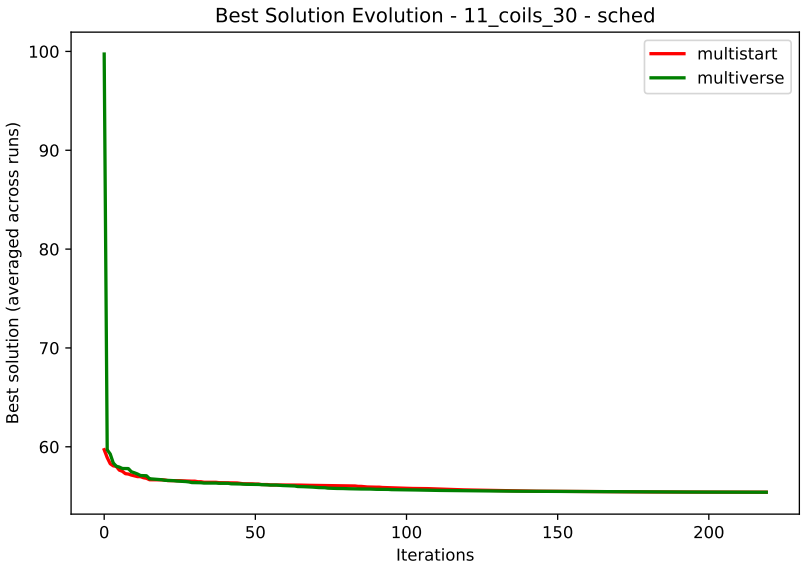

Figure C.4: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 11_coils_30

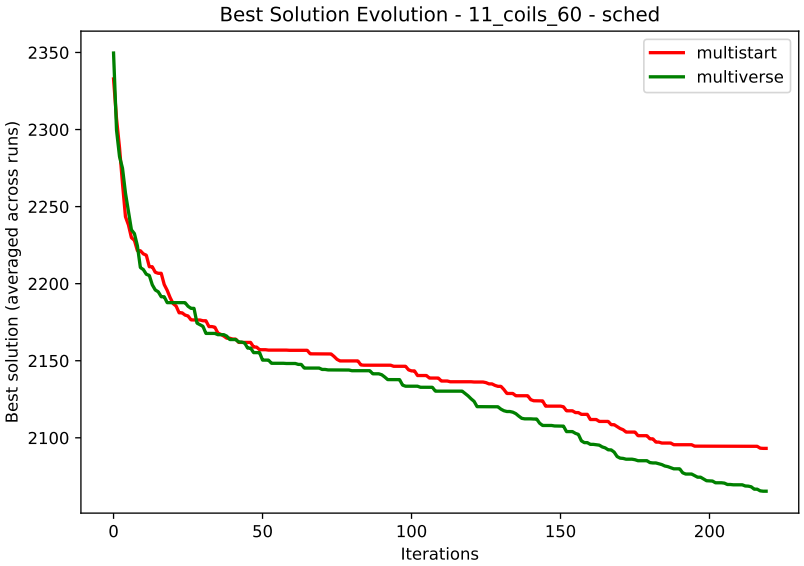

Figure C.5: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 11_coils_60

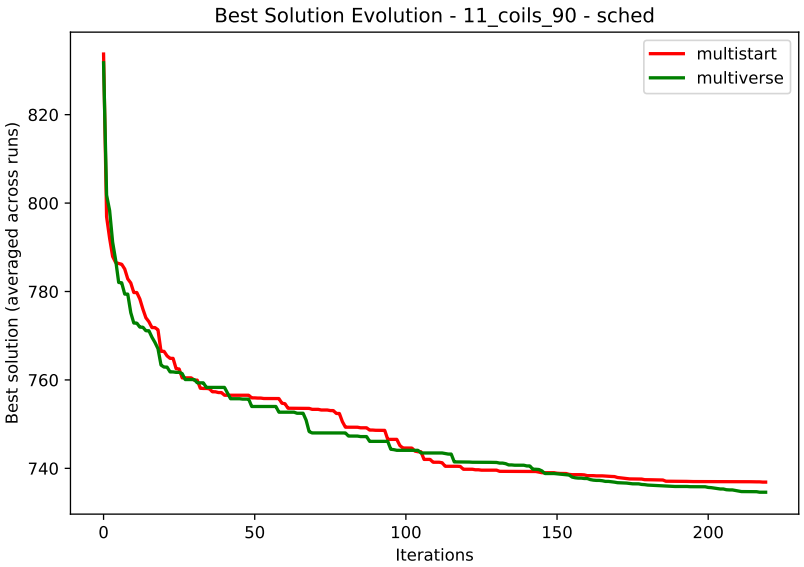

Figure C.6: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 11_coils_90

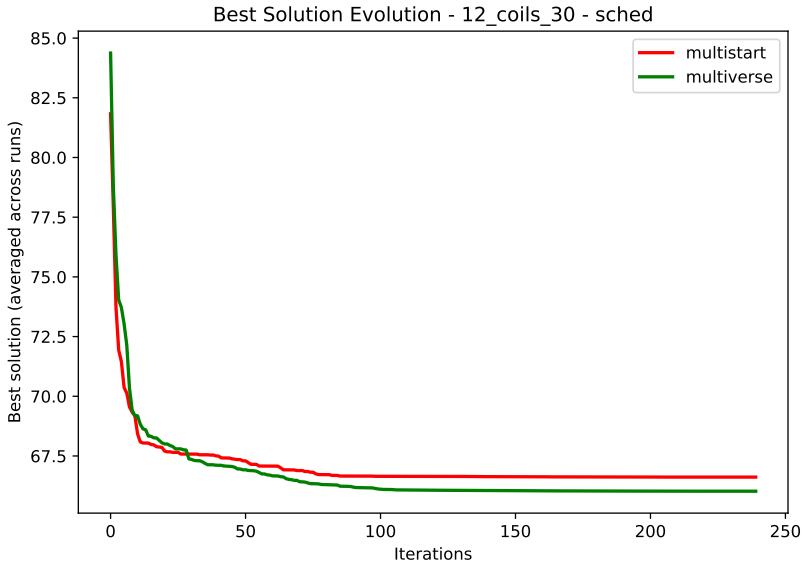

Figure C.7: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 12_coils_30

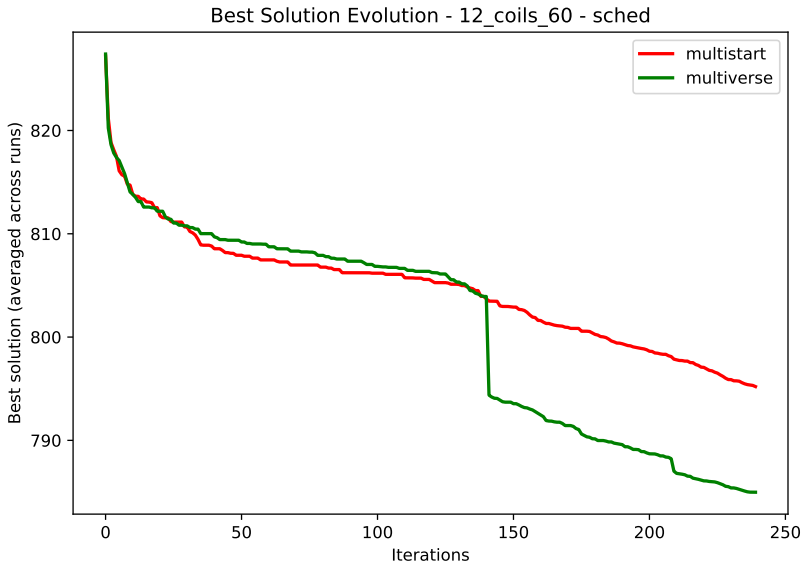

Figure C.8: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 12_coils_60

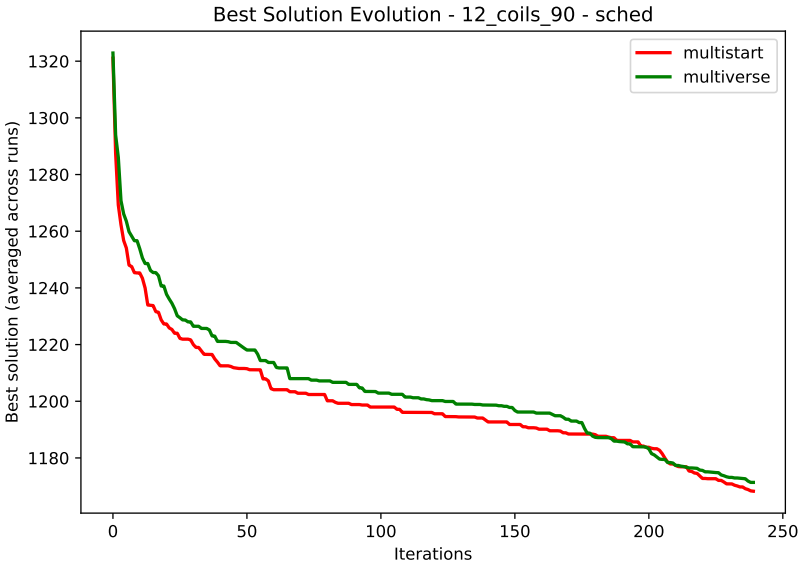

Figure C.9: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 12_coils_90

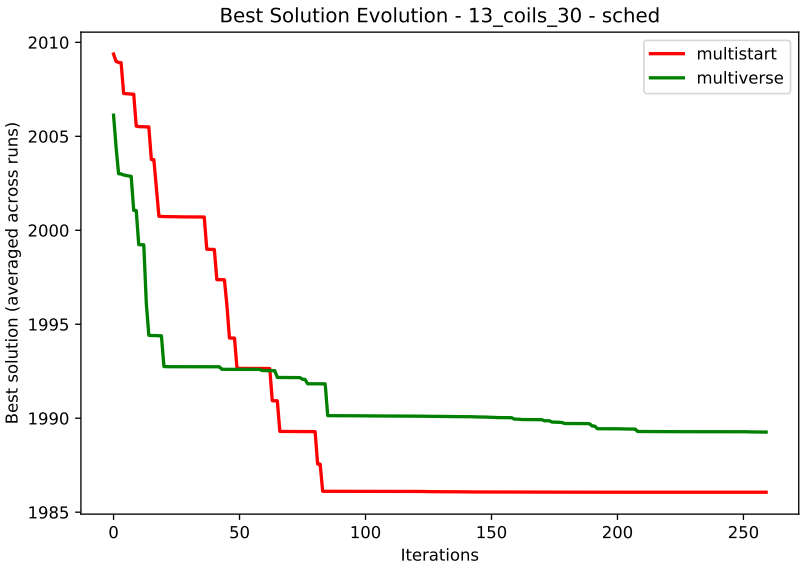

Figure C.10: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 13_coils_30

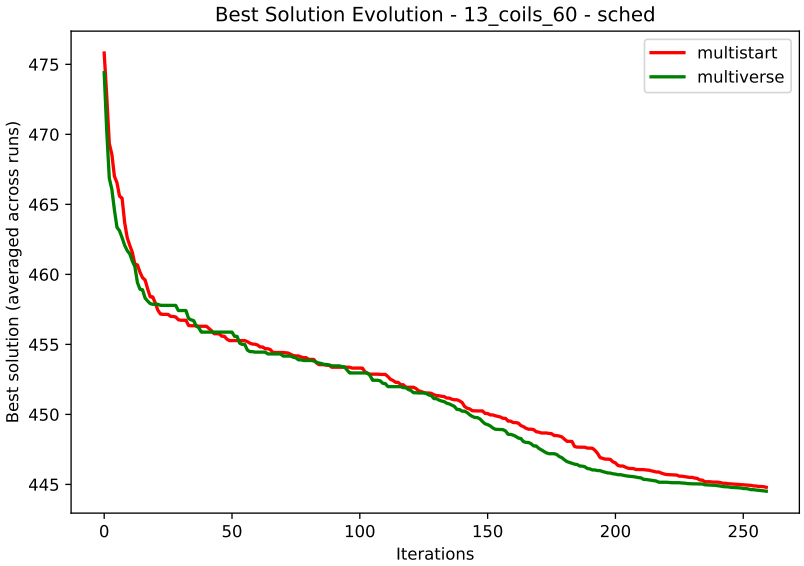

Figure C.11: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 13_coils_60

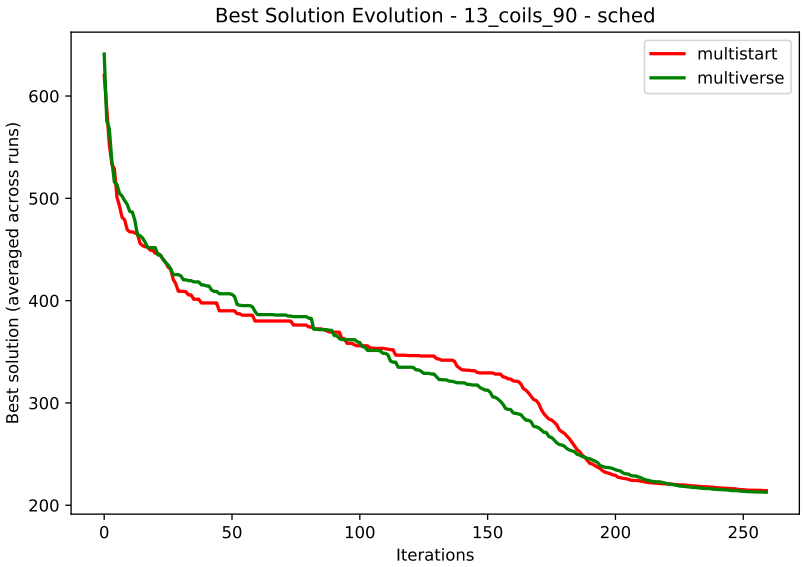

Figure C.12: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 13_coils_90

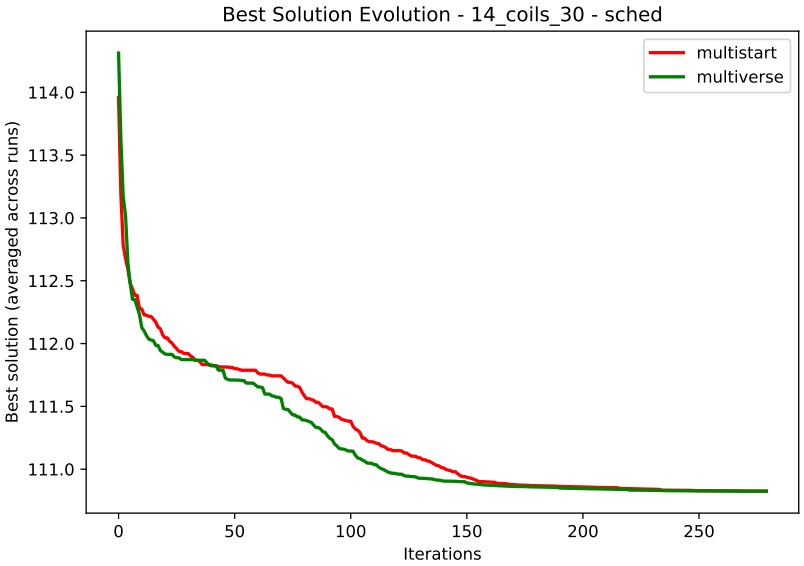

Figure C.13: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 14_coils_30

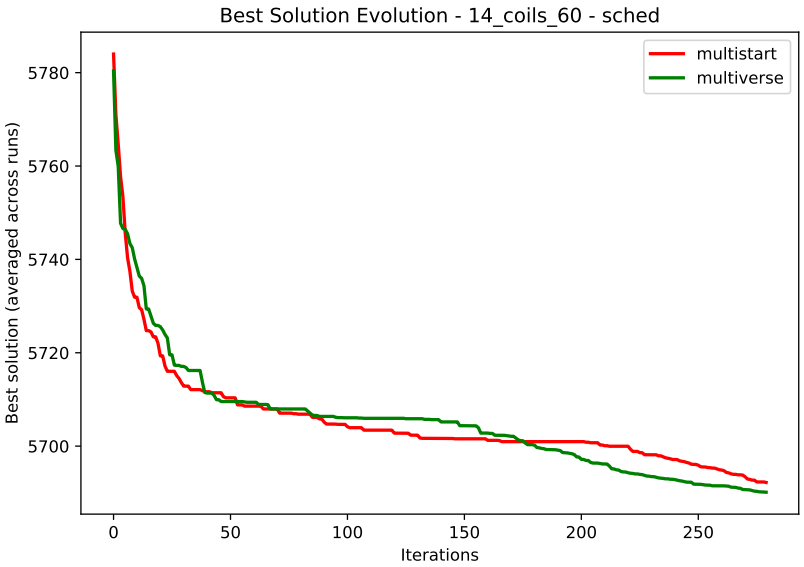

Figure C.14: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 14_coils_60

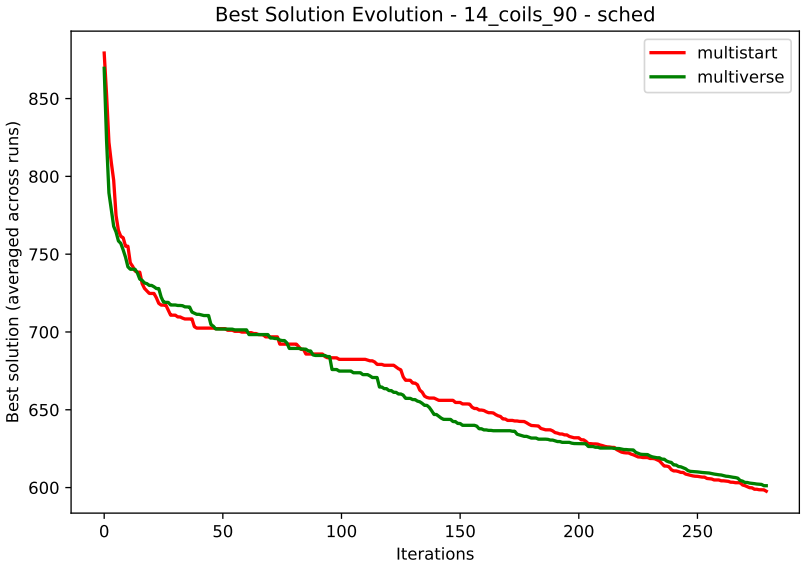

Figure C.15: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 14_coils_90

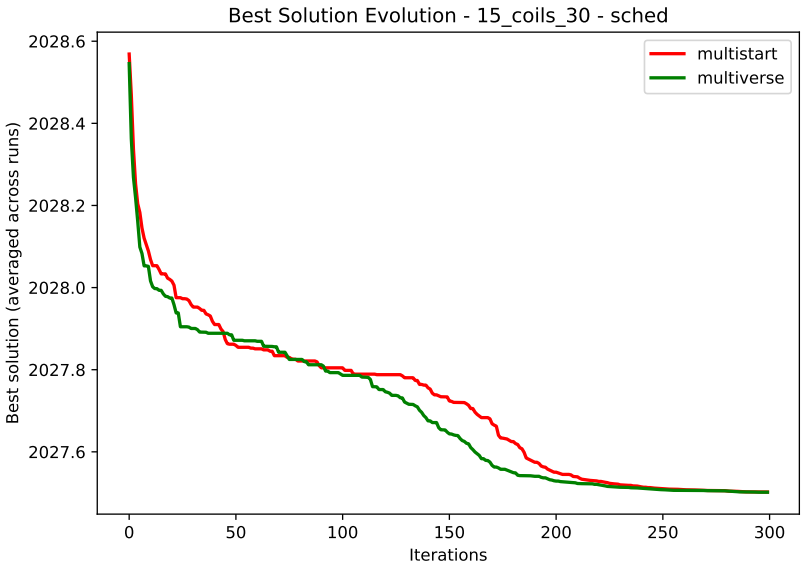

Figure C.16: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 15_coils_30

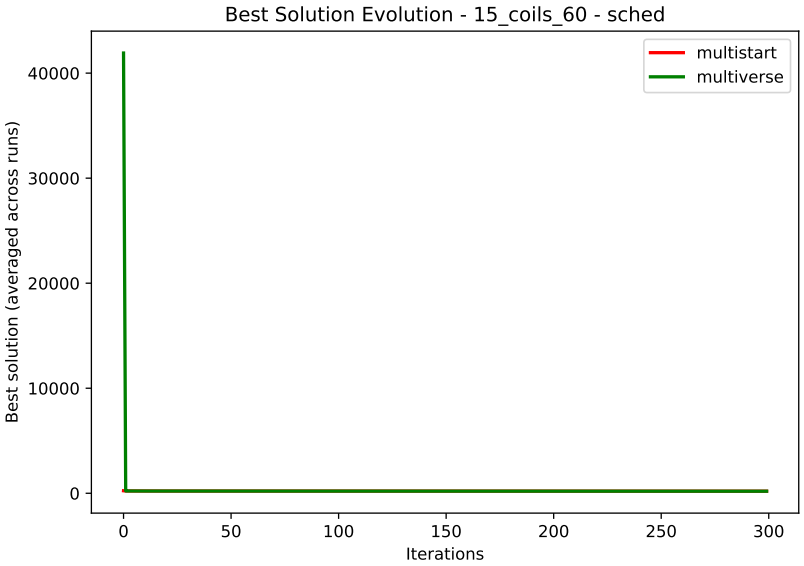

Figure C.17: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 15_coils_60

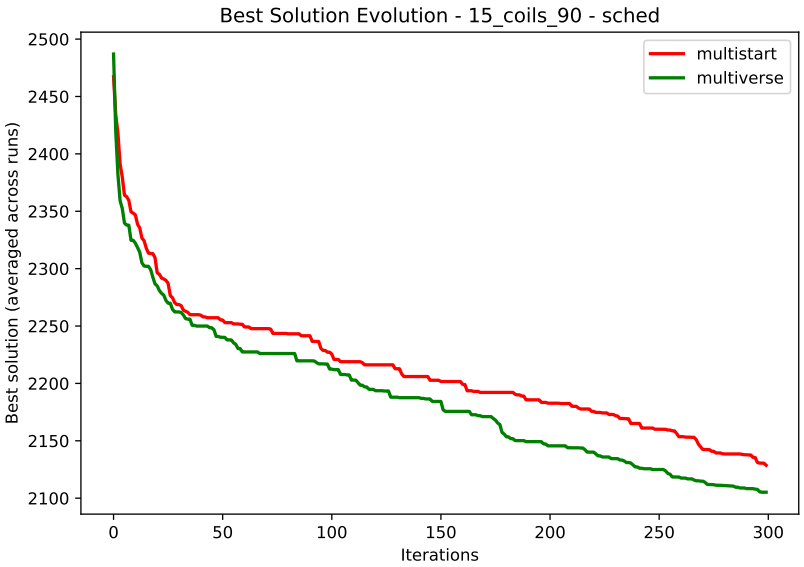

Figure C.18: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 15_coils_90

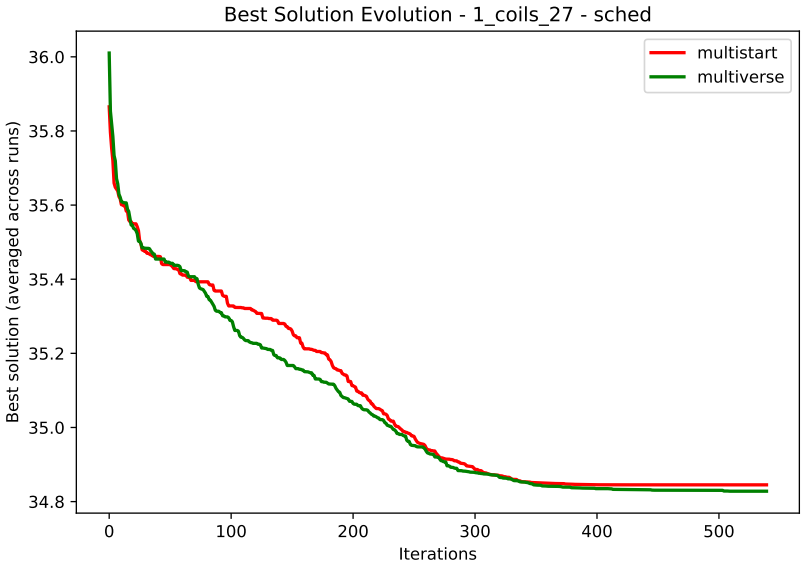

Figure C.19: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 1_coils_27

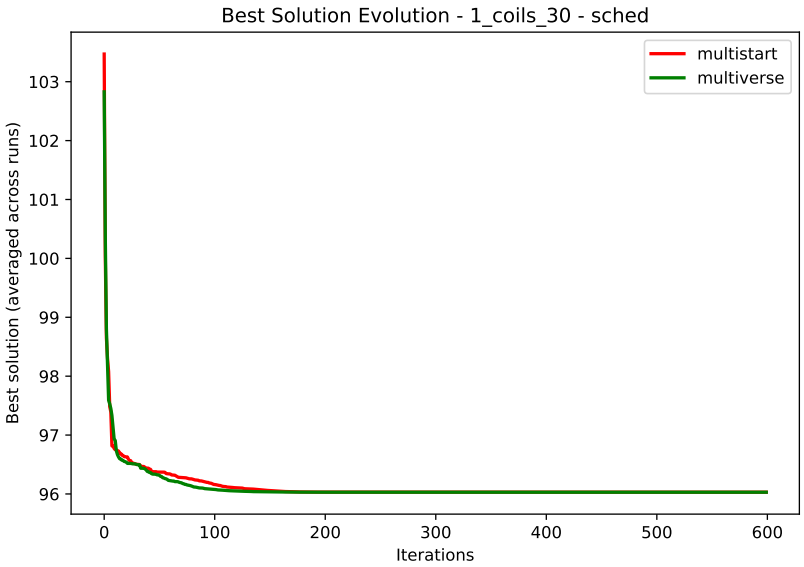

Figure C.20: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 1_coils_30

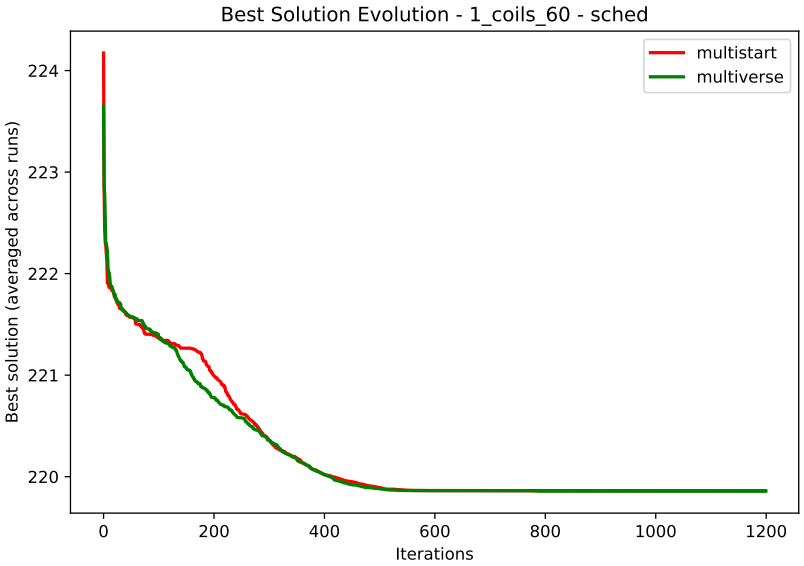

