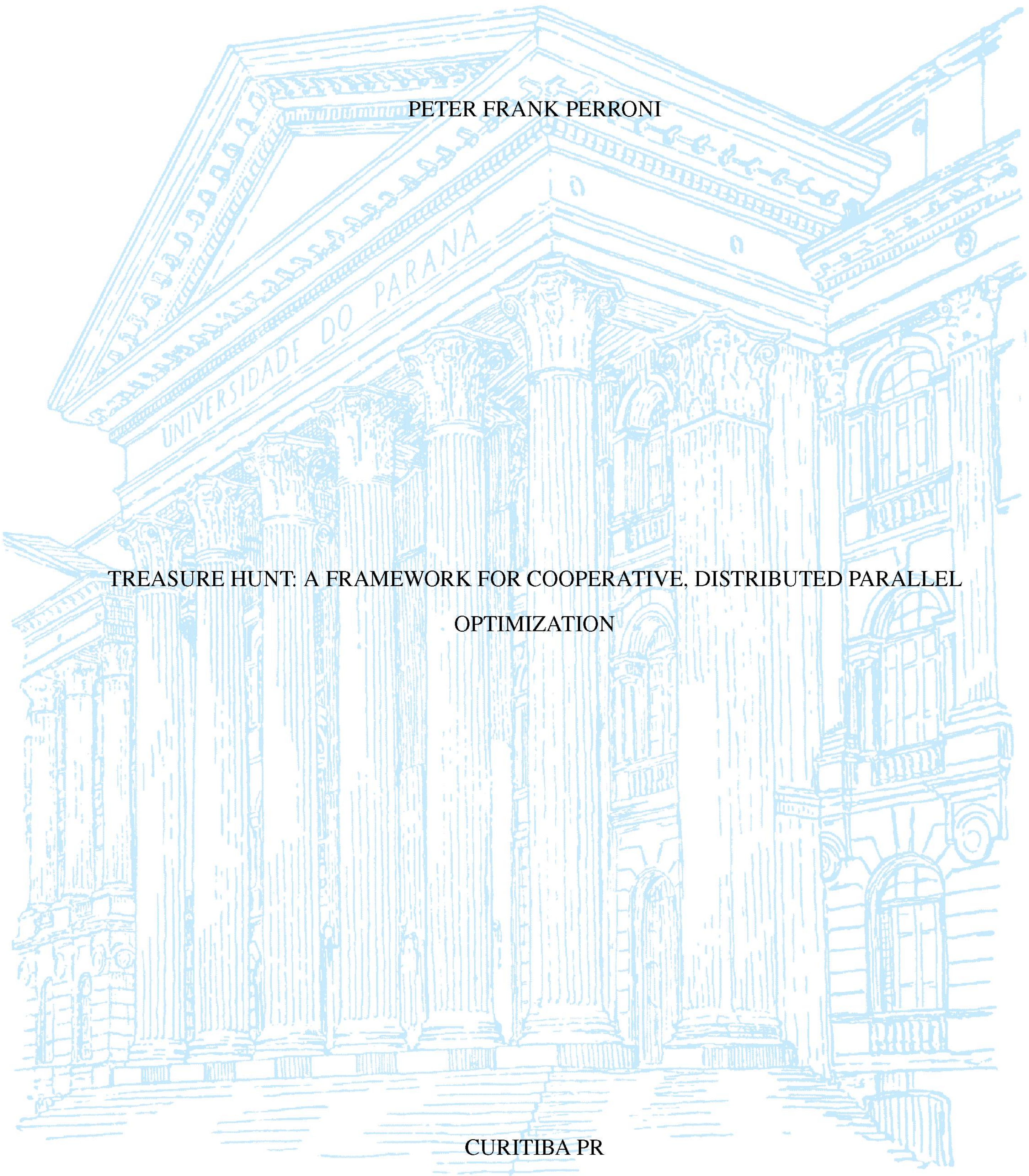
UNIVERSIDADE FEDERAL DO PARANÁ

PETER FRANK PERRONI

TREASURE HUNT: A FRAMEWORK FOR COOPERATIVE, DISTRIBUTED PARALLEL OPTIMIZATION

CURITIBA PR

2019



PETER FRANK PERRONI

TREASURE HUNT: A FRAMEWORK FOR COOPERATIVE, DISTRIBUTED PARALLEL OPTIMIZATION

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Daniel Weingaertner.

Coorientadora: Profa. Dra. Myriam Regattieri Delgado.

CURITIBA PR

2019

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

P459t Perroni, Peter Frank
Treasure hunt: a framework for cooperative, distributed parallel optimization [Recurso eletrônico] / Peter Frank Perroni – Curitiba, 2019.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Daniel Weingaertner
Coorientadora: Myriam Regattieri Delgado.

1. Inteligência artificial. 2. Algoritmos. 3. Métodos de otimização.(Informática). I. Universidade Federal do Paraná. II. Weingaertner, Daniel. III. Delgado, Myriam Regattieri. IV. Título.

CDD: 006.3

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **PETER FRANK PERRONI** intitulada: **Treasure Hunt: A Framework for Cooperative, Distributed Parallel Optimization**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 27 de Maio de 2019.

[illegible]
DANIEL WEINGAERTNER
Presidente da Banca Examinadora (UFPR)

[illegible]
LUIZ EDUARDO SOARES DE OLIVEIRA
Avaliador Interno (UFPR)

[illegible]
RICARDO LUDERS
Avaliador Externo (UTFPR)

[illegible]
GINA MAIRA BARBOSA DE OLIVEIRA
Avaliador Externo (UFU)

Chaos happens and makes things go wrong.

Science happens and makes chaos go wrong.

Things happen and make science go wrong.

Make things right, never give up from science,

Because today is not just consequence

from yesterday's decisions,

but from tomorrow's plannings.

For my grandfather, Sebastião Saltarello.
(in memoriam)

AGRADECIMENTOS

Em primeiro lugar, agradeço à minha família, que é a verdadeira razão de tudo. Agradeço à minha esposa, Patricia, por seu incentivo para que eu voltasse ao mundo acadêmico e científico, e por seu apoio irrestrito que permitiu que eu me mantivesse nas longas jornadas triplas diárias. Agradeço a todos os meus filhos, pelo seu amor incondicional. Amo todos vocês, incondicionalmente. Agradeço às minhas filhas Pamela e Luana, por terem sido a bússola que colocou um rumo na minha vida, e por terem se tornado essas pessoas tão maravilhosas que vocês são. Um agradecimento especial ao meu filho Miguel, por ter sido um guerreiro de fibra, e por sua incansável luta até os últimos minutos que proporcionou toda uma vida às suas irmãs. Agradeço ao meu filho Giovanni, por ter reacendido a chama da união familiar que a pesada rotina diária havia esfriado. Agradeço à minha mãe, Clarice, que me ensinou o valor do trabalho honesto, e por todo o seu esforço que possibilitou eu ser quem eu sou hoje. Agradeço ao meu pai, Mauro, por ter me ensinado a delicada arte da longa paciência. Agradeço à minha irmã, Vanessa, e ao meu irmão, Patrick, simplesmente por existirem na minha vida (saudades do meu irmão Alexandre, in memoriam). Agradeço ao meu avô, Sebastião, por ter sido meu único e verdadeiro herói da infância e da vida adulta, um exemplo inabalável da luta pelo que é certo apenas porque é o certo a ser fazer. E agradeço à minha sogra Amélia e sogro Vilson, por todo o suporte logístico ao longo desta jornada, e às minhas cunhadas, praticamente irmãs postiças, Elizangela e Alexsandra, pelo frequente apoio moral que sempre me ajudou a manter o ânimo elevado.

Agradeço ao meu orientador, Daniel, por ter aceito o desafio que foi me orientar nesta tese. Agradeço também por ter me aceito como mestrando, há algum tempo atrás, e por todas as outras vezes em que me estendeu a mão nos momentos de maiores dificuldades. Agradeço à minha coorientadora, Myriam, por ter acompanhado de perto a trajetória de todo o trabalho desenvolvido nesta tese, pelas suas incríveis revisões que tanto me ensinaram, e pela sua amizade. Agradeço tanto ao Daniel quanto à Myriam, por toda a paciência que tiveram para me ensinar o pensamento científico. E agradeço ao professor Maziero, por sua boa memória e grande consideração, que me auxiliaram a voltar ao mundo acadêmico.

Agradeço a todos os meus amigos de doutorado, pelas infindáveis horas que passamos juntos, e por toda a diversão que tivemos nas lendárias "*horas do café*". Agradeço aos técnicos administrativos do Departamento de Informática, por sua assistência e amizade ao longo de todos estes anos.

E por fim, e mais importante, agradeço à Grande Força Infinita do Universo, que me proporcionou toda essa grande emoção que é estar aqui e poder participar de tudo isso.

Obrigado !

RESUMO

Este trabalho propõe um framework multinível chamado Treasure Hunt, que é capaz de distribuir algoritmos de busca independentes para um grande número de nós de processamento. Com o objetivo de obter uma convergência conjunta entre os nós, este framework propõe um mecanismo de direcionamento que controla suavemente a cooperação entre múltiplas instâncias independentes do Treasure Hunt. A topologia em árvore proposta pelo Treasure Hunt garante a rápida propagação da informação pelos nós, ao mesmo tempo em que provê simutaneamente explorações (pelos nós-pai) e intensificações (pelos nós-filho), em vários níveis de granularidade, independentemente do número de nós na árvore. O Treasure Hunt tem boa tolerância à falhas e está parcialmente preparado para uma total tolerância à falhas. Como parte dos métodos desenvolvidos durante este trabalho, um método automatizado de Particionamento Iterativo foi proposto para controlar o balanceamento entre explorações e intensificações ao longo da busca. Uma Modelagem de Estabilização de Convergência para operar em modo Online também foi proposto, com o objetivo de encontrar pontos de parada com bom *custo/benefício* para os algoritmos de otimização que executam dentro das instâncias do Treasure Hunt. Experimentos em benchmarks clássicos, aleatórios e de competição, de vários tamanhos e complexidades, usando os algoritmos de busca PSO, DE e CCPSO2, mostram que o Treasure Hunt melhora as características inerentes destes algoritmos de busca. O Treasure Hunt faz com que os algoritmos de baixa performance se tornem comparáveis aos de boa performance, e os algoritmos de boa performance possam estender seus limites até problemas maiores. Experimentos distribuindo instâncias do Treasure Hunt, em uma rede cooperativa de até 160 processos, demonstram a escalabilidade robusta do framework, apresentando melhoras nos resultados mesmo quando o tempo de processamento é fixado (*wall-clock*) para todas as instâncias distribuídas do Treasure Hunt. Resultados demonstram que o mecanismo de amostragem fornecido pelo Treasure Hunt, aliado à maior cooperação entre as múltiplas populações em evolução, reduzem a necessidade de grandes populações e de algoritmos de busca complexos. Isto é especialmente importante em problemas de mundo real que possuem funções de fitness muito custosas.

Palavras-chave: Inteligência artificial. Métodos de otimização. Algoritmos distribuídos. Modelagem de convergência. Alta dimensionalidade.

ABSTRACT

This work proposes a multilevel framework called Treasure Hunt, which is capable of distributing independent search algorithms to a large number of processing nodes. Aiming to obtain joint convergences between working nodes, Treasure Hunt proposes a driving mechanism that smoothly controls the cooperation between the multiple independent Treasure Hunt instances. The tree topology proposed by Treasure Hunt ensures quick propagation of information, while providing simultaneous explorations (by parents) and exploitations (by children), on several levels of granularity, regardless the number of nodes in the tree. Treasure Hunt has good fault tolerance and is partially prepared to full fault tolerance. As part of the methods developed during this work, an automated Iterative Partitioning method is proposed to control the balance between exploration and exploitation as the search progress. A Convergence Stabilization Modeling to operate in Online mode is also proposed, aiming to find good *cost/benefit* stopping points for the optimization algorithms running within the Treasure Hunt instances. Experiments on classic, random and competition benchmarks of various sizes and complexities, using the search algorithms PSO, DE and CCPSO2, show that Treasure Hunt boosts the inherent characteristics of these search algorithms. Treasure Hunt makes algorithms with poor performances to become comparable to good ones, and algorithms with good performances to be capable of extending their limits to larger problems. Experiments distributing Treasure Hunt instances in a cooperative network up to 160 processes show the robust scaling of the framework, presenting improved results even when fixing a *wall-clock* time for the instances. Results show that the sampling mechanism provided by Treasure Hunt, allied to the increased cooperation between multiple evolving populations, reduce the need for large population sizes and complex search algorithms. This is specially important on real-world problems with time-consuming fitness functions.

Keywords: Artificial intelligence. Optimization methods. Distributed algorithms. Convergence modeling. High dimensionality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
AI	Artificial Inteligence
A-Team	Asynchronous Team
BB-BC	Big Bang-Big Crunch
CC	Cooperative Coevolution
CCPSO	Cooperatively Coevolving Particle Swarm Optimization
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CPSO	Cooperative Particle Swarm Optimization
CSMOn	Convergence Stabilization Modeling operating in Online mode
DE	Differential Evolution
EA	Evolutionary Algorithm
EDA	Estimation of Distribution Algorithms
EC	Evolutionary Computation
ES	Evolution Strategies
GA	Genetic Algorithm
GPU	Graphical Processing Unit
HPC	High Performance Computing
LSGO	Large Scale Global Optimization
NFL	No Free Lunch
NP	Nested Partition
NP-hard	Non-Polinomial hard
PDF	Probability Density Function
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
RS	Random Search
SI	Swarm Intelligence
TH	Treasure Hunt

SUMÁRIO

1 INTRODUCTION

For decades, computer science has used mathematical approaches to develop effective computational techniques, increasing responsiveness when solving complex real-world problems. Despite the fact that these techniques range from optimal specialized algorithms to distributed solutions to execute on high performance computing environments, they are still limited to the mathematical methods used to obtain the solutions.

As the most prominent computer science area that competes (or even overcomes in some cases) the results obtained by mathematical approaches, Artificial Inteligence (AI) encompasses many sub-areas of research. Each of them has its own interpretations and distinct approaches when solving (real world and benchmark) problems, showing different advantages and limitations. As a frequent AI's top performer when optimizing Non-Polinomial hard (NP-hard) problems, Metaheuristic is a relatively new sub-area of AI that does not focus on solving a specific problem, but rather on a broader category of problems.

Continuous functions of high complexity / dimensionality are of special interest for metaheuristics when creating methods to optimize real world problems, whose actual landscape is usually unknown. Search algorithms designed to deal with these functions typically focus on quickly finding strong local minima, instead of focusing on avoiding the stagnation caused by such minima, what consequently reduces the probability of obtaining long term convergences. Although this approach has proven to work efficiently for a large variety of problems, it might affect the capacity of the algorithm to reach a nearly global optimum.

To obtain continued convergences for longer periods after the first stagnation occurs, metaheuristics could combine different solutions, like those using the classic multi-start method. However, the rework is inherently present in these methods since the candidate solutions are usually generated through all the search space, what could potentially make the search algorithm to reach the same local minimum several times (Tu e Mayne (2002)). Besides, no automated method is usually employed by these algorithms to determine the best moment to make each restart, hence wasting processing resources.

Given that more than one good minimum can be present in the addressed problem, algorithms prepared for multimodal functions (like multi-population methods) provide an effective way to obtain multiple good solutions. Nonetheless, multi-population methods are only as efficient as the topology used to coordinate their work, because there are severe restrictions on the number of populations that can be supported by every topology. Despite the fact that this problem can be mitigated by increasing the population sizes, in practice, it is empirically known that there is a limit in the maximum number of individuals that can be used to improve the performance of a given search method, what consequently restricts the scalability of multi-population topologies.

The efficacy of suitable search algorithms for a given problem is not expected to be robust when increasing problem's complexity (eg. larger dimensionality), because a minimally reliable overview of the search space landscape could not be guaranteed at viable time (Crainic e Toulouse (2010)), particularly when the fitness function evaluation is time-consuming (eg. computing intensive equations and high number of decision variables). Although distributed search algorithms provide a good way of increasing search space's coverage, their performance are limited by the communication topology, given that the topology outlines the synchronization problems, the bottlenecks, the number of processing nodes and ultimately, the portion of the search space that can be visited.

In an attempt to provide a flexible and generic method with the capacity of dealing with high dimensional, complex, single-objective, continuous optimization problems, this thesis proposes a multi-level framework called Treasure Hunt (TH), which uses search methods as black boxes, and is capable of (i) distributing the search to a high number of processing nodes in a clustered computing environment and (ii) sustaining good long-term convergences, showing minimum communication penalties, high self-organization and self-adaptation (Dreo e Siarry (2007)), good scalability, inherent capacity of dealing with multimodal problems and time-consuming fitness functions.

This chapter summarizes the work proposed by this thesis. Section 1.1 describes the problem that this work proposes to tackle. Section 1.2 presents the motivations for developing TH framework. Finally, the Sections 1.3, 1.4, 1.5 and 1.6 describe the main objectives, challenges to be overcome, hypothesis and contributions of this thesis.

This project was partially supported by the R&D project PD-6491-0307/2013, proposed by Copel Geração e Transmissão S.A., under the auspices of the R&D Programme of Agência Nacional de Energia Elétrica (ANEEL).

1.1 PROBLEM DEFINITION

Real-world optimization problems, that in the past had to be over simplified so that they could be solved by mathematical methods on computers available at the time, nowadays rely on much more computing power and more effective numerical methods to solve them. This increased the processing speed and quality of results, making the numerical methods the main tool to solve these categories of problems. Given currently available methods and hardware capabilities, however, the complex category of such real-world problems are still too hard for direct numeric approaches, requiring further mathematical simplifications. As consequence of these "relaxations", the confidence on results is diminished.

Metaheuristics are AI-based iterative methods that organize and apply heuristics over candidate solutions (Nesmachnow (2014)), which are initially specified either by previous knowledge on the problem at hand, or by applying some random process to generate initial solutions. Metaheuristics' internal mechanics could be seen as "walks through neighborhoods" (Alba et al. (2013)), in the sense that given an initial solution, random modifications are made to the solution so it can move to good locations nearby (close to the candidate's position or to a group of candidates, in an Cartesian sense), thus obtaining a stochastic sampling of the objective function (Dreo e Siarry (2007)) and, over time, a minimally reliable global search behavior. Therefore, randomness is crucial to metaheuristics because it increases the exploratory effectiveness of the search process (Jr et al. (2015)), making individuals that start from different positions potentially explore distinct regions of the search space (Crainic e Toulouse (2010)). For that reason, metaheuristics cannot generally guarantee that an optimal solution will be found or that a comprehensive analysis of the search space will occur (Crainic e Toulouse (2010)), what makes it so hard proving the closeness to the problem's optimal solution.

Obtaining a good sampling of the search space is not an easy task, especially when the problem has a large number of variables or a complex landscape. To mitigate this problem, for example, collaborative methods have been created to improve "sample quality" (i.e., to obtain high quality candidate solutions) during optimization. To deal with the search stagnation, on the other hand, a well known technique is to balance between exploration (analysis of larger areas) and exploitation (intensification / local search). While many of these methods are recognized for obtaining local minima of good quality, they are not originally meant for escaping from these same minima.

The class of problems tackled by Treasure Hunt framework work can be summarized as having:

- **High Dimensionality:** the number of dimensions can easily raise to tens of thousands.
- **High Complexity:** typically requiring solutions for:
  - **Multimodality:** with an unknown number of optimal solutions.
  - **Large Search Spaces:** not much information is available to further reduce the problem's search space area.
  - **Distributed methods:** since shared memory parallel hardware are unable to solve the larger problems at viable time.

1.2 MOTIVATION

Metaheuristics have been used as robust methods that do not require in-depth knowledge about the problem being optimized. Nonetheless, for the classes of problems addressed in this thesis, better sampling of the search space and more effective ways of escaping from good local minima are still required for metaheuristics, so that good long term convergences can be obtained by the search methods. Given that real world problems tend to become more complex and parallel hardwares tend to be more affordable, any long term solution for improving the search space sampling would have to include parallel or distributed algorithms, to take advantage of simultaneous cooperations.

The cooperation between multiple parallel populations is a typical approach to improve convergence, because the larger coverage of the search space obtained by these populations (when compared to a single population) increases the diversity of candidate solutions, and very likely their quality. However, according to Crainic e Toulouse (2010), the evolution of cooperative parallel searches is much more controlled by the cooperation schema (i.e., the topology) than by the optimization logic itself. Topologies create cooperative environments on which potentially good information can flow more easily. According to Dreo e Siarry (2007), such collaborative environment entails two important characteristics: 1) self-organization, which occurs when the various levels of interactions between low-level components (that do not hold direct relation with global top-level components) cause a cascading effect over global components, thus producing a global pattern; 2) self-adaptation, which is the process of adapting automatically to new conditions during execution time, like for example balancing between the exploration and the exploitation as the search progress. Together, these two characteristics can highly influence not only the convergence speed but also the capacity of avoiding local minima traps.

As alternatives to obtain a good sampling of the search space landscape, a class of methods called Model-Based Optimization (MBO) uses surrogates or distribution-based algorithms to improve convergence. Surrogates are cheaper substitute functions that approximate the original function, while distribution-based algorithms use sequences of probability distributions to generate new candidate solutions. However, Bartz-Beielstein e Zaefferer (2017) strongly recommend using surrogates only when the classic optimization techniques (and their hybrids) fail to solve the problem at hand. Moreover, distribution-based algorithms are highly sensitive to the probabilistic models used, demanding correct feature selection and efficient methods for learning and sampling these models (Hauschild e Pelikan (2011)).

Regardless of the success obtained when sampling the landscape, most solutions fail on high dimensionality due to the *curse of dimensionality*[1]. Considering that the perception of what

[1]The performance deteriorates as the dimensionality of the search space increases.

is a large number of dimensions is relative for every search algorithm, decomposition frameworks like Cooperative Coevolution (CC) (Potter e De Jong (1994)) were created, in an attempt to overcome such algorithm-specific limitations, by evolving sub-sets of the dimensions separately. However, *divide-and-conquer*[2] methods like CC require efficient decomposition strategies and a considerably high communication frequency between the decomposed components (Mahdavi et al. (2015); Vu et al. (2011)), reducing their effectiveness in a distributed environment.

The methods that explore multi-processed hardware usually involve classic algorithms with profound changes in their original logic, so that the methods can run in parallel or distributed environments. This is usually accomplished by either spreading the problem's dimensions (more usual on parallel / shared memory systems) or the population (commonly used on distributed systems) into multiple processing units. Some of these methods increase their efficiency by exploring the communication topology between search algorithm's sub-populations, fine tuning the searches according to their location at the network (Sefrioui e Périaux (2000); Herrera e Lozano (2000)) (e.g. controlling the exploitation-versus-exploration behavior). There are a few exceptions that do not require expressive changes to the search logic, like for example the pool based algorithms. However, pool-based solutions have a clear limitation when expanding to a high number of processing nodes, due to problems like network delays and the limited capacity of the pool in dealing with too many distributed agents.

There is, therefore, a demand for new parallel, highly distributed and scalable methods, with efficient mechanisms to distribute optimization algorithms to processing nodes, without requiring deep changes in their logic. Moreover, the following characteristics are also expected: large sampling of the search space, efficient joint optimization between processing nodes in a controlled pace, high self-organization and self-adaptation, and reduced communication and propagation time.

1.3 OBJECTIVES

The objective of this thesis is to present a multilevel, distributed parallel framework called Treasure Hunt, which is capable of exploring a large portion of the search space by distributing independent search algorithms to a high number of processing nodes, providing a topology that allows quick propagation of good information and a decentralized joint optimization between the nodes.

1.4 CHALLENGES

For decades, mathematical methods have been extensively studied and applied to the class of problems addressed in this thesis. Despite improvements and good results, the quality of the optimization is still affected due to simplifications and adjusts required over original modeling. Although metaheuristics have been successfully applied to tackle this problem, whenever the number of decision variables (dimensions) is much larger than standard straightforward metaheuristics can efficiently handle, the need for good optimization algorithms, specialized on high dimensionality, becomes evident. Nevertheless, such algorithms usually have slower convergence and demand more processing power, hence requiring new methods to estimate good commitments between optimization improvements and processing times (a budget management problem). Considering the unpredictable nature of stochastic search algorithms and

[2]Large or complex problems are optimized by solving the smaller or simpler sub-problems that composes the original problem.

their convergence while optimizing solutions, limiting their processing resources is a challenge that needs to be addressed.

Recently, parallel variants of state-of-the-art metaheuristics have successfully reduced the overall processing time when compared with the respective serial versions. This is usually accomplished by dividing the original problem in sub-problems, either partitioning the variables or the search space domain. The division of the search space domain is not usually employed due to the complexity of providing an efficient division rule and maintaining good convergence while keeping the communication low. On the other hand, classic techniques are frequently employed to split the problem's decision variables into several sub-problems, each sub-problem containing a subset of the original dimensions (eg. Van den Bergh e Engelbrecht (2004); Li e Yao (2009)). Nonetheless, the partitioning of the dimensions into sub-problems potentially reduces the search space coverage, since the optimization outcomes are the combination of multiple sub-space optimizations, instead of the joint optimization of the decision variables' domains. Besides, this method adds data dependency, because the solutions mostly cannot be fully evaluated with only partial information.

When optimizing larger and more complex problems, more robust processing systems (e.g. distributed systems like High Performance Computing (HPC) environments) are required to obtain reasonable results in feasible time. However, scaling problems occur due to communication overhead, synchronization problems and (particularly for metaheuristic algorithms) due to inherent characteristics of candidate solution's evolution throughout iterations. Considering that metaheuristics which focus on exploring large distributed hardwares are not commonly found in the literature, there is a demand for effective scalable algorithms that distribute the work effort to a large number of processing nodes.

Considering the No Free Lunch (NFL) theorem[1], which states that there is no one model that works best for every problem, building a single solution or framework that covers all previously mentioned problems with a minimum acceptable quality is a complex challenge.

1.5 HYPOTHESIS

Given a complex / high dimensional problem and a search algorithm with known limitations for such problem, the first hypothesis is that the convergence can be maintained for longer periods when multistarting the search at stagnation points, and moving the population gradually toward a good region of the search space at every restart.

Extending this idea to multiprocessed environments, the second hypothesis is that a distributed joint convergence can be achieved between the processing nodes, in less *wall-clock* time and with better final results than the serial version of the search algorithm. To test this hypothesis, the collaborative optimization effort can be organized with a hierarchical distributed tree-topology, where every tree node contains a population initialized in (but not restricted to) a sub-area within the search space, and 1) no overlap occurs between the siblings' areas, 2) the children's areas are always restricted within parent's bounds, and 3) a budget restricted search algorithm is used at the nodes to increase cooperation and avoid processing waste. In this multiprocessed version, instead of multistarting toward its own good regions, every child population uses parent's good results as bias to influence its restarts, whilst the parents use children's good results as starting points for the search. The resulting effect is a smooth and continuous joint convergence obtained by simultaneous explorations (by parents) and exploitations

[1]In search and optimization research area (WOLPERT e MACREADY (2005)), NFL can be interpreted as "any two algorithms are equivalent when their performance is averaged across all possible problems".

(by children), on several levels of granularity, what greatly improves the optimization power of simple search algorithms.

1.6 CONTRIBUTIONS

This thesis proposes a multi-level, distributed framework called Treasure Hunt (TH - see Appendix A), which provides a flexible and generic method with the capacity of dealing with complex and high dimensional continuous optimization problems, presenting the following main contributions:

1. Search space organization: a hierarchical method to position populations throughout the search space, such that controlled overlap occurs between siblings, and a good commitment between exploration and exploitation occurs as consequence of parent / children organization.
2. Topology: a decentralized and asynchronous mechanism to communicate multiple independent search algorithms in a multilevel, hierarchical tree topology, providing fast propagation of information between any algorithm in the tree, allowing a massive number of optimization processes to be distributed in a high performance cluster.
3. Convergence: a controlled multi-start mechanism that reduces the waste of fitness function evaluation, allowing longer term convergence.

In support of the work developed in this thesis, an Iterative Partitioning method (CCPSO2-IP (Perroni et al. (2015)), Section 4.6) was developed to improve the performance on non-separable problems for the search algorithm specialized in high dimensionality CCPSO2 (Li e Yao (2012)). To reduce the waste of processing resources, an automated method to control the budget allocation was also developed, providing a quick estimation of the search stagnation during an optimization run (CSMOn (Perroni et al. (2017)), Section 4.7).

1.7 DOCUMENT STRUCTURE

This work is organized in 8 chapters (including this Introduction). Chapter 2 describes the most important methods used as base for the development of the framework proposed in this thesis. Chapter 3 presents the more relevant works found in literature regarding the main objectives of this thesis. The Treasure Hunt framework is described in Chapter 4, which also includes technical details and discussions. Chapter 5 illustrates the working principles of Treasure Hunt framework. Chapter 6 presents and discusses the experimental results for several different benchmarks, and Chapter 7 concludes this thesis. An Appendix is provided, presenting the metaphor that inspired the Treasure Hunt framework, and two articles published as outcomes of this thesis.

2 THEORETICAL FOUNDATION

The relevant theory required to understand the Treasure Hunt framework is discussed in this chapter. Next section presents a summary of Metaheuristics (2.1), which includes a set of approaches ranging from from simple search algorithms to large scale optimization algorithms. Section 2.2 describes A-Teams, a pool-based framework for cooperation between search algorithms. Section 2.3 presents brief overview of Hyperheuristics and its characteristics. Different distributed environments are presented in Section 2.4, and Section 2.5 describes the mathematical basis used by TH framework.

2.1 METAHEURISTICS

One basic mathematical optimization method is the Gradient Ascent/Descent, which takes the first derivative $f'(x)$ (the slope) of an evaluation function $f(x)$ and an arbitrary initial point x. At every iteration, a small displacement is made on the x position by adding a percentage α of the slope to the previous solution x_{old}, i.e., $x_{new} = x_{old} + \alpha f'(x_{old})$, until the point where the slope is evaluated as zero. At that point, the optimization has found a peak (an upper curve in a maximization problem or a lower curve in a minimization problem) and the process is complete. In the n-dimensional version of this method ($\overrightarrow{x} = [x_1, \cdots x_n]$), the gradient $\triangledown f'(\overrightarrow{x})$ is used instead, i.e., $x_{new} = x_{old} + \alpha \triangledown f'(\overrightarrow{x}_{old})$.

When the gradient of the function is too complex to be calculated or even the exact mathematical formulation is unknown, a robust sub-field in artificial intelligence called Metaheuristic can be employed. Metaheuristic is a research area that studies algorithms for optimization problems, generally by using a stochastic approach (Luke (2015)) and without in-depth knowledge on the problem at hand. It has been successfully used to optimize complex classes of problems, whose heuristic information is scarce or nonexistent, specially where more than one optimal solution is possible (multimodal problems).

Next sections give a quick overview of the most relevant metaheuristics used as base to build the framework proposed by this thesis.

2.1.1 A Brief Overview of Metaheuristics

Similar to the gradient ascent/descent displacement method, the metaheuristic algorithm Hill Climbing takes one initial possible candidate solution $\overrightarrow{x}$ (hereafter denoted simply as x), makes a small random modification on it (instead of using the gradient) and checks if the new solution (x_{new}) is better than the previous one (x_{old}). If the solution has improved, the new one is maintained ($x = x_{new}$), otherwise it keeps the previous solution ($x = x_{old}$). This process repeats until the local best solution is found or the time allocated for the optimization is exceeded. This algorithm is classified as a Local Search algorithm in the sense that it makes intensive analysis (i.e., exploitation) on the area nearby the starting point. Another important algorithm is the Tabu Search, that is similar to Hill Climbing with the addition of a tabu list, which is an attempt of preventing solutions similar to the already visited ones, by maintaining a list of temporarily forbidden solutions.

The detailed regional analysis made by local search algorithms result in a large gap in the search process, since most of the search space is not actually explored. To fill this gap, Global Optimization algorithms like Random Search (RS) rely on the nature of the stochasticity to evaluate large portions of the search space (i.e., exploration). Given enough time and appropriate randomness, these algorithms gradually improve the chance, as the search progresses, of generating superior and more diverse solutions. The Random search algorithm creates and evaluates, at every iteration, a random candidate solution x, saving x if it is better than the best solution found up to that point ($x_{best} = x$). Due to the unpredictable nature of randomness, this category of algorithms is sensitive to the number of evaluations. Besides, these algorithms lack satisfactory stopping criteria (Solis e Wets (1981)) (a budget management problem). To mitigate this scenario, a modified Controlled Random Search variant (CRS4) (Ali e Storey (1994)) uses a beta distribution to guide the generation of new solutions, according to the mean and standard deviation from previous best solutions. CRS4 stops the search when the fitness difference between the best (x_{best}) and the worst (x_{worst}) solutions are within a specified limit ϵ ($|f(x_{best}) - f(x_{worst})| < \epsilon$).

A more elaborated algorithm that combines both exploration (global optimization) at initial stages of the search and exploitation (local search) at latter stage is the Simulated Annealing (Cerny (1984)), which simulates a metal cooling process (an annealing process). Starting from an initial solution x, at every iteration this algorithm creates a slightly modified solution x_{new} and accepts it in case it is better than previous solution ($x = x_{new}$). Otherwise, the acceptance of a worse solution depends on an exponential probability $e^{\frac{f(x_{new})-f(x)}{t}}$ (where t is the temperature). The temperature is gradually reduced, following a cooling schedule that must be adjusted manually for each optimization problem (an empirical and time consuming job). This algorithm was an important breakthrough in the optimization research field due to its good capacity of escaping from local minima (i.e., branching[1] (Rattadilok et al. (2004))), besides automatically adjusting between exploration and exploitation.

The diversity provided by previous methods, however, is limited by the fact that only one solution is evaluated at every iteration. Populational methods were created to improve the results obtained throughout iterations, given that instead of evaluating one single solution, the algorithms evaluate an entire sample of the search space. Populational methods vary in many ways, for example: the population size, how each individual interacts with each other, how the solution kept by every population member is modified at every iteration, and if a community manager will be acting on behalf of the collective health. Evolutionary Computation (EC) is a subset of these algorithms that embraces two important groups of algorithms:

- Evolutionary Algorithm (EA): mostly composed of generational algorithms, where the next generation is re-sampled (re-created), based on results of previous (parent) generation, and using many biology-inspired techniques, like for example mutation, crossover and natural selection. Classic methods are Genetic Algorithm (GA), Evolution Strategies (ES) and Differential Evolution (DE).

- Swarm Intelligence (SI): each individual in the population evolves based both on social (population) and cognitive (individual) knowledge, and for that reason, no breeding occurs. Examples of SI methods are Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Cooperative Coevolution (CC) algorithms.

Genetic Algorithm is a generational algorithm, with moderate global search characteristics (Rudolph (1994)), that follows a sequential three-steps iterative process to obtain increasingly better adapted offspring: 1) selection of the best fit individuals to form a new population; 2) crossover (regional search), sharing pieces of information between individuals to generate slightly

[1]Branching is a small variation from a candidate solution, that helps escaping from strong local minima without losing much information from original solution.

different (and hopefully better) solutions; and 3) mutation (moderate global search), escaping from local minima by assigning random values to random dimensions from the solution vector.

Evolution Strategies (ES) are a category of algorithms similar to Genetic Algorithm that uses only mutation and truncation selection (i.e., it keeps only the best individuals) to generate new populations. Its strategies are composed of one selection operation (either full replace "," or join "+") and a combination of two numbers, one for the parents (μ) and other for the offspring (λ). For example, (μ, λ)-ES represents a strategy where μ parents are used to generate λ children (where λ is a multiple of μ), each parent creating λ/μ children through a simple mutation, and the selection occurring by dropping the parents and keeping only the offspring (full replace operation) (Weise (2009)). On the other hand, ($\mu + \lambda$)-ES strategy joins μ parents with their λ children and selects only the μ best individuals to become the next generation.