Figure C.21: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 1_coils_60

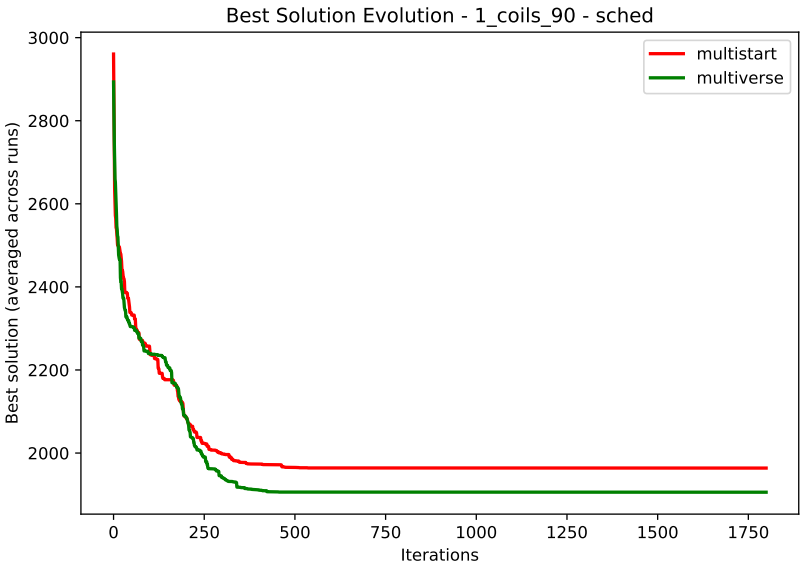

Figure C.22: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 1_coils_90

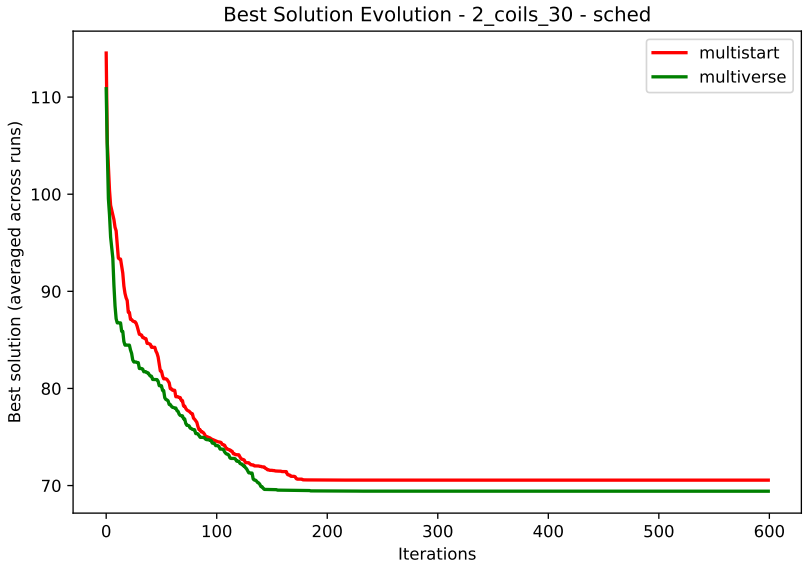

Figure C.23: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 2_coils_30

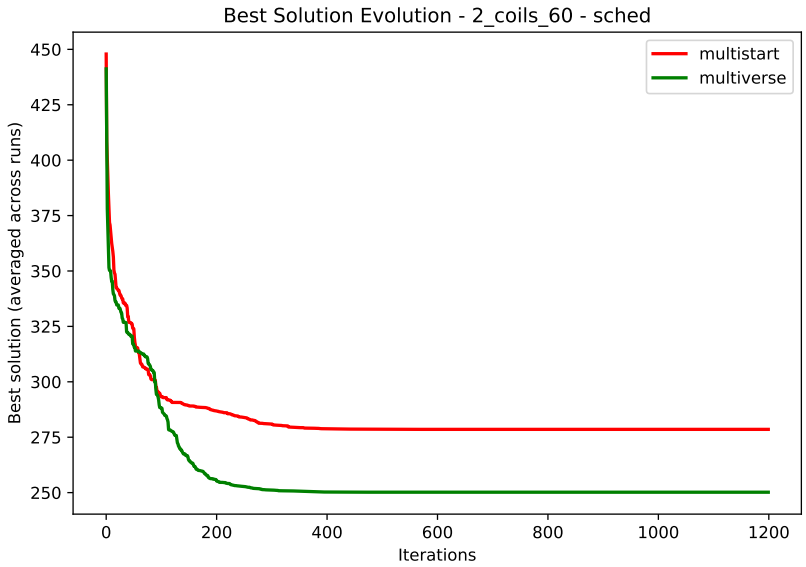

Figure C.24: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 2_coils_60

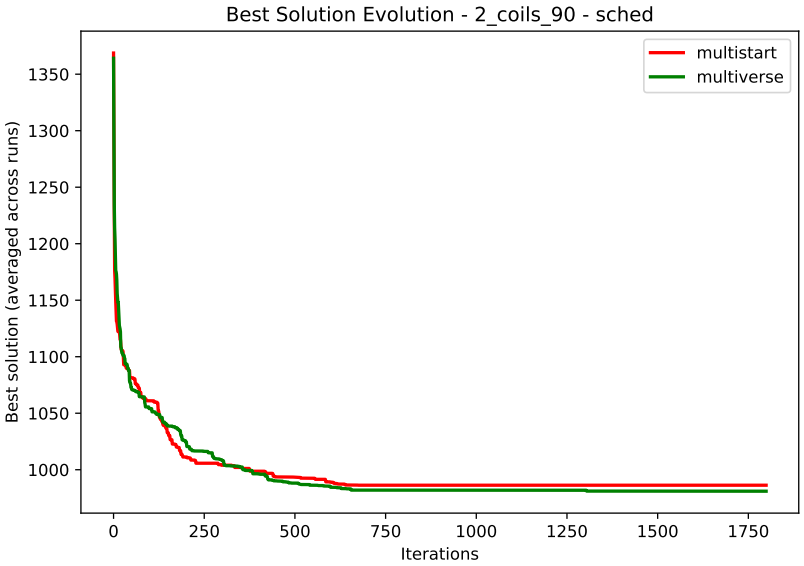

Figure C.25: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 2_coils_90

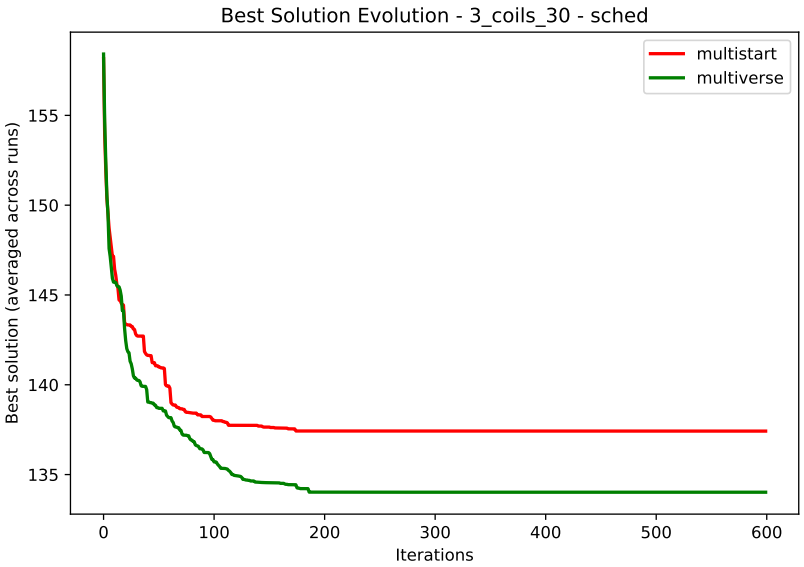

Figure C.26: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 3_coils_30

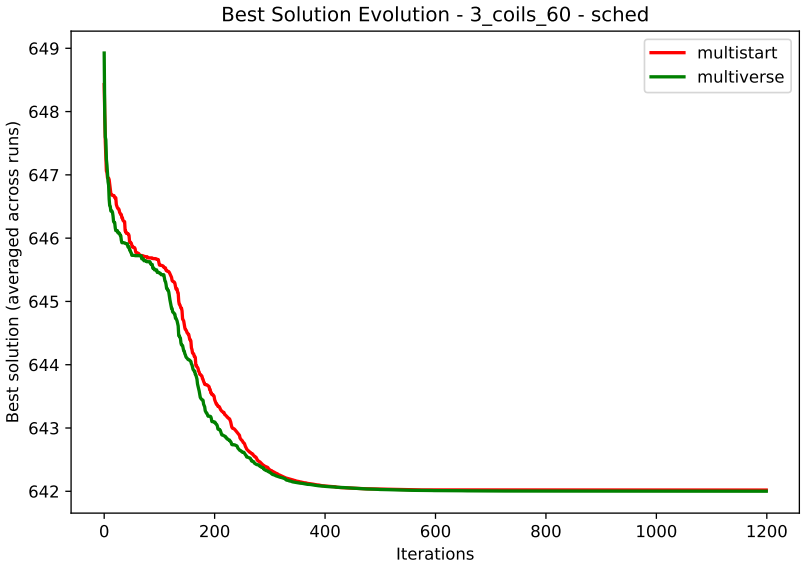

Figure C.27: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 3_coils_60

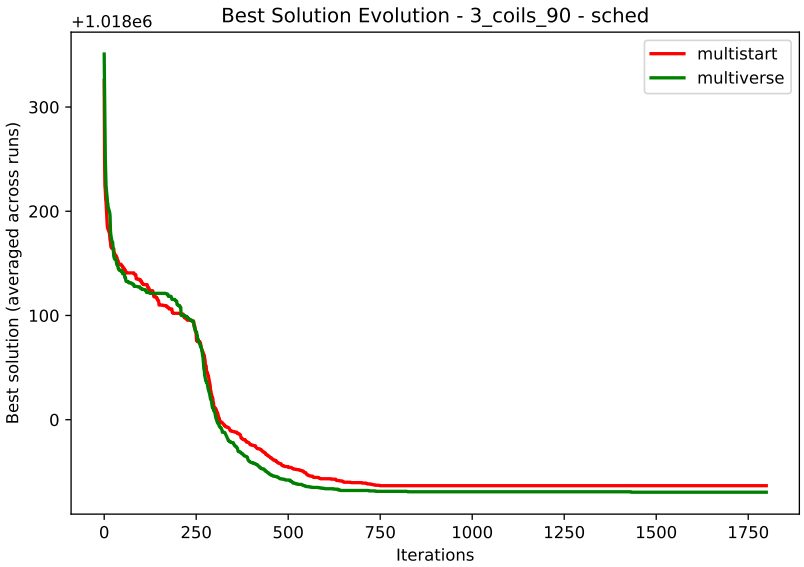

Figure C.28: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 3_coils_90

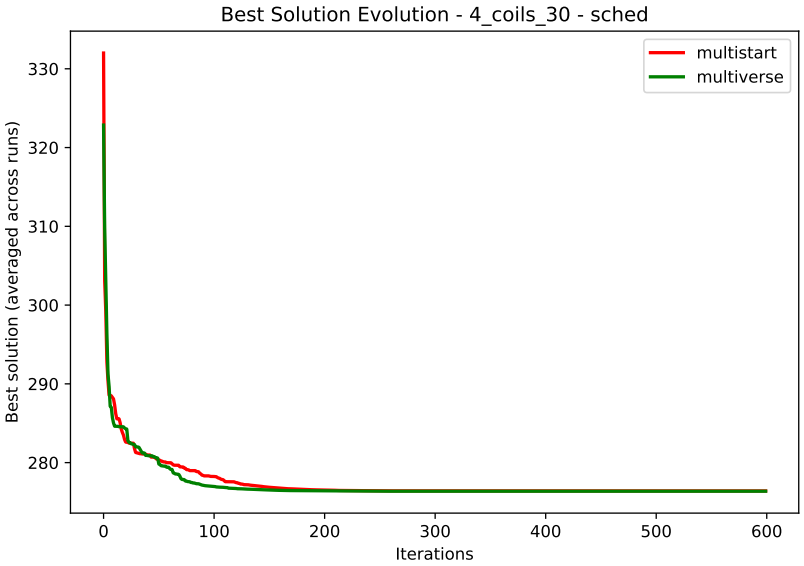

Figure C.29: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 4_coils_30

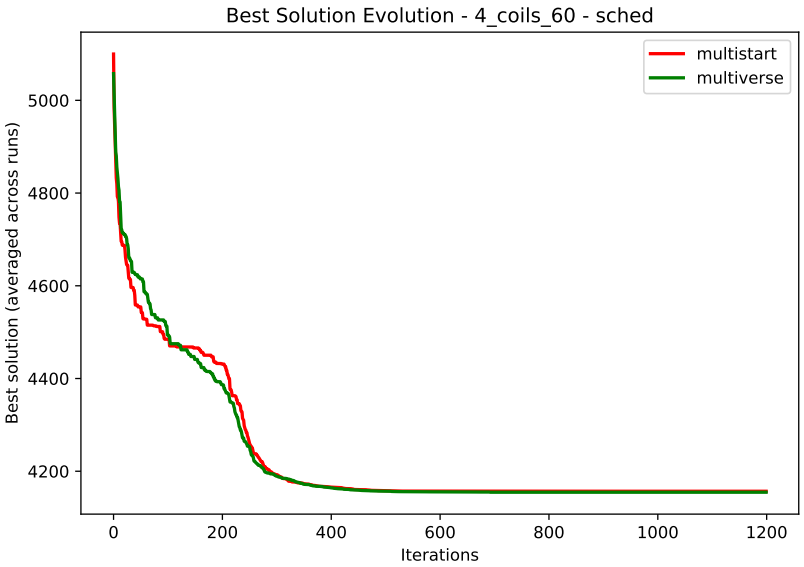

Figure C.30: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 4_coils_60

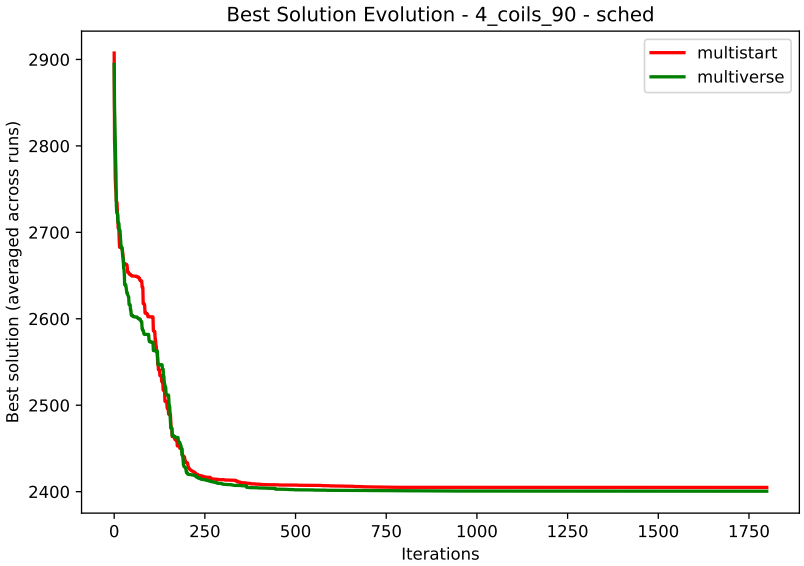

Figure C.31: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 4_coils_90

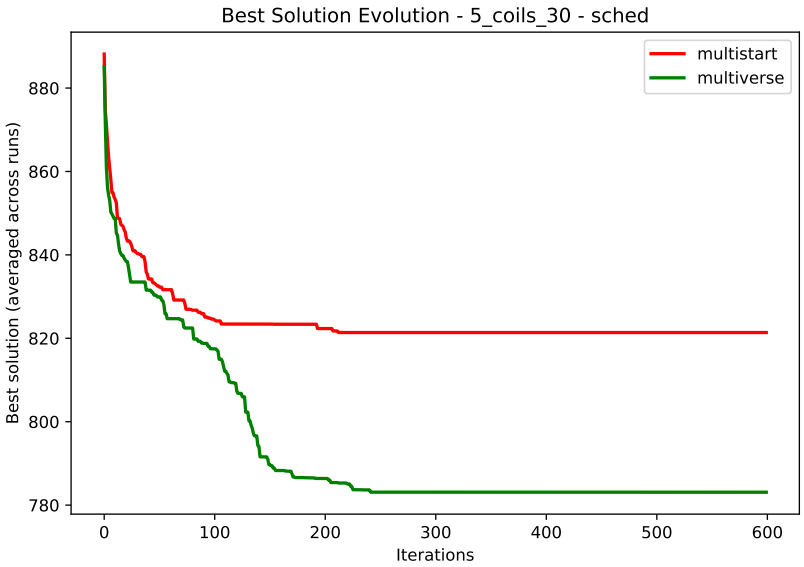

Figure C.32: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 5_coils_30

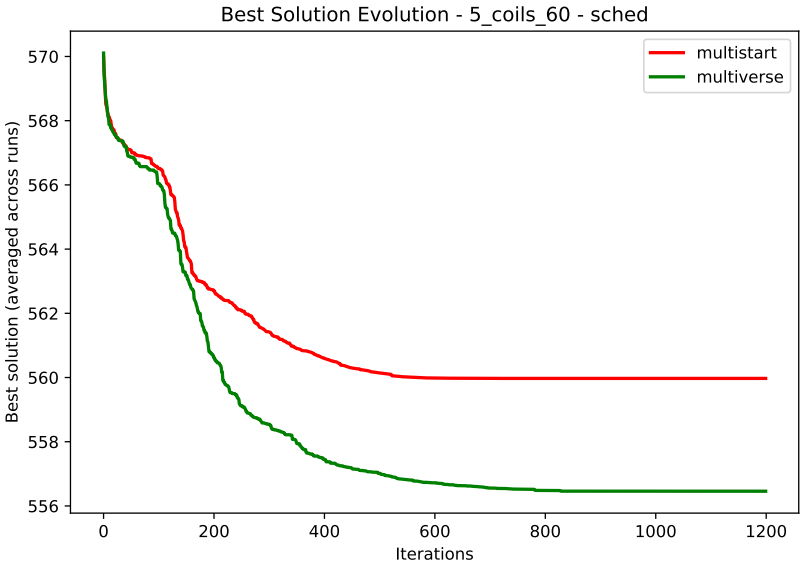

Figure C.33: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 5_coils_60

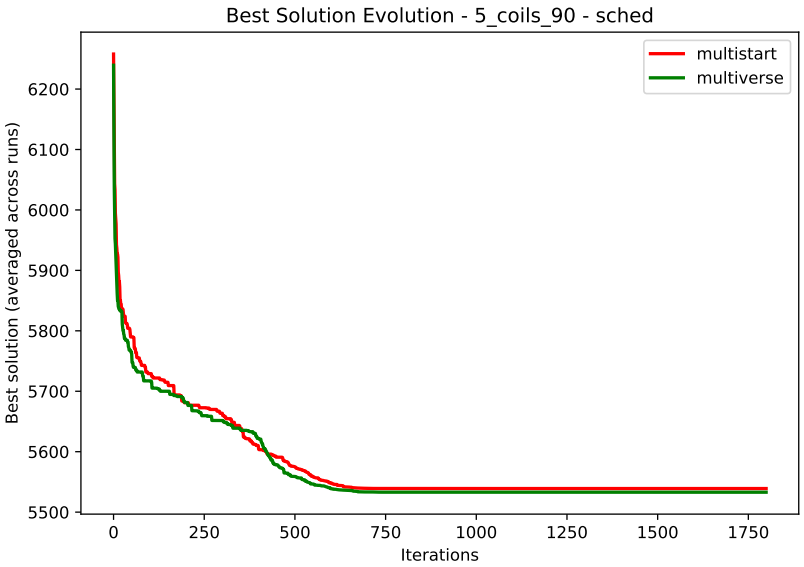

Figure C.34: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 5_coils_90

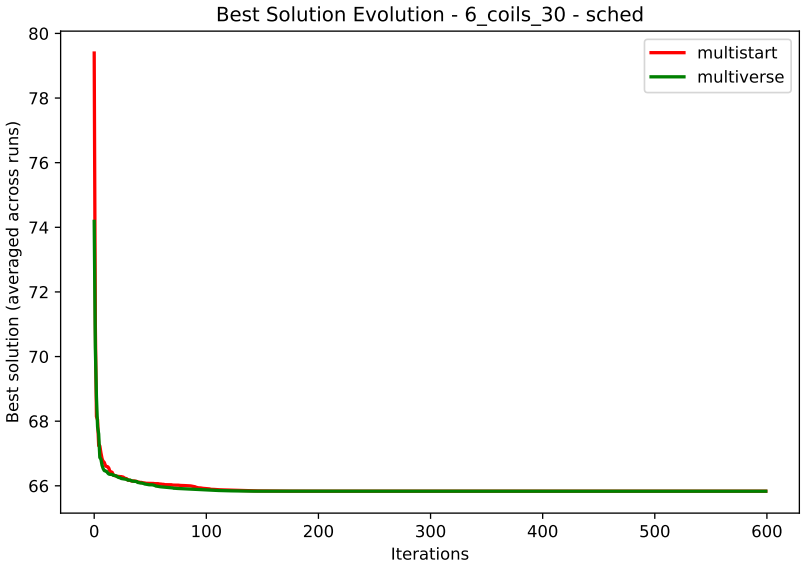

Figure C.35: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 6_coils_30

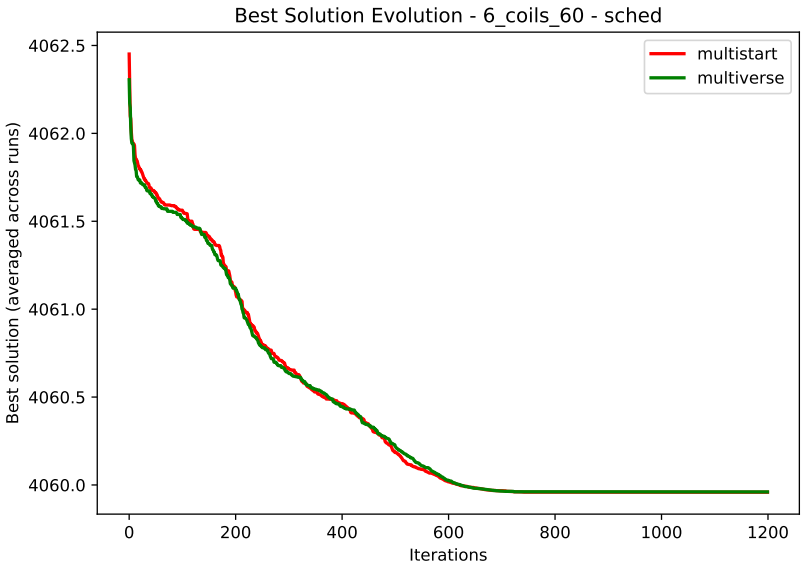

Figure C.36: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 6_coils_60

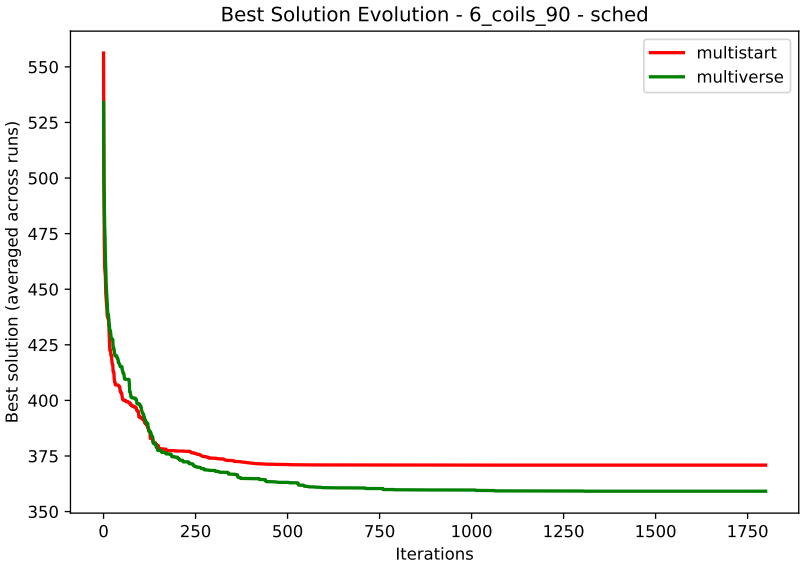

Figure C.37: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 6_coils_90

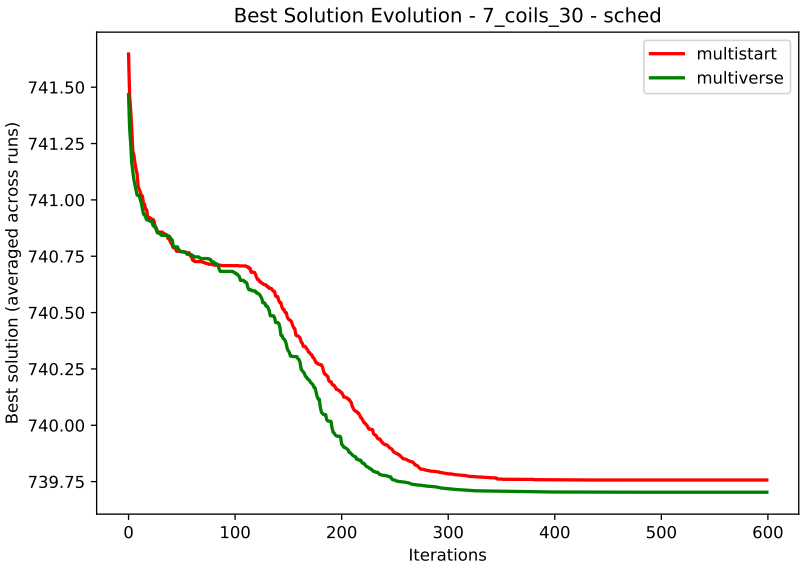

Figure C.38: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 7_coils_30