Differential Evolution (DE) method takes solutions already in place into the population and use them to create new solutions through a recombination operator (Weise (2009)). The recombination operator is responsible for the evolution of the population, computing the difference between any two solutions x_1 and x_2, applying a weight w over this difference and adding it to a third solution x_3, thus generating a new solution $x_{new} = x_3 + w(x_1 - x_2)$. Each population member generates a single child and the selection keeps the best fit solutions between both parents and offspring. The variants of DE are classified as DE/x/y/z, where x specifies the solution to be mutated (best, rand, etc), y is the number of different solutions used for recombination (1, 2, etc) and z denotes the crossover scheme (bin, *, etc) (Storn e Price (1997)).

Ant Colony Optimization (ACO) (Dorigo et al. (1996)) uses swarms ecosystem concepts to drive the evolution of the population. Instead of evolving through competition (as in EA methods), community members work cooperatively toward a common goal. In ACO, individuals act like ants looking for food, leaving trails of pheromone wherever they go. The more the ants walk through a specific path, the more pheromone will be enforced and more likely other ants will use the same route. Hence, ACO is more suitable for graph-like problems, where the idea of "path" can be applied. Although this is not a robust method for large dimensionality numerical optimization problems, this is a classic swarm algorithm that worths mention due to its concepts of evolving community (opposed to the generational population used by EAs).

Particle Swarm Optimization (PSO) (Eberhart et al. (1995)), in turn, is a milestone in continuous numerical global optimization, in the sense that its design presents an efficient way of quickly converging to local minima of good quality, besides allowing parallelization in most multiprocessed hardware. Its algorithm follows the concept of birds flock, where each bird (particle or individual) has its own current position (x , the solution) and velocity (v), flying toward both its known best personal direction (p_{best}) and swarm's global best direction (g_{best}). A weight is assigned for each direction (c_1 for personal direction and c_2 for global direction) and for the velocity (w). To reduce the risk of getting trapped on local minima, random factors computed at every iteration are added to the flight, one for personal direction (r_1) and one for global direction (r_2). Thus, the velocity update is calculated by preserving a portion of previous velocity, and adding a portion of the directions toward its personal best and the global best positions (2.1). Then, the new position is calculated as (2.2):

$$v_{new} = w\ v_{old} +\ c_1 r_1(p_{best} - x) +\ c_2 r_2(g_{best} - x) \tag{2.1}$$

$$x_{new} = x + v_{new} \tag{2.2}$$

The parameters c_1, c_2, and w allow the control of the swarm "on the fly", so that fine-tuned adjusts can be applied to the algorithm, to give it autonomy to decide (on-demand) for the best balance between exploration and exploitation as the search progresses. Due to its simplicity and high efficiency, PSO has been subject of extensive studies in academic community, and many highly competitive variants have been developed to compete with complex algorithms for large dimensionality problems. For search spaces with large dimensionality, a multi-swarm version of PSO called Cooperative Particle Swarm Optimization (CPSO (Van den Bergh e Engelbrecht (2004))) was created, where the dimensions are divided into subsets and each subset is given to a swarm. Then, the swarms execute (likely in parallel) with the same configuration, cooperating each other with their own best solutions. The final output of CPSO is the union of swarms' global best solutions, resulting in one single solution containing all original dimensions.

Inspired by the results obtained by PSO, many other nature-inspired swarm-based cooperative algorithms have been proposed. Grey Wolf optimization (Mirjalili et al. (2014)), for example, is an algorithm that mimics the movements of a worf pack chasing a prey, where the three higher hierarchy leaders (α, β and δ, representing the three best results) guide the whole pack during the hunt, using the search-than-attack wolves behavior to perform the exploration-than-exploitation automatically.

Artificial Bee Colony (ABC) algorithm (Karaboga (2005)) assigns random positions (candidate solutions) to a swarm of bees (individuals), looking for potentially good food sources (exploration), assigning more bees for investigation of the best regions (exploitation), and using scout bees (a multi-start mechanism) to investigate new locations when no better food source is found (i.e., when the search stagnates).

An SI algorithm that focuses on multimodal problems, namely Glow-worm Swarm Optimization (Krishnanand e Ghose (2005)), uses only the solution of the brightest neighbors to guide the search, thus reducing the chance of getting trapped on local minima.

There is actually an extensive list of bio-inspired swarm-based algorithms proposed (besides the small subset of SI algorithms previously mentioned), each of which presenting different mechanisms and inspired by different animals or insects (cockroaches, amoebas, mosquitoes, bats, bacteria, etc). A different approach, however, has been presented by Cooperative Coevolution (CC) algorithms. CC is a category of SI algorithms with context-sensitive fitness, which depends on the solutions presented by the available individuals. Two main groups of coevolution algorithms are used, one for competitive populations and other for cooperative populations. When one single population is used, for example, an individual solution is compared with some other random solution in the population to assign it a contextual fitness (winner solution has better fitness) (Luke (2013)). When two populations are used, the second population can be used to measure how good the solutions are on first population. When using multiple populations, a CC algorithm can split the problem into smaller subproblems (eg. assigning a subset of dimensions for each population) so that the populations have to work together to form a complete candidate solution for evaluation.

Physics-based algorithms bring physics concepts and adapt them into optimization algorithms, in an attempt to investigate the search space in ways that have not been tried before. A simple yet powerful physics-based optimization algorithm, Big Bang-Big Crunch (BB-BC) is a memoryless method that uses randomness to guide the search. Developed by Erol e Eksin (2006), BB-BC follows a two step procedure that iteratively calculates the center of mass of the population, then creates a new population around this center, repeating the process until some stopping criteria is met. This search algorithm ensures convergence by simply controlling the radius around the center of mass. Its overall steps are described below:

1. Create an initial random population of p individuals and evaluate their fitness.

2. Calculate the center of mass using Equation (2.3), where x_i is the n-dimensional solution of the individual i, f_i is the respective fitness value, and x_c is the population's n-dimensional center of mass.

3. Create a new population around the center of mass x_c, following a normal distribution with a decreasing standard deviation as the number of iterations increases.

4. Return to Step 2 if stopping criteria has not been met.

$$x_c = \frac{\sum_{i=1}^{p} \frac{1}{f_i} x_i}{\sum_{i=1}^{p} \frac{1}{f_i}} \tag{2.3}$$

2.1.2 Metaheuristics for Large Scale Global Optimization

Complex problems of large dimensionality have been addressed by optimization algorithms in many different ways. However, the meaning of what is a large number of dimensions varies depending on the considered method, and on the class of problems the algorithm was conceived for.

Math intensification algorithms, like the ones based on covariant matrix, are among the robust methods that most suffer with the *curse of dimensionality*[2]. One example is the state-of-the-art algorithm Covariance Matrix Adaptation Evolution Strategy (CMA-ES), that uses the covariant matrix to maximize genetic variations during mutation (Hansen e Ostermeier (1996)). Despite its high competence in finding good solutions even on global optimization problems, its inherent complexity makes it too slow when the number of dimensions is above 100 (Omidvar e Li (2010)). One of its versions adapted to deal with multimodal functions, IPOP-CMA-ES, has even shorter limit (50 dimensions). The algorithm sep-CMA-SE is a version specially created to reduce CMA's complexity that, in spite of outperforming CMA-ES when optimizing problem sizes above 100 dimensions, does not scale for problems above 500 dimensions.

PSO, on the other hand, has high convergence rate and natural ability to deal with multimodal separable and non-separable global optimization problems. However, it can handle efficiently complex problems above 500 dimensions only when adding specific mechanisms to deal with high dimensionality. In an attempt to overcome the difficulties imposed by the high number of dimensions, frameworks like Cooperative Coevolution (CC) have been created.

A competitive decomposition method (Mahdavi et al. (2015)) for large dimensionality (Caraffini et al. (2013a,b)), CCPSO2 (Cooperatively Coevolving Particle Swarm (Li e Yao (2012))) is a recent state-of-the-art algorithm based on the well known PSO metaheuristic (Eberhart et al. (1995)). It is a variant of a previous version called CCPSO (Li e Yao (2009); Goh et al. (2010)), and is recognized by its good capacity of dealing with multimodal complex functions and separable problems (Poikolainen et al. (2013); Poorjandaghi e Afshar (2014)). Similar to CPSO (Van den Bergh e Engelbrecht (2004)), CCPSO2 works by partitioning the dimensions into multiple swarms, each swarm evolving a population of candidate solution separately from the remaining swarms. At every iteration, the cooperative coevolution occurs by combining

[2]The performance deteriorates as the dimensionality of the search space increases because (Mahdavi et al. (2015)): 1) the search space exponentially increases with the problem size; 2) increasing the problem size also increases its landscape complexity.

the partial candidate solution of a particle with the best solutions from others swarms, at the moment of the fitness evaluation. However, differently from CPSO, the particle's positions are updated using a local best ring topology (i.e. particle's local best neighbors) and two different numeric distributions, Cauchy and Gaussian. Aiming to improve results when the problem has non-separable variables, the dimensions are randomly permuted at every iteration and, when no improvement occurs, CCPSO2 recreates all swarms selecting swarms size randomly from the parameters.

2.2 A-TEAM

Introduced in 1991 by de Souza e Talukdar (1993), Asynchronous Team (A-Team) is an architecture that describes a cyclic network of agents, their memories and how they share information. An agent is basically composed of two components: 1) an optimization algorithm; and 2) its asynchronous communication protocol, which specifies the inter-agent information flow.

The idea behind A-Team is to combine different algorithms, allowing them to benefit from each other's best characteristics. The agents on A-Team work independently from each other, therefore allowing them not only to run in parallel but also in distributed environments. Every agent has its own memory space (the pool) where it stores good results in, based on some insert / removal policy. To allow an effective asynchronous communication without affecting agents' algorithms, every agent reads an input solution from other agent's memory space, therefore creating a cyclic network of heterogeneous agents. Once the solution is read from the memory space, it is immediately removed from the pool of solutions. For that reason, a semaphore has to be implemented on the pool, avoiding simultaneous read and write over the same data block.

Before reading a solution, the agent has to follow some strategy to select the proper memory block (Rodrigues (1996)). If the agent always chooses the best solution, it will tend to converge too quickly and intensify the search around the good regions of the search space. If the agent always chooses the worst solution, the usually unexplored areas around the poor solutions will be exploited more carefully. The agent can also use an increasing linear probability, either from best to worst (worst having higher probability) or from worst to best. Using higher probability in the central area of the list of solutions (triangular distribution) would prioritize the intermediate solutions, what might intensify the search on regions not too distant from the best solution (because after a certain number of iterations, most of the solutions tend to approach the best solution). A dynamic selection strategy could combine previous strategies according to some specific need.

Considering that the memory space is limited and a large number of good solutions can be created quickly, there are destruction agents that follow predefined policies, deciding if a new solution can be inserted and, in a positive case, remove the old unused solutions according to some strategy. Strategies to remove old solutions are mostly probabilistic and include the removal of the worst solution, the removal of any solution in the memory with a uniform probability distribution, the removal of any solution (except the best result) with increasing linear probability (worst has higher chance of removal), and the removal of any except the best and the worst solutions using triangular distribution (higher probability on central area, thus holding both best solutions, for increasing convergence, and worst solutions, for increasing diversity).

2.3 HYPERHEURISTICS

Hyperheuristic can be defined as an optimization mechanism that operates on the (smaller) search space of heuristics rather than on the (much larger) solution space, having heuristics selected or generated according to the search progress.

When the hyperheuristic uses feedback information from the search process to chooses a heuristic, it is considered a learning algorithm. This learning can be online (when the learn occurs throughout the search) and offline (a training set is used to extract knowledge in the form of rules or programs).

Resembling the categorization of heuristic methods, the hyperheuristic metodologies used to select or generate the heuristics can be (Burke et al. (2013)):

- Constructive: the chosen heuristics iteratively extend partial solutions until complete solutions are obtained. The sequence of heuristics applied to the partial solutions are typically represented by chromosomes with variable sequence of characters, where each character represents a heuristic.
- Perturbative: the chosen heuristics modify complete candidate solutions. This methodology is composed of two stages:
  - Heuristic Selection: the heuristic is selected from a set of perturbative heuristics and then applied over a candidate solution (single-point search) or a population (multi-point search). The heuristic selection can be uninformed (through random or exhaustive process) or use some learning mechanism. An example of uninformed selection is the Simple Random, which selects randomly a heuristic at each step. The learning mechanism usually selects heuristic based on scores that are refined during execution. To assign these scores, methods like the Reinforcement Learning(RL) can be used. Given a state (candidate solution), RL takes an action (perturbation) based on a policy (heuristic), and then rewards (giving a positive score) or punishes (giving a negative score) the heuristic based on the results.
  - Move Acceptance: examples of acceptance criteria are *All Moves* (accepting all generated perturbations) and *Only Improvements* (accepting only the perturbations that generated improved solutions). Examples of methods that apply some acceptance criteria are the Random Descent (which applies the selected heuristic until no further improvement is obtained), Random Permutation (which creates a random cyclic list of heuristics and apply one at each step), Random Permutation Descent (same as random permutation, but only applies next heuristic when no improvements are obtained by the current one) and Greedy (applies separately all heuristics and chooses the best one) (Özcan et al. (2010)). A more complete acceptance criterion is the Choice Function, which ranks the heuristic based on two intensification criteria (its individual overall performance and the instant/current performance compared to the previously called heuristic) and an exploratory criterion (the elapsed time since last call), then accepts the move based on the combined scores.

2.4 DISTRIBUTED ENVIRONMENT

Good distributed search algorithms should aim particularly on (Gong et al. (2015)):

- Scalability: which comprises size scalabilility, meaning how performance increases as processors are added, and task scalability, referring to whether the algorithm provides tools to maintain efficiency on distributed environments when increasing problem size.
- Speedup: the ratio between sequential processing time and parallel processing time.
- Communication: the impact that checkpoints or information sharing cause on the overall processing time. The amount of data exchanged should be small enough to be efficiently handled even by low-speed network connections, and latencies should be hidden by sequential computation (Crainic e Toulouse (2010)).
- Fault-tolerance: refer to how robust is the algorithm to events of component failures, like for example network delays or node crashes.

Usual topologies applied to distributed metaheuristics include (Alba et al. (2013); Gong et al. (2015)):

- Master-slave: one master population delegates tasks to the slaves (Dubreuil et al. (2006)) (e.g., fitness evaluations).
- Island: also called coarse-grained model, composed of a global population distributed to multiple "islands" (populations), in which a serial search algorithm is performed, with communication occurring between islands according to some migration policy. Such policy comprises (Alba et al. (2013); Du et al. (2008)): topology (logical connection between islands), migration rate (number of solutions exchanged at every communication), migration period (time between communications) and selection of solutions (rules to select the solutions that will be exchanged).
- Cellular: usually one population member per cell in a grid of distributed nodes, with communication occurring only between neighbors, so that information can propagate to the entire population more slowly to avoid premature convergence (Alba e Dorronsoro (2005)).
- Hierarchical: also called hybrid model, this topology combines two or more models in a way that advantages can be combined to mitigate disadvantages. Hybrids include: Island - master-slave (with multiple interacting masters), Island - cellular (cellular formation inside every island) and Island - island (island formation inside island nodes) (Herrera et al. (1999); Vu et al. (2011)).
- Pool: uses a global shared resource pool to exchange information between autonomous processors (de Souza e Talukdar (1993)).
- Coevolution: divides the dimensions (instead of populations) into multiple simpler problems, requiring specialized fitness functions to aggregate and evaluate the solutions (Subbu e Sanderson (2004)).
- Multi-agent: based on the game theory, only individualized local goals are considered and a coordination process is used to manage these local goals (Bouvry et al. (2000)).

2.5 MATHEMATICAL BASIS

Next section describes the Beta numerical distribution, and Section 2.5.2 details the characteristics of a perfect k-ary tree.

2.5.1 Beta Probability Distribution

Probability distribution is an important statistical tool, used in many fields of science, that describes the dispersion of the values of a random variable. Probabilities for continuous distributions are measured over ranges of values rather than single points. A probability indicates the likelihood that a value will fall within an interval. For a single continuous random variable, the distribution is called Probability density function (PDF), whose curve shape varies according to its probability distribution. The total area under the curve is equals to 1 and the probability of the value of a random variable to fall inside a specific range is equals to the area under that curve, and limited by the lower and upper boundaries of such range.

The Beta distribution has been successfully used (eg. Ali e Storey (1994)) to probabilistically approximate individuals (gradually) to the best result found so far. It is a special type of probability distribution used to describe probability of probabilities, i.e., to model the behavior of random variables. Based on some knowledge about the problem (like initial success / failure probabilities), Beta PDF gives us the probability of such knowledge to remain unchanged. Equation (2.4) gives the Beta PDF, where x is the referred knowledge, α_p is the weight of the likelihood of a probability to hold true and β_p is the weight of the probability to hold false.

$$B_{pdf}(x; \alpha_p, \beta_p) = x^{\alpha_p - 1}(1 - x)^{\beta_p - 1} \tag{2.4}$$

$$\alpha_p = \beta_{max} - \beta_p,\ \beta_p \in [0 - \beta_{max}] \tag{2.5}$$

For the complementary probability presented on (2.5), the parameter β_{max} controls the shape of the beta distribution (Figura 2.1). A more controlled behavior can be seen with $\beta_{max} = 100$, whereas $\beta_{max} = 1$ yields a behavior that promotes more sparse distribution, favoring the boundaries.

2.5.2 Perfect k-ary Tree

Let L be the level (height) of a node in a tree, L_{root} be the height of a perfect *k-ary* tree and also its root level, $nChildren > 0$ the number of children that every node contains (except the leaves, where $L = 1$ and $nChildren = 0$), and $nNodes$ the total number of nodes in the tree. It can be said that each level L adds $nChildren^{L_{root} - L}$ nodes in the tree, with the entire tree having $nNodes$ (2.6) nodes at total.

$$nNodes = \frac{nChildren^{L_{root}} - 1}{nChildren - 1} \tag{2.6}$$

Figura 2.2 shows a perfect k-ary tree, where all nodes (except the leaves) have exactly k children and the tree has the diameter of $2(L_{root} - 1)$. It becomes evident, therefore, the relationship between the exponential growth on the number of nodes $nNodes$, and the linear growth on the diameter of the tree (the maximum shortest path between two nodes Crainic e Toulouse (2010)), as the number of levels L increases.

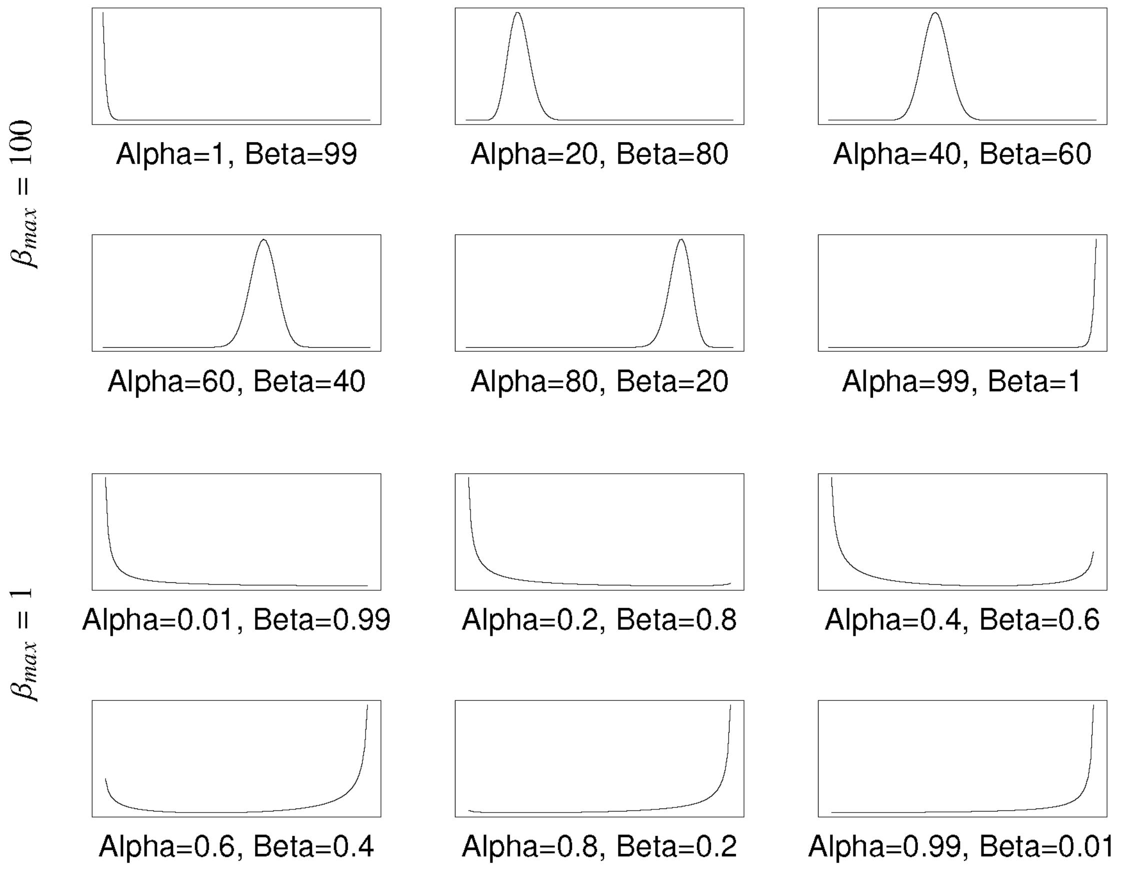


Figura 2.1: Beta PDF.

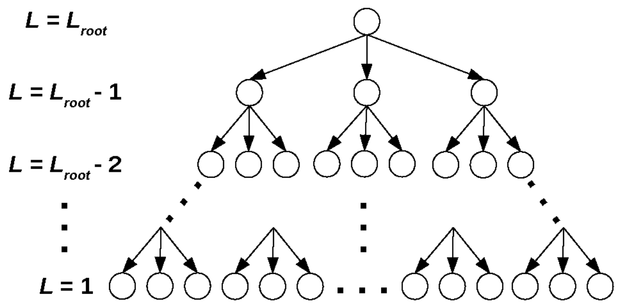


Figura 2.2: Perfect k-ary tree, with $k = 3$.

3 BIBLIOGRAPHIC REVIEW

This chapter describes the most relevant works found in the literature regarding parallel, distributed methods for optimization (Section 3.2), pool-based algorithms (Section 3.3) and hyperheuristics (Section 3.4). An exploratory research was conducted, aiming to find the robust methods available in literature for distributed optimization, and the existent gaps.

Comparing the work proposed in this thesis with the state of the art for distributed optimization (summarized in Table 3.1), Treasure Hunt (TH) can be classified as a Framework to distribute optimization methods, presenting the following characteristics which mostly differentiates from the existent methods, aiming to fill their gaps:

- Topology: Hierarchical Tree.
- Agents: Independent TH instances.
- Exploration / Exploitation: simultaneous explorations (by parents) and exploitations (by children), on several levels of granularity.
- Learning: *best-list*, a mechanism to control diversity while reinforcing the best sub-regions.
- Propagation Mechanism / Speed: Parent $\longleftrightarrow$ Children / Fast.
- Communication / Overhead: Asynchronous / Low.
- Joint Strategy: Guided Relocation.
- Memory / Tested Processes / Limited by: Distributed and Shared / 160 / Hardware.
- Fault Tolerance: High.

On next section, the state of the art for convergence modeling is discussed, whose main goal is to make predictions on the convergence behavior presented by the considered search algorithms.

Tabela 3.1: State of the Art for Distributed Optimization Algorithms

Authors	Topology	Agents	Exploration/ Exploitation	Learning	Propagation Mechanism/Speed	Communication / Overhead	Joint Strategy	Memory / Tested Processes / Limited by	Fault Tolerance
Pool-Based Algorithm									
Le Bouthillier e Crainic	Client-server	Independent	Pool / LS	None	Shared Pool / Slow	Asynchronous/Medium	None	Distributed / 8 / Server	High
Talukdar et al.	Client-server	Independent	Pool / LS	None	Cyclic Access to Shared Pool / Slow	Asynchronous/Low	None	Distributed / 4 /Server	High
Rodrigues	Client-server	Independent	Pool	Non-dominated Sorted Pool	Cyclic Access to Shared Pool / Slow	Asynchronous/Low	None	Distributed / 8 / Server	High
Talukdar e Ramesh	Client-server	Independent	Voyagers / Probes	None	Cyclic Access to Shared Pool / Slow	Asynchronous/Low	None	Distributed / 7 / Server	High
Ouelhadj et al.	Client-server	Hyperheuristic	None	None	Shared Pool / Slow	Asynchronous/Medium	None	Distributed / 10 / Server	High
Hyperheuristic									
Biazzini et al.	Island	Metaheuristic	None	Dynamic Probability Distribution	Random peer/Slow	Asynchronous/High	None	Distributed / 2^{13} / Agent	High
Henry Obit	Islands of Hyperheuristics	Independent	LLH Acceptance Criteria	None	Broadcast/Fast	Asynchronous/High	None	Shared / 6 / Agent	High
Rattadilok et al.	Master-slave-Island	Task	LLH Acceptance Criteria / Branching	RL	Inter-controllers peer-to-peer / Fast	Master-slave Asynchronous, Inter-controller Regular/Medium	None	Distributed / 12 / Agent	High
Meignan et al.	Independent	Independent	Search Operators	RL	Broadcast/Fast	Asynchronous/High	None	Distributed / 20 / Agent	High
Ouelhadj e Petrovic	Master-slave	LLH	Acceptance criteria	Greedy / Tabu	Slave$\rightarrow$Master$\rightarrow$Slave / Fast	Synchronous/High	None	Distributed / 20 / Agent	High
Lozano et al.	Master-slave	Metaheuristic	None	None	Slave$\rightarrow$Master/Fast	Synchronous/High	None	Shared / 250 / Hardware	Low
Metaheuristic									
Vu et al.	Master-slave	CC	Parallel evolutions of dimensions subsets	N/A	Slave$\rightarrow$Master$\rightarrow$Slave / Fast	Synchronous/High	Dimension Decomposition	Distributed / 6 / Master	Low
Alba e Dorronsoro	Cellular	Cells	Ratio	N/A	Neighborhood/Medium	Synchronous/High	None	Distributed / 400 /Hardware	No

3.1 CONVERGENCE ANALYSIS AND MODELING

Within the field of study called Theory of Randomized Search Heuristics (Yang (2011)), which tries to understand stochastic search algorithms (like EA and SI algorithms), Convergence Analysis is a sub-field that focuses on mathematical properties of the search methods in an attempt to prove convergence. Considering that such mathematical analysis is often difficult or even impossible in some cases, only a very few search algorithms have been actually studied mathematically, and most of these studies use simplified approaches and asymptotic approximations (Calafiore e Dabbene (2006)) for restricted classes of problems (Gutjahr (2009)). Since every randomized search method must have its own convergence analysis, there is no general purpose mathematical framework to provide insights into the search convergence.

The work developed by Pichitlamken e Nelson (2003) proposes a method for discrete optimization-via-simulation problems (i.e., stochastic systems represented by a computer simulation model). The method uses the classic explore-than-exploit approach, and incorporates the Nested Partition (NP) method (Shi e Ólafsson (2000)) as an attempt to guarantee the convergence. NP iteratively divides the search space into good sub-regions, intensifying the search only in the most promising partitions, and backtracking to re-explore its super-region when better solutions are found outside the current partition, thus escaping automatically from local traps. Although Shi e Ólafsson (2000) provides an extensive mathematical analysis, showing the high likelihood of the NP method to converge to a global optimum under certain conditions, they make no mention of the processing time required to find such result.

PSO convergence is tackled by Van Den Bergh e Engelbrecht (2006) through the analysis of the particles trajectories, according to the parameter settings. A formal analysis is provided to prove the particles convergence to a stable point of equilibrium, taking the inertial term (the particle's velocity) into consideration. The stable point is shown to be a weighted average of the personal best and global best positions, in which the weights are determined by the values of the acceleration coefficients. Although it is not a proof of convergence to a minimum, it states that the swarm will reach a point of equilibrium, under certain conditions. Trelea (2003) uses the dynamic system theory for a theoretical analysis of PSO, providing graphical guidelines for parameter selections that lead to good convergence behaviors.

As stated by Olafsson (2006), despite the fact that it is important to make statements about algorithms convergence under traditional asymptotic analysis for infinite time behavior, methods that target on short term solutions have much more practical usage. In that sense, Model Convergence is an approach that focuses on solutions instead of asymptotic analysis. Differently from the traditional focus on the improvement of an elite list of solutions, model convergence of population-based algorithms is obtained by promoting gradual improvements on the overall population. The work developed by Gutjahr (2009) presents a generic metaheuristic algorithm that encompasses good part of the currently existent metaheuristics as special cases, and then tries to apply model convergence to the generic algorithm.

A more efficient technique for solving expensive and time-demanding complex optimization problems, Model-based optimization (MBO) evaluates cheaper surrogate models instead of expensive objective functions, what could be used as a smoothing of the original function (Bartz-Beielstein e Zaefferer (2017)). MBOs generate the population by exploiting a model, which can be either a surrogate or a distribution. Surrogate models imitates the real function as much and as simple as possible, and is used when the computation of the original function is very expensive or time consuming. However, considering that they demand training for the accuracy and efficiency of results, quickly suffering from loss of performance as dimensions increase, Bartz-Beielstein e Zaefferer (2017) strongly recommend using MBOs only when the classic optimization techniques fail to solve the problem at hand. On the other hand, instead of propagating good candidate solutions to the next iterations, distribution-based models propagate the probability distribution induced or estimated from promising solutions in the population, hopefully assigning most of its probability mass to the set of optimal solutions. In that sense, distribution-based models are model convergence algorithms, given their focus on gradually improving the overall population instead of an elite list of solutions. According to Bartz-Beielstein e Zaefferer (2017), all distribution-based algorithms (like Estimation of Distribution Algorithm (EDA)) can also be classified as MBOs since the sequence of probability distributions generates approximations that describe the objective function.

A more computing intensive technique is the Optimal Computing Budget Allocation (OCBA), whose focus is on maximizing the usage of a limited computing budget. This method works by resampling candidate solutions in order to determine the best budget allocation for the iteration. It assigns the evaluations to the most promising solutions, instead of equally allocating the evaluations between search individuals. Zhang et al. (2016) present a PSO_OCBA allocation rule aiming to maximize PSO convergence rate, with samples allocated incrementally at each stage of PSO until the computing budget is exhausted. Despite the good results obtained, an extensive mathematical analysis is required in order to maintain its efficiency. Souravlias e Parsopoulos (2016) propose a method that allocates the budget to particles according to their neighborhood, using a rank-based criterion to assess the quality of the neighbors. Rada-Vilela et al. (2013), however, demonstrate how inefficient OCBA is from the search algorithm point of view, since a very high number of function evaluations is used for resampling (above 90%), while only a small portion is used for the optimization.

3.2 PARALLEL METAHEURISTIC ALGORITHMS

Proposed by Vu et al. (2011), a master-slave algorithm combining Cooperative Coevolution (CC) and Differential Evolution (DE) strategies uses the slave agents to evolve disjoint subsets of the problem dimensions. Exploration and exploitation occur as consequence of the parallel evolution of each subset of dimensions. The master is responsible for updating the best solution and for the termination criteria, with the communication occurring synchronously and only between the master and the slaves. For that reason, the information is propagated quickly, causing a high communication overhead. This is not a fault tolerant algorithm (since all slaves are required to evolve the entire solution) neither a scalable system (because the number of processes is limited by the number of dimensions). The method uses distributed memory and the authors tested it with up to 6 processes.

A cellular topology GA (cGA) presented by Alba e Dorronsoro (2005) uses the ratio between the radius of the cell neighborhood and the radius of the population, to control diversification (by reducing the ratio) and intensification (by increasing the ratio). This algorithm has a weak scalability, since the propagation occurs synchronously and only between neighbors, what reduces its propagation speed and increases its communication overhead. No mechanism to reduce failures is presented. Although the authors tested it with up to 400 processes in distributed memory, its topology is not as versatile as the topologies used by the other methods analyzed.

Herrera et al. (1999) propose a multi-level metaheuristic in a hybrid island-island topology, where every island is a Distributed GA (DGA), composed of multiple GAs running in parallel. All islands are weakly connected with neighbor islands through a direct GA-GA connection, forming a Hierarchical DGA system (HDGA). The expansion of the system to a multi-level HDGA is suggested (but not implemented or algorithmically described) by turning a DGA into a HDGA. Exploration and exploitation occur respectively through asynchronous global (inter-DGA) and local (intra-DGA) propagation. It has a high fault tolerance, since DGA's are independent, and medium to high scalability, due to sparsely connected topology.

3.3 COOPERATIVE POOL-BASED ALGORITHMS

A-Team (previously described in Section 2.2) is a scalable, fault-tolerant, cyclic architecture that shares solutions across independent optimization algorithms (Talukdar et al. (1998)), using pool of solutions for the explorations and Local Search (LS) for the intensification, with communication occurring asynchronously between agents. This method presents slow propagation of information and low communication overhead.

Talukdar e Ramesh propose an agent (Voyager) for A-Team systems that generates good starting solutions (Probes). A multi-objective variant of A-Team is proposed by Rodrigues (1996), presenting a learning mechanism that uses the non-dominated criterion to sort the pool of solutions.

Proposed by Le Bouthillier e Crainic (2005), Solution Warehouse is a client-server approach similar to A-Team that allows multiple metaheuristics to collaborate to improve results. The communication, however, is not required to be cyclic, and the solutions received are stored into an *in-training* population in the server, and then, a post-optimization process is applied to obtain local improvements. The improved solutions are then stored into the *adult* population, i.e. a pool which becomes available to be used by all metaheuristics. This increases the communication overhead for the server, and considering there is no mechanism to guarantee the complete propagation of information between the agents, the propagation occurs slowly. As long as the warehouse does not fail, this is a fault tolerant system. Considering the server can be implemented in many robust ways (like database systems), this algorithm is scalable.

3.4 PARALLEL HYPERHEURISTIC ALGORITHMS

Meignan et al. (2010) present a coalition-based hyper-heuristic composed by agents, which are independent algorithms that cooperate asynchronously through broadcast to improve the best coalition solution. Every agent uses Reinforcement Learning to select its search operators, which are divided in intensification and exploration ones. This is a fault-tolerant mechanism, since agents can be added or removed without perturbing the global functioning of the system, but broadcast communication reduces scalability.

Ouelhadj e Petrovic (2008) propose a Hyper-Heuristic Agent (HHA) that selects, based on greedy and tabu learning mechanism, the low level heuristic (LLH) to be applied at every iteration. The HHA broadcasts a starting solution and low level heuristics to all Low Level Heuristic Agents (LLHA), which will use the heuristics to improve the starting solution in parallel. The solutions are then sent back to HHA, which in turn stores synchronously the best result in a pool of best solutions. No decomposition is performed, and the exploration / exploitation mechanism is based on the acceptance criterion for the low level heuristic and for the best solutions. It is a fault tolerant algorithm, since HHA does not require all LLHA to be active to keep working, but it has weak scalability, given it is a master-slave topology.

As continuation of a previous work developed by Ouelhadj e Petrovic (2008), Ouelhadj et al. (2009) propose a controller that manages a set of different hyper-heuristics, which in turn uses a set of LLH to generate solutions. A pool of best solutions are maintained by the controller, which is used for asynchronous cooperation between HHAs. No refinement of the solutions or decomposition of the dimensions is performed, and no learning mechanism is employed by the controller. It is a fault-tolerant algorithm as long as the controller is active, but its scalability is limited since the controller cannot deal with too many agents.

Rattadilok et al. (2004) propose a distributed "Choice Function" (i.e., a Reinforcement Learning process) for parallel hyper-heuristic. A parallel hyper-heuristic multi-controller is also proposed, with every controller managing a set of tasks. Each task applies a selected LLH over the selected solution, until a termination criterion is met. Besides starting the tasks, the controller also selects the solution, chooses the LLH through the choice function and sends both to the tasks. The communication between the controller and the tasks is asynchronous in a master-slave topology, but when expanding the system to a multi-controller environment, the communication inter-controller occurs at fixed intervals. The exploration is implemented through LLH acceptance criteria, and the intensification occurs through branching (in this case, different strategies starting from the same solution to obtain different results). This algorithm has high falt tolerance, since at failure events of either a controller or a task, other tasks and controllers will keep their work going. It has good scalability, but limited to the quality of the information exchanged between controllers, since no decomposition mechanism is employed. Authors concluded that this kind of solution is sensitive to the interval of communication between controllers.

Henry Obit (2010) presents a multi-agent system called Asynchronous Cooperative Multi-Hyper-Heuristic (ACMHH), whose agents are independent hyper-heuristics that maintain their own solutions without interference. The agents are fully connected in an island topology, with communication occurring asynchronously. No decomposition or learning mechanism is employed, and exploration / exploitation is controlled through LLH acceptance criteria. Although this system has a high fault tolerance, the quality of results is dependent of frequent information exchange through broadcast, which affects its scalability.

A different approach is presented by Lozano et al. (2016), with a hyper-heuristic developed for parameter tunning of metaheuristics, running on a multi-core system. The proposed algorithm has low fault tolerance, since the system is built to optimize specific metaheuristic algorithms, which cannot fail (unless if multiple instances of same metaheuristic are kept running). For the same reason, it has low scalability. The authors tested this method with up to 250 processes using shared memory.

A distributed hyper-heuristic using a dynamic probability distribution for the learning was presented by Biazzini et al. (2009), whose distribution is updated by using agent's overall performances obtained after every agent execution. No decomposition or exploration / exploitation mechanism is used. Agents are metaheuristics connected in island topology, with asynchronous communication occurring at every fitness evaluation with a random peer, causing a high communication overhead. This is a fault tolerant algorithm, with medium to high scalability since the communication has small impact on the agents processing. Although experiments were conducted with up to 2^{13} processes, the time required for the information to propagate to all peers is an issue in this method ($O(log(n))$).

4 THE TREASURE HUNT FRAMEWORK

This chapter describes all details of the Treasure Hunt (TH) framework. Next section gives an overview of TH and all terms necessary to understand the framework. Section 4.2 details the proposed search space organization method. Section 4.3 illustrates the basic steps performed by TH during a joint optimization in a distributed environment. Section 4.4 describes the TH algorithm, whereas Section 4.5 describes the relocation strategy proposed for TH framework. Sections 4.6 and 4.7 present the methods created in this thesis in support of TH. Finally, Section 4.8 discusses several topics related to TH, including a sub-region selection mechanism that uses grouping dimensions as criterion.

4.1 TREASURE HUNT OVERVIEW

Treasure Hunt (TH) is a distributed framework for distributed framework for complex and high dimensional continuous optimization, comprised of independent TH instances with internal functions established as follows:

1. **Hierarchical organization (Section 4.2):** the root TH is assigned to the full search space, defined as a hyper-rectangular area $\mathcal{H}_0 \subset \mathbb{R}^n$, that can be hierarchically divided for the initialization of the child TH instances (see Figura 4.1):
   - TH is an independent instance of the TH framework.
   - $\mathcal{H}$ is the search space region associated with a TH instance, from TH's own perspective, regardless $\mathcal{H}$'s hierarchical location.
   - **S** (4.1) is a set which organizes the segments obtained by dividing each problem's dimension $j \in \{1, \cdots, n\}$ into K_j non-overlapping segments, forming $\prod_{j=1}^{n} K_j$ non-overlapping hyper-rectangular sub-regions $\mathcal{H}_{\mathfrak{h}}$ (4.3).
   - C is a selection mechanism which chooses from **S** a set $\mathcal{H}_{\mathfrak{h}}^{C}$ of child sub-regions (Figura 4.1a), forming a tree[1] (Figura 4.1b).
   - Every chosen sub-region $\mathcal{H}_{\mathfrak{h}}$ is associated with a distinct child TH instance.
   - Every TH instance has its own population, thus the Root TH's population is initialized inside $\mathcal{H}_0$, whereas every child TH's population is initialized within its chosen sub-region $\mathcal{H}_{\mathfrak{h}}$.
   - Finer granularity can be achieved in this hierarchical structure by applying this process recursively to any selected sub-region.
2. **Distributed optimization (Section 4.3):** Multiple independent populations are hierarchically coordinated (Figura 4.1b), such that an asynchronous joint convergence between all independent populations is obtained as if they were a single distributed population:
   - Every TH instance (parent or children THs) contains its own search group G, that encompasses both the population and the search method.

[1]The terms *node* and *sub-region* hereafter will be used interchangeably, given direct correlation between these terms for TH framework.

  - G's search method is regarded as a black box, that performs full search over the entire search space $\mathcal{H}_0$.
  - Children's populations, however, are initialized within non-overlapping THs' sub-region $\mathfrak{h}$. This increases the chance of finding a local optimum nearby $\mathfrak{h}$, besides reducing the chance of work overlap between the search groups.
- A multi-start process is performed by each TH instance at every TH iteration:
  - G collects the fitness improvements obtained at every search method's iteration, and uses them to detect the convergence stagnation. Once stagnated, the search is interrupted.
  - Each TH instance collects G's best candidate solution. Then, TH controls the local propagation of the information about promising regions by asynchronously sending / receiving information to / from its parent and children.
  - Every TH instance repositions G's population to positions reported by children, and also biased toward information received from the parent (or toward its own best, in case it is the root TH).
  - G re-initiates the search.

Notice that if $\mathcal{H}_0$ is not hierarchically divided, there will be only one TH instance in the search, and therefore, no local propagation of information will occur. In such case, the multi-start method will always occur toward TH instance's own best solution. Also, the term "solution" used here denotes the underlying search space position and any related information (e.g. configurations and statistics).

4.2 HIERARCHICAL ORGANIZATION

Let $\mathbf{D} = \{d_j \mid j = 1, \cdots, n\}$ be the set of n dimensions of a hyper-rectangular continuous search space $\mathcal{H}_0 \subset \mathbb{R}^n$, where d_j is associated with the j-th decision variable of an optimization problem. Every dimension d_j can be divided into non-overlapping segments S_{j,k_j}, such that the set of all segments for all dimensions is given by $\mathbf{S}$:

$$\mathbf{S} = \bigcup_{j=1}^{n} \bigcup_{k_j=1}^{K_j} S_{j,k_j}\ ,\ S_{j,k_j} \subset [S_j^l, S_j^u] \tag{4.1}$$

where S_{j,k_j} denotes the segment k_j of dimension d_j, S_j^l and S_j^u are the lower and upper boundaries of d_j, and K_j is the number of segments the dimension d_j is partitioned into.

Considering segments of equal size (Δ_j) for dimension d_j, $S_{j,k_j}$is given by (4.2).

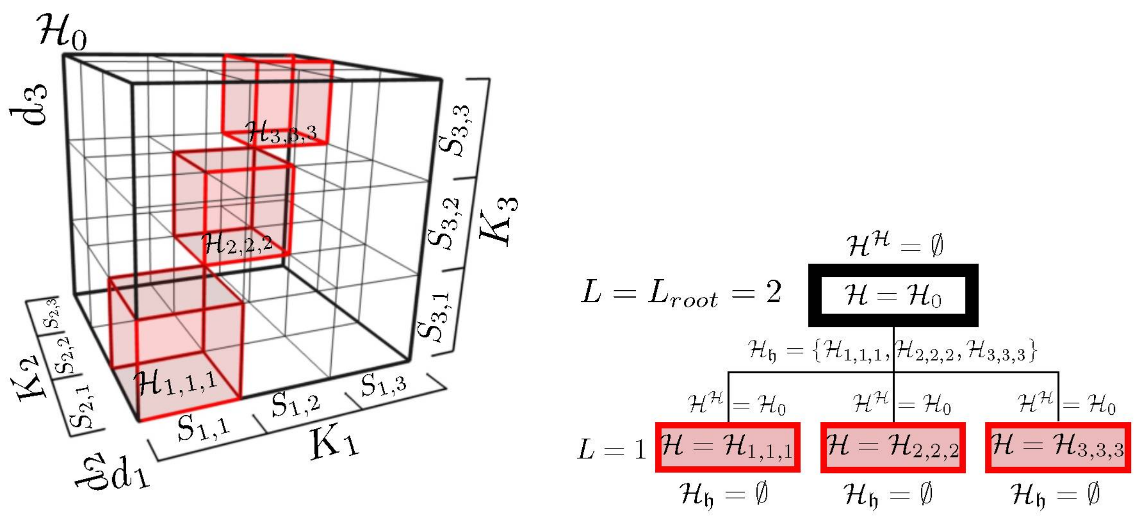
$$S_{j,k_j} = [S_j^l + (k_j - 1)\Delta_j, S_j^l + k_j\Delta_j] \tag{4.2}$$

$$\Delta_j = (S_j^u - S_j^l)/K_j$$

A specific sub-region $\mathcal{H}_{k_1,\ldots,k_n}$ is defined by the Cartesian product of one segment from every dimension d_j:

$$\mathcal{H}_{k_1,\ldots,k_n} = S_{1,k_1} \times \cdots \times S_{n,k_n}\ ,\ \mathcal{H}_{k_1,\ldots,k_n} \subset \mathcal{H}_0 \tag{4.3}$$

such that the union of all possible non-overlapping sub-regions comprises the entire search space $\mathcal{H}_0$. Thus, $\mathcal{H}_0$ can be divided into $\prod_{j=1}^{n} K_j$ disjoint sub-regions delimited by $\mathbf{S}$.


$\mathcal{H}_0$
d_3
$\mathcal{H}_{3,3,3}$
$\mathcal{H}_{2,2,2}$
$\mathcal{H}_{1,1,1}$
$S_{3,3}$
$S_{3,2}$
$S_{3,1}$
K_3
$S_{2,3}$
$S_{2,2}$
$S_{2,1}$
K_2
$S_{1,1}$
$S_{1,2}$
$S_{1,3}$
K_1
d_2
d_1
$\mathcal{H}^{\mathcal{H}} = \emptyset$
$L = L_{root} = 2$
$\mathcal{H} = \mathcal{H}_0$
$\mathcal{H}_{\mathfrak{h}} = \{\mathcal{H}_{1,1,1}, \mathcal{H}_{2,2,2}, \mathcal{H}_{3,3,3}\}$
$\mathcal{H}^{\mathcal{H}} = \mathcal{H}_0$
$\mathcal{H}^{\mathcal{H}} = \mathcal{H}_0$
$\mathcal{H}^{\mathcal{H}} = \mathcal{H}_0$
$L = 1$
$\mathcal{H} = \mathcal{H}_{1,1,1}$
$\mathcal{H} = \mathcal{H}_{2,2,2}$
$\mathcal{H} = \mathcal{H}_{3,3,3}$
$\mathcal{H}_{\mathfrak{h}} = \emptyset$
$\mathcal{H}_{\mathfrak{h}} = \emptyset$
$\mathcal{H}_{\mathfrak{h}} = \emptyset$

(a) Sub-regions chosen at diagonal by criterion (4.4). (b) Hierarchical tree topology obtained by $C(\mathbf{S})$.

Figura 4.1: Cartesian division of the search space $\mathcal{H}_0$. An example with $n = 3$ ($D = \{d_1, d_2, d_3\}$) is partitioned in ($K_1 = K_2 = K_3 = 3$) contiguous intervals of equal sizes (Δ_j).

Given that the total number of sub-regions (or nodes) can be intractable, and aiming to associate a sub-region $\mathcal{H}_{\mathfrak{h}}$ to every TH instance, a selection criterion $C(\mathbf{S})$ must choose a subset $\mathcal{H}_{\mathfrak{h}}^{C} = \{\mathcal{H}_{\mathfrak{h}} | \mathcal{H}_{\mathfrak{h}} \subseteq \mathcal{H}_0, \forall \mathcal{H}_{\mathfrak{h}}\}$. For better coverage of the search space, the partitioning procedure (presented in a simplified form in Figura 4.1) can be recursively applied to create a hierarchical subdivision of the problem, forming a tree topology of height L_{root} in which the root node $\mathcal{H}_0$ contains the whole search space, each parent node $\mathcal{H}^{\mathcal{H}}$ contains its own set of $\mathcal{H}_{\mathfrak{h}}^{C}$ chosen child nodes, and each node $\mathcal{H}$ knows its parent, its children and its level L in the tree.

4.2.1 A simple 3-D example

Consider a three-dimensional problem ($D = \{d_1, d_2, d_3\}, n = 3$) where the variables' domains are segmented in contiguous intervals of equal sizes Δ_j and cardinalities $K_1 = K_2 = K_3 = 3$. The Cartesian product of such intervals produces 27 three-dimensional sub-regions $\mathcal{H}_{\mathfrak{h}}$, and the union of all sub-regions composes the entire search space $\mathcal{H}_0$. Figura 4.1a shows three sub-regions of $\mathcal{H}_0$ chosen at diagonal by selection criterion:

$$C(\mathbf{S}) = \bigcup_{k=1}^{3} S_{1,k} \times S_{2,k} \times S_{3,k} \tag{4.4}$$

forming a tree topology (Figura 4.1b) of height $L_{root} = 2$ (i.e., levels $L = 2$ for root node and $L = 1$ for leaf nodes), with one parent region $\mathcal{H}_0$ and three chosen sub-regions $\{\mathcal{H}_{1,1,1}, \mathcal{H}_{2,2,2}, \mathcal{H}_{3,3,3}\} = \mathcal{H}_{\mathfrak{h}}^{C}$.

4.3 DISTRIBUTED OPTIMIZATION

The hierarchical organization described in the previous section enables a distributed optimization, which can be summarized in six main steps (described here with $L_{root} = 2$, for simplicity aspects):

1. The full search space $\mathcal{H}_0$ assigned to the root TH instance is subdivided, and $nChildren$ child sub-regions ($\mathcal{H}^C_{\mathfrak{h}}$) are chosen according to a selection criterion $C(\mathbf{S})$ (Figura 4.2 Step 1). Each child sub-region is then associated with a subordinate child TH instance.

2. At first iteration ($t = 1$), all p individuals from each search group G (including the root TH) are randomly positioned into TH's designated sub-region (for clarity purpose, root TH's population is hidden from Figura 4.2 Step 2).

3. Despite the restricted initialization, the search group G of every TH performs an independent full search (ignoring the boundaries of its associated search space) and obtains one local minimum from its own perspective (Figura 4.2 Step 3).

4. The search group of the root TH performs its own full search, starting from the following locations: (i) the children's best solutions; (ii) the best solution of the iteration (gb_t) found by its personal search group; (iii) locations generated using the β-relocation strategy (described on Section 4.5), pointing to its own general best solution gb (Figura 4.2 Step 4).

5. The root TH's best solution (Figura 4.2 Step 5) is a result from an exploration based on the collaborative search performed. It is added to the root TH's best-list, and then a random solution is selected from this list and sent back to the children, to influence their movement.

6. Each child TH instance uses the solution received from the root TH as bias to reposition its individuals (except gb_t), using the β-relocation strategy pointing to the received solution (Figura 4.2 Step 6).

7. All TH instances re-initiate their search (step 3).

Given TH's distributed and asynchronous characteristics, all TH instances execute steps 2 to 6 in parallel with each other. It is important to emphasize that these steps imply two types of iterations occurring during a TH optimization: 1) the search algorithm's iterations, which are managed by the search group G until the search algorithm's completion; 2) the TH's iteration, which is comprised of G's final result, a communication process with parent and children, and a population repositioning process.

As a requirement for the optimization to occur jointly between all TH instances, there must be cooperation between search groups to allow the propagation of potentially good information through all the nodes in the tree. However, some groups might take much longer than others to converge, due to difficulties the search method may face when optimizing particularly harder sub-regions. Considering the asynchronous cooperation between TH instances only occurs after each search groups execution has completed, slow convergences might reduce the cooperation between search groups and, consequently, affect the joint optimization. To mitigate this problem, the communication frequency could be improved by using CSMOn (described on Section 4.7), which detects the convergence stagnation and stops the search after a nearly optimal number of fitness evaluations is performed. Hence, the cooperation, which is of high importance in a parallel environment, yields high-quality low-cost results on TH.

4.4 TH ALGORITHM

Treasure Hunt framework works through the execution of distributed TH instances, which are initialized in disjoint sub-regions selected by criterion $C(\mathbf{S})$. TH framework is comprised of three phases (Algorithm 1):

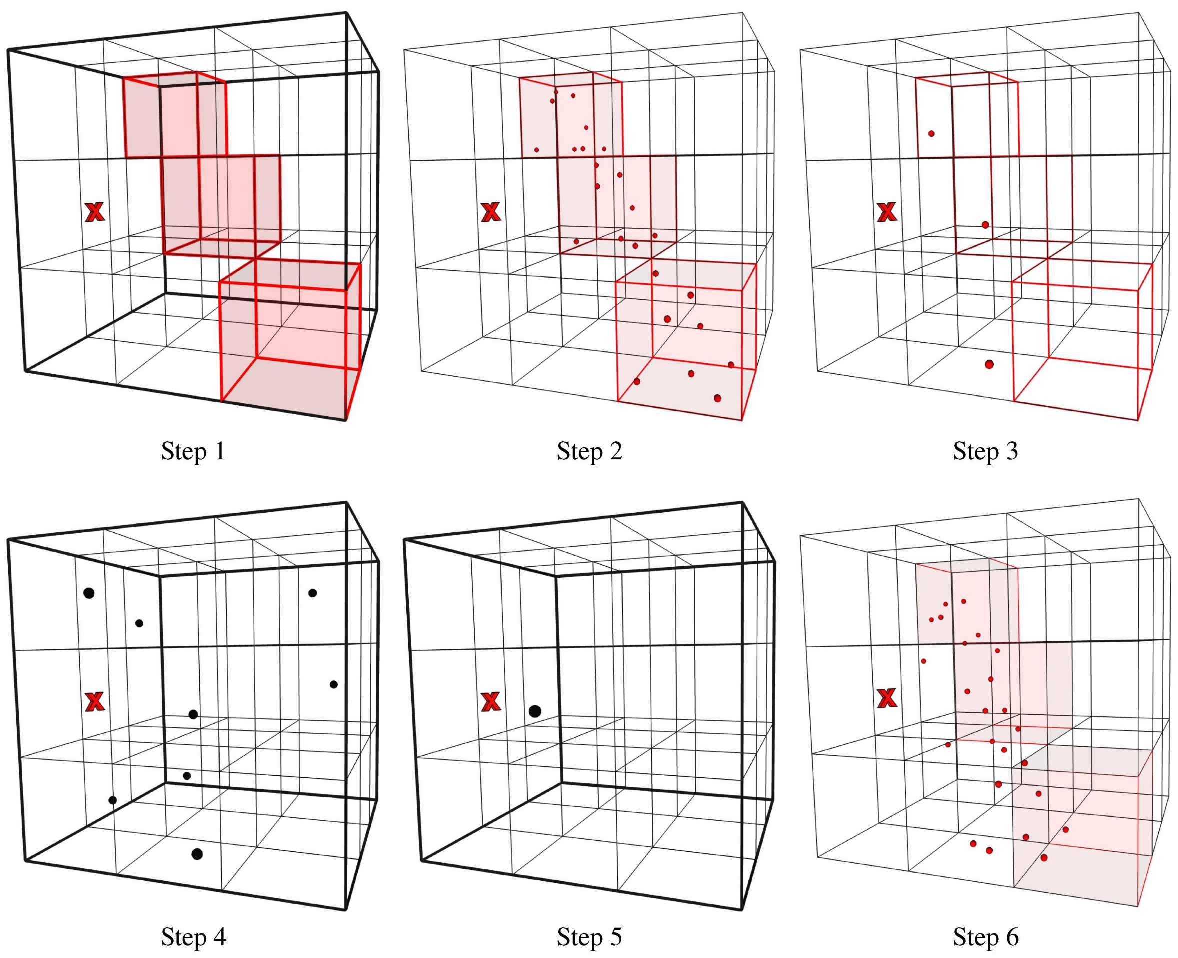


Figura 4.2: Simplified iteration of TH's joint optimization mechanism. Configuration = $[D : \{d_1, d_2, d_3\}, n : 3, \mathbf{K} : 3, p : 8]$. "X" is the global optimum. Step 1: $\mathcal{H}_0$ (the external cube) is subdivided in $nChildren = 27$ sub-regions and $\mathcal{H}_{\mathfrak{h}}^{C}$ (the red cubes) are chosen by selection criterion (4.4). Step 2: Search groups' individuals are initialized into respective constrained sub-regions (showing only children search groups, for clarity). Step 3: Unconstrained full searches find a best solution for each search group. Step 4: Root TH's search group starts from its group's and children's best results, and from locations generated using the β-relocation strategy. Step 5: Best solution found by root TH is sent back to children. Step 6: Children search groups reposition individuals biased toward root TH's best and start a new iteration (i.e., restart the search, returning to Step 3).

- **Initialization:** the p individuals of G's population are uniformly initialized at the assigned region $\mathcal{H}$ (Algorithm 1, line 2).
- **Full search:** a method to control the joint convergence between the TH instances runs the search group iteratively, controls the propagation of the information between parent and children, and manages the repositioning of population's individuals between iterations (Algorithm 1, lines 4 to 21).
- **Residual Communication:** if a TH instance has finished but its children ($\mathcal{H}_{\text{TH}}$) have not, it continues propagating children's information to its parent ($\mathcal{H}^{\text{TH}}$) and active children, until all its children are inactive (Algorithm 1, lines 22 to 28).

  The following additional representation is used:

- t is the number of the current TH iteration and T is the maximum number of iterations.
- gb_t is the best candidate of current TH iteration t.
- gb is a copy of TH's overall best candidate solution ($gb \leq gb_t$ in a minimization problem). Due to self-organization, gb is also the best solution for the current TH's subtree.
- $\mathcal{H}^{\text{TH}}$ is the parent TH instance, and $\mathcal{H}^{\text{TH}_b}$ is a copy of a random result from parent's best-list (or a copy of TH's gb, in case it is the first iteration or the root instance).
- $\mathcal{H}_{\text{TH}}$ are children TH instances and $\mathcal{H}_{\text{TH}_\text{b}}$ is a list with a copy of children's best solutions.
- LS is a local search function that receives a list of candidate solutions and optimizes them individually.

  The Full Search phase performs the actual optimization:

1. G's full search (Algorithm 1, line 5):
   - G executes the search algorithm over its population, then updates TH's gb and best-list.
   - Despite being initialized inside $\mathcal{H}$, the search group moves are constrained to $\mathcal{H}_0$.
   - G controls the running time of the search algorithm through CSMOn method (Section 4.7).
   - gb_t is always kept for the next iteration. As long as the search algorithm returns its best individual gb_t, there is no restriction on the algorithm used by G (e.g. population-based metaheuristic, hyperheuristic algorithm and mathematical method - see Section 2 for a comprehensive list).
2. A local cooperation is performed between current TH instance, its parent and children, so that (given enough budget) a joint convergence is obtained with reduced cooperative effort. If current TH instance has no new information to be reported, its parent and children use their last known information, so that the search can continue without delay. All communications occur asynchronously to allow independence between TH instances. In order to communicate with:
   - Parent (Algorithm 1, line 8), the TH instance:

Algoritmo 1 Treasure Hunt

```
1: Input: [$\mathcal{H}_0$, $\mathcal{H}$, $\mathcal{H}^{\mathsf{TH}}$, $\mathcal{H}_{\mathsf{TH}}$, $p$, $T$]
2: Initialize G with $p$ and $\mathcal{H}$    ▷ First phase: Initialization
3: $t \leftarrow 1$
4: do    ▷ Second phase: Full Search
5:     G performs a complete search over $\mathcal{H}_0$
6:     if $t = 1$ then    ▷ Initialize $\mathcal{H}^{\mathsf{TH}_b}$
7:         $\mathcal{H}^{\mathsf{TH}_b} \leftarrow gb$
8:     if TH has parent then    ▷ $\mathcal{H}^{\mathsf{TH}} \neq \emptyset$
9:         Asynchronous: sends [$gb$, Active] to $\mathcal{H}^{\mathsf{TH}}$
10:        Asynchronous: receives $\mathcal{H}^{\mathsf{TH}_b}$
11:    else    ▷ The root level
12:        $\mathcal{H}^{\mathsf{TH}_b} \leftarrow gb$
13:    if TH has child then    ▷ $\mathcal{H}_{\mathsf{TH}} \neq \emptyset$
14:        Asynchronous: receives $\mathcal{H}_{\mathsf{TH_b}}$ and status
             from active children
15:        $\mathcal{H}_{\mathsf{TH'_b}} \leftarrow LS(\mathcal{H}_{\mathsf{TH_b}})$
16:        Move $nChildren$ individuals in G
             to $\mathcal{H}_{\mathsf{TH'_b}}$ (except $gb_t$)
17:        Update best-list and $gb$ with $\mathcal{H}_{\mathsf{TH'_b}}$
18:        Asynchronous: sends 1 random solution
             from best-list to all active $\mathcal{H}_{\mathsf{TH}}$
19:    Perform the $\beta$-relocation strategy
         on $(p - nChildren - 1)$ individuals
20:    $t \leftarrow t + 1$
21: while $(t \leq T)$
22: while has active children do    ▷ Third phase: Residual Communication
23:    Asynchronous: receives $\mathcal{H}_{\mathsf{TH_b}}$ and status
         from active children
24:    if $\mathcal{H}_{\mathsf{TH_b}}$ is better than $gb$ then
25:        Update $gb$ with$\mathcal{H}_{\mathsf{TH_b}}$
26:        Asynchronous: sends $gb$ to $\mathcal{H}^{\mathsf{TH}}$ (if $\mathcal{H}^{\mathsf{TH}} \neq \emptyset$)
             and to active $\mathcal{H}_{\mathsf{TH}}$
27: if TH has parent then    ▷ $\mathcal{H}^{\mathsf{TH}} \neq \emptyset$
28:    Asynchronous: sends [$gb$, Inactive] to $\mathcal{H}^{\mathsf{TH}}$
29: return [$gb$, best-list]
```

   - Sends gb and the status *active* to the parent.
   - Receives $\mathcal{H}^{\mathsf{TH}_b}$ from the parent.

- Children (Algorithm 1, line 13), the TH instance:
   - Receives the best candidate solution and the status from every active children, updating the lists $\mathcal{H}_{\mathsf{TH_b}}$ and **status**, respectively. A local optimization of children's results $LS(\mathcal{H}_{\mathsf{TH_b}})$ yields $\mathcal{H}_{\mathsf{TH'_b}}$ (Algorithm 1, line 15). Then, both gb and best-list are updated with $\mathcal{H}_{\mathsf{TH'_b}}$, and $nChildren$ individuals on G (except gb_t) are relocated to $\mathcal{H}_{\mathsf{TH'_b}}$.

   – Selects a single random solution from current best-list and sends to all children ($\mathcal{H}_{\mathsf{TH}}$).

3. Remaining individuals in G are repositioned biased toward $\mathcal{H}^{\mathsf{TH}_b}$, using β-relocation strategy (Section 4.5) to guide the joint convergence (Algorithm 1, line 19). Since at least one individual in G must be repositioned through β-relocation strategy, if $\mathcal{H}_{\mathsf{TH}} \neq \emptyset$, $p \geq nChildren + 2$ (i.e., at least 1 for β-relocation, 1 for gb_t and $nChildren$ for $\mathcal{H}_{\mathsf{TH}'_b}$).

The best-list is a pool (memory) mechanism that stores solutions from multiple perspectives, playing a special role in the control of the convergence stability (Rudolph (1994)). It is updated at every TH iteration with its own group's and children's best results (Algorithm 1, lines 5 and 17), keeping the candidate solutions that either: (i) maximize the diversity (e.g. replacing solutions with worse fitness and smallest Cartesian distances from the new candidate solutions), what is specially important on multimodal fitness functions; (ii) minimize the diversity (e.g. replacing the largest distance instead), what can be used to accelerate the convergence; or (iii) combine both (i) and (ii), to control the exploration/exploitation behavior. Considering this thesis focuses more on convergence behavior, the second method is used for the experiments.

As stated previously on Section 2.3, branching helps escaping from strong local minima without losing important information (e.g. Simulated Annealing (Kirkpatrick et al. (1983)). Therefore, to reduce the chance of the parent to be stagnated in the same trap as the child in its last iteration, before adding child's candidate solution to the best-list, the solution is locally optimized with an LS function.

Once the TH instance has completed its optimization, the residual communication is propagated to its parent until all its children are inactive (Algorithm 1, line 22):

1. Results from parent are ignored, forcing children into exploitation.

2. Results from active children and respective statuses are read, and gb is updated if necessary (no LS is performed neither the best-list is updated or replicated, allowing a smooth convergence stabilization).

3. If gb is updated, it is sent to parent and active children.

Once all children are inactive (and consequently, the subtree at current hierarchical level), TH informs its parent that local cooperation has completed (Algorithm 1, line 28), returning a list of variate good results (Alba et al. (2013)) containing both gb and best-list.

4.5 β-RELOCATION STRATEGY

A dynamic beta distribution is used to ensure weakly but increasingly overlapping starting positions for the search groups' populations, such that areas between these populations could also be evaluated. It also controls the probability of proximity to the best position, ensuring joint convergence to a global point of equilibrium for all search groups (notice that the largest the distance between the solutions inside the best-list, the smallest the chance of reaching such point of equilibrium).

After relocating individuals to children's best positions (in case it is a parent), the remaining individuals in the population (except gb_t) are repositioned by following these two steps:

a) Individuals are uniformly distributed ($r_{i,j}$) within $\mathcal{H}$.

b) These random initial positions $r_{i,j}$ are then repositioned $(x_{i,j})$ closer to the parent's best position $\mathcal{H}^{\mathrm{TH}_b}$ (or to its own best position, in case it is the root TH), with probability of no repositioning β_p.

$$\begin{aligned} x_{i,j}(t) &= r_{i,j} - \mathcal{B}[r_{i,j} - \mathcal{H}^{\mathrm{TH}_b,j}] \\ i = 1,\cdots,p & \qquad j = 1,\cdots,n \\ r_{i,j} &= \mathcal{U}(s(\mathcal{H},j)) \\ \mathcal{B} &= B(\beta_{max} - \beta_p, \beta_p) \\ \beta_p &= p_s \times \beta_{max} \times {p_\delta}^{p_w} \end{aligned} \tag{4.5}$$

$$p_\delta = 1 - t/T \tag{4.6}$$

where $x_{i,j}$ is the new position for dimension d_j of the individual i, $r_{i,j}$ is a random position generated in the interval $s(\mathcal{H},j)$, s is the function that extracts $\mathcal{H}$'s boundaries on its dimension d_j, $\mathcal{U}$ is a random number generator for uniform distribution within the specified interval, $\mathcal{H}^{\mathrm{TH}_b,j}$ is the dimension d_j of the best solution sent by the parent (or from itself, in case it is the root TH), B is a random number generator for the Beta distribution with complementary probability β_p, and β_{max} controls the dispersion of the beta distribution (i.e., its shape). Examples of β shapes are centric ($\beta_{max} = 100$) and sparse ($\beta_{max} = 1$). The parameters p_s ($p_s \lessapprox 0.99$) and p_w ($p_w > 0$) relate to how fast the child TH's convergence needs to go toward the best results sent by the respective parent. Considering the distance between an individual's random position $r_{i,j}$ and its parent's best $\mathcal{H}^{\mathrm{TH}_b,j}$, the starting percentage p_s specifies how far the relocated position $x_{i,j}$ is expected to be from parent's best at first TH iterations (i.e. a large p_s would force a more detailed evaluation on assigned region $\mathcal{H}$). The acceleration coefficient p_w, in turn, configures an exponential probability (the weight) of the individuals to be closer to parent's best as p_δ decreases, gradually enforcing a joint convergence to a common region of the search space. In a general case, $p_s \in [0.9, 0.99]$ and $p_w \in [1,2]$, however, when information flow between nodes in the tree topology is required to occur more aggressively (due to multiple reasons like network problems, large processing times and large number of tree levels), these two parameters can be adjusted (by reducing p_s and increasing p_w) to achieve the desired convergence speed.

Equation (4.6) presents a linear strategy to fine tune the beta movement strategy. The displacement rate p_δ is a time-dependent parameter, based on TH's current iteration t and the maximum number of iterations T, that controls the percentage of p_s that should remain over time as the search progresses, i.e., it dynamically controls throughout iterations the proximity of the individual to the parent's best $\mathcal{H}^{\mathrm{TH}_b}$, hence forcing all child THs to converge to a common area of the search space.

4.6 ITERATIVE PARTITIONING METHOD

PSO algorithm is well known for its good convergence capability, balanced with its ability of exploring large portions of the search space. However its performance usually deteriorates as the number of dimensions increases. The multi-swarm PSO variant helps to mitigate this problem by splitting dimensions into several collaborative swarms, each swarm exploring its own piece of the search space, and all swarms working together to improve the results. Nevertheless, the cooperative version may also get trapped into poor local minima when the number of dimensions is too high. CCPSO2 is an algorithm that tries to mitigate this problem by both permutating the dimensions and dynamically changing the number of swarms.

As a state-of-the-art algorithm, CCPSO2 is used as a Cooperative Coevolution representative for the experiments in this thesis. However, as discussed in Poorjandaghi e Afshar (2014), CCPSO2 does not perform on non-separable problems as good as it performs on separable ones. On Omidvar et al. (2014) a Differential Grouping was proposed to find interdependencies among variables, with good improvements for large scale problems. As recommended in Li e Yao (2012), a good choice for the swarm sizes would have a significant impact on CCPSO2 performance. Hence, a mechanism to dynamically control dimensions and swarms creation might consider more potential variable interactions (Yang et al. (2008)), therefore increasing the chance of obtaining better performances on non-separable problems.

For this reason, the Iterative Partitioning Method for CCPSO2 (CCPSO2-IP (Perroni et al. (2015))) has been proposed in this thesis to improve CCPSO2 performance, by using combinations of number of swarms that promote diversity at the beginning of the search, and intensification at later stages. Three boost functions are proposed to control the method behavior: $Boost_L$ (Linear), $Boost_E$ (Exponential) and $Boost_S$ (Sigmoid). For fine tuning, it also provides a boost rate (B_r) and the number of tries before boosting ($maxTries$).

Considering the IP (Iterative Partitioning) method provides an automated balancing mechanism between exploration and exploitation, other search algorithms besides CCPSO2 could also benefit from such feature during an optimization run. For this reason, most experiments presented in this thesis use the IP method as a dynamic weight function to fine tune the convergence, instead of using the linear equation (4.6).

The article presented in Appendix A.3 details the IP method, which was was proposed during the research activities of this thesis, and published in the Brazilian Conference on Intelligent Systems - BRACIS 2015 (Perroni et al. (2015)).