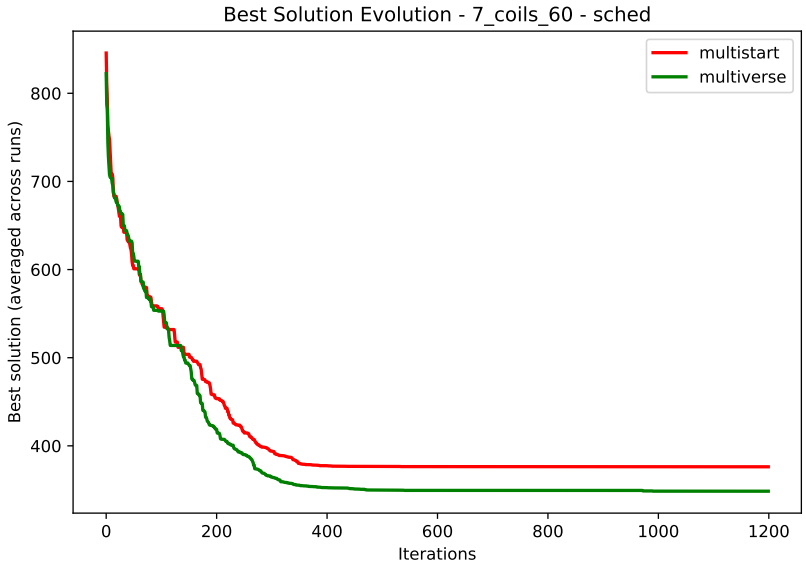

Figure C.39: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 7_coils_60

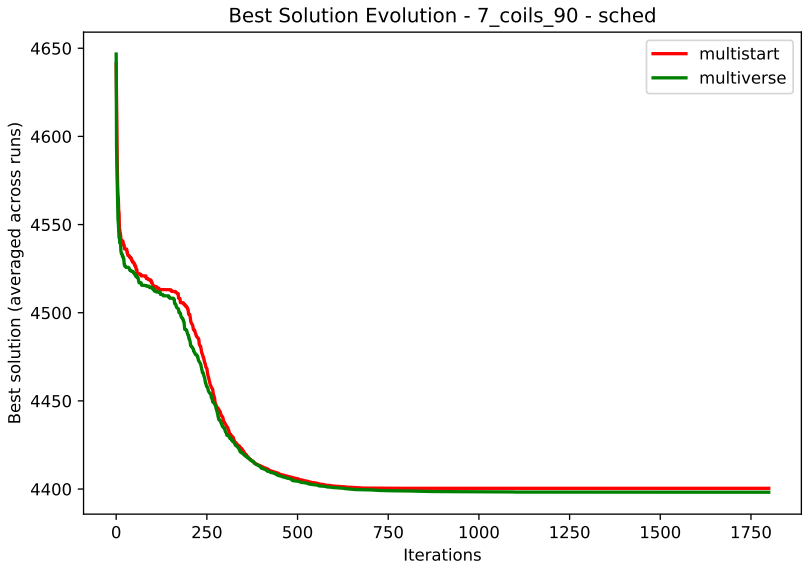

Figure C.40: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 7_coils_90

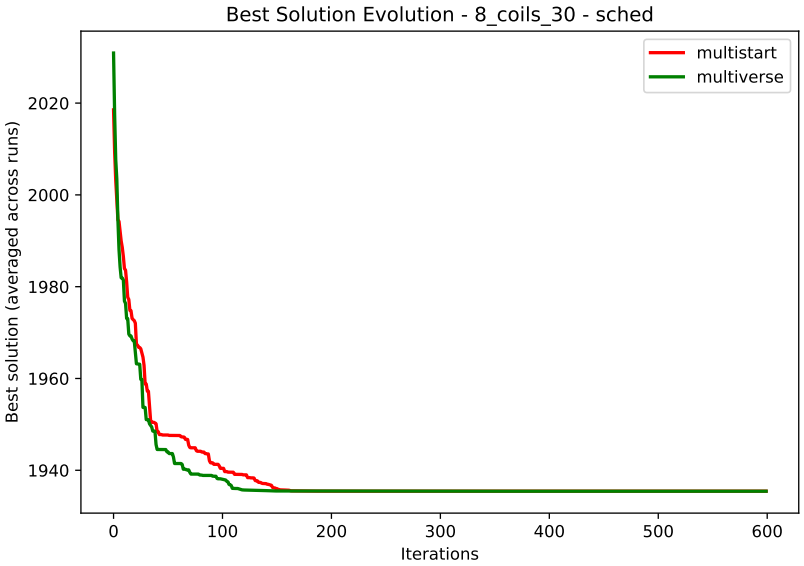

Figure C.41: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 8_coils_30

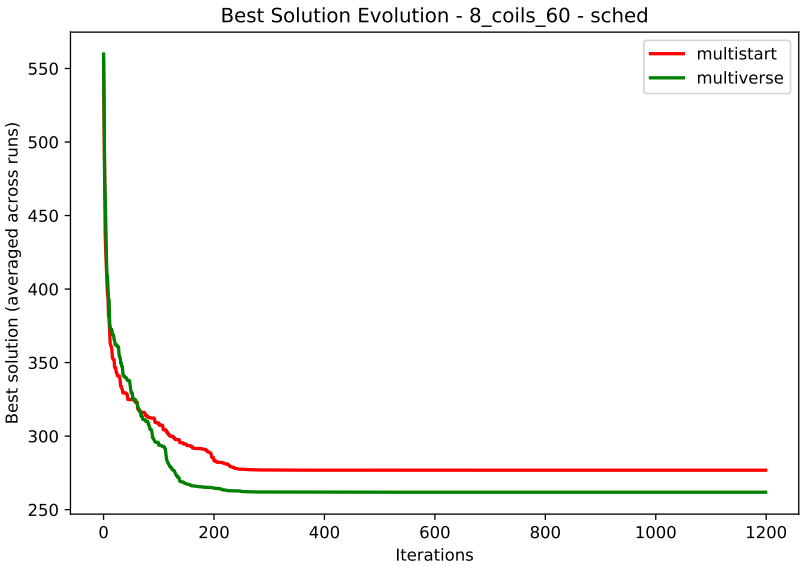

Figure C.42: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 8_coils_60

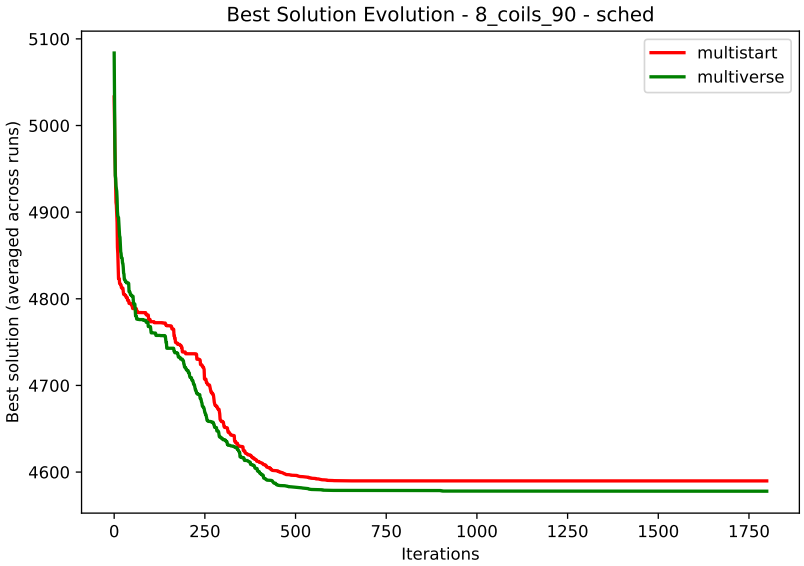

Figure C.43: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 8_coils_90

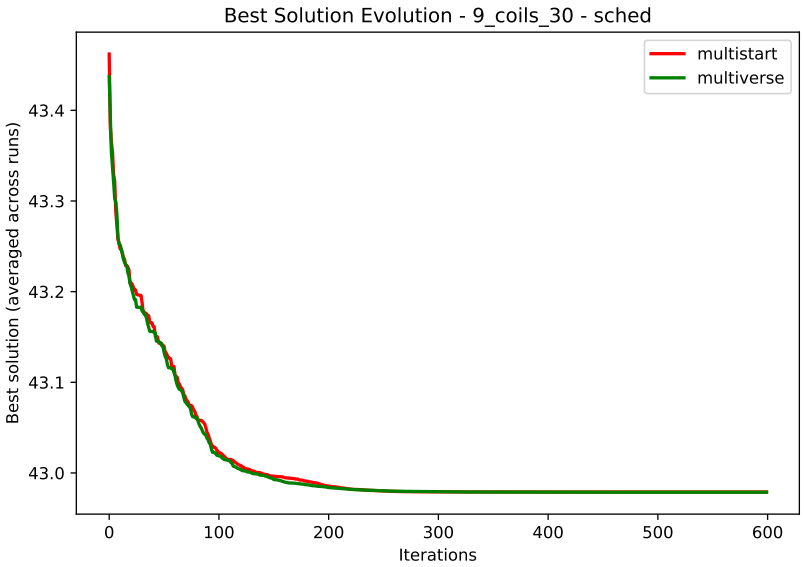

Figure C.44: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 9_coils_30

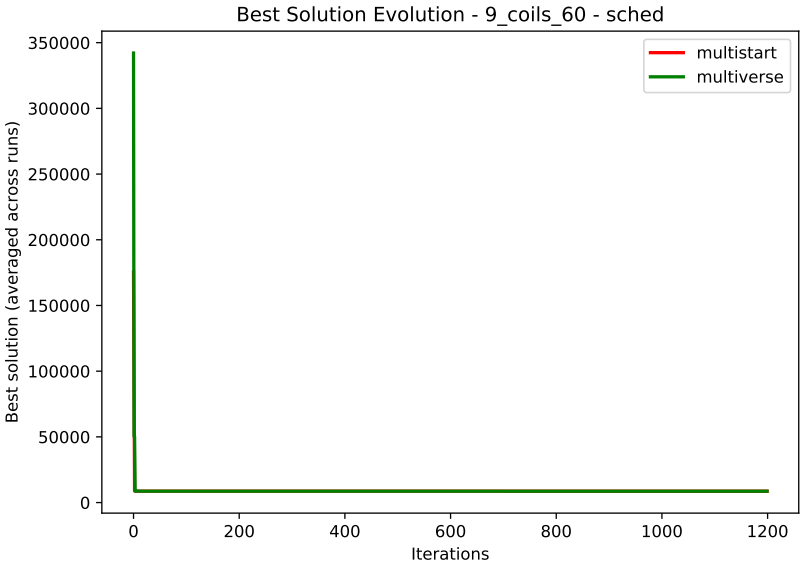

Figure C.45: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 9_coils_60

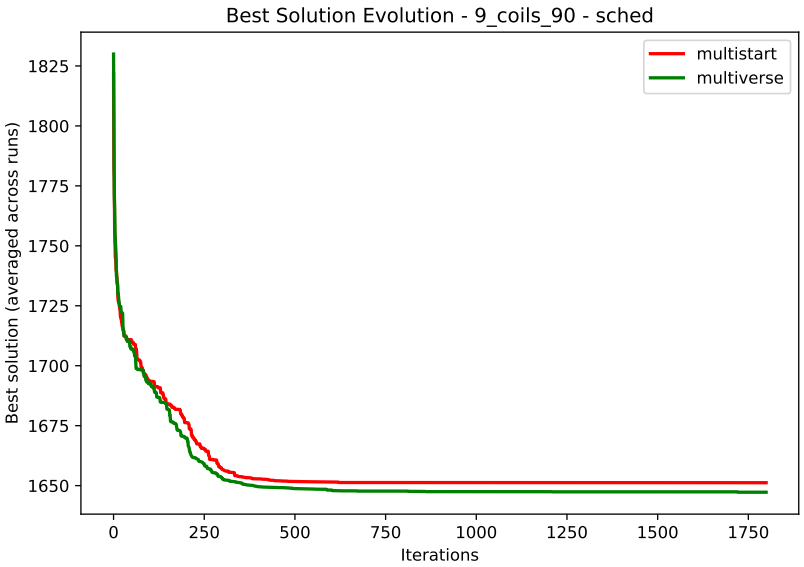

Figure C.46: Evolution of SCHED average best solution versus iteration for Multistart and Multiverse for 9_coils_90

Bibliography

[11] *Third World Congress on Nature & Biologically Inspired Computing, NaBIC 2011, Salamanca, Spain, October 19-21, 2011*. IEEE, 2011. ISBN: 978-1-4577-1122-0.

[ABD17] M. T. Adham, P. J. Bentley, and D. Diaz. "Evaluating Decomposition Strategies to Enable Scalable Scheduling for a Real-World Multi-Line Steel Scheduling Problem". In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017 IEEE Symposium Series on Computational Intelligence (SSCI). Nov. 2017, pp. 1–8. DOI: 10.1109/SSCI.2017.8285185.

[AK89] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY, USA: John Wiley & Sons, Inc., 1989. ISBN: 0-471-92146-7.

[AL06] Enrique Alba and Gabriel Luque. "Evaluation of Parallel Metaheuristics". In: (Jan. 1, 2006).

[Alb02] Enrique Alba. "Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance". In: *Information Processing Letters*.

Evolutionary Computation 82.1 (Apr. 15, 2002), pp. 7–13. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(01)00281-2. URL: https://www.sciencedirect.com/science/article/pii/S0020019001002812.