4.7 CONVERGENCE STABILIZATION MODELING IN ONLINE MODE (CSMON)

Current methods that try to model the convergence behavior, like MBOs and OCBAs, focus on improving the global convergence, requiring extensive changes to the original metaheuristic, what makes them specific for every search algorithm. Aiming on local convergence analysis instead of global one, the algorithm-generic method called Convergence Stabilization Modeling operating in Online mode (CSMOn) is proposed by this thesis.

CSMOn proposes an automatic algorithm-independent approach for the budget allocation problem, focusing on the economy of the budget instead of maximizing its usage. It uses the standard behavior presented by general memory-based metaheuristics (like swarm algorithms) to provide an on-line estimation of a good cost/benefit stopping point for every optimization run, so that the waste on the function evaluations is minimum.

Part of the good results obtained by the swarm algorithms are due to the social knowledge obtained by interacting individuals within swarm. This knowledge, however, acts like a constraint over individuals movements, gradually restricting their positioning on the search space (assuming that the convergence stable point is possible (Van Den Bergh e Engelbrecht (2006))). Considering that at least one local minimum is reachable by the individuals, the quick displacement toward the minimum causes an increase in the convergence rate that matches the behavior of an exponential curve, whereas a log-like convergence behavior can bee seen when reaching the stable point in the local minimum (Figura 4.3).

Such convergence behavior when moving through a local minimum can be used to model, in online mode, a plateau curve that describes the current convergence. Linear regression methods can therefore be used to determine, for every run, the moment when the modeled curve reaches a smooth decrease in its values.

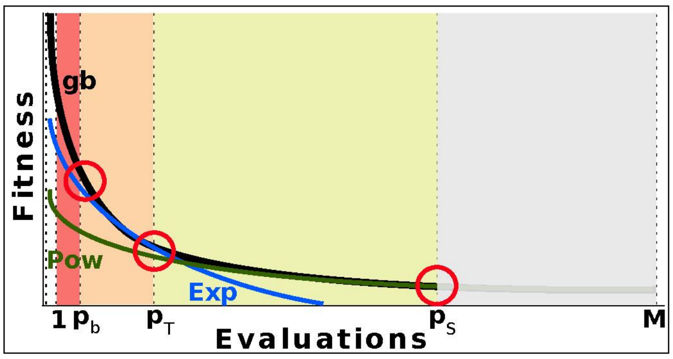


Figura 4.3: CSMOn method
Transition points for fast, intermediate and slow convergence.

Different from existent methods, CSMOn treats the search algorithm as a black-box, without requiring any modification to its original behavior. For that reason, CSMOn has been used by TH's search group (G) to control the number of evaluations (the budget) for all search algorithms used by TH framework.

Although TH can use other methods to detect the convergence stagnation, the high efficiency of CSMOn is a critical factor that makes it very important for TH's good long term convergence. The article presented in Appendix A.4 details the method CSMOn, which was was proposed during the research activities of this thesis, and published in The Genetic and Evolutionary Computation Conference - GECCO 2017 (Perroni et al. (2017)).

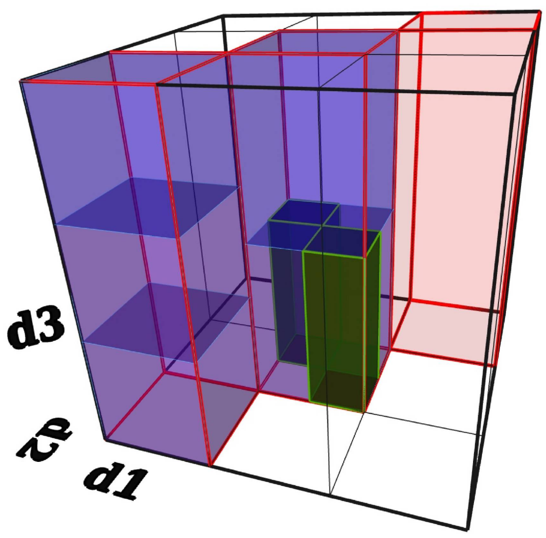
4.8 DISCUSSIONS

The following sections discuss relevant topics about Treasure Hunt framework's architectural design, distributed communication, convergence behavior and configuration.

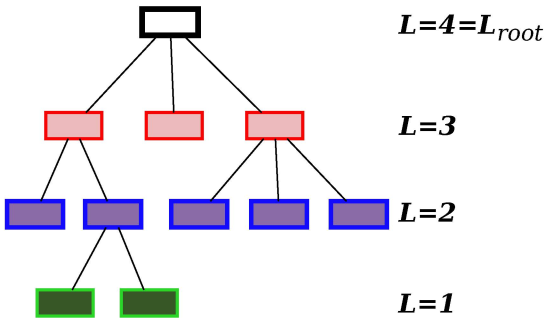
4.8.1 Hierarchical Topology

The fact that TH search groups can execute independently from each other provides a seamless mechanism to integrate optimization processes into large HPC environments, allowing the utilization of the full potential of a cluster by simply running one search group per processor. In order to match the cluster structure, both the hierarchical partitioning of the search space and the selection criterion must take into consideration physical characteristics of the hardware and known properties (if any) of the search space (Figura 4.4a).

Visualizing TH's hierarchical structure as a graph modeling, a tree topology is uncovered (Figura 4.4b), forming a system where any communication between different subtrees requires intermediate coordination from a common higher level node. This mechanism allows only a select group of information to flow between subtrees, filtering bad candidate solutions and obtaining a smooth joint convergence between TH instances. The time required for the complete propagation of the good information is very small, $2 \times (L_{root} - 1)$ TH iterations (i.e., the diameter of the tree (Crainic e Toulouse (2010), depending on the best-list size and on the β-relocation strategy configured), meaning that for every new level added to the tree height, only 2 TH iterations are added to the propagation time. Besides, in this tree topology the number of sub-regions grows exponentially, whereas the number of levels increases linearly (except in the linear topology). Considering that the only information required to flow between the nodes is the best solution from each node and its status, high scaling can be achieved by TH framework.



(a) A 3 variables problem (n = 3).



(b) Respective tree topology.

Figura 4.4: TH hierarchical search space partitioning. Hardware characteristics, like nodes intercommunications with various response times, can be considered when partitioning the search space and selecting sub-regions. Notice the partitioning presented on figures is merely for demonstration purpose and do not reflect any specific hardware or selection criterion.

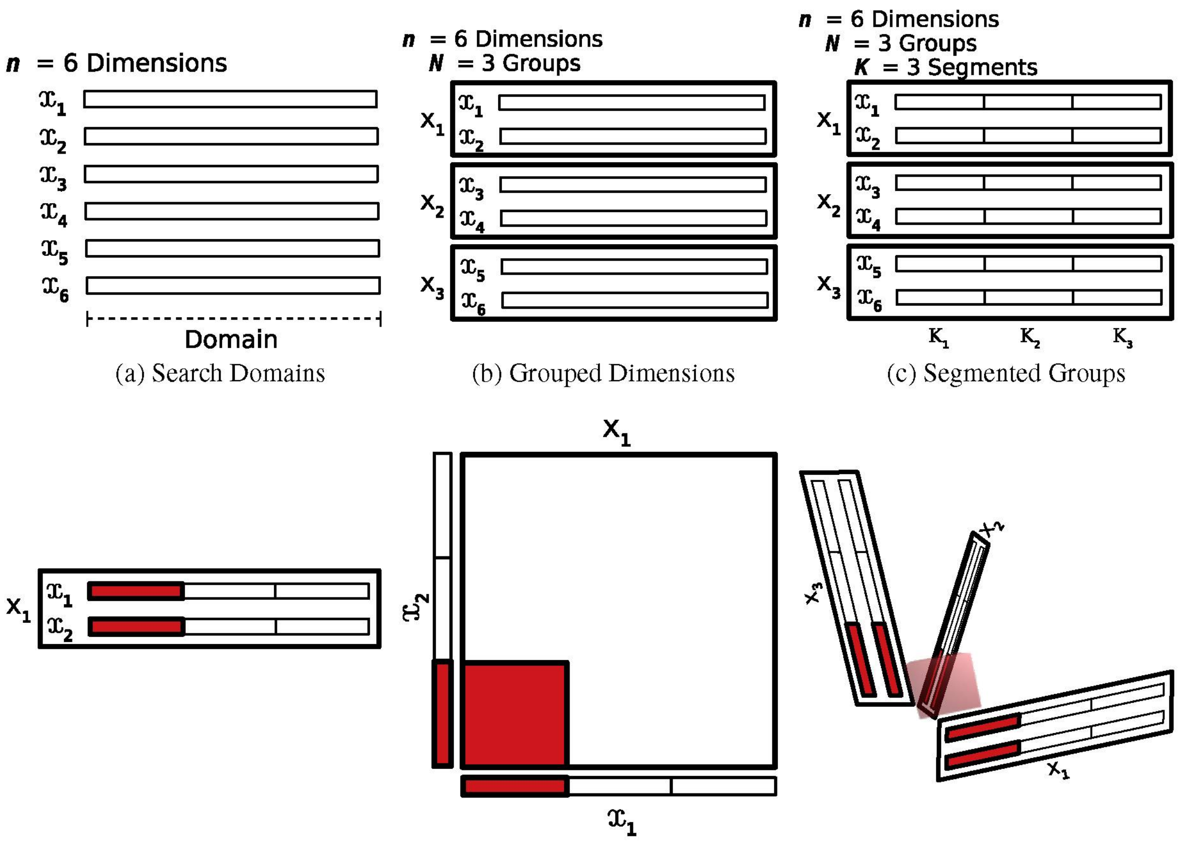
The linear topology is obtained when every parent has only one child, and all nodes have $\mathcal{H} = \mathcal{H}_0$ (with any value of L_{root}). As result, no segmentation is performed and no restriction is imposed over the initial positions generated for the individuals. In this topology, the complete propagation of the good information can occur in $L_{root} - 1$ TH iterations, and every search group provides a refinement over the results obtained by its parent or child, hence acting as a filtering sequence.

Although the diameter of the tree can be very small when compared with the total number of nodes, premature convergence (as mentioned by Crainic e Toulouse) can be mitigated by TH framework since the data flows only after the search algorithm has finished its optimization, meaning that only the final result of each optimization is exchanged between TH instances.

This hierarchical topology, when used with the gradual repositioning method (detailed in Section 4.3), ensures a global cooperative and iterative optimization, performing simultaneously explorations by parents and exploitations by children (on several levels of granularity), making TH capable of quickly visiting a large number of promising regions of the search space (i.e., a comprehensive landscape sampling), and dealing with multimodal problems through the usage of the best-list mechanism.

TH framework can be regarded as a hierarchical island-island topology (Alba et al. (2013)), in the sense that there is a “virtual global population” spread into multiple instances of a serial search algorithm for isolated optimization, with low communication rate between sub-populations, following a specific migration policy to add some diversity. However, there is no global population in TH framework, since each TH instance generates a new population at every TH iteration, and its search group G, responsible for managing the multi-start over the search algorithms, imposes almost no restriction on which search algorithm to use. Typical island topologies, in turn, are mostly classic search algorithms modified for parallel environments.

Despite its advantages, the hierarchical partitioning of the search space domain still does not reduce the complexity of evaluating costly fitness functions (eg. time-consuming / processor intensive equations or problems with very large numbers of dimensions). However, since the search algorithm is independent from TH framework, it can use its own techniques to



(a) Search Domains (b) Grouped Dimensions (c) Segmented Groups

(d) Group's Segment (e) Segment's Cartesian Product (f) Group's Cartesian Product

Figura 4.5: Selection Criterion by Group for a Large Number of Dimensions.
(a) Search domain for the n variables of the problem. (b) Problem's dimensions are grouped in N groups. (c) Each group is segmented in K_l segments (see Equation (4.7)). (d) Variables within the same group only interact at segment k_l. (e) The *"hiper-diagonal"* of the grouped dimensions are selected. (f) The Cartesian product of all *"hiper-diagonals"* form the sub-region selected by this criterion.

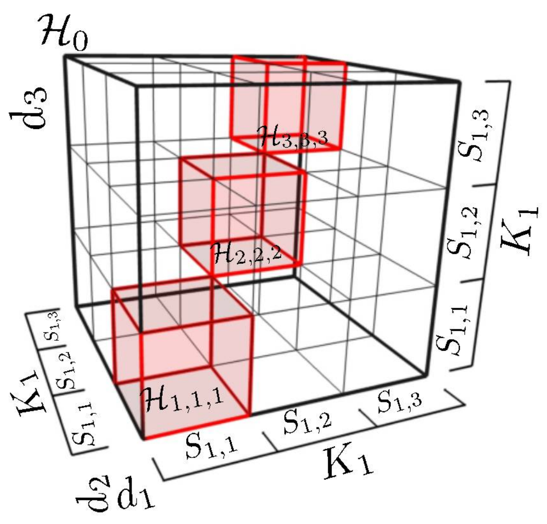
reduce the processing time, e.g., by using trivial parallelism on multiple processors, cooperative coevolutionary methods on multiple processors, or graphical processing unit (GPU) cards.

4.8.2 Selection Criterion by Grouping Dimensions

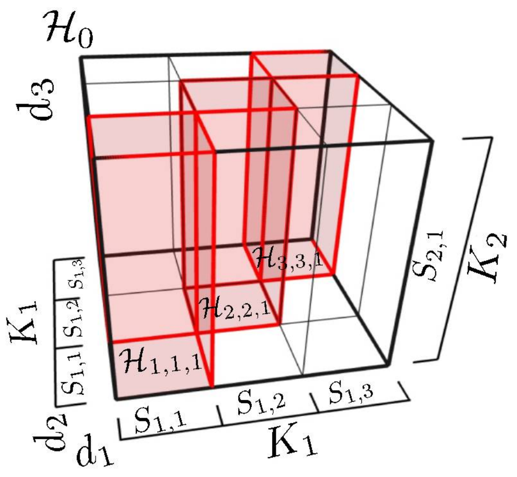
A straightforward criterion to select sub-regions uses dimension grouping when segmenting the search space (Figura 4.5). Let $\mathbf{D} = \{\mathbf{D}_l, l = 1, \ldots, N\}$, where $\mathbf{D}_l$ is a disjoint subset (a group) of the n dimensions and $N \leq n$ is the number of groups. Extending the idea presented in Figura 4.1a, the selection criterion (4.7) ensures that all dimensions within group $\mathbf{D}_l$ use the same number of segments K_l (Figura 4.5c) but only interact at the segments number k_l (Figura 4.5d). As result, this selection criterion chooses only the sub-regions located at the diagonal of the grouped dimensions (Figuras 4.5e and 4.6a), and all sub-regions of groups with one single dimension (Figura 4.6b).

$$C(\mathbf{S}) = \bigcup_{k_1=1}^{K_1} \cdots \bigcup_{k_N=1}^{K_N} \prod_{l=1}^{N} \prod_{j|d_j \in D_l} S_{j,k_l} \tag{4.7}$$

In order to keep a good tree balancing when subdividing the sub-regions, all parents must maintain the same number of child nodes in all tree levels, such that the resulting hierarchy



(a) $N = 1, K_1 = 3$.



(b) $N = 2, K_1 = 3, K_2 = 1$.

Figura 4.6: Selection Criterion by Grouping $n = 3$ Dimensions. Both selections generate the same tree as in Figura 4.1b. (a) A single group ($N = 1$) selects sub-regions at diagonal of $\mathcal{H}_0$. (b) d_1 and d_2 are on same group, thus only their diagonal is combined with d_3. Larger coverage (but less detailed sampling) is obtained when initializing TH instances with (b) than with (a).

presents characteristics of a perfect k-ary tree of height L_{root} (Section 2.5.2). Each node has its own depth level L: leaves have $L = 1$ (sub-regions without children), non-leaves have $L \geq 2$ (parent regions) and $L = L_{root}$ indicates the root node $\mathcal{H}_0$. Notice that the grouping of dimensions into $\mathbf{D}_l$ is problem dependent, hence every node can have its own grouping set $\mathbf{D}$ as long as N is kept the same.

Ideally, an optimal dimension grouping for any given problem is $N = n$ with a large number of partitions K_l. However, considering the unfeasibility of such aggregation for most problems (given the high number of sub-regions generated), $N \ll n$ and its value should consider the computing budget (the maximum number of function evaluations) and the available processing power (number of processors, time limit, etc).

Nevertheless, the reduction in the number of chosen sub-regions necessarily means that some combinations of domain intervals are not covered by non-root instances during the initialization of the first TH iterations (e.g. Figura 4.1a has 27 sub-regions but only 3 are chosen). When subdividing the search space recursively (considering segments of equal sizes), the children of each region are covering a rate of $\prod_{l=1}^{N} K_l / (\prod_{l=1}^{N} K_l^{size(\mathbf{D}_l)})$ of parent's search area (e.g., Figura 4.6a covers $^3/_{27}$, whereas Figura 4.6b covers $^1/_3$ of the search space). On the other hand, considering that every parent region covers a larger portion of the search space than its children, and $\mathcal{H}_0$ covers the entire search space, the β-relocation strategy allows children's populations to "escape" from the area nearby the designated sub-region, causing a smooth long term convergence toward parent's best results and, consequently, a larger sampling of the search space.

It is well known that search methods that aggregate dimensions as a technique for improving results have better chance of success when interacting variables are grouped into the same aggregating component (Yang et al. (2008); Omidvar et al. (2014); Li e Yao (2012); Omidvar e Li (2010)). However, as TH framework does not follow strict *divide-and-conquer* rules, there is not an actual separation of variables on this selection criterion, given that all dimensions are always available for the TH instance to evolve. Nonetheless, the evolution could benefit from this selection criterion by separating interacting variables into different groups (since a segment of a group interacts with all segments of other groups) and grouping variables of similar expected range of values (since variables on the same group interacts on their diagonal).

TH framework does not dictate how each dimension should be segmented, therefore, variables of different domains can be aggregated under the same group D_l either by: (i) segmenting group's domains through proportional segment sizes, instead of using absolute sizes; or ignoring proportions by (ii) using the same number of segments but different segments sizes (narrower segments will benefit from more detailed searches).

This selection criterion obtains the linear topology by using $N = 1$ and $K = 1$ (with any value of L_{root}). For larger configurations, this criterion allows an exponential growth on the search space sampling whilst keeping a linear increase in the communication time. For example, by grouping the n dimensions into $N = 5$ groups and dividing the search space domain in $\mathbf{K} = 3$ segments, for one level of parallelism ($L_{root} = 2$) there will be 243 selected sub-regions at the leaf level ($L = 1$) plus the full search space at the root level ($L = L_{root}$). By assigning 1 TH instance per region, there will be a total of 244 THs with maximum communication time of 2 TH iterations. Raising the depth of parallelism by 1 level ($L_{root} = 3$), the number of TH instances increases to ($L = 1 \rightarrow$ 59,049 THs) + ($L = 2 \rightarrow$ 243 THs) + ($L = 3 \rightarrow$ 1 TH) = 59.293 TH instances with maximum communication time of 4 TH iterations. For $L_{root} = 4$, there will be $14,408,200$ TH instances created at the tree, with maximum communication time of 6 TH iterations, and so on.

4.8.3 Fault Tolerance

Considering that the only critical values for any TH instance are its best-list and gb, to improve fault tolerance on TH framework without affecting the overall communication performance, the parent instances could perform periodic synchronizations with backup nodes.

However, it has to be considered that every parent TH instance contains the best results from its children. Therefore, even in the event of a node failure (hence losing the communication with its entire subtree), the TH instance in the parent node would still contain its subtree's known best results. Besides, temporary communication failures (like communication delays) can be quickly recovered by TH, since the cooperation is immediately recovered as soon as the communication is restored. Regarding the search algorithm's evolution that is lost on faulty nodes, it is not a problem on TH since at every TH iteration, the search algorithm's evolution is entirely reset. For this reason, TH framework can be classified as partially tolerant to communication failures and, except for the root node, no node is crucial for the successful completion of the optimization.

4.8.4 Scalability

Since all communication is asynchronous and all TH instances are independent from each other (fully occupying the processors), TH framework is size scalable and (except when the parents have a large number of children or the search groups converge too quickly) speedup should be close to maximum. Task scalability is achieved as long as the hierarchical search space division makes sense from the problem perspective (i.e. the chosen configuration for TH tree topology and the dimensions partitioning matches the problem requirements).

Despite the fact that TH's tree-topology presents scaling after increasing the problem size (either by adding more dimensions or increasing the search space size), there is a commitment between the problem size and TH's sub-region granularity that must be considered. This is because too fine-grained sub-regions tend to make the search groups in the leaf nodes to converge too quickly. Although the communication mechanism on TH framework can partially mitigate this problem, poor samplings obtained at locations nearby TH's sub-regions would slow down the joint convergence between TH instances.

4.8.5 Guidelines for TH Configuration

Population Size: Populational optimization methods generally use the social interactions between individuals to cooperate or compete in the pursuit of improved results. For that reason, the reduction of the population size can affect such internal interactions of the search algorithm. One way of reducing this strong dependency on the population size is to balance between the reduction in the number of individuals and the increase in the number of TH instances, given that the distributed joint convergence provided by TH allows the multiple independent populations to interact like a "virtual" distributed population. However, ultimately the minimum number of individuals in the leaf nodes should be dictated by the search algorithm used in every TH instance. For example, DE/*best/1/bin* has a minimum limit of 6 individuals imposed by the algorithm, whereas local search algorithms usually can use as few as 1 individual. On parent nodes, the population size is constrained to a minimum of $nChildren+2$ individuals (see Section 4.4 for details).

Variable Correlation: Considering that TH does not decompose the dimensions, there is not a direct advantage in segmenting the search space based on variables dependencies. Instead, the variables should be segmented based on range of values, allowing more exploration on larger segments and more detailed analysis on narrower segments. For example, given that the selection criterion that groups dimensions (Section 4.8.2) only chooses the hyper-diagonal of each group for Cartesian product while generating the sub-regions, variables with similar expected range of values could be grouped into the same group.

Convergence Stagnation: The method proposed to detect the convergence stagnation in this work is CSMOn. Adjusting CSMOn parameters is more related to the processing budget available than to the distributed joint convergence or the optimization problem itself. Although small relaxations is usually the preferred configuration, it might not comply with the requirements of response time. For that reason, it is recommended the use of a dynamic relaxation on CSMOn, allowing quick improvements (i.e. large relaxations) in the initial stages of the search and long term convergences (small relaxations) on the final stages.

Termination criteria: Usual termination criteria are number of evaluations and *wall-clock* time. Considering that one of the main focus of TH is to keep a continuous convergence throughout the entire optimization, there is no direct recommendation for the termination criteria. However, there is a direct relation between the continuous long-term convergence and the number of TH instances, given that additional instances usually means more diversity of results. Such diversity, in turn, might produce longer but less aggressive convergence. Therefore, if the number of TH instances is increased but the processing budget is not, it is recommended to set up more aggressive β-relocation parameter values (see Section 4.5) to balance between diversity and convergence speed.

Hierarchical Tree: While the selection of additional sub-regions (in the same tree level) increases the diversity of results, the sub-division of selected sub-regions (adding a new tree level) intensifies the analysis in those regions. There is no direct recommendation for the number of selected sub-regions neither for the levels of intensification in a hierarchical tree (its height). However, when adding nodes (and consequently, TH instances) to the tree, it has to be considered that the selection of a new sub-region also requires (a slight) additional work for the respective parent TH instance to propagate the information. Furthermore, the addition of a new tree level also requires (slightly) more time for the information to propagate to the entire tree. This time can be increased when using larger *best-list* sizes. Therefore, when distributing additional TH instances, it has to be considered a balance between the joint convergence speed obtained by adjusting the β-relocation parameter values (eg. adjusting the convergence aggressiveness according to the node height), and the number of child TH instances per node (eg. adding child nodes when optimizing computing intensive problems, and adding tree levels in the opposite case).

Multi-Objective Problems: Despite the fact that TH framework is not meant for multi-objective optimization, multi-objective problems can be adapted for TH by combining the multiple objectives into a single-objective function. Considering TH's dynamic internal mechanics, the recommended way to combine the objectives is by assigning dynamic weights to each objective, which should be adjusted automatically as the search progresses.

5 AN EXAMPLE OF TH WORKING PRINCIPLES

Two distinct testing sets are used aiming to illustrate TH's controlled multi-start for 1 single TH instance (Section 5.1) and the joint convergence for the tree-topology (Section 5.2). All tests are performed using the selection criterion by grouping dimensions (4.7) as the selection mechanism.

Given that the experiments are conducted over minimization benchmarks and no parameter tuning is performed for every problem, all tests in this work use the same number of partitions (K) for the N groups, with equal partition sizes. Although the search may benefit from group formations by taking advantage of interacting variables (as previously explained in Section 4.8.2), no specific aggregation rule was used to assign the n dimensions to N groups, and the assignments just followed the sequential order (i.e., $D_1 = \{d_1, \dots\}, \cdots, D_N = \{\dots, d_n\}$).

A comprehensive analysis of the parameters is out of the scope of this work. Considering that each test set may be expressively different from each other, some parameters (described in every experiment) are empirically adjusted solely to suit TH behaviour to the characteristics of every benchmark.

The following general configuration is used (for notation simplicity, L_{root} is henceforth denoted as L):

- TH: best-list size = $max(L, nChildren)$, CSMOn= [relaxation R : 0.8 and maximum number of fitness function evaluations per TH iteration M : $p \times 50$], p_δ using linear equation and p_δ using IP=[$Boost_L$, B_r : 1, $maxTries$: 5].
- Search Algorithms: DE/*best/1/bin* = [F : 0.05, CR : 0.01], PSO = [w : 0.8, $c1$: 0.2, $c2$: 0.2] and Random Search (RS).

Remaining parameters are specified in each experiment. Since DE algorithm is known for its good convergence behavior for a large variety of problems, it was chosen to illustrate most of experiment of this section.

5.1 CONTROLLED MULTI-START

This experiment is a two-dimensional example that demonstrates how TH repositions population's individuals at every multi-start. The configuration used is:

- Benchmark: Rastrigin.
- Search Algorithm: DE/*best/1/bin*.
- TH Topology: $N = 1$, $K = 1$, $L = 1$ (1 TH instance).
- TH: $n = 2$, $p = 36$, $p_s = 0.7$, $\beta_{max} = 100$, p_δ using linear equation, $p_w = 7$, no LS.

In an ideal case, p_w is set to a small value and p_s is set to a high value, such that information can flow smoothly between populations. However, in this test p_w is set to a high value and p_s to a small value to force more aggressive transitions between iterations, so that the full convergence can occur in only 6 TH iterations for demonstration purposes. At every TH instance's multi-start, a snapshot of the population is taken before starting the next full search phase. Although the

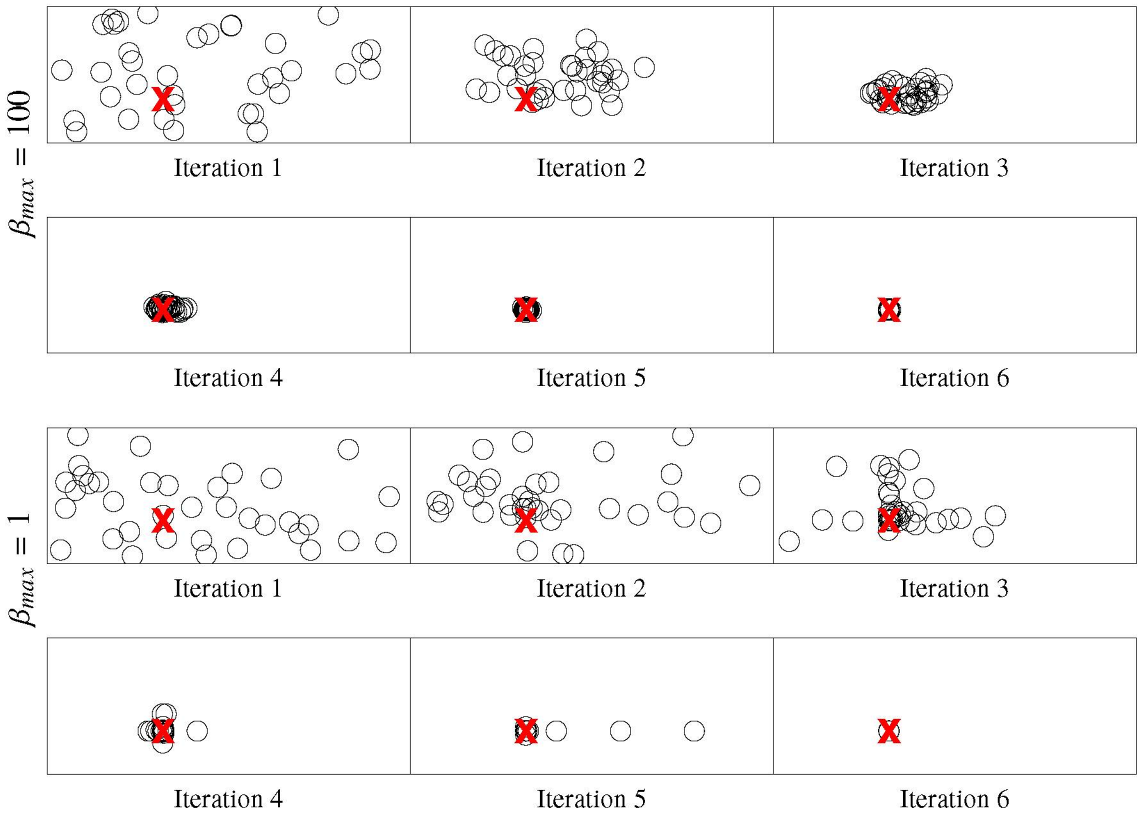


Figura 5.1: Snapshots of the relocation strategy at every TH iteration for 1 TH instance. Configuration = [n : 2, N : 1, K : 1, L : 1, p : 36], p_δ using linear equation. Population individuals are the circles and problem's global optimum is the "X".

global optimum is found at the first few iterations, TH is kept running solely to demonstrate how the β-strategy consistently moves the population toward promising regions.

The resulting effect of using two different distribution shapes (as illustrated in Figura 2.1) is compared in Figura 5.1. Iteration 1 is the random uniform initialization, whereas iterations 2 to 6 show the population moving gradually toward the global optimum, following a centric shape when $\beta_{max} = 100$, and a more sparse distribution when $\beta_{max} = 1$.

5.2 TREE-TOPOLOGY

This testing set illustrates how TH improves simple serial search algorithms by distributing and organizing the optimization effort in a hierarchical tree-topology.

5.2.1 Distributed optimization

This is a 2D example, aiming at demonstrating how TH's tree topology ensures that multiple parallel populations, with asynchronous communication, converge gradually at global level toward the known global best. The configuration used is:

- Benchmark: Rastrigin.
- Search Algorithm: DE/*best/1/bin*.
- TH Topology: $N = 2$, $\mathbf{K} = 3$, $L = 2$ (10 TH instances).

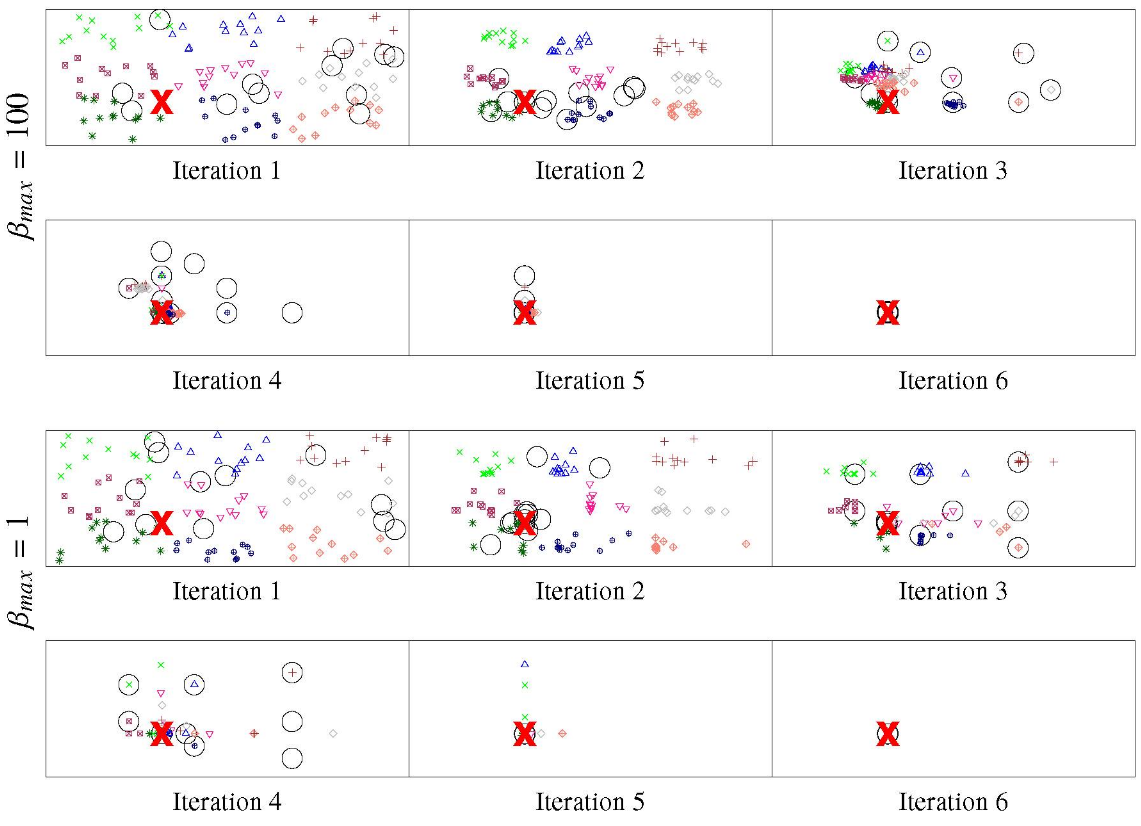


Figura 5.2: Snapshots of the relocation strategy at every TH iteration for 10 TH instances.
Configuration = [n : 2, N : 2, $\mathbf{K}$: 3, L : 2, p : 12], p_δ using linear equation. All sub-regions are selected for demonstration purposes. The population of each of the 9 children is represented by a different symbol, the parent's population is the larger circles and the problem's global optimum is the "X".

- TH: $n = 2$, $p = 12$, $p_s = 0.7$, $\beta_{max} = 100$, p_δ using linear equation, $p_w = 7$, no LS.

As seen in Figura 5.2, the configuration with $\beta_{max} = 100$ enables more detailed intensification in multiple different locations, whereas $\beta_{max} = 1$ provides a larger coverage of the search space. In both cases, all populations converge jointly to the same region of the search space, where the global optimum is located at. A random uniform initialization is performed at the first iteration. Iteration 2 shows one effect of the communication delay between parent and children TH instances, which causes the children to reposition their populations toward their own best, until they receive an input from the parent. Iteration 3 illustrates the effect of exchanging good information between parent and children, with children moving toward parent's best and parent initializing individuals at children's best positions. The resulting effect is a smooth migration of the individuals' positions, from a full search space coverage to an increasingly dimensionwise-restricted search area, following a beta-like repositioning toward the current global best result.

The beta distribution with sparse shape ($\beta_{max} = 1$) is used in conjunction with the IP method to improve p_δ, and results are presented in Figura 5.3. Snapshots are taken at every 5 TH iterations, and no different symbol is used for every population to illustrate the actual search space coverage. We can see that all 9 populations provide a throughout coverage, even after several TH iterations. Nonetheless, despite apparent chaotic repositioning, all TH instances jointly converge to the same region of the search space in a steady pace.