[Alb05] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, Oct. 3, 2005. 574 pp. ISBN: 978-0-471-73937-1. Google Books: bUJXzkhg5s8C.

[AM11] L. Araujo and J. J. Merelo. "Diversity Through Multiculturality: Assessing Migrant Choice Policies in an Island Model". In: *IEEE Transactions on Evolutionary Computation* 15.4 (Aug. 2011), pp. 456–469. ISSN: 1941-0026. DOI: 10.1109/TEVC.2010.2064322.

[Amd67] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). New York, NY, USA: Association for Computing Machinery, Apr. 18, 1967, pp. 483–485. ISBN: 978-1-4503-7895-6. DOI: 10.1145/1465482.1465560. URL: https://doi.org/10.1145/1465482.1465560.

[ANT02] Enrique Alba, Antonio Nebro, and José Troya. "Heterogeneous Computing and Parallel Genetic Algorithms". In: *Journal of Parallel and Distributed Computing* 62 (Sept. 1, 2002), pp. 1362–1385. DOI: 10.1006/jpdc.2002.1851.

[App+06] David L. Applegate et al. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006. 606 pp. ISBN: 978-0-691-12993-8. Google Books: vhsJbqomDuIC.

[ASG07] I. Alaya, C. Solnon, and K. Ghedira. "Ant Colony Optimization for Multi-Objective Optimization Problems". In: *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*. 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007). Vol. 1. Oct. 2007, pp. 450–457. DOI: 10.1109/ICTAI.2007.108.

[AT01] Enrique Alba and José Troya. "Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms". In: *Future Generation Computer Systems* 17 (Jan. 31, 2001), pp. 451–465. DOI: 10.1016/S0167-739X(99)00129-6.

[AT02] E. Alba and M. Tomassini. "Parallelism and Evolutionary Algorithms". In: *IEEE Transactions on Evolutionary Computation* 6.5 (Oct. 2002), pp. 443–462. ISSN: 1941-0026. DOI: 10.1109/TEVC.2002.800880.

[AT99] Enrique Alba and José Troya. "A Survey of Parallel Distributed Genetic Algorithm". In: *Complexity* 4 (Mar. 1, 1999), pp. 31–52. DOI: 10.1002/(SICI)1099-0526(199903/04)4:43.0.CO;2-4.

[AW09] Daniel Angus and Clinton Woodward. "Multiple Objective Ant Colony Optimisation". In: *Swarm Intelligence* 3.1 (Mar. 1, 2009), pp. 69–85. ISSN: 1935-3820. DOI: 10.1007/s11721-008-0022-4. URL: https://doi.org/10.1007/s11721-008-0022-4.

[BD05] C. Blum and M. Dorigo. "Search Bias in Ant Colony Optimization: On the Role of Competition-Balanced Systems". In: *IEEE Transactions on Evolutionary Computation* 9.2 (Apr. 2005), pp. 159–174. ISSN: 1941-0026. DOI: 10.1109/TEVC.2004.841688.

[BEN14] R. Bellman, A. O. Esogbue, and I. Nabeshima. *Mathematical Aspects of Scheduling and Applications: Modern Applied Mathematics and Computer Science*. Elsevier, May 20, 2014. 345 pp. ISBN: 978-1-4831-3744-5. Google Books: iW_iBQAAQBAJ.

[Ber] Dimitri P Bertsekas. "Network Optimization: Continuous and Discrete Models". In: (), p. 607.

[BFM97] Thomas Bäck, David Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. Jan. 1, 1997. DOI: 10.1887/0750308958.

[Bia+09] Leonora Bianchi et al. "A Survey on Metaheuristics for Stochastic Combinatorial Optimization". In: *Natural Computing: an international journal* 8.2 (June 1, 2009), pp. 239–287. ISSN: 1567-7818. DOI: 10.1007/s11047-008-9098-4. URL: https://doi.org/10.1007/s11047-008-9098-4.

[BKM94] Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. "A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations". In: *Operations Research Letters* 16.2 (Sept. 1, 1994), pp. 101–113. ISSN: 0167-6377. DOI: 10.1016/0167-6377(94)90065-5. URL: https://www.sciencedirect.com/science/article/pii/0167637794900655.

[BKS98] Bernd Bullnheimer, Gabriele Kotsis, and Christine Strauß. "Parallelization Strategies for the Ant System". In: *High Performance Algorithms and Software in Nonlinear Optimization*. Ed. by Renato De Leone et al. Applied Optimization. Boston, MA: Springer US, 1998, pp. 87–100. ISBN: 978-1-4613-3279-4. DOI: 10.1007/978-1-4613-3279-4_6. URL: https://doi.org/10.1007/978-1-4613-3279-4_6.

[Blu+11] Christian Blum et al. "Hybrid Metaheuristics in Combinatorial Optimization: A Survey". In: *Applied Soft Computing* 11.6 (Sept. 1, 2011), pp. 4135–4151. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2011.02.032. URL: https://www.sciencedirect.com/science/article/pii/S1568494611000962.