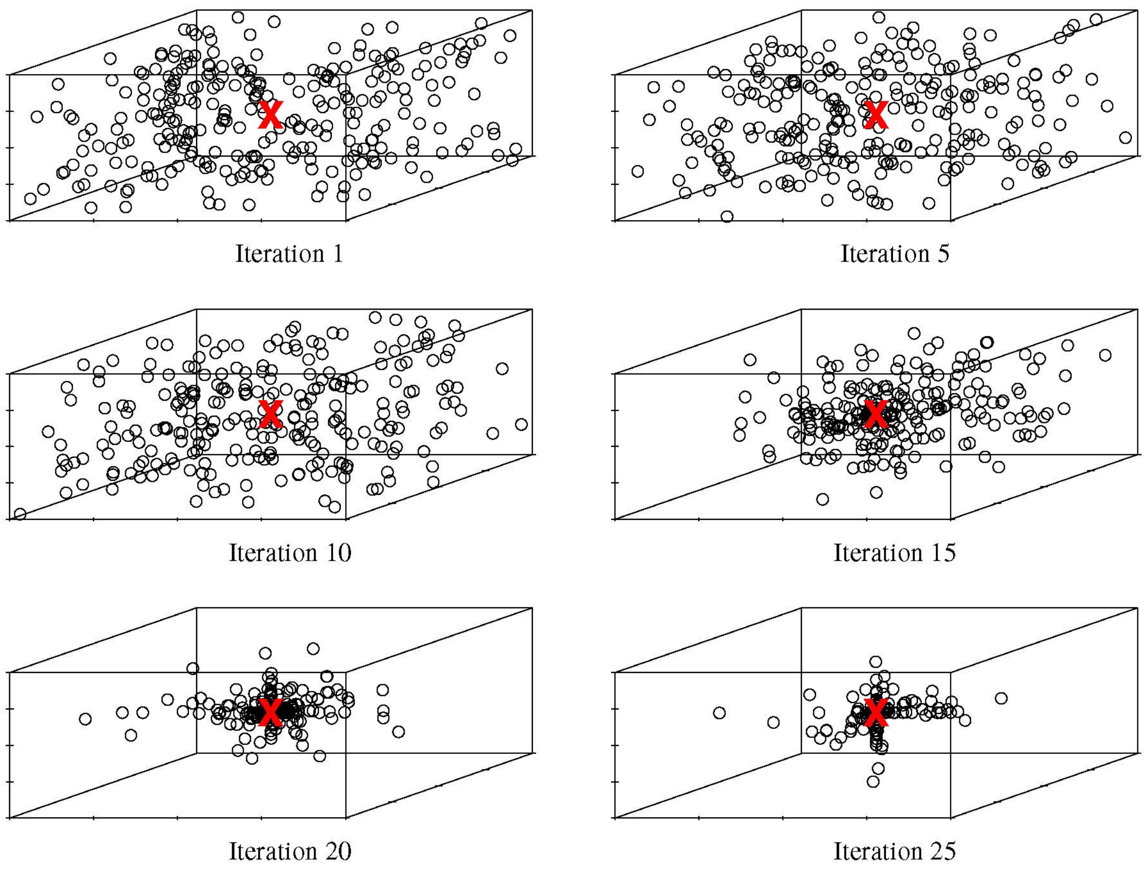


Figura 5.3: Snapshots of the relocation strategy at every 5 TH iterations using IP for p_δ. Configuration = [n : 3, N : 3, $\mathbf{K}$: 2, L : 2 (9 TH instances), β_{max} : 1, p : 30]. All sub-regions are selected for demonstration purposes. All 9 populations are plotted on same color to illustrate the actual search space coverage. The problem's global optimum is the "X".

5.2.2 Information flow between nodes

This example uses a more complex function with $n = 1000$ dimensions (from the competition benchmark CEC13 Li et al. (2013)) to demonstrate how TH topology affects the transfer of good information between the nodes. The following configuration is used to compare the convergence of DE with and without TH:

- Benchmark: F12 (overlapping multimodal function).
- Search Algorithm: *DE/best/1/bin* and Random Search (RS).
- TH Topologies: [$N = 1$, $K = 1$, $L = 1$] (1 TH instance), [$N = 1$, $K = 2$, $L = 2$] (3 TH instances), [$N = 1$, $K = 2$, $L = 3$] (7 TH instances).
- TH: $p_s = 0.99$, $\beta_{max} = 1$, p_δ using linear equation and IP, $p_w = 1$, shared budget of $Mtotal = 5 \times 10^5$ evaluations, LS = [Hill Climbing controlled by CSMOn, with R : 0.01 and M : n], $p = 30$ for DE (standalone), and $p = 6$ for RS with TH and DE with TH (both parallel).

Figura 5.4 shows a quick convergence of standalone DE at the very beginning of the search, followed by a sudden stagnation, whereas DE with TH presents a smooth and continuous convergence throughout the search.

Figura 5.4a compares the results of TH using a pure Random Search (RS) algorithm (RS using normal distribution) and the classic DE algorithm. It is normally expected that a DE algorithm behaves better than a pure random search on a complex problem, however, the resulting relocation mechanism of RS with TH allows the simple randomness of a uniform distribution to perform better than the much more elaborated DE. Besides, it can be seen that RS with TH even follows closely the DE with TH up to half of the optimization, stagnating after that point. Notice that the relocation mechanism provided by RS with TH resembles the CRS4 algorithm (described previously in 2.3), although much more elaborated than CRS4.

Figura 5.4b shows the joint convergence and the information flow between tree nodes, where the TH Root instance always possesses the best information, and its children (Child 1 and Child 2) follow its convergence throughout the search (although at certain distance).

Despite the fact that the final result of TH in Figura 5.4c is only slightly better than DE without TH, it must be considered that the limited budget ($MTotal$) of 5×10^5 evaluations is shared between all 7 TH instances, shortening the evolution of their populations to 1/7th of the standalone DE. It means that every TH instance is allowed to evolve solely 7×10^4 fitness evaluations, nearly the same amount of evaluations that DE standalone required to stagnate. Just like seen in Figura 5.4b, the Root has the best information, and Child 1 and Child 2 follow its convergence. The leaf instances (Child 11, Child 12, Child 21 and Child 22) also follow the convergence, but at a proportionally larger distance, which is the expected behavior of TH's topology because child instances perform the sampling of the search space and report to parents, which in turn refine these samples and send them back to children as a bias to guide their search. This mechanism ensures that parent's good results gradually flow to children, allowing them to escape from their own local optima.

In order to accelerate information flow in a limited budget scenario, the search space sampling can be dynamically maximized by optimizing p_δ with the IP method (Perroni et al. (2015)) (described in Section 4.6), instead of using the linear strategy given by (4.6). IP provides multiple different distances from the known global best (as illustrated in Figura 5.3), resulting in a more stable convergence (as seen in Figura 5.4d, with children following Root instance more quickly), and for that reason, it is used in the next experiments.

5.2.3 Improving simple serial search algorithm

To demonstrate that TH can improve search algorithms not prepared for the problem at hand, the standard PSO (known to execute poorly on high dimensionality) is used to perform a more detailed test over CEC13 1000-dimension functions. The following configuration is used:

- CEC13 Benchmark (cec (2017)): a competition benchmark for Large Scale Global Optimization (LSGO) prepared for IEEE Congress on Evolutionary Computation 2013/2015, comprised of 15 complex functions of 1000-dimension:
  - Fully-separable Functions: F1, F2, F3.
  - Partially Additively Separable Functions: F4, F5, F6, F7, F8, F9, F10, F11.
  - Overlapping Functions: F12, F13, F14.
  - Non-separable Function: F15.
- Search Algorithm: standard PSO.
- TH Topologies: 1 Single TH instance ($N = 1$, $K = 1$, $L = 1$), linear topology with 2 THs ($N = 1$, $K = 1$, $L = 2$) and 3 THs ($N = 1$, $K = 1$, $L = 3$), and pyramidal topology with 3 THs ($N = 1$, $K = 2$, $L = 2$).

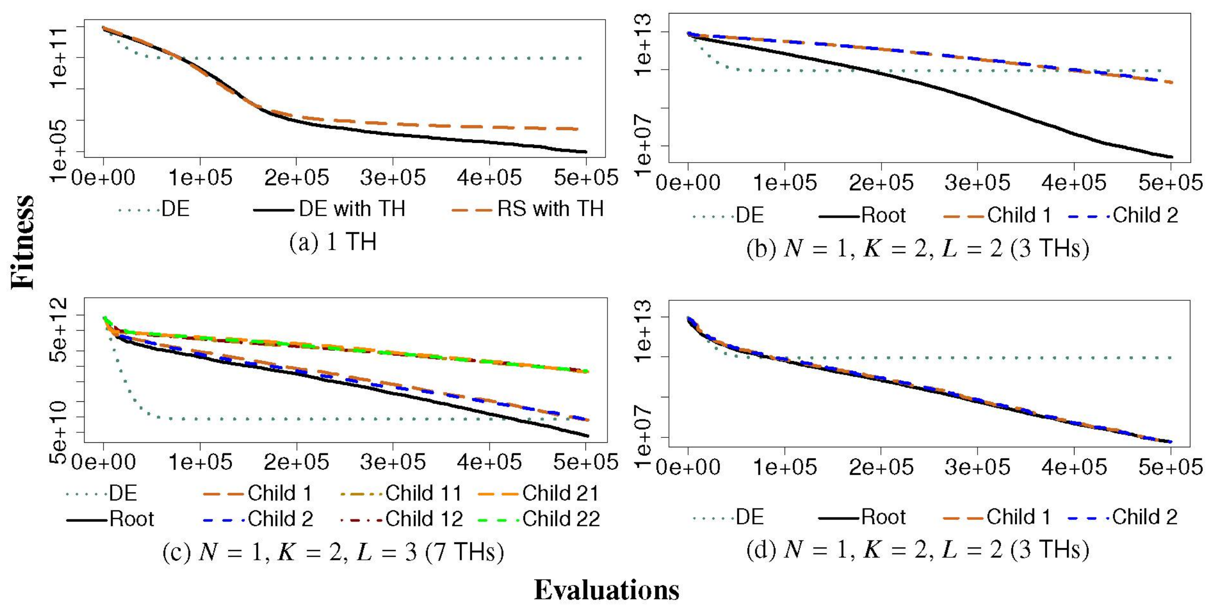


Figura 5.4: Convergence behavior of TH instances, compared with standalone DE.
(a) DE with TH compared with RS with TH (1 TH), with p_Δ using linear repositioning. (b) DE with TH (3 TH), with p_Δ using linear repositioning. (c) DE with TH (7 TH), with p_Δ using linear repositioning. (d) DE with TH (3 TH), with p_Δ using IP. Root is the TH instance at tree's root node, {Child 1, Child 2} are Root's children, {Child 11, Child 12} are Child 1's children and {Child 21, Child 22} are Child 2's children. Search algorithms: DE/*best/1/bin* and RS. Function: F12 (CEC13). p = 30 for standalone DE, and p = 6 for DE with TH and RS with TH.

- TH: $p_s = 0.99$, $\beta_{max} = 1$, p_δ using IP, $p_w = 1$, LS = [Hill Climbing controlled by CSMOn, with $M : n$, $R : 0.01$], 30 independent executions, shared budget of $Mtotal = 5 \times 10^5$ evaluations, $p = 30$ for standalone PSO and $p = 6$ for PSO with TH.

Figura 5.5 shows that TH improves PSO's results in all cases, increasing its stability (lower standard deviation) for most functions. Comparing with Figura 5.6, it can be seen that PSO with TH obtains large improvements in 10 from 15 functions, reducing the fitness above 15 orders of magnitude in F11 and F13. When comparing the number of nodes in the tree, we can conclude that additional nodes tend to reduce TH's performance (although slightly), because the overall budget of 5×10^5 evaluations is shared between all TH instances. The budget available for every TH instance, per configuration, is 5×10^5 for PSO and 1SP, 2.5×10^5 for 2LP, and 1.66×10^5 for 3LP. The function F15, however, presents better results for pyramidal configuration, although it uses the same number of THs as the linear configuration 3LP, suggesting that different problems might benefit from different topologies.

One of the reasons for PSO with TH demand only $p = 6$ individuals to obtain much better results, in same overall computing budget than PSO (which uses $p = 30$), is that smaller populations tend to stagnate more quickly, forcing a larger number of restarts and, consequently, raising collaboration.

Considering that CEC13 functions are complex high dimensional problems and the standard PSO is a known weak algorithm for such category of problems, we can conclude that TH is capable of improving the optimization power of simple search algorithms on the studied benchmarks.

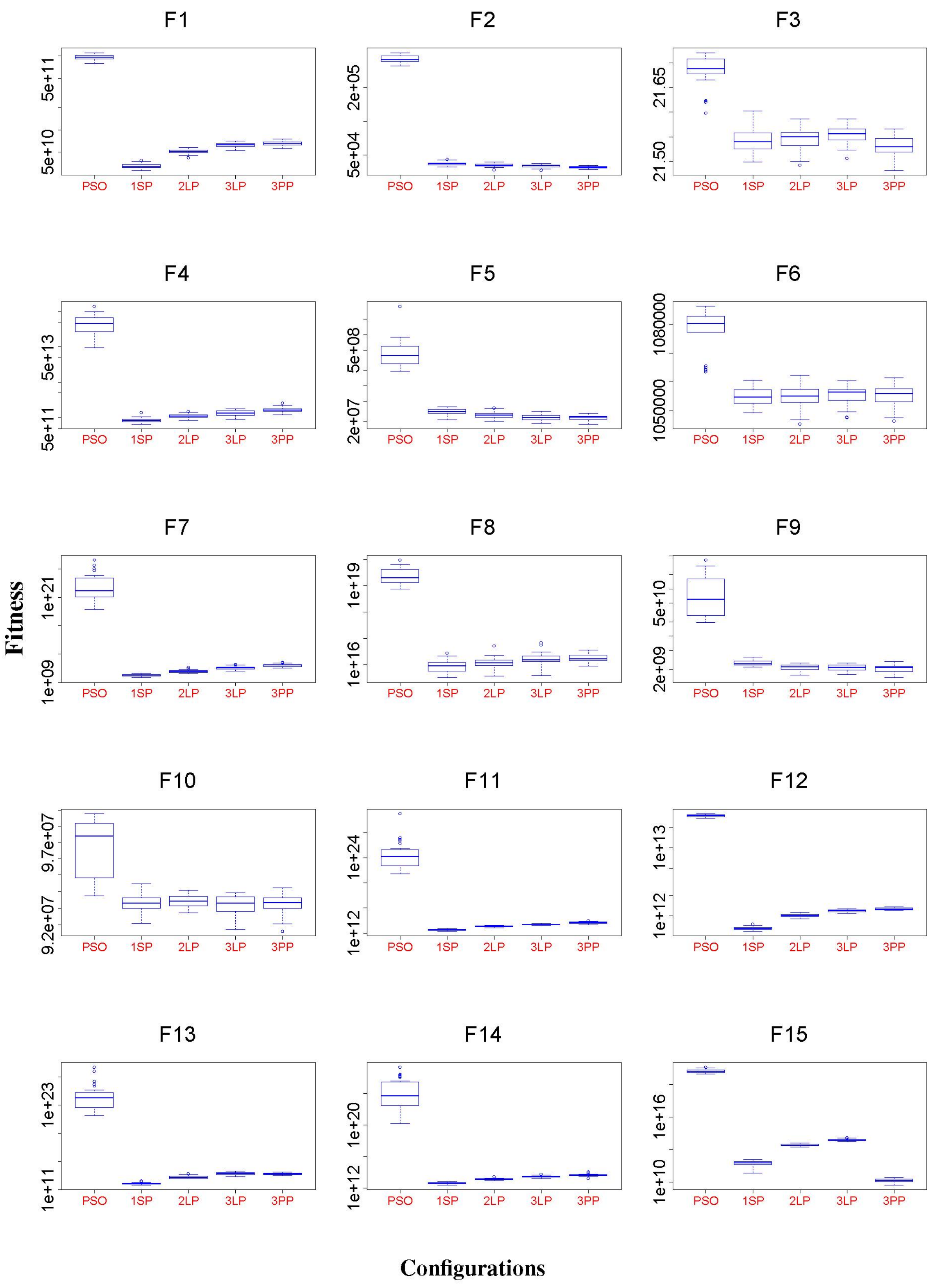


Figura 5.5: PSO with and without TH.
Standalone PSO = $[p : 30]$, PSO with TH = $[p : 6]$, p_δ using IP, $MTotal = 5 \times 10^5$, 30 independent executions. Topologies: 1SP = 1 single TH instance, 2LP = 2 THs (linear topology), 3LP = 3 THs (linear topology), 3PP = 3 THs (pyramidal topology).

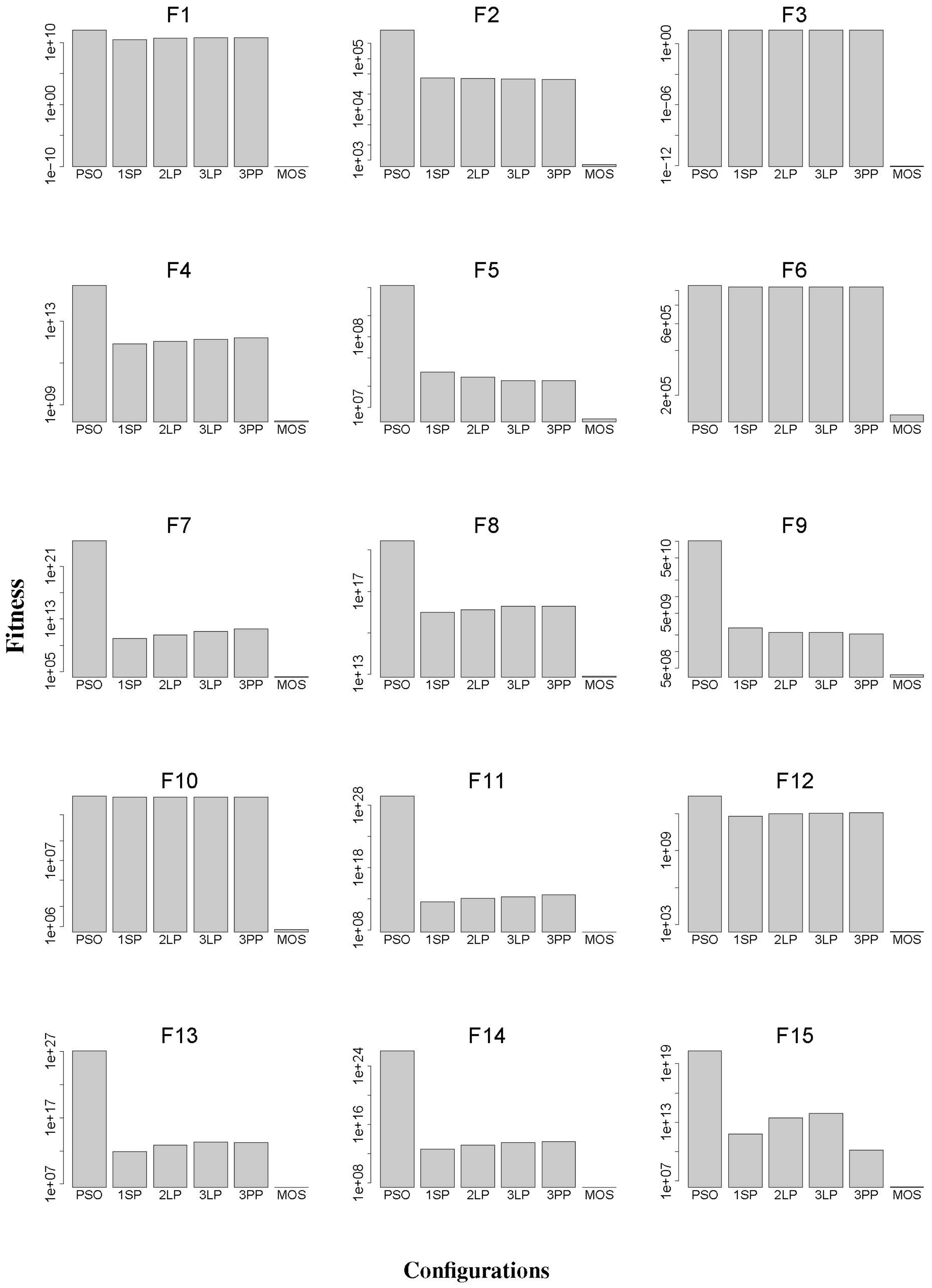


Figura 5.6: PSO with and without TH compared with CEC 2013/2015 Competition Winner. Competition winner: MOS (CEC (2015)). Configuration: see Figura 5.5

6 EMPIRICAL STUDIES

Aiming to prove the effectiveness of Treasure Hunt framework when optimizing various problem sizes with distinct characteristics, two testing sets are considered, one to evidence the long term convergence obtained by TH's controlled multi-start capabilities, and one to demonstrate the power of TH's distributed optimization in an HPC environment:

- Controlled Multi-start: TH instance provides long term convergence through a controlled multi-start process that minimizes the waste of evaluations in poor regions, relocating individuals gradually toward promising regions.
- Distributed Optimization: Treasure Hunt is designed to allow distribution of a large number of parallel TH instances, such that a large sampling of the search space is obtained, with better results than the standalone TH and in shorter *wall clock* time.

The configuration below is used for both testing sets:

- Search Algorithms: CCPSO2 with IP = $[B_r : 0.05, maxTries : 3]$, standard PSO= $[w : 0.7, c1 : 0.2, c2 : 0.2]$, DE/*best/1/bin* = $[F : 0.05$ and $CR : 0.01]$.
- TH: best-list size = $max(L,nChildren)$, p_δ using IP=$[Boost_L, B_r : 1, maxTries : 5]$, LS = [Hill Climbing controlled by CSMOn, with $R : 0.01$].

Similarly to Section 5, parameters are empirically adjusted (and presented in each experiment) only to match benchmark's characteristics, since test sets differences are relevant. PSO, DE and CCPSO2 are used because they are good representatives for swarm, evolutionary and CC search algorithms, respectively. Except for the benchmark COCO (described in next section), all results are obtained based on statistics collected from 30 independent executions of every configuration. CCPSO2 uses the IP method in all experiments to control the number of swarms in each search group. All tests are performed over minimization benchmarks.

6.1 CONTROLLED MULTI-START

To validate the proposed multi-start method under different scenarios, three benchmarks of increasing dimensionality (with medium and high complexities) are used to test the search algorithms with and without a single-instance TH, then results are compared to show the influence of TH's multi-start on search algorithms' behavior:

1. COCO: Comparing Continuous Optimizer (Hansen et al. (2016)), an automated framework with 24 low dimensionality functions of high complexity.
2. MSG Generator: an automated and parameterized test-problem (landscape) generator called Max-Set of Gaussians (MSG) (Gallagher e Yuan (2006); Bartz-Beielstein (2015)), generating nonlinear, nonseparable random test functions of low and medium dimensionality for minimization, based on the sum of Gaussians.
3. Classic benchmark: 2 multimodal and separable (Ackleys and Rastrigin) and 1 unimodal, non-convex and non-separable (Rosenbrock) high-dimensional functions.

   - Ackleys: $f(\mathbf{x}) = -20.exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + exp(1)$

- Rosenbrock: $f(\mathbf{x}) = \sum_{i=1}^{n}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$
- Rastrigin: $f(\mathbf{x}) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$

The following configuration is used for the three benchmarks ($nEvals$ is the number of evaluations already performed during the optimization):

- TH Topology: $N = 1, K = 1, L = 1$ (1 TH).
- TH: $\beta_{max} = 100$, $p = 30$, CSMOn = $[M : 10^5, R : 1 - nEvals/MTotal]$, LS = $[M : 10 \times n]$.
- CCPSO2 with IP: $Boosts$.

and results are presented for each benchmark.

6.1.1 COCO

The specific configuration for this benchmark is: $n = \{2, 3, 5, 10, 20, 40\}$, $MTotal = 50000 \times n$, $p_w = 2$, with statistics and number of executions statically enforced by COCO framework. Figura 6.1, created by COCO framework, shows the summarized results of 24 hard benchmark functions with 51 targets, with curves representing the combined proportion of functions and targets solved by the optimization runs (i.e., larger area under the curve is better).

Despite the fact that DE algorithm obtains the fourth position on larger dimensionalities (in a rank of 6 algorithms, i.e., DE, PSO and CCPSO2, with and without TH), and the worst position in the lower dimensionalities (2D and 3D), Figura 6.1 shows that DE with TH clearly reaches superior performances in all cases (2D to 40D). PSO obtained poor results in all 6 dimensionalities, and similarly to DE algorithm, the performance raises in all cases when PSO is used with TH.

However, the algorithm CCPSO2 does not benefit from TH in any case. The reason is that CCPSO2's internal mechanisms already provide good performance, obtaining the second best results on smaller dimensionalities and the third best results on 20D and 40D. Therefore, TH's population reposition mechanisms delay CCPSO2's convergence speed during most of the search, consequently affecting CCPSO2's final result.

Results without TH shown in Figura 6.1 are used as baselines to generate Figura 6.2, and the improvements obtained with TH are plotted as curves. TH's long term convergence can be seen for PSO (Figura 6.2a) and DE (Figura 6.2b), especially on larger dimensionalities, where the gains with TH reached 40% for PSO and 50% for DE. Although TH suffers from a performance decay around half of the search, it quickly recovers, converging to much better results. This is due to TH's mechanisms to assist the search method, that could cause the slow down of the convergence for a short period, while looking for promising regions to reposition the population. A second decay on PSO with TH and DE with TH occurs at the end of the search on problems with $n = \{2, 3, 5\}$, caused by improvements obtained by the standalone versions (the baseline) on smaller dimensionalities.

While PSO and DE obtain increasing improvement rates with TH, CCPSO2's results deteriorate when using TH (up to -37% - Figura 6.2c), most likely because the CCPSO2's sophisticated algorithm, developed to deal with complex search regions, does not benefit from TH's assistance on low dimensionality.

Considering the different improvement rates obtained by TH on each search algorithm, we can conclude that, for low dimensionality, TH gains are directly related to the optimization algorithm used. Except for CCPSO2, this test demonstrates that the long term convergence

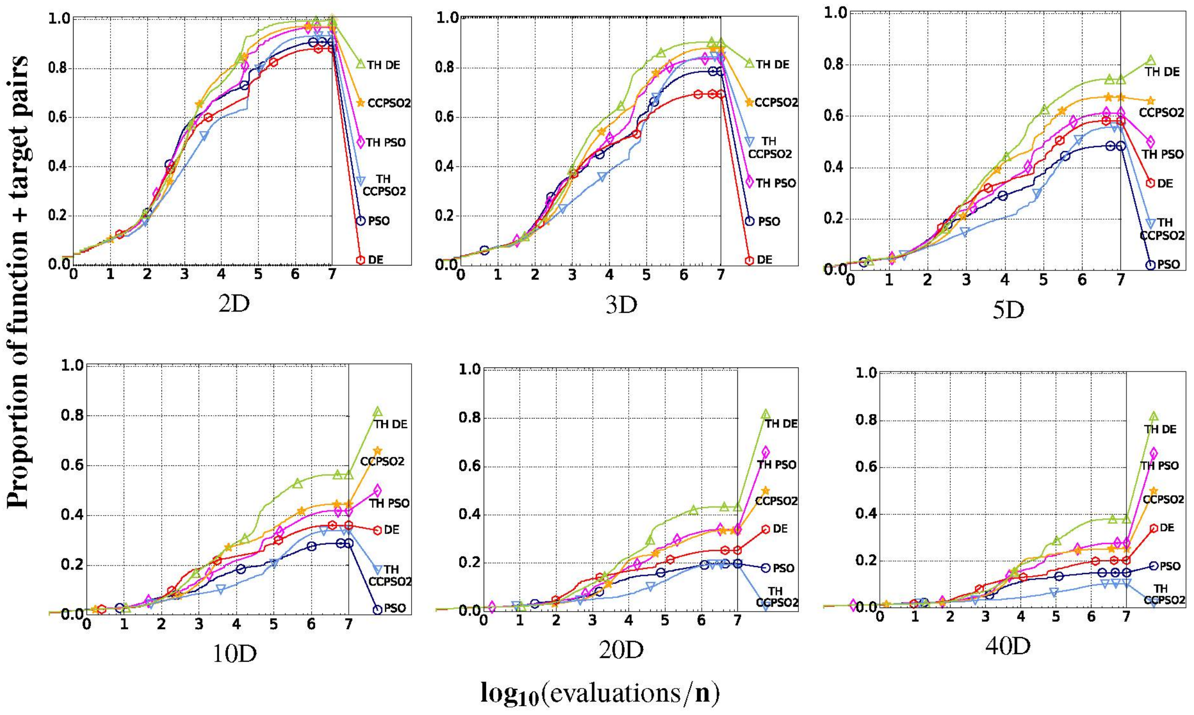


Figura 6.1: COCO's Empirical Cumulative Distribution Function (ECDF) per Dimension.
Results summarized for 24 noiseless functions, for DE, PSO and CCPSO2 with and without TH, using 51 relative targets uniformly distributed in a log scale between 10^2 and 10^{-8}. Number of evaluations is given in $10^x \times n$. The small dots represent the overall fraction of all successfully solved functions-target pairs. The curves represent the combined proportion of functions and targets solved by the optimization runs (larger AUC is better).

provided by TH achieves much superior results than the standalone search algorithm, and better improvements occur for problems with higher dimensionalities.

6.1.2 MSG Generator

Two complexity levels are configured:

- Medium difficulty: $n = 50$, $MTotal = 5000 \times n$, $S = [-5,5]$, random local minima at fitness = 50.
- High difficulty: $n = 100$, $MTotal = 40000 \times n$, $S = [-10,10]$, random local minima at fitness = 90.
- For both difficulty levels: $p_w = 1$, $\mathcal{U}(n, 2n)$ gaussians for every random problem, worst fitness=100.

Figura 6.3 summarizes the results with a mixed model based on confidence intervals (Chiarandini e Goegebeur (2010)), creating a simple pairwise comparison plot between versions with and without single-instance TH, showing how significant is the difference between results.

According to random problems' results in Figura 6.3a (medium difficulty) and Figura 6.3b (high difficulty), results are significantly better when using TH (except for TH with DE on medium difficulty). This good performance obtained by standalone DE on medium difficulty random problems, however, does not persist when increasing difficulty and dimensionality. On high difficulty configurations, all tests with TH escape from the stagnation caused by the random

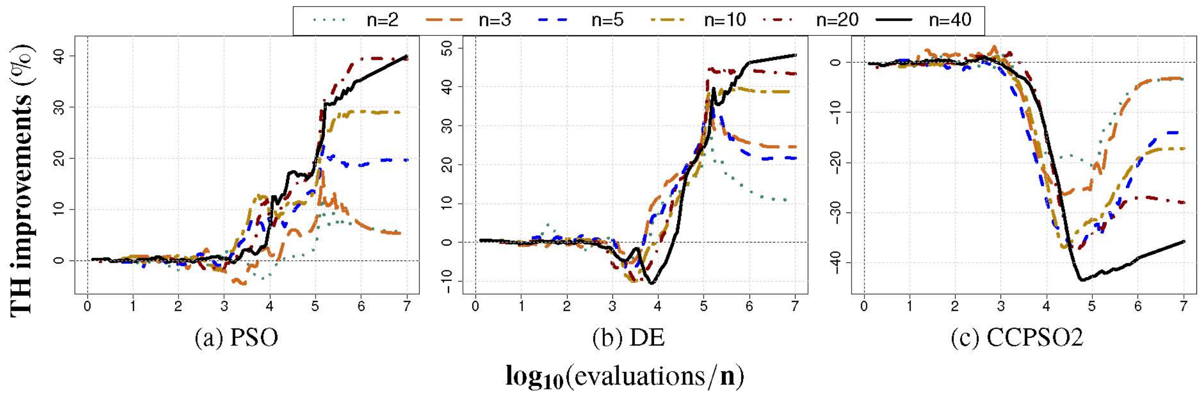


Figura 6.2: Differences between search algorithms with and without TH on complex, low dimensionality COCO problems.
(a) PSO (b) DE/*best/1/bin* (c) CCPSO2. Results without TH are the baselines and the improvements obtained with TH are the plotted curves. Actual number of evaluations is given by $10^x \times n$. Statistics and number of executions are enforced by COCO. Configuration: [1 TH instance, $n = \{2, 3, 5, 10, 20, 40\}$, $MTotal = 50,000 \times n$, $p_w = 2$, $\beta_{max} = 100$ (centric shape), $p = 30$].

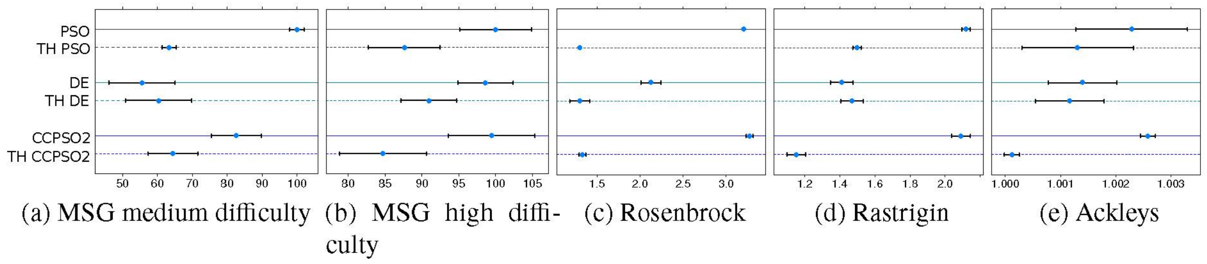


Figura 6.3: Paired comparison of search algorithms with and without TH.
(a) Random test (MSG) with medium difficulty ($n = 50$), local minima at fitness=50, $MTotal = 5,000 \times n$ and search space boundaries $S = [-5, 5]$. (b) Random test (MSG) with high difficulty ($n = 100$), local minima at fitness=90, $MTotal = 40,000 \times n$ and $S = [-10, 10]$. (c) Rosenbrock. (d) Rastrigin. (e) Ackleys. Average and standard deviation for 30 runs. Crossed bars between algorithms means the results are not significantly different. Every result in (c), (d) and (e) are obtained by optimizing the respective functions with $n = 2000$ and $n = 10000$ dimensions, then normalizing the results with equation $1 + \log(1 + fit/n)$, such that results close to 1 are also close to the global optimum. Configuration for (a) and (b): [$p_w = 1$, $\mathcal{U}(n, 2n)$ gaussians for every random problem, worst fitness set as 100]. Configuration for (c), (d) and (e): [$p_w = 2$, $MTotal = 5,000 \times n$]. General configuration: [1 TH instance, $\beta_{max} = 100$ (centric shape), $p = 30$].

local minima (set to fitness = 90), whereas the standalone search methods stagnate quickly at the very beginning of the search.

Comparing solely TH approaches, their performances in both Figura 6.3a and Figura 6.3b are not significantly different between search algorithms, and for that reason, we can conclude that TH also reduces the importance of using specific search methods for each MSG problem at hand.

6.1.3 Classic functions

The specific configuration is: $n = \{2000, 10000\}$, $MTotal = 5000 \times n$ and $p_w = 2$.

In this benchmark, the algorithm that most benefits from TH is CCPSO2, with best results on multimodal Rastrigin (Figura 6.3d) and multimodal Ackleys functions (Figura 6.3e, approaching the global optimum in all executions). On Ackleys function, although TH only slightly improves PSO's results, PSO becomes comparable to DE. For DE algorithm, TH is effective on Rosenbrock unimodal function (Figura 6.3c) and it does not significantly affect results for the other functions, very likely because only one TH instance (hence 1 population) is used, what reduces the effectiveness of TH on multimodal problems.

In general, TH improves results on these high dimensionality functions. On non-convex Rosenbrock function, the very similar performance obtained by all search algorithms means that TH can reduce the importance of using specific search algorithms also on high dimensionality classic problems.

6.2 DISTRIBUTED OPTIMIZATION

Based on the empirical validation of the proposed multi-start method (presented in the previous section), two testing sets are performed to validate TH's capability of maintaining a convergence on a parallel distributed HPC environment, as the number of TH instances increases:

1. Convergence behavior on scaling: compares the effect of an increasing number of TH instances over the quality of the convergence, to demonstrate its capability of benefiting from a highly distributed environment.

2. Scaling with fixed *wall clock* time: given a limited time in an HPC environment, this test compares the improvements as the number of TH instances is increased.

The following configuration is used in each TH instance:

- Benchmark: Ackleys, Rastrigin, Rosenbrock.
- TH: $\beta_{max} = 1, p_w = 2^{L_{root}-L}, MTotal = 5000 \times n$, CSMOn = $[M : 10^3 \times (L_{root} - L + 1)^2, R : 1 - nEvals/MTotal]$, LS = $[M : p]$.
- CCPSO2 with IP: $Boost_E$.

All tests are performed over minimization problems, using the selection criterion by group (4.7), the same number of segments (**K**) for all groups and equal segment sizes.

The budget $MTotal$ is not shared between TH instances, therefore, every instance performs $MTotal$ function evaluations. Empirical tests have demonstrated that the fraction of the budget provided by CSMOn to the search algorithms (M) can be small on TH, because main improvements occur mostly at first stages of the search. Hence, for the root TH, $M = 1 \times 10^3$ to increase the number of relocations.

The experimental results are described in next sections.

6.2.1 Convergence on scaling

To obtain similar convergence behavior as previous section, population size is set to $p = 30$. Problem size is set to $n = 2000$. TH topologies used are: [$N = 1$, $K = 1$, $L = 1$] (1 TH instance), [$N = 2$, $\mathbf{K} = 2$, $L = 2$] (5 THs), [$N = 2$, $\mathbf{K} = 2$, $L = 3$] (21 THs), [$N = 1$ and $K = 63$, $L = 2$] (64 THs).

Most of configurations in Figura 6.4 show increasing improvements with 5 and 21 THs, demonstrating that good results can be further improved (especially at the beginning of the optimization) by using an increasing number of TH instances. When using 64 THs, expressive improvements are obtained along all the optimization processes.

Although a single-instance TH does not significantly improve DE on Rastrigin (Figura 6.3d) and Ackleys functions (Figura 6.3e), when adding more TH instances (Figura 6.4e and Figura 6.4f) the performance is improved. On Rosenbrock function, the single-instance TH performs better than raising to 5 and 21 TH instances on DE (Figura 6.4d), and obtains similar final result on PSO (Figura 6.4a) and CCPSO2 (Figura 6.4g). The reason is related to the fact that the results presented in Section 6.1.3 show large improvements for all three search algorithms with 1 single TH instance on Rosenbrock function (Figura 6.3c). Therefore, much more detailed sampling of the search space would be required to obtain significant additional improvements, and consequently, a larger number of TH instances is required (as seen with 64 TH instances in Figura 6.4).

Internally, CCPSO2 consumes all M evaluations provided by CSMOn at every TH iteration, reducing the number of restarts and communications performed. As a consequence, CCPSO2 stagnates quickly when compared with PSO and DE. However, given that each restart consumes processing resources due to relocation strategy, processing times for PSO and DE are larger than for CCPSO2. Nevertheless, CCPSO2 with 1 TH reaches good results much more quickly than the other algorithms, showing that the number of restarts required for each search algorithm might be different. Although not desirable, CCPSO2's quick stagnation can be mitigated by adjusting M according to the optimization problem.

Considering that the only difference between configurations is the number of THs, we can conclude that the superior results obtained, when increasing the number of TH instances, are due to the larger sampling of the search space, which is provided by TH's multilevel topology.

6.2.2 Scaling with Increased Problem Size

To investigate TH's capability of scaling with larger problem sizes, the search algorithms PSO, DE and CCPSO2 are used to optimize the functions Rosenbrock, Rastrigin and Ackleys with the dimensionality increased to $n = 10000$. Instead of demonstrating the usual convergence behavior of the algorithms, this experiment shows the average number of fitness function evaluations required to obtain a significant improvement in the solution (reductions of at least one order of magnitude in the fitness).

TH topologies used are: [$N = 1$, $K = 1$, $L = 1$] (1 TH instance), [$N = 1$, $\mathbf{K} = 2$, $L = 2$] (3 THs), [$N = 1$, $\mathbf{K} = 2$, $L = 3$] (7 THs), [$N = 2$, $K = 4$, $L = 2$] (17 THs), [$N = 2$, $K = 2$, $L = 3$] (21 THs) and [$N = 3$, $K = 3$, $L = 2$] (28 THs). Population size is set to $p = 30$

In general, Figura 6.5 shows that most of the multi-instance configurations overcome the standalone configuration (with 1 TH instance), either by finding better solutions or by obtaining similar solutions more quickly (except in Figura 6.5c, where PSO with 1 TH instance obtains an unexpected superior performance). The configuration that obtained the best overall performance is with 21 THs (instead of the larger configuration, with 28 THs), what can be explained by their topologies and the respective information flow (the configuration with 21 THs uses 4 children per

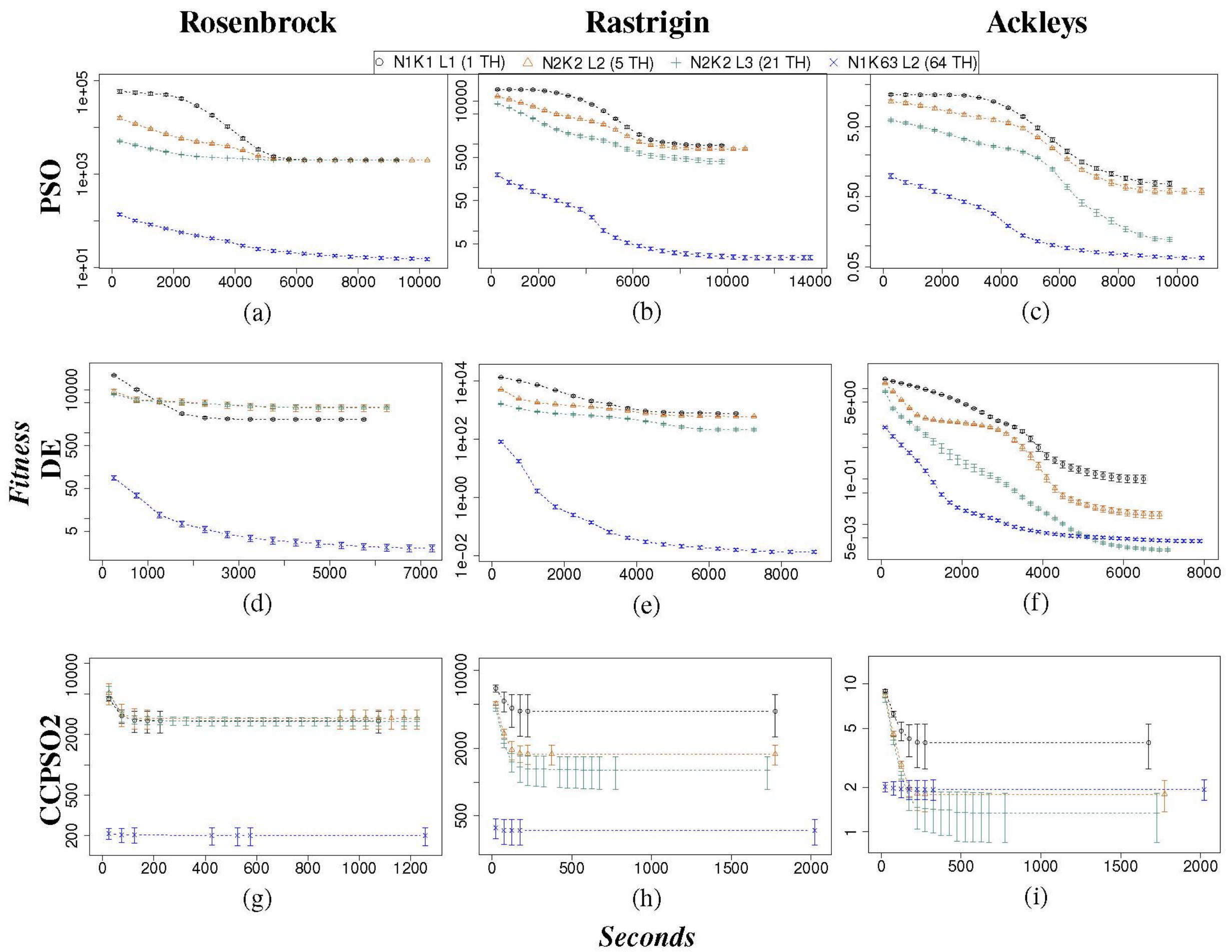


Figura 6.4: TH convergence on HPC environment.
Average and standard deviation for 30 runs. Configuration: [n = 2000, p = 30, β_{max} = 1 (sparse shape), $p_w = 2^{L_{root}-L}$, p = 30, $MTotal = 5000 \times n$].

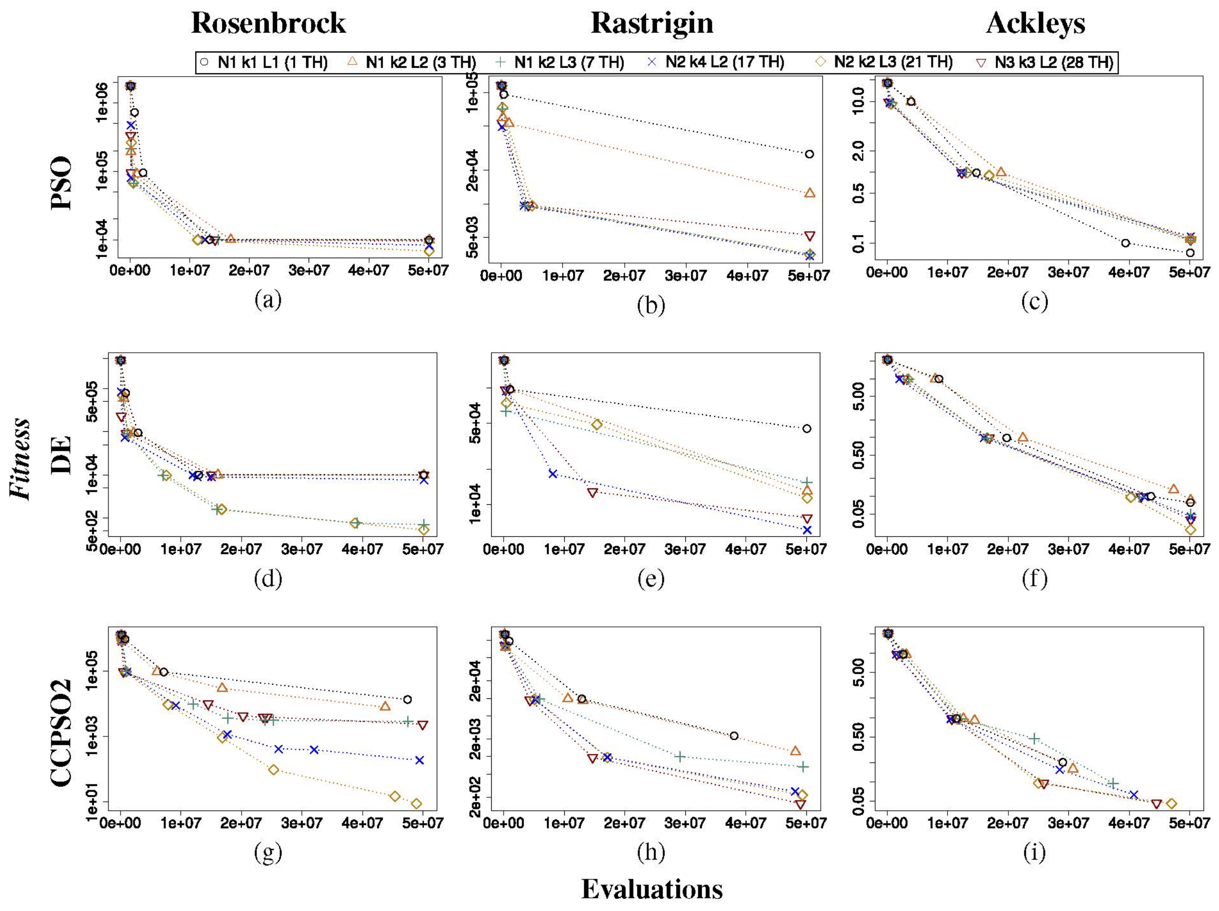


Figura 6.5: Average number of evaluations required to obtain a significant improvement (reductions of at least one order of magnitude).
Average for 30 runs. Configuration: [n = 10000, p = 30, β_{max} = 1 (sparse shape), $p_w = 2^{L_{root}-L}$, p = 30, $MTotal = 5000 \times n$].

parent and 2 parent levels, whereas the configuration with 28 THs uses 27 children per parent and only 1 parent level).

Despite the fact that the configuration with 17 THs uses only two groups (N = 2), its performance follows the larger configuration with 28 THs (with N = 3 groups), probably due to the reduced number of children to communicate with the Root TH.

The quick stagnation of CCPSO2 seen in Figura 6.4 is not seen in Figura 6.5, very likely due to the much increased difficulty of the functions, and the CCPSO2's good capability of dealing with complex functions. The only case that stagnates in all configurations is PSO on Rosenbrock function (Figura 6.5a), obtaining the worst result for Rosenbrock between all search algorithms. Nonetheless, PSO's configuration with 21 THs reaches the stable location more quickly than the other configurations.

6.2.3 Scaling with fixed *wall clock* time

It is well known that a larger sampling of the search space increases the chance of finding good regions (Dreo e Siarry (2007)), and final results are closely related to the initial sampling (Alba et al. (2013); Jr et al. (2015); Crainic e Toulouse (2010)). For this reason, instead of optimizing the functions for long periods, this experiment optimizes the functions Rosenbrock, Rastrigin and Ackleys for 100 seconds only (Alba et al. (2013)). This is an attempt to evidence the commitment between the number of TH instances and the resulting initial sampling, showing this way the potential for future long term optimizations.

TH topologies used: [$N = 1$, $K = 1$, $L = 1$] (1 TH instance), [$N = 1$, $K = 31$, $L = 2$] (32 THs), [$N = 1$, $K = 63$, $L = 2$] (64 THs) and [$N = 1$, $K = 95$, $L = 2$] (96 THs). To verify the importance of large population sizes when increasing the number of TH instances, $p = 30$ is compared with $p = 6$ (the minimum allowed by DE's requirements). Problem size is set to $n = 2000$.

Figura 6.6 shows that the influence of the number of individuals (in each search group) on results is progressively reduced as the number of TH instances increases, due to the larger coverage provided when distributing populations throughout the search space. Considering that smaller populations tend to stagnate more quickly and synchronize more often than larger populations, they also perform less fitness function evaluations per time frame. Given that results for both population sizes are similar, it can be said that part of these evaluations can be replaced by more frequent cooperation between TH instances, favouring problems with time-consuming fitness functions.

Figura 6.6 also shows that the difference between search algorithms' results are not significantly relevant as the number of TH instances increases (although CCPSO2 presents a slight advantage with 96 TH instances), demonstrating that TH's guided sampling of the search space can be more beneficial than focusing on using complex search algorithms. The only configuration that does not hold such characteristics is CCPSO2 with 32 TH instances, given that an unexpected outperformance (yet unstable) is obtained.

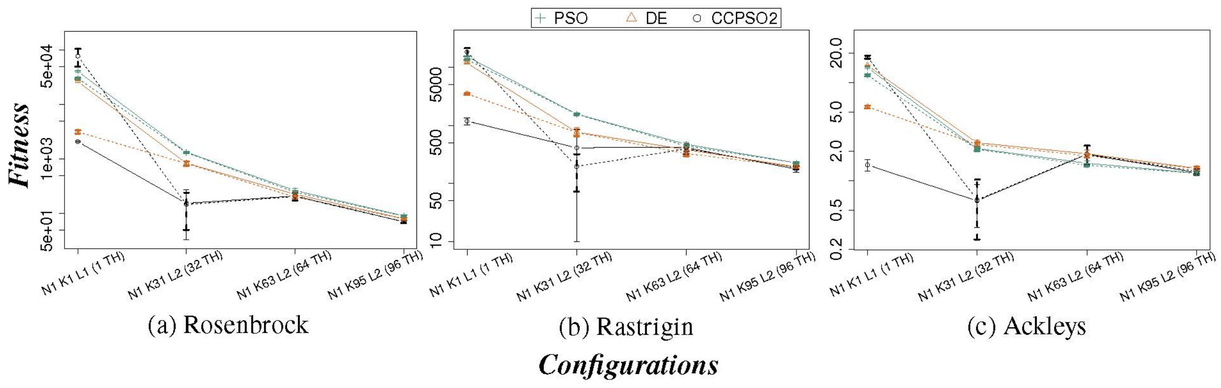
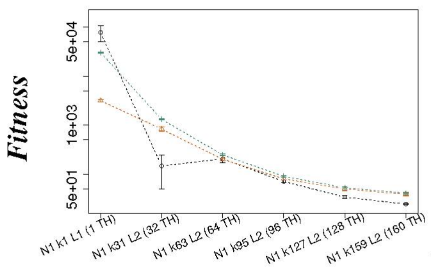


Figura 6.6: TH Scaling on HPC environment.
Average and standard deviation for 30 runs, processing during 100 seconds and up to 96 THs. Configuration: [$n = 2000$, $p = 6$ (dashed lines) and $p = 30$ (solid lines), $\beta_{max} = 1$ (sparse shape), $p_w = 2^{L_{root}-L}$, $p = 30$, $MTotal = 5000 \times n$].

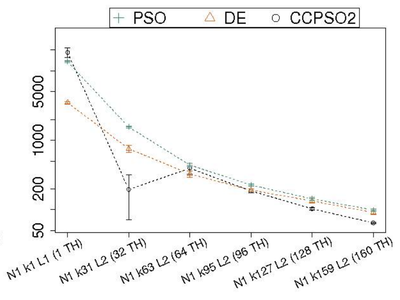
To investigate TH's capability of continuous convergence with a reduced population size ($p = 6$), the experiment presented in Figura 6.6 was expanded up to 160 TH instances: [$N = 1$, $K = 127$, $L = 2$] (128 THs), [$N = 1$ and $K = 159$, $L = 2$] (160 THs).

Figura 6.7 shows that, besides consistently improving the optimization, the additional TH instances also reduce the importance of using complex metaheuristics, since the differences between the search algorithms' results are negligible when using 64 TH instances or more (although CCPSO2 still presents a slight advantage).

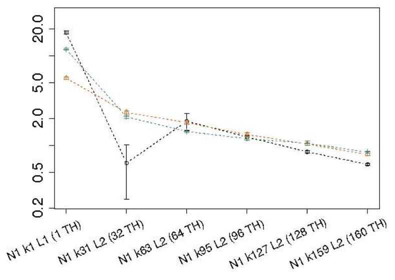
A key conclusion that can be drawn from Figuras 6.6 and 6.7 is that TH framework is scalable, allowing the distribution of serial search algorithms to a high number of processing nodes to optimize separable, non-separable, unimodal and multimodal high dimensionality problems, obtaining superior results in same *wall clock* time than the standalone versions of the search algorithms.



(a) Rosenbrock



(b) Rastrigin



(c) Ackleys

Figura 6.7: TH Scaling on HPC environment.
Average and standard deviation for 30 runs, processing during 100 seconds and up to 160 THs. Configuration: [n = 2000, p = 6, β_{max} = 1 (sparse shape), $p_w = 2^{L_{root}-L}$, p = 30, $MTotal = 5000 \times n$].

7 CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

This thesis proposed a distributed framework called Treasure Hunt (TH), aiming to optimize high dimensional, complex, single-objective, continuous numerical functions. TH is capable of distributing independent serial optimization algorithms to a high number of processing nodes, imposing low communication penalties, using a tree topology, a search space organization method and a driving mechanism, in such a way that a large sampling of the search space is obtained.

TH's local asynchronous communication schema is capable of providing quick propagation of good information to all processing nodes, even when using clusters with millions of nodes, ensuring simultaneous explorations (by parents) and exploitations (by children), on several levels of granularity, with only small overhead to the nodes in the tree. Given a suitable computational budget, TH ensures a joint convergence, at global level, between all distributed populations.

Considering that only the root node is crucial for the successful completion of the optimization, and all parents replicate the best solutions from their children, TH has a good fault tolerance. However, full fault tolerance is possible on TH by backing up small data structures between nodes, like for example the pool of best solutions.

A selection criterion by group has been proposed to chose sub-regions of the search space located at the diagonal of the grouped dimensions, such that virtual "hyper-diagonals" are chosen automatically for Cartesian products, thus forming distributed topologies with a k-ary tree shape.

An Iterative Partitioning (IP) method has been proposed to partition arbitrary intervals iteratively, producing an "*elastic*" effect on the partitions obtained. IP has been used on TH to control the displacement rate, promoting diversity on the initial stages of the search and intensification on the later stages. It was used also to improve CCPSO2's performance (CCPSO2-IP) on non-separable problems, and its results were published on BRACIS 2015 (Perroni et al. (2015)).

A Convergence Stabilization Modeling to operate in Online mode has also been proposed to control TH's multi-start mechanism, aiming to detect the convergence stagnation and to stop the search, hence avoiding waste of processing budget. A relaxation mechanism to adjust CSMOn behavior provides controlled compliance with stagnation points, allowing fine tuned results from quick major improvements to long term continuous convergences. CSMOn method was published on GECCO 2017 (Perroni et al. (2017)).

Simple experiments have been presented to demonstrate TH's internal mechanisms. To show TH's capability of dealing with unimodal and multimodal landscapes that present non-separable, partially separable, fully separable and non-convex characteristics, several testing sets have been conducted with PSO, DE and CCPSO2 search algorithms. These sets include small, medium and large complexities and dimensionalities on classic, random and competition benchmarks

Results show that TH framework consistently boosts search algorithm's results, making poor performer algorithms to become comparable to good ones. Good performer algorithms, in turn, can benefit from TH when the search space is increased up to a size that is too large for the algorithm.

As experimentally demonstrated, TH's characteristics can be leveraged by distributing the search to cooperative TH instances, with test results presenting stable, long term convergences. Experiments with increasing numbers of TH instances present a robust scaling, showing that the proposed search space organization method is capable of obtaining a good sampling of the search space, dealing efficiently with large dimensionality and small population sizes. A robust scaling is also verified when fixing the same *wall clock* time for all tested search algorithms (tested up to 160 TH instances), with negligible differences between poor and good performers after 64 THs.

As a final conclusion, it can be said that the large sampling and the cooperation mechanism between multiple evolving populations reduce the importance of large population sizes and complex search algorithms, which are critical concerns on real-world problems with time-consuming fitness functions.

7.2 FUTURE WORK

This work is a proof of concept to introduce the TH framework, and as such, we consider that TH's capabilities were only partially explored and many open questions still remain. Future works include, for example, hyperheuristic algorithms for the search group, sub-region's selection criteria that consider hardware environment's and problem's characteristics, relocation strategies using different distributions, different methods to set the displacement rate, aggregate functions to combine multiple objectives into a single-objective function, methods to create new knowledge from the information exchanged, robust fault tolerance mechanisms and experiments on real-world complex high dimensional problems, like for example the configuration of deep neural networks.

7.2.1 Displacement Rate

The gradual repositioning method employed by TH behaves in a way that resembles the Nested Partition method, since the automatic repositioning of group's individuals settles the populations only at the most prominent locations within not too sparse regions of the search space. However, if a better location is found elsewhere, instead of doing a backtracking (like Nested Partition), TH uses the beta distribution to smoothly displace the entire population toward the promising location, favoring the region in between.

Considering that the beta distribution influences the search algorithms used by the search group G, the displacement rate p_δ is subject to refinements through any convenient technique, to adjust its convergence behavior as required. For example, instead of setting p_δ with (4.6), it could be optimized by search algorithms like Hill Climbing or Simulated Annealing. It is worth to mention that once this fine tuning is focused on finding the best distance from parent's best result, given that such best position is not static over TH iterations, finding a good value for p_δ can be regarded as a uni-dimensional dynamic optimization problem.

7.2.2 Hyperheuristics

TH instances are independent from each other, and the search groups impose almost no restriction on the search algorithms to be used. The reason why the search group G have no thorough definition on TH framework is to allow its implementation by any convenient method, like for example by using a hyperheuristic algorithm. When using such hyperheuristic component in more than one TH instance, the resulting framework could be classified as a distribution of hyperheuristics. Notice that to obtain a distributed hyperheuristic instead, the solutions exchanged between TH instances must include hyperheuristic-specific information, like for example the operators (or search methods) used and the scoring process for the algorithm selection. However, regardless what hyperheuristic component type is implemented, it has to be considered that any implementation-specific detail of G must be hindered from the TH instance.

REFERÊNCIAS

(2017). Congress on evolutionary computation. `http://www.cec2015.org/`. Accessed in: 14/Mar/2017.

Alba, E. e Dorronsoro, B. (2005). The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142.

Alba, E., Luque, G. e Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48.

Ali, M. e Storey, C. (1994). Modified controlled random search algorithms. *International Journal of Computer Mathematics*, 53(3-4):229–235.

Bartz-Beielstein, T. (2015). How to create generalizable results. Em *Springer Handbook of Computational Intelligence*, páginas 1127–1142. Springer.

Bartz-Beielstein, T. e Zaefferer, M. (2017). Model-based methods for continuous and discrete global optimization. *Applied Soft Computing*, 55:154–167.

Biazzini, M., Bánhelyi, B., Montresor, A. e Jelasity, M. (2009). Distributed hyper-heuristics for real parameter optimization. Em *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, páginas 1339–1346. ACM.

Bouvry, P., Arbab, F. e Seredynski, F. (2000). Distributed evolutionary optimization, in manifold: Rosenbrock's function case study. *Information Sciences*, 122(2):141–159.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. e Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

Calafiore, G. e Dabbene, F. (2006). *Probabilistic and randomized methods for design under uncertainty, Chapter 3*, volume 3. Springer.

Caraffini, F., Neri, F., Gongora, M. e Passow, B. (2013a). Re-sampling search: A seriously simple memetic approach with a high performance. Em *Memetic Computing (MC), 2013 IEEE Workshop on*.

Caraffini, F., Neri, F., Passow, B. N. e Iacca, G. (2013b). Re-sampled inheritance search: high performance despite the simplicity. *Soft Computing*, 17(12).

CEC, I. (2015). Competition on: Large scale global optimization. `http://titan.csit.rmit.edu.au/~e46507/lsgo-competition-cec15/lsgo-competition-summary-2015.pdf`. Accessed in: 03/Jun/2019.

Cerny, V. (1984). Minimization of continuous functions by simulated annealing. Relatório técnico.

Chiarandini, M. e Goegebeur, Y. (2010). Mixed models for the analysis of optimization algorithms. Em *Experimental Methods for the Analysis of Optimization Algorithms*, páginas 225–264. Springer.

Crainic, T. G. e Toulouse, M. (2010). Parallel strategies for meta-heuristics. Em *Handbook of metaheuristics chapter 17*, volume 146. Springer.

de Souza, P. S. e Talukdar, S. N. (1993). Asynchronous organizations for multi-algorithm problems. Em *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, páginas 286–293. ACM.

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

Dreo, J. e Siarry, P. (2007). *Stochastic Metaheuristics as sampling techniques using swarm intelligence*. INTECH Open Access Publisher.

Du, X., Ding, L. e Jia, L. (2008). Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm. Em *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, volume 1, páginas 433–437. IEEE.

Dubreuil, M., Gagné, C. e Parizeau, M. (2006). Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(1):229–235.

Eberhart, R. C., Kennedy, J. et al. (1995). A new optimizer using particle swarm theory. Em *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, páginas 39–43. New York, NY.

Erol, O. K. e Eksin, I. (2006). A new optimization method: big bang–big crunch. *Advances in Engineering Software*, 37(2):106–111.

Gallagher, M. e Yuan, B. (2006). A general-purpose tunable landscape generator. *IEEE transactions on evolutionary computation*, 10(5):590–603.

Goh, C. K., Tan, K. C., Liu, D. e Chiam, S. C. (2010). A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design. *EJOR*, 202(1).

Gong, Y.-J., Chen, W.-N., Zhan, Z.-H., Zhang, J., Li, Y., Zhang, Q. e Li, J.-J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300.

Gutjahr, W. J. (2009). Convergence analysis of metaheuristics. Em *Matheuristics*, páginas 159–187. Springer.

Hansen, N., Auger, A., Mersmann, O., Tušar, T. e Brockhoff, D. (2016). COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, arXiv:1603.08785.

Hansen, N. e Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. *Proceedings of IEEE International Conference on Evolutionary Computation*, (June 1996):312–317.

Hauschild, M. e Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128.

Henry Obit, J. (2010). *Developing novel meta-heuristic, hyper-heuristic and cooperative search for course timetabling problems*. Tese de doutorado, University of Nottingham.

Herrera, F. e Lozano, M. (2000). Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63.

Herrera, F., Lozano, M. e Moraga, C. (1999). Hierarchical distributed genetic algorithms. *International journal of intelligent systems*, 14(11):1099–1121.

Jr, I. F., Yang, X.-S., Brest, J., Fister, D. e Fister, I. (2015). Analysis of randomisation methods in swarm intelligence. *International journal of bio-inspired computation*, 7(1):36–49.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Relatório técnico, Technical report-tr06, Erciyes university, engineering faculty, computer . . . .

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

Krishnanand, K. N. e Ghose, D. (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. Em *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, páginas 84–91.

Le Bouthillier, A. e Crainic, T. G. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32(7):1685–1708.

Li, X., Tang, K., Omidvar, M., Yang, Z. e Qin, K. (2013). Benchmark functions for the cec'2013 special session and competition on large scale global optimization. Evolutionary Computation and Machine.

Li, X. e Yao, X. (2009). Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. Em *Evolutionary Computation, 2009. CEC '09. IEEE Congress on.*

Li, X. e Yao, X. (2012). Cooperatively coevolving particle swarms for large scale optimization. *Evolutionary Computation, IEEE.*

Lozano, J. M. C., Giménez, D. e García, L. P. (2016). Optimizing metaheuristics and hyperheuristics through multi-level parallelism on a many-core system. Em *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, páginas 786–795. IEEE.

Luke, S. (2013). *Essentials of metaheuristics*. Lulu, second edition. Available for free at `http://cs.gmu.edu/~sean/book/metaheuristics/`. Accessed in: Mar/04/2019.

Luke, S. (2015). *Essentials of Metaheuristics*. Lulu, 2.2 edition. Available at http://cs.gmu.edu/∼sean/book/metaheuristics/.

Mahdavi, S., Shiri, M. E. e Rahnamayan, S. (2015). Metaheuristics in large-scale global continues optimization: A survey. *Information Sciences*, 295:407–428.

Meignan, D., Koukam, A. e Créput, J.-C. (2010). Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879.

Mirjalili, S., Mirjalili, S. M. e Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61.

Nesmachnow, S. (2014). An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320–347.

Olafsson, S. (2006). Metaheuristics. *Handbooks in operations research and management science*, 13:633–654.

Omidvar, M. N. e Li, X. (2010). A comparative study of cma-es on large scale global optimisation. Em *Australasian Joint Conference on Artificial Intelligence*, páginas 303–312. Springer.

Omidvar, M. N., Li, X., Mei, Y. e Yao, X. (2014). Cooperative co-evolution with differential grouping for large scale optimization. *Evolutionary Computation, IEEE Transactions on*, 18(3).

Ouelhadj, D. e Petrovic, S. (2008). A cooperative distributed hyper-heuristic framework for scheduling. Em *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, páginas 2560–2565. IEEE.

Ouelhadj, D., Petrovic, S. e Ozcan, E. (2009). A multi-level search framework for asynchronous cooperation of multiple hyper-heuristics. Em *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, páginas 2193–2196. ACM.

Özcan, E., Misir, M., Ochoa, G. e Burke, E. K. (2010). A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. *Int. J. Appl. Metaheuristic Comput.*, 1(1):39–59.

Perroni, P. F., Weingaertner, D. e Delgado, M. R. (2015). Automated iterative partitioning for cooperatively coevolving particle swarms in large scale optimization. Em *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, páginas 19–24. IEEE.

Perroni, P. F., Weingaertner, D. e Delgado, M. R. (2017). Estimating stop conditions of swarm based stochastic metaheuristic algorithms. Em *The Genetic and Evolutionary Computation Conference, GECCO*. ACM.

Pichitlamken, J. e Nelson, B. L. (2003). A combined procedure for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 13(2):155–179.

Poikolainen, I., Iacca, G., Caraffini, F. e Neri, F. (2013). Focusing the search: a progressively shrinking memetic computing framework. *International Journal of Innovative Computing and Applications*, 5(3).

Poorjandaghi, S. S. e Afshar, A. (2014). A robust evolutionary algorithm for large scale optimization. Em *World Congress*, volume 19.

Potter, M. A. e De Jong, K. A. (1994). *A cooperative coevolutionary approach to function optimization*, páginas 249–257. Springer Berlin Heidelberg.

Rada-Vilela, J., Zhang, M. e Johnston, M. (2013). Optimal computing budget allocation in particle swarm optimization. Em *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, páginas 81–88. ACM.

Rattadilok, P., Gaw, A. e Kwan, R. S. (2004). Distributed choice function hyper-heuristics for timetabling and scheduling. Em *International Conference on the Practice and Theory of Automated Timetabling*, páginas 51–67. Springer.

Rodrigues, R. d. F. (1996). *Times Assíncronos Para a Resolução de Problemas de Otimização Combinatória com Múltiplas Funções Objetivo*. Tese de doutorado, Tese de Mestrado. Campinas-São Paulo, UNICAMP.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5(1):96–101.

Sefrioui, M. e Périaux, J. (2000). A hierarchical genetic algorithm using multiple models for optimization. Em *Parallel Problem Solving from Nature PPSN VI*, páginas 879–888. Springer.

Shi, L. e Ólafsson, S. (2000). Nested partitions method for global optimization. *Operations research*, 48(3):390–407.

Solis, F. J. e Wets, R. J.-B. (1981). Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30.

Souravlias, D. e Parsopoulos, K. E. (2016). Particle swarm optimization with neighborhood-based budget allocation. *International Journal of Machine Learning and Cybernetics*, 7(3):451–477.

Storn, R. e Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359.

Subbu, R. e Sanderson, A. C. (2004). Modeling and convergence analysis of distributed coevolutionary algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):806–822.

Talukdar, S., Baerentzen, L., Gove, A. e De Souza, P. (1998). Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321.

Talukdar, S. e Ramesh, V. (1993). A multiagent technique for contingency constrained optimal power flows. Em *Power Industry Computer Application Conference, 1993. Conference Proceedings*, páginas 188–194. IEEE.

Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, 85(6):317–325.

Tu, W. e Mayne, R. (2002). Studies of multi-start clustering for global optimization. *International journal for numerical methods in engineering*, 53(9):2239–2252.

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

Van Den Bergh, F. e Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. *Information sciences*, 176(8):937–971.

Vu, C. C., Nguyen, H. H. e Bui, L. T. (2011). A parallel cooperative coevolution evolutionary algorithm. Em *Knowledge and Systems Engineering (KSE), 2011 Third International Conference on*, páginas 48–53. IEEE.