[Blu04] Christian Blum. *Theoretical and Practical Aspects of Ant Colony Optimization*. IOS Press, 2004. 298 pp. ISBN: 978-3-89838-282-3. Google Books: ut9qw0SUeMkC.

[BR03] Christian Blum and Andrea Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Computing Surveys* 35.3 (Sept. 1, 2003), pp. 268–308. ISSN: 0360-0300. DOI: 10.1145/937503.937505. URL: https://doi.org/10.1145/937503.937505.

[BSZ02] Christian Blum, Michael Sampels, and Mark Zlochin. "On a Particularity in Model-Based Search". In: (Oct. 6, 2002).

[Can00] Erick Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Springer Science & Business Media, Nov. 30, 2000. 192 pp. ISBN: 978-0-7923-7221-9. Google Books: UXkGGQbmsfAC.

[Car] Gianni Di Caro. "Ant Colony Optimization and Its Application to Adaptive Routing in Telecommunication Networks". In: (), p. 374.

[Cec+13] José M. Cecilia et al. "Enhancing Data Parallelism for Ant Colony Optimization on GPUs". In: *Journal of Parallel and Distributed Computing*. Metaheuristics on GPUs 73.1 (Jan. 1, 2013), pp. 42–51. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2012.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0743731512000032.

[CG97] Erick Cantu-Paz and David Goldberg. "Predicting Speedups of Ideal Bounding Cases of Parallel Genetic Algorithms." In: Jan. 1, 1997, pp. 113–120.

[CH05] Teodor Gabriel Crainic and Nourredine Hail. "Parallel Metaheuristics Applications". In: *Parallel Metaheuristics*. John Wiley & Sons, Ltd, 2005, pp. 447–494. ISBN: 978-0-471-73938-8. DOI: 10.1002/0471739383.ch19. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471739383.ch19.

[CHS02] Oscar Cordon, Francisco Herrera, and Thomas Stützle. "Special Issue on Ant Colony Optimization". In: *Mathware & soft computing* 4.2-3 (2002). ISSN: 1134-5632. URL: http://hdl.handle.net/2013/.

[CJM07] A. Catala, J. Jaen, and J. A. Mocholi. "Strategies for Accelerating Ant Colony Optimization Algorithms on Graphical Processing Units". In: *2007 IEEE Congress on Evolutionary Computation*. 2007 IEEE Congress on Evolutionary Computation. Sept. 2007, pp. 492–500. DOI: 10.1109/CEC.2007.4424511.