Weise, T. (2009). *Global Optimization Algorithms - Theory and Application*. Self-Published, second edition.

WOLPERT, D. H. e MACREADY, W. G. (2005). Coevolutionary free lunches. *IEEE transactions on evolutionary computation*, 9(6):721–735.

Yang, X.-S. (2011). Metaheuristic optimization: algorithm analysis and open problems. Em *International Symposium on Experimental Algorithms*, páginas 21–32. Springer.

Yang, Z., Tang, K. e Yao, X. (2008). Multilevel cooperative coevolution for large scale optimization. Em *Evolutionary Computation, 2008. CEC 2008. IEEE World Congress on Computational Intelligence*.

Zhang, S., Xu, J., Lee, L. H., Chew, E. P., Wong, W. P. e Chen, C.-H. (2016). Optimal computing budget allocation for particle swarm optimization in stochastic optimization. *IEEE Transactions on Evolutionary Computation*.

APÊNDICE A – THE METAPHOR

The framework presented in this thesis is based on a previous simplified version of this proposal (with one level of parallelism and a bi-dimensional virtual search space, detailed on next section) that used the classical “Treasure Hunt” story as metaphor. Hence, for historical reasons, the expanded framework proposed in this thesis kept its original name.

A.1 CHASING THE TREASURE

Looking for an important hidden treasure, a pirate fleet, managed by its Admiral pirate (Figura A.1, step 1), berths in a large unknown island which is suspected to have a complex landscape with dangerous regions. In an attempt to combine efforts and speed up the search for the treasure, the admiral assigns each Captain pirate (and respective crew members / pirates) to an exclusive portion of the island, which must be scanned as detailed as possible using all means that each captain has on hands.

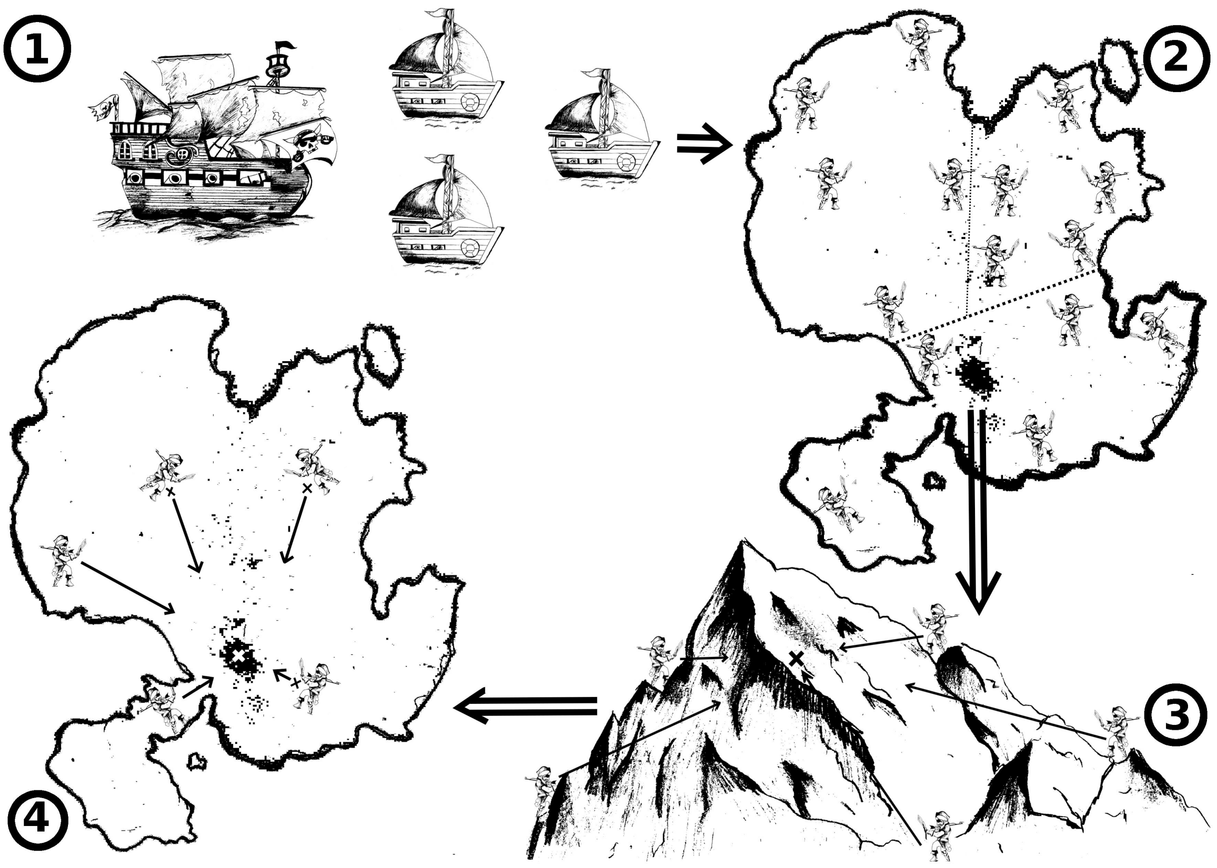


Figura A.1: Sketch of the Treasure Hunt story.
(1) Admiral’s ship and his Pirate fleet. (2) Captains’ crews are assigned to delimited areas in the island. (3) Each crew scans its region and obtains a best location. (4) Admiral’s crew scans the entire island based on Captains crews’ results.

Since initially the captains have no knowledge about their assigned search regions, they define arbitrary positions within their areas from where every pirate in their group will begin the search (Figura A.1, step 2). Every captain has the freedom to decide how his search group will move in the island, including entering inside neighbour regions. When his pirates “agree” with a common location as a good candidate for the place where the treasure might be hidden (Figura A.1, step 3), the position is then reported back to the captain. It is important to mention that pirates can always move through dangerous locations, but the treasure is never hidden there since these locations are unfeasible for the treasure.

The captains report their best found locations to the admiral, which will compare them with his personal best-list and make necessary replacements in the list. The admiral maintains a short list of the most probable sites where the treasure might be hidden in the island (the best-list), trying to keep a high diversity of positions in his list to cover as much of the territory as possible.

The positions reported by the captains are then assigned by the admiral to each pirate of his personal crew, which in turn will make a new independent search themselves (Figura A.1, step 4). After admiral’s search group has completed the search, the resulting location will replace one position from admiral’s best-list (if necessary). Then, one random location from the best-list is sent back to all captains, which in turn will use it as reference to start over the search in random positions in their sub-regions, but this time biased toward admiral’s recommendation.

A.2 EXPANDING TO AN N-DIMENSIONAL ISLAND

By correlating the metaphor presented on previous section with an optimization scenario, the island can be thought of as the full search space, the crews as the search algorithms, the pirates as the individuals of populations, every captain as an independent Treasure Hunt instance (responsible for each individual hunt), and the admiral as the root Treasure Hunt instance.

A.3 CCPSO2-IP: AUTOMATED ITERATIVE PARTITIONING FOR CCPSO2

Automated Iterative Partitioning for Cooperatively Coevolving Particle Swarms in Large Scale Optimization

Peter Frank Perroni
Universidade Federal do Paraná
Email: pfperroni@inf.ufpr.br

Daniel Weingaertner
Universidade Federal do Paraná
Email: danielw@inf.ufpr.br

Myriam Regattieri Delgado
Universidade Tecnólogica Federal do Paraná
Email: myriamdelg@utfpr.edu.br

Abstract

Particle Swarm Optimization (PSO) is a relatively recent meta-heuristic inspired by the swarming or collaborative behaviour of biological populations. It is known by its capacity of obtaining important fitness improvements on a short period of time. A cooperative version named CPSO has been used to deal with high dimensional search spaces and CCPSO2 is one of its variants that has achieved high performances in large scale optimization problems (above 500 dimensions). This paper proposes an Iterative Partitioning (IP) method for CCPSO2 that takes advantage of the CCPSO2 characteristics. The resulting approach, named CCPSO2-IP, also joins some well known good practices into one single algorithm. Boost functions are included to fine tune the search steps. The competition benchmark CEC13 for large scale global optimization (LSGO) is used to validate the proposed method. Results show that the IP-based method outperforms the standard CCPSO2 and the single swarm PSO, where the exponential boost function presents the highest performance.

I. Introduction

Metaheuristics are important tools when dealing with difficult optimization problems. However, in many cases, as more complex real-world issues are addressed more dimensions have to be considered, what in turn reduces the optimization effectiveness obtained by classical metaheuristic tools.

Particle Swarm Optimization (PSO), introduced in 1995 by Kennedy and Eberhart, brought a new perspective for the search algorithms and allowed complex problems to be solved more quickly. Nevertheless, as the number of dimensions increases, PSO also suffers from the *Curse of Dimensionality*[1]. Cooperative Particle Swarm Optimization (CPSO) was created to mitigate this problem. It divides the search space into multiple swarms, allowing for a more exploratory search and reducing the probability of bad local minima.

However, even CPSO may not be robust enough to tackle highly complex real-world problems, with hundreds or thousands of dimensions. For this reason, a Cooperatively Coevolving (CC) PSO version (CCPSO2) has been developed in an attempt to gather automatically promising groups of dimensions into swarms so that better resulting fitness can be attained. Good results were obtained in [1], where it showed competitive performance when compared with state-of-the-art algorithms for high dimensional space, like sep-CMA-ES [2] and CC Differential Evolution (CC DE) [3].

This work proposes an automatic Iterative Partitioning (IP) for CCPSO2, which relies on the optimization behaviour itself and on a boost function that dictates the balance between exploratory and intensification phases. In CCPSO2-IP, exploration and intensification are performed iteratively throughout the entire search, so that the probability of being trapped in a poor local minimum is reduced.

The rest of this paper is organized as follows. Section II gives an overview of the original PSO version as well as some recent variations developed for large scale optimization problems. Section III details the proposed approach. Experiments and results are discussed in Section IV, while Section V concludes the paper.

II. Particle Swarm Optimization Methods

This section describes PSO-based algorithms, specially those suitable for large scale optimization problems. Thus, in addition to the original version, two recent cooperative versions are detailed: Cooperative Particle Swarm Optimization (CPSO) and Cooperatively Coevolving PSO (CCPSO2).

[1]The performance deteriorates as the dimensionality of the search space increases.

A. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population based algorithm, inspired by the dynamic of a bird flock, which has the characteristic of quickly converging to a local minimum [4]. It consists of one global best position $\hat{\mathbf{y}}$ ($gbest$) and a population P of p particles, where every particle has n dimensions and each dimension is composed of the attributes: the current position in the search space $x_{i,j}$ ($i = 1..p$, $j = 1..n$), the current velocity $v_{i,j}$ and the personal best position achieved so far in the search space $y_{i,j}$ ($pbest_{i,j}$) [5].

$$\begin{aligned} v_{i,j}(t+1) = wv_{i,j}(t) \quad &+ c_1 r_{1,i}(t)[y_{i,j}(t) - x_{i,j}(t)] \\ &+ c_2 r_{2,i}(t)[\hat{y}_j(t) - x_{i,j}(t)] \end{aligned} \tag{1}$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{2}$$

The parameters c_1 and c_2 are the acceleration coefficients that control the particle direction, the parameter w is the amount of inertia preserved from the previous iteration, and r_1 and r_2 are elements from two uniform random sequences in the interval $[0..1]$. At every iteration t, all dimensions j of the particles must be updated by (1) and (2). Equations (3) and (4) show the $pbest$ ($\mathbf{y}_i$) and $gbest$ ($\hat{\mathbf{y}}$) calculations (respectively) for a minimization function f.

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t), & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)), \\ \mathbf{x}_i(t+1), & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)). \end{cases} \tag{3}$$

$$\hat{\mathbf{y}}(t+1) = \underset{\mathbf{y}_i}{arg\ min}\ f(\mathbf{y}_i(t+1)), \quad 1 \leq i \leq p \tag{4}$$

To limit particles movement and avoid their escape from the search space, the velocity limit $[-v_{max}, v_{max}]$ and the search area limit $[x_{min}, x_{max}]$ are defined.

Linear Decreasing Weight (LDW) is the strategy used to control particles inertia [6] (5), where T_{max} is the maximum number of iterations, w_{ini} is the initial inertia weight and w_{end} is the final inertia weight (i.e., when T_{max} is achieved).

$$w = (w_{ini} - w_{end}) \left(\frac{T_{max} - t}{T_{max}} \right) + w_{end} \tag{5}$$

For problems with high dimensional search space, the Cooperative version of PSO (CPSO) can be used [5], where the dimensions are divided into subsets and each subset is given to a swarm. Then, the swarms execute in parallel with the same configuration, cooperating each other with their own best solutions. The final output of CPSO is the union of swarms' global best positions, resulting in one single solution containing all original dimensions.

B. CCPSO2

CCPSO2 [1] (Cooperatively Coevolving Particle Swarm) is a recent state-of-the-art CPSO variant that has been successfully used to solve complex optimization problems and can be considered as a reference for newly developed algorithms for high scale problems [7] [8]. It is based on a previous version called CCPSO [9] [10]. CCPSO2 was developed as an attempt to tackle issues encountered during optimization processes where the number of dimensions is much higher than the ones found in trivial problems. It presents good performance despite its low computational cost when compared to some other optimization techniques for large scale problems [11].

Algorithm 1 describes the CCPSO2 framework. It uses the same idea of the $\mathbf{b}$ function (line 1) existent in CPSO. $P_k.\hat{\mathbf{y}}$ represents the $gbest$ for each swarm k, K is the current total number of swarms and $\mathbf{z}$ represents the vector $\mathbf{x}$ for the current particle i within the swarm k under evaluation. The function $\mathbf{b}$ joins n/K dimensions managed by the swarms, forming a complete candidate solution for evaluation by the fitness function f. Therefore, $\mathbf{b}$ is used to aggregate solutions generated by each particle from each swarm. CCPSO2 has two fundamental differences from CPSO, one in the particle's position calculations and other in the swarms management.

Instead of using the global best to update the particle's positions, CCPSO2 uses a $lbest$ ring topology to obtain the particle's local best neighbour (function $localBest$, line 18 in Algorithm 1), which is stored in $P_k.\hat{\mathbf{y}}'_i$ ($k \in [1..K], i \in [1..p]$) and is used to calculate the particles' next position (6). Equation (6) uses random numbers from Cauchy ($\mathcal{C}$) and Gaussian ($\mathcal{N}$) distributions, besides a random number $rand$ uniformly generated from $[0, 1]$. The ring topology is an attempt to reduce the convergence speed of the optimization, that is a major issue for PSO when dealing with high dimensional search spaces. The r value is a parameter usually set to

0.5.

$$x_{i,j}(t+1) = \begin{cases} y_{i,j}(t) + \mathcal{C}(1)|y_{i,j}(t) - \hat{y}'_{i,j}(t)|, & if\ rand \leq r \\ \hat{y}'_{i,j}(t) + \mathcal{N}(0,1)|y_{i,j}(t) - \hat{y}'_{i,j}(t)| & otherwise. \end{cases} \tag{6}$$

CCPSO2 dynamically changes the number of swarms K and permutates the dimensions to improve their grouping. The dimensions are evenly divided between the swarms so that each swarm contains s dimensions ($s \in \mathbf{S}$).

Algorithm 1 Pseudocode of CCPSO2 [1]

```
1: $\mathbf{b}(k,\mathbf{z}) = (P_1.\hat{\mathbf{y}}, \cdots, P_{k-1}.\hat{\mathbf{y}}, \mathbf{z}, P_{k+1}.\hat{\mathbf{y}}, \cdots P_K.\hat{\mathbf{y}})$
2: Create and initialize $K$ swarms with $s$ dimensions (where $s$ is randomly chosen from a set $\mathbf{S}$, and $n = K \times s$
   always being true); the $k$th swarm is denoted as $P_k, k \in [1 \cdots K]$
3: **repeat**
4:     **if** $f(\hat{\mathbf{y}})$ has not improved **then** randomly choose $s$ from $\mathbf{S}$ and let $K = n/s$
5:     **end if**
6:     Randomly permutate all $n$ dimension indices
7:     Construct $K$ swarms, each with $s$ dimensions
8:     **for** each swarm $k \in [1 \cdots K]$ **do**
9:         **for** each particle $i \in [1 \cdots p]$ **do**
10:            **if** $f(\mathbf{b}(k, P_k.\mathbf{x}_i)) < f(\mathbf{b}(k, P_k.\mathbf{y}_i))$ **then**
11:                $P_k.\mathbf{y}_i \leftarrow P_k.\mathbf{x}_i$
12:            **end if**
13:            **if** $f(\mathbf{b}(k, P_k.\mathbf{y}_i)) < f(\mathbf{b}(k, P_k.\hat{\mathbf{y}}))$ **then**
14:                $P_k.\hat{\mathbf{y}} \leftarrow P_k.\mathbf{y}_i$
15:            **end if**
16:        **end for**
17:        **for** each particle $i \in [1 \cdots p]$ **do**
18:            $P_k.\hat{\mathbf{y}}'_i \leftarrow localBest(P_k.\mathbf{y}_{i-1}, P_k.\mathbf{y}_i, P_k.\mathbf{y}_{i+1})$
19:        **end for**
20:        **if** $f(\mathbf{b}(k, P_k.\hat{\mathbf{y}})) < f(\hat{\mathbf{y}})$ **then** the $k$th part of $\hat{\mathbf{y}}$ is replaced by $P_k.\hat{\mathbf{y}}$
21:        **end if**
22:    **end for**
23:    **for** each swarm $k \in [1 \cdots K]$ **do**
24:        **for** each particle $i \in [1 \cdots p]$ **do**
25:            Update $P_k.\mathbf{x}_i$ using (6)
26:        **end for**
27:    **end for**
28: **until** termination criterion is met
```

Initially, the algorithm randomly chooses from $\mathbf{S}$ the number of dimensions s of the swarms and then creates K swarms (restricted to $n = K \times s$). Then, it permutates the particle's dimensions and executes one iteration. Every iteration is composed of one typical CPSO particle best update procedure, followed by the verification of the local best particle (ring topology). The kth swarm's best is also updated if necessary. Once the best particles for all swarms have been updated, all particle's positions are updated through (6). If no improvement is obtained at the end of the iteration, the algorithm makes a new random selection from $\mathbf{S}$ for a new s value and then repeats the previous steps. The process is repeated until the termination criterion is met.

The PSO algorithm is well known for its convergence capability balanced with its ability to explore the search space. However its performance usually deteriorates as the number of dimensions increases. The Cooperative PSO variant helps to relieve this problem by splitting dimensions into several collaborative swarms, each one exploring its own piece of the search space, all swarms working together to improve the results. Nevertheless, the cooperative version may also get trapped into poor local minima when the number of dimensions is too high. CCPSO2 tries to mitigate this problem by both permutating the dimensions and dynamically changing the number of swarms. However, as discussed in [12], CCPSO2 does not perform on non-separable problems as good as on separable problems. On [13] a Differential Grouping was proposed to find interdependencies among variables, with good improvements for large scale problems. As recommended in [1], a good choice for the swarm sizes in $\mathbf{S}$ would have a significant impact on CCPSO2 performance. Hence, a mechanism to dynamically control dimensions and swarms creation might consider more potential variable interactions [3], therefore increasing the chance of obtaining higher performances.

III. CCPSO2-IP: THE PROPOSED APPROACH

Aiming to improve the CCPSO2 performance, this work proposes a method which iteratively splits the dimensions of the problem into constantly reduced number of swarms (thus, constantly increasing swarm sizes). The reduction process named Iterative Partitioning (IP) is based on parameters that define (i) the number of reasonable tries for a unsuccessful configuration, (ii) the minimum acceptable improvement rate to be obtained at every iteration, and (iii) a boost function.

Algorithm 2 describes the CCPSO2-IP method, which is basically Algorithm 1 with lines 2 to 7 replaced by the iterative procedure. The parameter $minImprovement$ defines the minimum improvement obtained at every iteration that is still advantageous for the search keep using the same swarm configuration (usually between 0% to 5%). If the improvement is lower than this value, it is considered as no improvement (for higher tolerance, set it towards zero). Algorithm 2 executes the CCPSO2 search on line 32.

Initially, the maximum number of swarms $maxK(0)$ is calculated by a boost function (line 2), which is used to set the current number of swarms K. The parameter $maxTries$ defines the number of iterations to be tried under no acceptable improvement. Hence, if no acceptable improvement is obtained after $maxTries$ iterations, the number of swarms K is reduced by a fixed rate K_r (11). If the search continues but no acceptable improvement is obtained after additional $maxTries$ tries, K is reduced once more (and so on) until its minimum possible value is reached. At this point, the number of swarms K increases to a new starting point $maxK(t)$ (line 13).

Iterative Partitioning is based on the idea of boost functions, where the optimization step size can be indirectly controlled by a function and a speed-up value (Fig.1). We propose the boost functions Linear (8), Sigmoid (9) and Exponential (10), however additional functions can be used to address specific types of optimization problems. Equation (7) is used to calculate $maxK$ and makes reference (through $Boost(t)$) to the boost functions (8), (9) or (10).

$$maxK(t) = MIN(MAX(n * Boost(t), 1), n) \tag{7}$$

$$Boost_L(t) = -B_r * \left(\frac{t}{T_{max}}\right) + B_r \tag{8}$$

$$Boost_S(t) = \frac{B_r}{1.0 + \exp(12 * B_r * \left(\frac{t}{T_{max}}\right) - 6 * B_r)} \tag{9}$$

$$Boost_E(t) = \frac{B_r}{exp(12 * B_r * \left(\frac{t}{T_{max}}\right)} \tag{10}$$

$$K = MIN(MAX(K - MAX(maxK * K_r, 1), 1), maxK) \tag{11}$$

$$n = \sum_{k=1}^{K} s_k \tag{12}$$

The parameter B_r is a boost rate value from $[0, 1]$, where 0 means no boost and 1 means maximum boost. Notice that the higher the boost rate, the larger will be $maxK(0)$ and more aggressive will be the iterative reduction in the number of swarms. This happens because K is reduced based on K_r (11), which is dependent on $maxTries$, that is fixed at the beginning of the algorithm. Therefore, a high boost rate will begin the search with a high number of swarms (K), then when the reduction happens on K (line 17), it will be a large decrease on the number of swarms. However, if a small boost rate is specified, a small number of swarms will be created at beginning and only small decreases will occur. Together with the auto decrease of $maxK$ (7), which is controlled by a boost function that dictates its reduction, CCPSO2-IP method ensures a dynamic

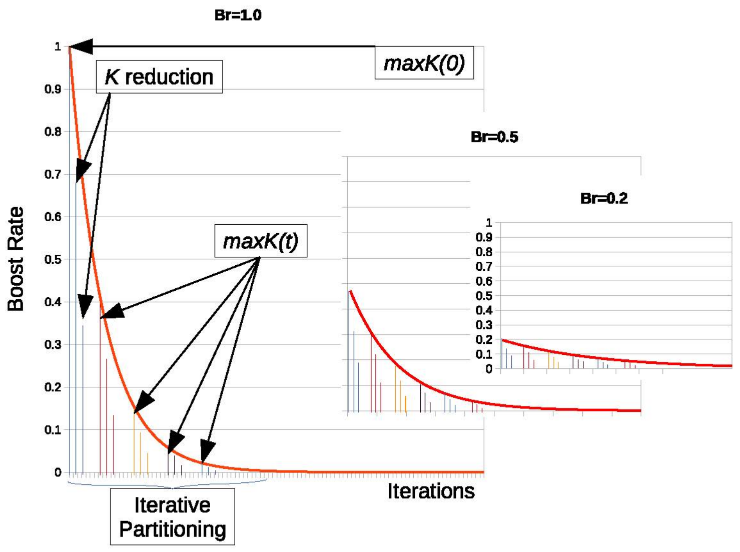


Fig. 1: Iterative Partitioning Method.
For Exponential boost function with $maxTries = 3$. The parameter $maxK(t)$ will determine the initial value for K at every round.

iterative partitioning that will try several different combinations of variables, thus increasing the probability of obtaining better results.

For complex optimization problems, CCPSO2-IP mechanism is not always enough to reduce the chance of the search to be trapped in local minima. For this reason, Algorithm 2 also has a fail-safe mechanism that tries to force the exploration of other regions, setting $maxK = maxK(0)$ (line 11) if after $maxTries$ reductions on $maxK$ no improvement can be obtained. This might force a new exploratory search during one round in an attempt to increase the chance of escaping from a local minimum. Two permutation points, one fixed (line 22) and other probabilistic (line 26) are also added to ensure the known benefits of the permutations [1] will be maintained.

CCPSO2-IP joins the advantages of using dynamically changing swarm sizes, permutations and the well known benefit of exploring the search space at initial stages (through the use of more partitions) and intensifying the search at later stages (through reduced number of partitions, what is ensured by the boost functions and their parameterized boost rates). For the proposed boost function, the Exponential focuses on the exploration only at very early stages, while the Sigmoid makes exploration at the first half of the search and then focus on the intensification at the other half, and the Linear constantly decreases the iterative partitioning throughout the search.

Notice the swarms can be resized to uneven number of dimensions (lines 22 and 27), i.e., randomly chosen based on restriction (12) (where s_k is the number of dimensions for swarm k), allowing for a better fitting of the swarms for problem specific dependencies. Also, every time the swarms are created (line 4) or recreated (line 20), their dimensions are not permutated (permutations only occur on lines 22 and 26), so that the original sequence of dimensions is tried at least once. To reduce the overhead caused by permutations, one additional mapping can be employed between the aggregation function b and the actual particle's positions, so that the change on the number of swarms, the swarms sizes and the dimension's grouping can be done with only minor performance impact.

IV. Experimental Results

In order to compare the proposed CCPSO2-IP method with the previously discussed particle swarm optimization algorithms, the same experiments were conducted for all algorithms (PSO, CCPSO2, CCPSO2-IP L, CCPSO2-IP S and CCPSO2-IP E, the last three algorithms represent the different boost functions being considered) then their results were compared.

A. *Configurations*

The objective of this paper is to compare PSO, CCPSO2 and the three proposed boost functions. The competition benchmark CEC13 (Congress on Evolutionary Computation 2013, also used by CEC15) for Large Scale Global Optimization (LSGO) [14] was adopted to measure the optimization power of the algorithms.

Algorithm 2 CCPSO2-IP

```
1: $K_r = 1/maxTries$
2: Calculate $maxK(0)$ with (7)
3: $K = maxK$
4: Create $K$ swarms
5: $fitImprovement = 1$                                   ▷ bypass first iteration
6: for $t$ in $[1 \cdots T_{max}]$ do
7:     if $fitImprovement < minImprovement$ then
8:         if $maxTries$ iterations without improvement then
9:             if $K \leq MAX(maxK * K_r, 1))$ then
10:                if $maxTries$ updates on maxK without improvement then   ▷ force exploration
11:                    $maxK = maxK(0)$
12:                else                                   ▷ Iteratively reduce maxK
13:                    Calculate $maxK(t)$ with (7)
14:                end if
15:                $K = maxK$
16:            else                                       ▷ Iteratively reduce K
17:                Calculate $K$ reduction with (11)
18:            end if
19:            if new $K$ is different from previous $K$ then
20:                Recreate swarms with new $K$
21:            else
22:                Permutate dimensions and resize swarms
23:            end if
24:            Recalculate PBest's and KBest's fitness values
25:        else                                 ▷ give it a 50% chance of permutation
26:            if $rand < 0.5$ then
27:                Permutate dimension and resize swarms
28:                Recalculate PBest's and KBest's fitness
29:            end if
30:        end if
31:    end if
32:    Execute CCPSO2 (Algorithm 1), lines 8 to 27
33:    Calculate fitImprovement
34: end for
```

In the future we intend to compare CCPSO2-IP with CEC13/CEC15 LSGO winners, although this demands specific tuning for each problem, whilst we are now interested on a more generic solution. CEC13 benchmark is composed of 15 specially difficult functions of 1000 dimensions that represent different real world large-scale optimization problems. It provides convenience and flexibility for comparing various evolutionary algorithms specifically designed for large-scale global optimization. CEC13 benchmark functions and their respective bounds comprise:

- Fully-separable Functions: F1 (Shifted Elliptic Function $[-100, 100]$), F2 (Shifted Rastrigin's Function $[-5, 5]$), F3 (Shifted Ackley's Function $[-32, 32]$).
- Partially Additively Separable Functions:
  - Functions with a separable subcomponent: F4 (7-nonseparable, 1-separable Shifted and Rotated Elliptic Function $[-100, 100]$), F5 (7-nonseparable, 1-separable Shifted and Rotated Rastrigin's Function $[-5, 5]$), F6 (7-nonseparable, 1-separable Shifted and Rotated Ackley's Function $[-32, 32]$), F7 (7-nonseparable, 1-separable Shifted Schwefel's Function $[-100, 100]$).
  - Functions with no separable subcomponents: F8 (20-nonseparable Shifted and Rotated Elliptic Function $[-100, 100]$), F9 (20-nonseparable Shifted and Rotated Rastrigin's Function $[-5, 5]$), F10 (20-nonseparable Shifted and Rotated Ackley's Function $[-32, 32]$), F11 (20-nonseparable Shifted Schwefel's Function $[-100, 100]$).
- Overlapping Functions: F12 (Shifted Rosenbrock's Function $[-100, 100]$), F13 (Shifted Schwefel's Func-

tion with Conforming Overlapping Subcomponents $[-100, 100]$), F14 (Shifted Schwefel's Function with Conflicting Overlapping Subcomponents $[-100, 100]$).

- Non-separable Function: F15 (Shifted Schwefel's Function 1.2 $[-100, 100]$).

Initially, *10* independent runs were executed for every algorithm, which processed each of the benchmark functions up to *500K* fitness evaluations, using *15* particles, then their averages were collected. The parameters were empirically set for all algorithms, based on previous experiments. For PSO, the following parameter values were used: *w=0.7, c1=0.8, c2=1.1*. The parameters for all four CCPSO2 algorithms were set to allow up to 500 swarms. For CCPSO2, the S set was configured as $\{$*2,5,10,50,100,250*$\}$. For CCPSO2-IP, the number of dimensions for each swarm was chosen randomly, the boost parameters were set as E[$B_r = 0.5, maxTries = 2$] (for Exponential), S[$B_r = 0.521, maxTries = 3$] (for Sigmoid) and L[$B_r = 0.5, maxTries = 5$] (for Linear), and *minImprovement* was set to *0.05*. For all CCPSO2 algorithms, the value of r was set to *0.5*.

A Mann-Whitney test [15] was executed in a second experiment with the null hypothesis assuming no difference between the original CCPSO2 and the best performer boost function for CCPSO2-IP, having a significance level of *5%*.

Considering that CCPSO2 performs better on separable functions, one separable benchmark function (F2 [Shifted Rastrigin's Function]) was selected to run the algorithms up to 1 million fitness evaluations with 30 particles and 25 independent runs to investigate the convergence behaviour.

B. Results and Analysis

Table I shows the averaged results (cost of solutions) for PSO, CCPSO2 and CCPSO2-IP algorithms for 500K fitness evaluations in all benchmark functions. From the 15 addressed functions, the Exponential boost function obtained the best results in 9 cases and similar results in others 6 when compared with the best algorithm, while the Sigmoid obtained the best results in 2 benchmarks, the Linear was best in 1 and PSO was best in 3 benchmarks. All algorithms obtained comparable results in F3, F6 and F10, with small performance difference on F8 and F9. For simple comparison with real competition algorithms, the algorithm VMO DE was included from CEC13/15 published results [16] for guidance only, altough CCPSO2 is not intended for competitions.

TABLE I: Averaged Results for PSO, CCPSO2 and CCPSO2-IP.

Average and Standard Deviation (Cost of solutions). Best results in bold. Configuration used for PSO, CCPSO2, CCPSO2-IP S/E/L: [15 benchmark functions, 500K fitness eval., 15 particles, 10 independent runs]. VMO DE results for 600K fitness evaluations (available on CEC15 website [16]).

	F1	**F2**	**F3**	**F4**	**F5**
PSO	6.73E+010 (1.17E+010)	5.55E+004 (2.99E+003)	2.12E+001 (5.85E-002)	6.85E+011 (2.71E+011)	2.79E+007 (3.60E+006)
CCPSO2	1.42E+010 (1.02E+010)	1.09E+004 (6.26E+003)	2.03E+001 (6.97E-002)	2.41E+012 (1.11E+012)	1.83E+007 (1.58E+006)
CCPSO2-IP S	3.07E+008 (3.14E+008)	1.85E+003 (3.43E+002)	2.02E+001 (3.59E-002)	8.34E+011 (5.01E+011)	1.54E+007 (3.35E+006)
CCPSO2-IP L	3.34E+008 (4.44E+008)	1.76E+003 (2.91E+002)	2.03E+001 (3.61E-002)	1.17E+012 (1.21E+012)	**1.51E+007 (2.77E+006)**
CCPSO2-IP E	**8.55E+007 (6.35E+007)**	**1.44E+003 (2.59E+002)**	**2.01E+001 (4.42E-002)**	**6.08E+011 (3.62E+011)**	1.56E+007 (1.62E+006)
VMO DE	*2.26E+04 (3.55E+03)*	*2.36E+04 (8.26E+02)*	*2.47E+01 (9.59E-01)*	*1.15E+10 (1.49E+09)*	*7.28E+14 (0.00E+00)*
	F6	**F7**	**F8**	**F9**	**F10**
PSO	**1.05E+006 (5.05E+003)**	1.31E+010 (6.73E+009)	**1.10E+016 (1.19E+016)**	1.97E+009 (2.96E+008)	**9.31E+007 (3.99E+005)**
CCPSO2	1.07E+006 (2.92E+003)	2.40E+010 (8.79E+009)	2.97E+016 (2.05E+016)	1.52E+009 (1.55E+008)	9.49E+007 (8.70E+005)
CCPSO2-IP S	1.07E+006 (2.82E+003)	5.90E+009 (4.28E+009)	3.05E+016 (3.90E+016)	**1.05E+009 (1.24E+008)**	9.53E+007 (8.36E+005)
CCPSO2-IP L	1.07E+006 (2.56E+003)	6.27E+009 (2.35E+009)	1.53E+016 (8.89E+015)	1.06E+009 (1.77E+008)	9.45E+007 (8.92E+005)
CCPSO2-IP E	1.06E+006 (4.14E+003)	**2.13E+009 (6.82E+008)**	1.52E+016 (8.43E+015)	1.36E+009 (3.01E+008)	9.44E+007 (1.00E+006)
VMO DE	*7.75E+05 (1.91E+05)*	*2.38E+07 (8.25E+06)*	*1.21E+14 (4.55E+13)*	*1.60E+09 (3.34E+08)*	*2.08E+07 (3.13E+06)*
	F11	**F12**	**F13**	**F14**	**F15**
PSO	3.02E+012 (1.46E+012)	1.16E+012 (8.60E+010)	1.07E+012 (7.91E+011)	6.29E+012 (3.83E+012)	7.30E+009 (5.49E+009)
CCPSO2	5.66E+012 (4.78E+012)	2.20E+011 (2.01E+011)	2.84E+012 (5.63E+012)	8.14E+012 (7.60E+012)	3.34E+012 (7.27E+012)
CCPSO2-IP S	1.07E+012 (6.66E+011)	**3.38E+007 (1.91E+007)**	8.17E+010 (2.35E+010)	1.75E+012 (6.16E+011)	3.02E+008 (3.13E+008)
CCPSO2-IP L	9.85E+011 (3.85E+011)	6.74E+007 (6.39E+007)	8.73E+010 (1.82E+010)	1.66E+012 (4.57E+011)	5.09E+008 (3.18E+008)
CCPSO2-IP E	**5.42E+011 (2.56E+011)**	4.54E+007 (3.58E+007)	**2.51E+010 (1.22E+010)**	**7.09E+011 (2.28E+011)**	**1.48E+008 (3.70E+007)**
VMO DE	*9.87E+08 (1.90E+08)*	*7.73E+03 (5.18E+03)*	*8.36E+08 (2.64E+08)*	*3.39E+08 (5.51E+07)*	*8.43E+06 (6.82E+05)*

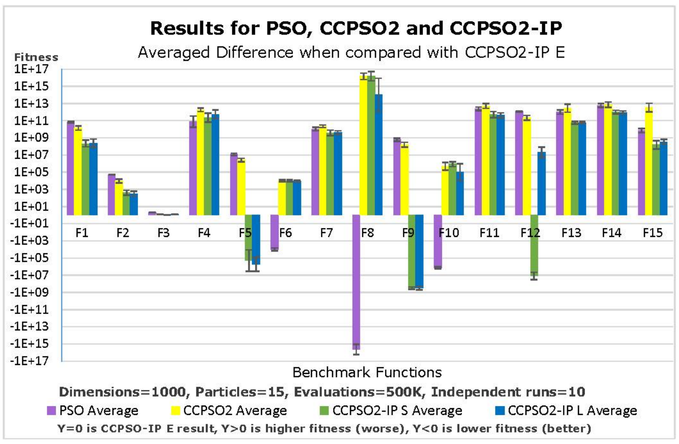


Fig. 2: Averages and Std.Dev. on logarithmic scale compared to CCPSO2-IP E.
Configuration: 15 benchmark functions, 500K fitness eval., 15 particles, 10 independent runs.