[CJP08] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2008. 378 pp. ISBN: 978-0-262-53302-7. Google Books: MeFLQSKmaJYC.

[Cra19] Teodor Crainic. "Parallel Metaheuristics and Cooperative Search". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Cham: Springer International Publishing, 2019, pp. 419–451. ISBN: 978-3-319-91086-4. DOI: 10.1007/978-3-319-91086-4_13. URL: https://doi.org/10.1007/978-3-319-91086-4_13.

[CT10] Teodor Gabriel Crainic and Michel Toulouse. "Parallel Meta-Heuristics". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 497–541. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_17. URL: https://doi.org/10.1007/978-1-4419-1665-5_17.

[Cun+02] Van-Dat Cung et al. “Strategies for the Parallel Implementation of Metaheuristics”. In: *Essays and Surveys in Metaheuristics*. Ed. by Celso C. Ribeiro and Pierre Hansen. Operations Research/-Computer Science Interfaces Series. Boston, MA: Springer US, 2002, pp. 263–308. ISBN: 978-1-4615-1507-4. DOI: 10.1007/978-1-4615-1507-4_13. URL: https://doi.org/10.1007/978-1-4615-1507-4_13.

[CZ05] Ling Chen and Chunfang Zhang. “Adaptive Parallel Ant Colony Algorithm”. In: *Advances in Natural Computation*. Ed. by Lipo Wang, Ke Chen, and Yew Soon Ong. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 1239–1249. ISBN: 978-3-540-31858-3. DOI: 10.1007/11539117_165.

[DB05] Marco Dorigo and Christian Blum. “Ant Colony Optimization Theory: A Survey”. In: *Theoretical Computer Science* 344.2 (Nov. 17, 2005), pp. 243–278. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.05.020. URL: https://www.sciencedirect.com/science/article/pii/S0304397505003798.

[DBP99] V. Donaldson, Francine Berman, and R. Paturi. “Program Speedup in a Heterogeneous Computing Network”. In: *Journal of Parallel and Distributed Computing* 21 (Jan. 2, 1999), pp. 316–322. DOI: 10.1006/jpdc.1994.1062.

[DDG99] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. “Ant Algorithms for Discrete Optimization”. In: *Artificial Life* 5.2

(1999), pp. 137–172. DOI: http://dx.doi.org/10.1162/106454699568728.

[Deb08] Kalyanmoy Deb. "Introduction to Evolutionary Multiobjective Optimization". In: *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Ed. by Jürgen Branke et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 59–96. ISBN: 978-3-540-88908-3. DOI: 10.1007/978-3-540-88908-3_3. URL: https://doi.org/10.1007/978-3-540-88908-3_3.

[Deb11] Kalyanmoy Deb. "Multi-Objective Optimisation Using Evolutionary Algorithms: An Introduction". In: *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*. Ed. by Lihui Wang, Amos H. C. Ng, and Kalyanmoy Deb. London: Springer, 2011, pp. 3–34. ISBN: 978-0-85729-652-8. DOI: 10.1007/978-0-85729-652-8_1. URL: https://doi.org/10.1007/978-0-85729-652-8_1.

[Del+] Pierre Delisle et al. "PARALLEL IMPLEMENTATION OF AN ANT COLONY OPTIMIZATION METAHEURISTIC WITH OPENMP". In: (), p. 7.

[Del+05] Pierre Delisle et al. "Comparing Parallelization of an ACO: Message Passing vs. Shared Memory". In: *Hybrid Metaheuristics*. Ed. by María J. Blesa et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 1–11. ISBN: 978-3-540-31898-9. DOI: 10.1007/11546245_1.

[Del+13] Audrey Delévacq et al. "Parallel Ant Colony Optimization on Graphics Processing Units". In: *Journal of Parallel and Distributed Computing* 73.1 (2013), pp. 52–61. ISSN: 0743-7315. DOI: http://dx.doi.org/10.1016/j.jpdc.2012.01.003. URL: http://www.sciencedirect.com/science/article/pii/S0743731512000044.

[DG97a] M. Dorigo and L. M. Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 53–66. ISSN: 1941-0026. DOI: 10.1109/4235.585892.

[DG97b] Marco Dorigo and Luca Maria Gambardella. "Ant Colonies for the Travelling Salesman Problem". In: *Biosystems* 43.2 (July 1, 1997), pp. 73–81. ISSN: 0303-2647. DOI: 10.1016/S0303-2647(97)01708-5. URL: https://www.sciencedirect.com/science/article/pii/S0303264797017085.

[Día+14] Diego Díaz et al. "An ACO Algorithm to Solve an Extended Cutting Stock Problem for Scrap Minimization in a Bar Mill". In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 13–24. ISBN: 978-3-319-09952-1. DOI: 10.1007/978-3-319-09952-1_2.

[Día+20] Diego Díaz et al. "Improved Method for Parallelization of Evolutionary Metaheuristics". In: *Mathematics* 8.9 (9 Sept. 2020),

p. 1476. DOI: 10.3390/math8091476. URL: https://www.mdpi.com/2227-7390/8/9/1476.

[Dor+14] Marco Dorigo et al., eds. *Swarm Intelligence - 9th International Conference, ANTS 2014, Brussels, Belgium, September 10-12, 2014. Proceedings*. Vol. 8667. Lecture Notes in Computer Science. Springer, 2014. ISBN: 978-3-319-09951-4. DOI: 10.1007/978-3-319-09952-1. URL: http://dx.doi.org/10.1007/978-3-319-09952-1.

[Dor92] Marco Dorigo. "Optimization, Learning and Natural Algorithms (in Italian)". PhD thesis. Milan, Italy: Dipartimento di Elettronica, Politecnico di Milano, 1992.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004. ISBN: 0-262-04219-3.

[DS13] L. Dawson and I. Stewart. "Improving Ant Colony Optimization Performance on the GPU Using CUDA". In: *2013 IEEE Congress on Evolutionary Computation*. 2013 IEEE Congress on Evolutionary Computation. June 2013, pp. 1901–1908. DOI: 10.1109/CEC.2013.6557791.

[DS19] Marco Dorigo and Thomas Stützle. "Ant Colony Optimization: Overview and Recent Advances". In: *Handbook of Metaheuristics*. 2019, pp. 311–351. DOI: 10.1007/978-3-319-91086-4_10. URL: https://app.dimensions.ai/details/publication/pub.1107124659%20and%20http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2009-013r001.pdf.

[ECB07] Issmail Ellabib, Paul Calamai, and Otman Basir. "Exchange Strategies for Multiple Ant Colony System". In: *Information Sciences*. Including: The 3rd International Workshop on Computational Intelligence in Economics and Finance (CIEF'2003) 177.5 (Mar. 1, 2007), pp. 1248–1264. ISSN: 0020-0255. DOI: 10.1016/j.ins.2006.09.016. URL: https://www.sciencedirect.com/science/article/pii/S0020025506002908.

[Ekl04] Sven E. Eklund. "A Massively Parallel Architecture for Distributed Genetic Algorithms". In: *Parallel Computing*. Parallel and Nature-Inspired Computational Paradigms and Applications 30.5 (May 1, 2004), pp. 647–676. ISSN: 0167-8191. DOI: 10.1016/j.parco.2003.12.009. URL: https://www.sciencedirect.com/science/article/pii/S0167819104000365.

[Fer+14] Silvino Fernandez et al. "Scheduling a Galvanizing Line by Ant Colony Optimization". In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 146–157. ISBN: 978-3-319-09952-1. DOI: 10.1007/978-3-319-09952-1_13.

[Fer+15] Silvino Fernández et al. "Performance Comparison of Ant Colony Algorithms for the Scheduling of Steel Production Lines". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO Companion '15. New York, NY, USA: Association for Computing Machinery, July 11, 2015, pp. 1387–1388. ISBN: 978-1-4503-3488-

4. DOI: 10.1145/2739482.2764658. URL: https://doi.org/10.1145/2739482.2764658.

[Fer+16] Silvino Fernández et al. "Criticality of Response Time in the Usage of Metaheuristics in Industry". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. GECCO '16 Companion. New York, NY, USA: Association for Computing Machinery, July 20, 2016, pp. 937–940. ISBN: 978-1-4503-4323-7. DOI: 10.1145/2908961.2931649. URL: https://doi.org/10.1145/2908961.2931649.

[GCH07] C. García-Martínez, O. Cordón, and F. Herrera. "A Taxonomy and an Empirical Analysis of Multiple Objective Ant Colony Optimization Algorithms for the Bi-Criteria TSP". In: *European Journal of Operational Research* 180.1 (July 1, 2007), pp. 116–148. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2006.03.041. URL: https://www.sciencedirect.com/science/article/pii/S0377221706002451.

[GD96] L. M. Gambardella and M. Dorigo. "Solving Symmetric and Asymmetric TSPs by Ant Colonies". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. Proceedings of IEEE International Conference on Evolutionary Computation. May 1996, pp. 622–627. DOI: 10.1109/ICEC.1996.542672.

[GLM00] F. Glover, M. Laguna, and R. Marti. “Fundamentals of Scatter Search and Path Relinking”. In: *Control and Cybernetics* 29.3 (2000), pp. 653–684.

[GM02] Michael Guntsch and Martin Middendorf. “A Population Based Approach for ACO”. In: *Applications of Evolutionary Computing*. Ed. by Stefano Cagnoni et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 72–81. ISBN: 978-3-540-46004-6. DOI: 10.1007/3-540-46004-7_8.

[GP10] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. 2nd Edition. Springer Publishing Company, Incorporated, 2010. ISBN: 1-4419-1663-6 978-1-4419-1663-1.

[GP19] Michel Gendreau and Jean-Yves Potvin, eds. *Handbook of Metaheuristics*. 3rd ed. International Series in Operations Research & Management Science. Springer International Publishing, 2019. ISBN: 978-3-319-91085-7. DOI: 10.1007/978-3-319-91086-4. URL: https://www.springer.com/gp/book/9783319910857.

[GP78] Martin Grötschel and Manfred W. Padberg. “On the Symmetric Travelling Salesman Problem: Theory and Computation”. In: *Optimization and Operations Research*. Ed. by Rudolf Henn, Bernhard Korte, and Werner Oettli. Lecture Notes in Economics and Mathematical Systems. Berlin, Heidelberg: Springer, 1978, pp. 105–115. ISBN: 978-3-642-95322-4. DOI: 10.1007/978-3-642-95322-4_12.

[Gro+99] William Gropp et al. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999. 414 pp. ISBN: 978-0-262-57132-6.

[Gue+14] Ginés D. Guerrero et al. "Comparative Evaluation of Platforms for Parallel Ant Colony Optimization". In: *The Journal of Supercomputing* 69.1 (July 1, 2014), pp. 318–329. ISSN: 1573-0484. DOI: 10.1007/s11227-014-1154-5. URL: https://doi.org/10.1007/s11227-014-1154-5.

[Har+06] A. Hara et al. "Effective Diversification of Ant-Based Search Using Colony Fission and Extinction". In: *2006 IEEE International Conference on Evolutionary Computation*. 2006 IEEE International Conference on Evolutionary Computation. July 2006, pp. 1028–1035. DOI: 10.1109/CEC.2006.1688422.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press (re-issued 1992), 1975. ISBN: 0-262-08213-6.

[Igl+19] Miguel Iglesias-Escudero et al. "Planning and Scheduling with Uncertainty in the Steel Sector: A Review". In: *Applied Sciences* 9.13 (13 Jan. 2019), p. 2692. DOI: 10.3390/app9132692. URL: https://www.mdpi.com/2076-3417/9/13/2692.

[JMM05] Stefan Janson, Daniel Merkle, and Martin Middendorf. "Parallel Ant Colony Algorithms". In: *Parallel Metaheuristics: A New*

Class of Algorithms. Ed. by Enrique Alba. Wiley, 2005, pp. 171–201. DOI: 10.1002/0471739383.ch8.

[JSV91] P. Jog, J. Y. Suh, and D. Van Gucht. "Parallel Genetic Algorithms Applied to the Traveling Salesman Problem". In: *SIAM Journal of Optimization* 1 (1991), pp. 515–529.

[JTS10] Raka Jovanovic, Milan Tuba, and Dana Simian. "Comparison of Different Topologies for Island-Based Multi-Colony Ant Algorithms for the Minimum Weight Vertex Cover Problem". In: 9.1 (2010), p. 10.

[Kaw+00] Hidenori Kawamura et al. "Multiple Ant Colonies Algorithm Based on Colony Level Interactions". In: *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* E83-A.2 (Feb. 25, 2000), pp. 371–379. ISSN: , 0916-8508. URL: https://search.ieice.org/bin/summary.php?id=e83-a_2_371&category=A&year=2000&lang=E&abst=.

[KGV83] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. "Optimization by Simulated Annealing". In: *Science (New York, N.Y.)* 220 (June 1, 1983), pp. 671–80. DOI: 10.1126/science.220.4598.671.

[KLQ14] B. Kazimipour, X. Li, and A. K. Qin. "A Review of Population Initialization Techniques for Evolutionary Algorithms". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. 2014 IEEE Congress on Evolutionary Computation (CEC). July 2014, pp. 2585–2592. DOI: 10.1109/CEC.2014.6900618.

[KPS12] Pavel Krömer, Jan Platos, and Václav Snásel. "Evolutionary Clustering on CUDA". In: *ECAI*. 2012, pp. 909–910.

[Krö+11] Pavel Krömer et al. "A Comparison of Many-Threaded Differential Evolution and Genetic Algorithms on CUDA". In: *NaBIC*. 2011, pp. 509–514.

[Lar+99] P. Larrañaga et al. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". In: *Artificial Intelligence Review* 13.2 (Apr. 1, 1999), pp. 129–170. ISSN: 1573-7462. DOI: 10.1023/A:1006529012972. URL: https://doi.org/10.1023/A:1006529012972.

[Lóp+16] Manuel López-Ibáñez et al. "The Irace Package: Iterated Racing for Automatic Algorithm Configuration". In: *Operations Research Perspectives* 3 (Jan. 1, 2016), pp. 43–58. ISSN: 2214-7160. DOI: 10.1016/j.orp.2016.09.002. URL: https://www.sciencedirect.com/science/article/pii/S2214716015300270.

[LS09] Maria Lucka and Piecka Stanislav. "Parallel Posix Threads Based Ant Colony Optimization Using Asynchronous Communication". In: *Proceedings of the 8th International Conference on Applied Mathematics* 2 (Jan. 1, 2009).

[LS12a] M. Lopez-Ibanez and T. Stutzle. "The Automatic Design of Multiobjective Ant Colony Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 16.6 (Dec. 2012), pp. 861–875. ISSN: 1941-0026. DOI: 10.1109/TEVC.2011.2182651.

[LS12b] Manuel López-Ibáñez and Thomas Stützle. "An Experimental Analysis of Design Choices of Multi-Objective Ant Colony Optimization Algorithms". In: *Swarm Intelligence* 6.3 (Sept. 1, 2012), pp. 207–232. ISSN: 1935-3820. DOI: 10.1007/s11721-012-0070-7. URL: https://doi.org/10.1007/s11721-012-0070-7.

[LS69] C. F. Long and J. D. Schoeffler. "Dynamic Scheduling in the Process Industries by Predictive Control". In: *Automatica* 5.2 (Mar. 1, 1969), pp. 235–238. ISSN: 0005-1098. DOI: 10.1016/0005-1098(69)90017-X. URL: https://www.sciencedirect.com/science/article/pii/000510986990017X.

[Man+06] Max Manfrin et al. "Parallel Ant Colony Optimization for the Traveling Salesman Problem". In: *Ant Colony Optimization and Swarm Intelligence*. Ed. by Marco Dorigo et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 224–234. ISBN: 978-3-540-38483-0. DOI: 10.1007/11839088_20.

[MC03] Pablo Moscato and Carlos Cotta. "A Gentle Introduction to Memetic Algorithms". In: *Handbook of Metaheuristics*. Ed. by Fred Glover and Gary A. Kochenberger. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2003, pp. 105–144. ISBN: 978-0-306-48056-0. DOI: 10.1007/0-306-48056-5_5. URL: https://doi.org/10.1007/0-306-48056-5_5.

[MLY17] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. "A Survey of Swarm Intelligence for Dynamic Optimization: Algo-

rithms and Applications". In: *Swarm and Evolutionary Computation* 33 (Apr. 1, 2017), pp. 1–17. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2016.12.005. URL: https://www.sciencedirect.com/science/article/pii/S2210650216302541.

[MO99] R. Maclin and D. Opitz. "Popular Ensemble Methods: An Empirical Study". In: *Journal of Artificial Intelligence Research* 11 (Aug. 1, 1999), pp. 169–198. ISSN: 1076-9757. DOI: 10.1613/jair.614. arXiv: 1106.0257. URL: http://arxiv.org/abs/1106.0257.

[MRS02] Martin Middendorf, Frank Reischle, and Hartmut Schmeck. "Multi Colony Ant Algorithms". In: *Journal of Heuristics* 8.3 (May 1, 2002), pp. 305–320. ISSN: 1572-9397. DOI: 10.1023/A:1015057701750. URL: https://doi.org/10.1023/A:1015057701750.

[MS10] R. Mallipeddi and P. N. Suganthan. "Ensemble of Constraint Handling Techniques". In: *IEEE Transactions on Evolutionary Computation* 14.4 (Aug. 2010), pp. 561–579. ISSN: 1941-0026. DOI: 10.1109/TEVC.2009.2033582.

[MSB91] H. Muhlenbein, M. Schomisch, and J. Born. "The Parallel Genetic Algorithm as Function Optimizer". In: *Proceedings on an International Conference on Genetic Algorithms*. 1991.

[ND10] J. Nickolls and W. Dally. "The GPU Computing Era". In: *IEEE Micro* (2010). DOI: 10.1109/MM.2010.41.

[OSH87] I. M. Oliver, D. J. Smith, and J. R. C. Holland. "Study of Permutation Crossover Operators on the Traveling Salesman Problem".

In: *Genetic algorithms and their applications : proceedings of the second International Conference on Genetic Algorithms : July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA* (1987). URL: https://agris.fao.org/agris-search/search.do?recordID=US201301782179.

[PL02] D. A. L. Piriyakumar and P. Levi. "A New Approach to Exploiting Parallelism in Ant Colony Optimization". In: *Proceedings of 2002 International Symposium on Micromechatronics and Human Science*. Proceedings of 2002 International Symposium on Micromechatronics and Human Science. Oct. 2002, pp. 237–243. DOI: 10.1109/MHS.2002.1058041.

[PLG87] Chrisila B. Pettey, Michael R. Leuze, and John J. Grefenstette. "A Parallel Genetic Algorithm". In: *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. USA: L. Erlbaum Associates Inc., Oct. 1, 1987, pp. 155–161. ISBN: 978-0-8058-0158-3.

[PLS14] Leslie Pérez Cáceres, Manuel López-Ibáñez, and Thomas Stützle. "Ant Colony Optimization on a Budget of 1000". In: *Swarm Intelligence - 9th International Conference, ANTS 2014, Brussels, Belgium, September 10-12, 2014. Proceedings*. 2014, pp. 50–61. DOI: 10.1007/978-3-319-09952-1_5. URL: http://dx.doi.org/10.1007/978-3-319-09952-1_5.

[PNC11] Martín Pedemonte, Sergio Nesmachnow, and Héctor Cancela. "A Survey on Parallel Ant Colony Optimization". In: *Applied*

Soft Computing 11.8 (Dec. 1, 2011), pp. 5181–5197. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2011.05.042. URL: https://www.sciencedirect.com/science/article/pii/S156849461100202X.

[PT99] P. Preux and E.-G. Talbi. "Towards Hybrid Evolutionary Algorithms". In: *International Transactions in Operational Research* 6.6 (1999), pp. 557–570. ISSN: 1475-3995. DOI: 10.1111/j.1475-3995.1999.tb00173.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-3995.1999.tb00173.x.

[Pun98] William Punch. "How Effective Are Multiple Populations in Genetic Programming". In: *Genetic Programming* 98 (Jan. 1, 1998).

[Rae+12] Luc De Raedt et al., eds. *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012. ISBN: 978-1-61499-097-0.

[Rai06] Günther R. Raidl. "A Unified View on Hybrid Metaheuristics". In: *Hybrid Metaheuristics*. Ed. by Francisco Almeida et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 1–12. ISBN: 978-3-540-46385-6. DOI: 10.1007/11890584_1.

[Ree10] Colin Reeves. "Genetic Algorithms". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. 2nd Edi-

tion. Springer Publishing Company, Inc., 2010, pp. 55–82. ISBN: 1-4419-1663-6 978-1-4419-1663-1.

[Rei91] Gerhard Reinelt. "TSPLIB—A Traveling Salesman Problem Library". In: *INFORMS Journal on Computing* 3.4 (Nov. 1991), pp. 376–384. DOI: 10.1287/ijoc.3.4.376. URL: https://ideas.repec.org/a/inm/orijoc/v3y1991i4p376-384.html.

[RL02] Marcus Randall and Andrew Lewis. "A Parallel Implementation of Ant Colony Optimization". In: *Journal of Parallel and Distributed Computing* 62.9 (Sept. 1, 2002), pp. 1421–1432. ISSN: 0743-7315. DOI: 10.1006/jpdc.2002.1854. URL: https://www.sciencedirect.com/science/article/pii/S074373150291854X.

[Rob49] J. B. Robinson. *On the Hamiltonian Game (a Traveling-Salesman Problem).* Santa Monica, CA: RAND Corporation, 1949.

[RPE99] Mauricio G.C. Resende, Panos M. Pardalos, and Sandra Duni Ekscanioglu. "Parallel Metaheuristics for Combinatorial Optimization". In: *International School on Advanced Algorithmic Techniques for Parallel Computations with Applications* (1999).

[Ryo+08] Shane Ryoo et al. *Program Optimization Carving For . . .* 2008.

[Sal09] Carolina Salto. "Metaheurísticas Híbridas Paralelas Para Problemas de Corte, Empaquetado y Otros Relacionados". PhD thesis. San Luis, Argentina: Universidad Nacional de San Luis, 2009.

[SH97] T. Stutzle and H. Hoos. "MAX-MIN Ant System and Local Search for the Traveling Salesman Problem". In: *Proceedings of*

1997 IEEE International Conference on Evolutionary Computation (ICEC '97). Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97). Apr. 1997, pp. 309–314. DOI: 10.1109/ICEC.1997.592327.

[Sho93] Ron Shonkwiler. "Parallel Genetic Algorithms." In: Proceedings of the Fifth International Conference on Genetic Algorithms. Jan. 1, 1993, pp. 199–205.

[Stü98] Thomas Stützle. "Parallelization Strategies for Ant Colony Optimization". In: *Parallel Problem Solving from Nature — PPSN V*. Ed. by Agoston E. Eiben et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 722–731. ISBN: 978-3-540-49672-4. DOI: 10.1007/BFb0056914.

[Sut05] Herb Sutter. *Software and the Concurrency Revolution*. ACM, 2005. URL: http://queue.acm.org/detail.cfm?id=1095421.

[Tal09] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, May 27, 2009. 625 pp. ISBN: 978-0-470-49690-9. Google Books: SIsa6zi5XV8C.

[Tej+17] Enric Tejedor et al. "PyCOMPSs: Parallel Computational Workflows in Python". In: *The International Journal of High Performance Computing Applications* 31.1 (Jan. 1, 2017), pp. 66–82. ISSN: 1094-3420. DOI: 10.1177/1094342015594678. URL: https://doi.org/10.1177/1094342015594678.

[Two+10] C. Twomey et al. "An Analysis of Communication Policies for Homogeneous Multi-Colony ACO Algorithms". In: *Information*

Sciences 180.12 (June 15, 2010), pp. 2390–2404. ISSN: 0020-0255. DOI: 10.1016/j.ins.2010.02.017. URL: https://www.sciencedirect.com/science/article/pii/S0020025510000824.

[Van11] Thé Van Luong. "Parallel Metaheuristics on GPU". PhD thesis. Université Lille1, 2011.

[Wan+20] Yu Wang et al. "Multi-Objective Optimization of Rolling Schedule for Tandem Cold Strip Rolling Based on NSGA-II". In: *Journal of Manufacturing Processes* 60 (Dec. 1, 2020), pp. 257–267. ISSN: 1526-6125. DOI: 10.1016/j.jmapro.2020.10.061. URL: http://www.sciencedirect.com/science/article/pii/S1526612520307416.

[Whi19] Darrell Whitley. "Next Generation Genetic Algorithms: A User's Guide and Tutorial". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Cham: Springer International Publishing, 2019, pp. 245–274. ISBN: 978-3-319-91086-4. DOI: 10.1007/978-3-319-91086-4_8. URL: https://doi.org/10.1007/978-3-319-91086-4_8.