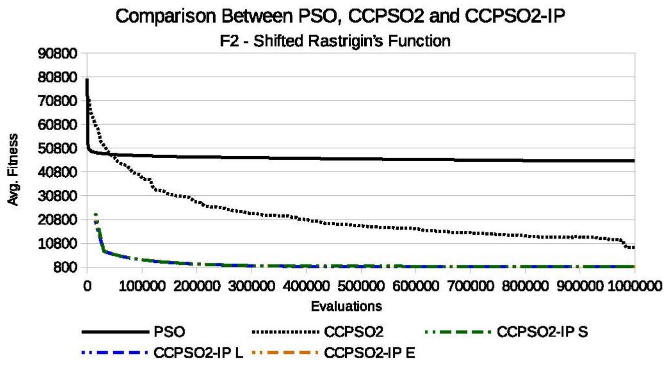


Fig. 3: Comparison between methods for F2 benchmark function.
General Configuration: [1M fitness evaluations, 30 particles, 25 independent runs]. PSO: [w=0.7; c1=0.8; c2=1.1]. CCPSO2: S={2,5,10,50,100,250}. CCPSO2-IP: E[Br=0.5, maxTries=2]; S[Br=0.521, maxTries=3]; L[Br=0.5, maxTries=5].

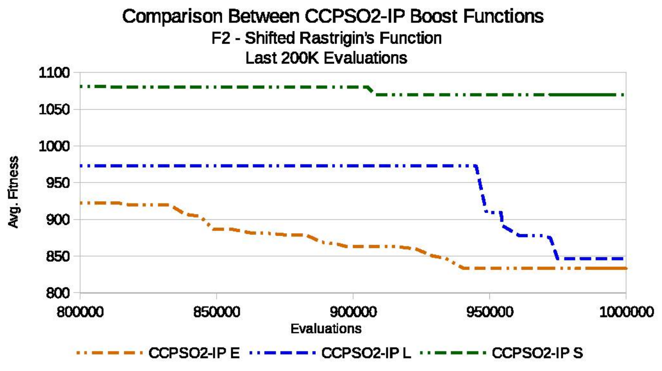


Fig. 4: Last 200K fitness evaluations for CCPSO2-IP methods on F2.
General Configuration: [1M fitness eval., 30 particles, 25 independent runs]. CCPSO2-IP: E[Br=0.5, maxTries=2]; S[Br=0.521, maxTries=3]; L[Br=0.5, maxTries=5].

In general (Fig.2), CCPSO2-IP was significantly better in 8 of the benchmarks and was similar to other algorithms in all other functions (only Sigmoid boost function obtained averaged worst result on F8, what would indicate some specific adjustment for it would be required for this benchmark function).

Table II shows the Mann-Whitney test results for 10 independent runs (up to 500K fitness evaluations) of CCPSO2 and the best performer CCPSO2-IP (Exponential). With exception of function F10, the null hypothesis was rejected in all cases, showing a significant difference between algorithms while optimizing these benchmark functions.

TABLE II: Mann-Whitney Test for CCPSO2 and CCPSO2-IP E on All Benchmark Functions

P-values for Mann-Whitney. Configuration used: [15 benchmark functions, 500K fitness eval., 15 particles, 10 independent runs].

F1	**F2**	**F3**	**F4**	**F5**
0.0007253	7.578e-05	1.083e-05	7.578e-05	0.0003248
F6	**F7**	**F8**	**F9**	**F10**
0.002089	1.083e-05	0.001505	0.0002057	*0.2176*
F11	**F12**	**F13**	**F14**	**F15**
7.578e-05	0.002089	1.083e-05	1.083e-05	1.083e-05

Fig.3 shows the averaged convergence behaviour for 25 independent runs on F2 benchmark function for all algorithms running up to 1M fitness evaluations. We can see a clear distinctions between PSO and CCPSO2 algorithms. PSO gets trapped right at the beginning of the search. CCPSO2 algorithms not only show the best performance but also provide continuous convergence.

Fig.4 shows the ability of IP-based algorithms to escape from good local minima even at the end of the evolutionary process.

V. Conclusion

Considering the advances obtained by multi-swarm optimization methods on complex high scale problems, and taking advantage of the good performance obtained by CCPSO2 method on high dimensional functions, an Iterative Partitioning method was developed to join some well known good practices into one single algorithm.

Results show a superior performance of the IP-based method on almost all benchmark functions, specially on the most difficult ones. It showed a good capacity of escaping from local minima even after long stagnation periods. More detailed studies are required to investigate the behaviour of each boost function on specific optimization problems and to create best practices to regulate the boost parameter.

References

[1] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *Evolutionary Computation, IEEE*, 2012.

[2] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, 2001.

[3] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Evolutionary Computation, 2008. CEC 2008. IEEE World Congress on Computational Intelligence*, 2008.

[4] F. V. D. Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, Faculty of Natural and Agricultural Science, 2001.

[5] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, 2004.

[6] Y. Zhang, Z. Jia, H. Jiang, and Z. Liu, "Image restoration based on robust error function and particle swarm optimization-bp neural network," *2008 Fourth International Conference on Natural Computation*, vol. 7, 2008.

[7] F. Caraffini, F. Neri, M. Gongora, and B. Passow, "Re-sampling search: A seriously simple memetic approach with a high performance," in *Memetic Computing (MC), 2013 IEEE Workshop on*, 2013.

[8] F. Caraffini, F. Neri, B. N. Passow, and G. Iacca, "Re-sampled inheritance search: high performance despite the simplicity," *Soft Computing*, vol. 17, no. 12, 2013.

[9] X. Li and X. Yao, "Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009.

[10] C. K. Goh, K. C. Tan, D. Liu, and S. C. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *EJOR*, vol. 202, no. 1, 2010.

[11] I. Poikolainen, G. Iacca, F. Caraffini, and F. Neri, "Focusing the search: a progressively shrinking memetic computing framework," *International Journal of Innovative Computing and Applications*, vol. 5, no. 3, 2013.

[12] S. S. Poorjandaghi and A. Afshar, "A robust evolutionary algorithm for large scale optimization," in *World Congress*, vol. 19, no. 1, 2014.

[13] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 3, 2014.

[14] X. Li, K. Tang, M. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the cec'2013 special session and competition on large scale global optimization," Evolutionary Computation and Machine, 2013.

[15] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[16] "Congress on evolutionary computation," http://www.cec2015.org/, 2015, accessed in: 08/Jul/2015.

A.4 CSMON: ESTIMATING STOP CONDITIONS OF STOCHASTIC METAHEURISTICS

Abstract

When dealing with metaheuristics, one important question is how many evaluations are worth spending in the search for better results. This work proposes a method to estimate the best moment to stop swarm iterations based on the analysis of the convergence behavior presented during optimization, aiming to provide an effective balance between saving fitness evaluations and keeping the optimization quality. An automated Convergence Stabilization Modeling operating in Online mode (CSMOn) is proposed based on a sequence of linear regressions using exponential and log-like curves. The method was tested on the CEC13 benchmark with CCPSO2-IP E algorithm and on 30 random Max-Set functions with the swarm algorithms PSO, ABC and CCPSO2-IP E. CEC13 results show that up to 90% less fitness evaluations are performed for functions where CCPSO2-IP E has a steady convergence, and up to 49% for functions where convergence is erratic, while penalties for fitness are kept small. Max-Set results demonstrates the robustness of CSMOn for the search algorithms tested. We conclude that CSMOn is capable of adapting to an optimization in progress, producing a good trade-off between result quality and evaluation savings.

Index terms— Convergence stabilization modeling, parameter tuning, curve fitting, online algorithms, swarm intelligence.

`http://dx.doi.org/10.1145/3071178.3071326`

Estimating Stop Conditions of Swarm Based Stochastic Metaheuristic Algorithms

Peter Frank Perroni, Daniel Weingaertner, Myriam Regattieri Delgado[*][†][‡]

July 20, 2017

1 Introduction

The increasing complexity of real world optimization problems requires powerful tools with robust configuration sets. Among these tools, stochastic metaheuristics play a special role, and there is a wide variety of literature concerning configuration methods and usage. However, despite their importance, not much effort has been done toward algorithm-independent procedures to determine a good cost/benefit number of fitness function evaluations, specially for continuous problems. Apart from being quite contextual (the solution relies on the strengths and weaknesses of the optimization algorithms and the problem being optimized), their behavior also obeys the unpredictable random numbers logic.

Traditionally, convergence modeling is directed toward asymptotic analysis, severely restricted to algorithm properties [1] where rigorous assumptions must be made, and mostly looking for global best solution under a theoretical infinite number of evaluations [2]. This has a limited practical value since real world complex optimizations normally expect high quality solutions (local instead of global ones) with minimal number of evaluations (instead of accepting large processing times). Some works like [3] ally asymptotic analysis with practical usage, but still under hard restrictions and assumptions and mostly for discrete problems with no large search spaces. In [4], a general framework using model convergence is proposed for discrete problems, which instead of using the best solutions only for overall convergence analysis, it uses the population history to interfere in new generations for better quality

[*]P. F. Perroni is with the Department of Informatics, Federal University of Paraná, Curitiba, PR (email: pfperroni@gmail.com).

[†]D. Weingaertner is with the Department of Informatics, Federal University of Paraná, Curitiba, PR (email: danielw@inf.ufpr.br).

[‡]M. R. Delgado is with the Department of Electrical Engineering, Federal University of Technology, Curitiba, PR (email: myriamdelg@utfpr.edu.br).

of solutions. In [5], a deep analysis is made of which conditions and parameters selection guarantee local convergence for Particle Swarm Optimization (PSO), but no novel process is provided for convergence modeling. Model-based optimization (MBO) methods [6] use either distribution-based models or surrogate models to improve convergence. The former includes the well-known Estimation of Distribution Algorithms (EDAs), which generate new candidate solutions according to the probability distribution estimated from the promising candidate solutions of the current population. In the latter, generally used when the evaluation of the cost function is very expensive or time consuming, the original function is replaced by a cheaper coarse-grained function that simplifies the search space, however its performance on high dimensionality problems and the selection of the correct surrogate model are still open research questions to be addressed. A more elaborated approach that also requires specific mathematical formulation for every search algorithm for a good performance is the Optimal Computing Budget Allocation (OCBA) [7], which operates by iteratively allocating function evaluations to individuals (until the total budget is exhausted) based on an allocation rule that identifies the most promising solutions, so that noisy/costly solutions are discarded and accuracy is improved, thus maximizing the convergence rate [8] by estimating the fitness.

This work proposes a different approach for local convergence analysis that is not dependent on a particular algorithm but on the standard behavior presented by general memory-based metaheuristics (like swarm algorithms). It treats the optimization algorithm as a black-box, thus exploring several aspects present in the search algorithm instead of requiring core changes to it. The main goal is to provide an on-line estimation of a good cost/benefit stopping point for every optimization run of a swarm based search algorithm, aiming to save function evaluations.

The hypothesis is that the inherent convergence behavior presented by these search algorithms when entering a local minimum can be modeled by a sequence of auto-adapted exponential and log-like curves.

The reason behind this hypothesis is that the social knowledge provided by the swarms gradually restrict the movement of each individual within a portion of the search area (given that the algorithm guarantees that individuals will reach a stable point [5]). Such movement toward the stable point (and potentially toward a local minimum) yields reasonable exponential convergence rates (when major improvements occur) followed by log-like convergence rates (when improvements slow down), providing run time data to model the convergence speed through linear regression.

When approaching a local minimum, due to the typical large sequence of small improvements in this region of the search space, the curve calculated through linear regression will be biased toward recent data points and away from initial (fast convergence) points, indicating the beginning of the convergence slow down. When the slow down happens during an exponential convergence, the majority of the improvements already occurred and next improvements tend to be modest. When the slow down happens during a log-like convergence, only minor incremental improvements are expected on subsequent evaluations, meaning that the convergence is stabilizing and further search might not be advantageous.

Considering this hypothesis, and given a restricted computational budget (limited swarm iterations, fitness evaluation or run time) on a multiple run scenario, CSMOn can automatically determine a good moment to stop evolving a population of solutions on a particular run and start searching on the next run. In this case, we consider that more important than the budget allocated to the fitness calculation of a specific particle, the challenge here is associated with the analysis of the convergence of the best fitness achieved by the swarm. This way, a good commitment could be found between the drawback (performance loss) of not having optimized until the end of the budget allocated for each independent run, and the advantages of saving computational budget (notice that using the saved effort on subsequent optimization runs is out of the scope of this work).

Therefore, the proposed method Convergence Stabilization Modeling in Online mode (CSMOn) consists in detecting local minimum traps of evolution by collecting convergence data (points comprising best fitness values and their respective number of evaluations) during the optimization run (online). A sequence of 2D exponential and log-like curves is adjusted over this data through linear regression, and a good cost/benefit stopping point is estimated by finding out the moment when the log-like curve takes distance from the initial convergence improvements. The source code of the CSMOn implementation is available online at [9].

2 Mathematical Basis

Taking the convergence data as a list $\ell = \{(x_j, y_j)\}, \forall j \in \{1, \cdots, s\}$ of size s of strictly monotonically decreasing 2D coordinate points, y_j as the jth fitness value obtained at fitness evaluation x_j, p_1 as an initial position in this list, p_2 as a final position and p_c as a position close to p_2 ($1 \leq p1 < p_c \lessapprox p2 \leq s$), one way of defining a convergence stabilization is observing the relative differences of the convergence improvements. When the difference $[y_{p_c}, y_{p_2}]$ is very small compared to the difference for the whole interval $[y_{p_1}, y_{p_2}]$, we can observe a convergence stabilization. In such definition, the stabilization is contextual and depends on how many convergence improvements have been obtained and the relationship between them. Hence, the greater the number of improvements at beginning of the sequence and the difference between them, more improvements with small differences will be required to state that a convergence is reaching a stabilization point at a later stage.

This section describes the curves, the linear regression method and the properties used in this work to model the convergence stabilization behavior.

2.1 Curves

Two curve functions are used to model the general convergence stabilization behavior of a swarm based stochastic metaheuristic: the Exponential and the Power functions. Large improvements typically seen at the initial of the search or when escaping from local minimum traps are better modeled by exponential curves, whereas log-like curves are well suited for moderate and small improvements seen on the remaining of the search.

The natural Exponential Function Exp is defined as:

$$Exp(x) = \alpha e^{-\beta x} \tag{1}$$

where α is the Intersect point with Y axis ($\alpha = Exp(0)$) and β is the relative growth speed of the function.

The deterioration of an exponential convergence can be detected by discovering the point when the sublinear convergence begins. The list ℓ is said to converge sublinearly when:

$$\lim_{s\to\infty} \frac{|y_s - L|}{|y_{s-1} - L|} \to 1 \tag{2}$$

where L denotes the last possible y value of the sequence. Assuming the last possible value as L_M for a maximum number of M evaluations, this criteria can be re-written for a minimization problem by an approximation as the decay equation (3):

$$\mathcal{D}_e(\ell) = 1 - \frac{y_s - L_M}{y_{s-1} - L_M} \tag{3}$$

indicating how much the current value y_s differs from the previous value y_{s-1} concerning the estimation of the limit achieved due to computational limitation.

The log-like Power Function Pow can be defined as

$$Pow(x) = \alpha x^{-\beta} \tag{4}$$

where α is the Scaling factor ($\alpha = Pow(1)$) and β is the growth speed of the function.

A logarithmic convergence commences only after the deterioration of the exponential convergence. Thus, once the criteria (2) is satisfied, the list ℓ is said to converge logarithmically when:

$$\lim_{s\to\infty} \frac{|y_s - y_{s-1}|}{|y_{s-1} - y_{s-2}|} \to 1 \tag{5}$$

As in (3), this limit can be simplified by decay equation (6):

$$\mathcal{D}_l(\ell) = 1 - \frac{y_s - y_{s-1}}{y_{s-1} - y_{s-2}} \tag{6}$$

2.2 Linear Regression

Least Square (LS) is a linear regression method used to find the best-fitting curve that adjusts to a data set ℓ of points (x, y).

Due to the rigid shape of the exponential curve, the precision of the curve calculated through LS is increased if the data set is translated to bring the convergence values $\mathbf{y}$ closer to the X axis. Let $\gamma_j = y_j - min(\mathbf{y}) + 1$, $\forall j \in [p_1 \cdots p_2]$ be the new translated value, then equations (7a) and (7b) can be used to estimate the parameters β and α of the exponential curve [10] between $[p_1, p_2]$:

$$\beta_e(\ell, p_1, p_2) = \frac{\sum\limits_{j=p_1}^{p_2} (x_j - \overline{x})(ln(\gamma_j) - \overline{ln(\gamma)})}{\sum\limits_{j=p_1}^{p_2} (x_j - \overline{x})^2} \tag{7a}$$

$$\alpha_e(\ell, p_1, p_2) = \left(\sum_{j=p_1}^{p_2} ln(\gamma_j) - \beta_e \sum_{j=p_1}^{p_2} x_j \right) / \, (p_2 - p_1 + 1) \tag{7b}$$

where $\overline{x}$ is the average of x_j and $\overline{ln(\gamma)}$ is the average of $ln(\gamma_j)$.

The exponential curve calculated through LS tends to adjust better to the regions where the majority of points are located, quickly moving away from the other points. Considering that in a typical convergence there are more moderate and small improvements than initial major improvements, there are fewer data points available to represent the beginning of the exponential decay than the remaining of the convergence. Therefore, considering that the exponential curve (1) decreases at a rate proportional to its current value, it can be said that the beginning of the convergence slow down occurs when the parameter α_e (intersection with Y axis) has its value reduced, when compared to the previous regression, after adding one more point to the sequence.

The second part of the convergence stabilization modeling uses a power curve whose parameters β and α can be estimated as follows:

$$\beta_p(\ell, p_1, p_2) = \frac{\sum\limits_{j=p_1}^{p_2} (log(x_j) - \overline{log(x)})(log(y_j) - \overline{log(y)})}{\sum\limits_{j=p_1}^{p_2} (log(x_j) - \overline{log(x)})^2} \tag{8a}$$

$$\alpha_p(\ell, p_1, p_2) = \left(\sum_{j=p_1}^{p_2} log(y_j) - \beta_p \sum_{j=p_1}^{p_2} log(x_j) \right) / \, (p_2 - p_1 + 1) \tag{8b}$$

where $\overline{log(x)}$ and $\overline{log(y)}$ are the average of $log(x_j)$ and $log(y_j)$.

Because the growth speed β_p is constant along X axis, the power curve has a behavior similar to the exponential curve on the intersect (parameter α_p) when distantiating from initial data points (since the majority of the points are moderate and small improvements), but occurring more slowly than for the exponential curve (given the growth speed is constant here).

3 CSMOn: Convergence Stabilization Modeling Operating in Online Mode

The proposed method (CSMOn) controls the execution of the optimization algorithm by continuously requesting the metaheuristic to process until the next best result is found. Every new best fitness value (fit) and its associated number of evaluations (ne) are appended to a list of points $\mathbf{gb} = \{(ne_j, fit_j)\}, \forall j \in \{1, \cdots, s\}$, replacing ℓ as the convergence list for the LS (section 2.2). The process is divided in three main phases (Figure 1):

i Initial/Major Improvements: Determine through criteria (2) and (5) the moment when major improvements cease, ensuring the search will stop only after first improvements have occurred. Positions 1 and p_b identify this phase (red area in Figure 1) where no regression is performed.

ii Intermediate Improvements: After phase i, the regression process is started and finds the point where the exponential curve (blue) takes distance from initial convergence data (first red circle). Positions p_b and p_T identify this phase (orange area), with position p_T defining the transition from exponential to logarithmic decay.

iii Final/Convergent Improvements: Once phase ii finishes, a log-like curve (green curve) is adjusted to the remaining improvements until the curve departs from initial points (second red circle). Positions p_T and p_S identify this phase (yellow area), with p_S defining the point where the logarithmic sequence stabilizes (third red circle), saving unnecessary fitness evaluations (grayed area).

Considering the search algorithm can escape from a particular stagnation point, phases i, ii and iii can repeat along evolution.

Metaheuristics like PSO typically show the greatest improvements at the very beginning of the search, and initial fitness values can be quite distant from the general fitness found on the remaining of the search. In such cases, the initial points (two points would suffice) may be considered outliers and are not included in $\mathbf{gb}$.

The following plateau-curve is proposed to model the general convergence stabilization behavior of swarm based optimization algorithms (with

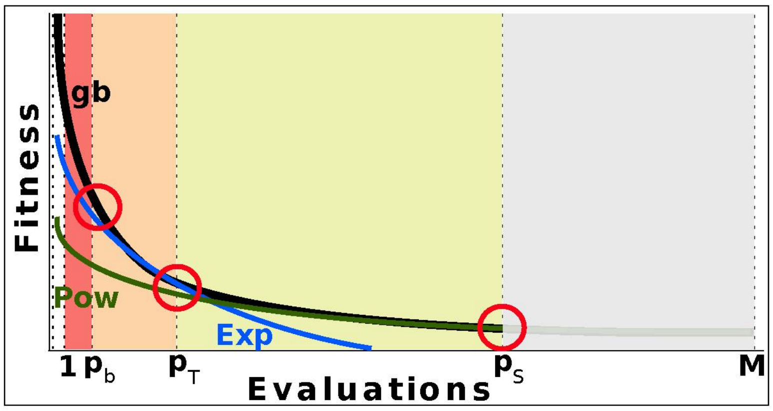


Figure 1: CSMOn Method defining transition points for fast, intermediate and slow convergence
log-like curve represented by the power curve):

$$F_{app}(ne_j) = \begin{cases} Exp(ne_j) \ \ (1) & \text{if } j \leq p_T \\ Pow(ne_j) \ \ (4) & \text{if } j > p_T \end{cases} \tag{9}$$

where F_{app} is the function that approximates the real fitness fit_j at ne_j evaluations.

Algorithm 1 CSMOn Algorithm

```
1: Input: $\{M, R\}$
2: $p_T \leftarrow -1, p_S \leftarrow -1$
3: $r \leftarrow 0.99$                                ▷ Set $r \leftarrow R$ for a fixed relaxation
4: $append(\mathbf{gb}, GetBest(1, M))$
5: repeat
6:     $r \leftarrow max(r^2, R)$                     ▷ Remove this line for a fixed relaxation
7:     if $p_S = -1$ then                             ▷ Look for Exponential convergence
8:         $p_T \leftarrow AdjustExp(\mathbf{gb}, M, r)$
9:     if $p_T > 0$ then                              ▷ Log-like convergence started
10:        $p_S \leftarrow AdjustLog(\mathbf{gb}, M, r, p_T)$      ▷ At this point,
    either finishes for a given $r$ ($p_S > 0$) or a new exponential convergence
    is detected ($p_S = -1$) and returns to line 5
11: until $ne_{p_S} >= M$ or ($r = R$ and $p_S > 0$)
```

These are the basic CSMOn steps (Algorithm 1):

- Two problem-dependent input parameters are required: the maximum number of fitness function evaluations for every run M (i.e. the portion of the total budget for this run) and the relaxation parameter $R \in]0, 1[$. The relaxation controls the limit sensibility, specifying how close the equations (3) and (6) should be from satisfying the respective criteria (2) and (5).

- Function $GetBest(nBest, M)$ returns from the search algorithm a list $\{(ne_1, fit_1), \cdots\ (ne_{nBest},\ fit_{nBest})\}$. To deal with search algorithms that do not show a minimum initial convergence, if $\mathbf{gb}$ is empty, the function will start to add elements to the list only after $\mathcal{D}_e(\mathbf{gb}) \geq .0001$ (0.01% improvement). If the maximum number of evaluations M is reached before filling the list, the search algorithm stops and the function appends to the list a dummy entry containing the last best fitness (M, fit_M).

- Initially one best value is obtained from the search algorithm and the initial value for the relaxation is defined: $r \in [R, R_s]$, with $R > 0$ and $R_s < 1$. If a fixed relaxation should be used, then $r = R$, otherwise it will decrease in a quadratic way from the superior to the inferior limit. A decreasing relaxation accepts an initially less compliant exponential and an increasingly more suitable log-like convergence, while fixed relaxation uses the same compliance level throughout the search.

- Function $AdjustExp(\mathbf{gb}, M, r)$ requests new elements for $\mathbf{gb}$ to detect the end of the major improvements (p_b) and the transition point p_T between an exponential and a log-like convergence.

- Function $AdjustLog(\mathbf{gb}, M, r, p_T)$ requests new elements until the log-like convergence stabilizes at point p_S, indicating a possible end of the search. However, if a new exponential convergence occurs, control is returned to *AdjustExp* (line 8 in Algorithm 2) and p_b and p_T are reset.

- If a fixed relaxation is used, the method finishes at the first p_S found. If a quadratic decreasing relaxation is used, p_S is reset, r has its value reduced for the calculation of a new p_S and the method finishes when the relaxation reaches R. In any case, the method stops when the maximum number of evaluations M is reached.

Algorithm 2 AdjustExp

```
1: Input: ${\mathbf{gb}, M, r}$
2: $s_{prev} \leftarrow s$
3: $append(\mathbf{gb}, GetBest(2, M))$                ▷ Request next 2 bests results
4: if $s - s_{prev} < 2$ then return $-1$
5: $p_b \leftarrow -1$
6: while $ne_s < M$ do
7:     if $\mathcal{D}_e(\mathbf{gb}) < r$ and $\mathcal{D}_l(\mathbf{gb}) < r$ then
8:         if $p_b = -1$ then                     ▷ Initialize the regression process
9:             $p_b \leftarrow s - 2$
10:            $\alpha_2 \leftarrow \alpha_e(\mathbf{gb}, p_b, s)$ (7b)
11:        else              ▷ Check the incremental difference on regression
12:            $\alpha_1 \leftarrow \alpha_2$
13:            $\alpha_2 \leftarrow \alpha_e(\mathbf{gb}, p_b, s)$
14:            if $\alpha_2 < \alpha_1$ then
15:                return $s$                     ▷ Transition to log-like curve
16:    else                                       ▷ Stop the regression
17:        $p_b \leftarrow -1$
18:    $append(\mathbf{gb}, GetBest(1, M))$             ▷ Request next best result
   return $-1$
```

The adjustment of the exponential curve (Algorithm 2) has the following steps:

1. A sequence of best values is requested and appended to **gb** as long as the fitness improvements are substantial (i.e. until criteria (2) and (5) are satisfied) (line 7). Then, the start of the weak part of the exponential decay is set $p_b = s - 2$ (line 9), marking the beginning of the regression calculation (7b) (line 10).

2. For every next best value (line 18), a new regression is calculated in the interval $[p_b, s]$ (line 13) and the incremental difference to previous regression is compared to detect if the curve is biased toward recent best values, representing a slow down on the exponential convergence (line 14).

3. When the bias is detected, the end of the exponential curve is set $p_T = s$ (returns the control to line 8 in Algorithm 1).

4. However, if the convergence speed increases above the defined relaxation r, the regression stops (line 17) and the process proceeds requesting the sequence of best values.

Algorithm 3 AdjustLog

```
1: Input: ${\mathbf{gb}, M, r, p_T}$
2: $s_{prev} \leftarrow s$
3: $append(\mathbf{gb}, GetBest(3, M))$
4: if $s - s_{prev} < 3$ then return $-1$
5: $\alpha_1 \leftarrow \alpha_p(\mathbf{gb}, p_T, s-1)$ (8b)
6: $\alpha_2 \leftarrow \alpha_p(\mathbf{gb}, p_T, s)$
7: while $\alpha_2 \geq \alpha_1$ and $ne_s < M$ do          ▷ Check incremental difference
8:     if $\mathcal{D}_e(\mathbf{gb}) \geq r$ or $\mathcal{D}_l(\mathbf{gb}) \geq r$ then
9:         return $-1$                          ▷ Exponential convergence detected
10:    $append(\mathbf{gb}, GetBest(1, M))$
11:    $\alpha_1 \leftarrow \alpha_2$
12:    $\alpha_2 \leftarrow \alpha_p(\mathbf{gb}, p_T, s)$
   return $s$
```

Finnally, log-like curves are adjusted (Algorithm 3):

1. A sequence of best values is requested and the respective incremental regressions (8b) are calculated for the interval $[p_T, s]$ (lines 5, 6 and 12) and then compared to detect if the curve is biased toward recent best values, representing a logarithmic stabilization (line 7).

2. When the bias is detected, the respective position $p_S = s$ is returned, marking the end of the relevant logarithmic decay (line 10 in Algorithm 1).

3. However, if the search algorithm escapes logarithmic convergence moving towards an exponential convergence (line 8), then the whole process starts over from current position s, ignoring previous convergence points.

In a multiple run scenario, the total budget has to be partitioned in a way that every independent run has its own portion of the evaluations. A simplistic schema to use CSMOn under this scenario would set M to the number of evaluations assigned to every run, and the saved evaluations be used on upcoming runs by creating a different update mechanism for relaxation r that takes the overall remaining budget into consideration.

4 Experimental Results

Two test sets are used to experiment with CSMOn: one fixed and one randomly generated. The fixed test set comprises specially difficult competition benchmark functions for optimization called CEC13 [11] and is composed of 15 minimization functions of 1000 dimensions with global best at $L_M = 0$:

- Fully-separable Functions: F1, F2, F3.
- Partially Additively Separable Functions: F4, F5, F6, F7, F8, F9, F10, F11.
- Overlapping Functions: F12, F13, F14.
- Non-separable Function: F15.

The randomized test set uses an automated and parameterized test-problem (landscape) generator called Max-Set of Gaussians (MSG) [12, 13], which generates nonlinear, nonseparable random test functions based on the sum of Gaussians.

A competitive decomposition method [14] for high dimensional problems [15, 16], CCPSO2 [17] (Cooperatively Coevolving Particle Swarm) is a recent state-of-the-art algorithm based on the well known PSO [18] metaheuristic. It is a variant of a previous version called CCPSO [19, 20], being recognized by its computational simplicity and good capacity of dealing with multimodal complex functions and separable problems [21] [22]. CCPSO2-IP [23] is an Iterative Partitioning method for CCPSO2 that improves its capacity of dealing with non-separable problems by automating the task of finding combination of dependent dimensions, so that CCPSO2-IP can escape more easily from local minima traps caused by poor variable interaction. For a comprehensive test of CSMOn under complex optimization scenarios, CCPSO2-IP with exponential boost function (CCPSO2-IP E) has been used to optimize CEC13 functions due to its strong capacity of escaping from local minima and keep converging after long periods of stagnation on high dimensional problems, producing convergence behaviors of hard prediction. For the random MSG functions, the swarm algorithms CCPSO2-IP E, PSO [24] and ABC [25] (Artificial Bee Colony) have been tested.

The results are then summarized with the mixed model based on confidence intervals proposed by [26][26], which creates a simple pairwise comparison plot (between versions with and without CSMOn) that shows how significant is the difference between results.

4.1 Configurations

Considering that the CSMOn stop condition does not entirely depend on the number of evaluations M (i.e. it mainly depends on the convergence behavior and the relaxation parameter), the search algorithms are processed in two phases to compare the fitness obtained with and without the use of CSMOn:

1. under the control of CSMOn method (i.e., until the last convergence stabilization p_S is found).

2. it continues the optimization on the same run until M fitness evaluations are performed (i.e., processing additional $M - p_S$ fitness evaluations).

For the CEC13 test set, CCPSO2-IP E configuration is set as: $p = 30$, $B_r = 0.5$, $maxTries = 2$. The following values are used for M: $M1 = 1e6$, $M2 = 3e6$, $M3 = 6e6$ and $M4 = 1e7$. For each M, the following values are used for R (both for fixed and quadratic decreasing relaxations): {0.01, 0.02, ..., 0.09, 0.1, 0.2,..., 0.9}. Every benchmark function F is processed 30 times for each combination of $M \times R$ and an analysis has been conducted aiming to evaluate:

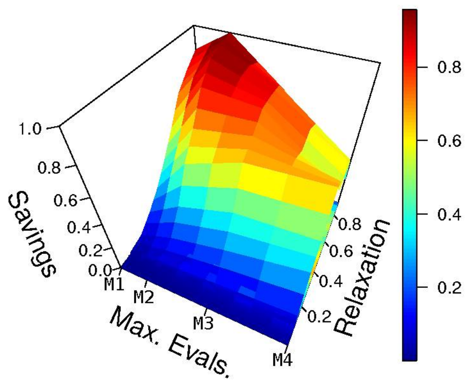
1. The effect of the relaxation parameter: For every function F and each combination of $M \times R$, both the economy on the number of fitness evaluations ($(M - ne_{p_S})/M$) and the complement of the relative approximation error of the fitness ($1 - |(fit_M - fit_{p_S})/fit_M|$) are multiplied to obtain the trade-off matrix. Matrices for all 15 functions F are averaged to illustrate the effect of the relaxation parameter as the maximum number of evaluations increases, where higher values mean better commitment between the fitness approximation and the economy on function evaluations.

2. Overall advantages/drawbacks for best relaxations: From the previous experiment, two relaxation parameters are chosen (R_{low}, R_{high}). For each function F, the best from these two parameters is used to compare the best fitness obtained in every execution (for both M and p_S evaluations) and the averaged evaluation savings for each p_S.

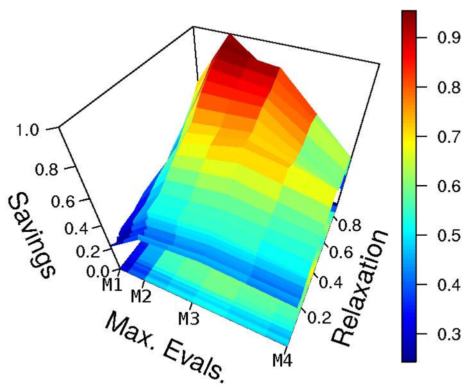
3. Detailed advantages/drawbacks for best functions: This experiment aims to visualize how far the stabilization points (p_S) found by CSMOn are from the best possible values (restricted to M). For the best performed functions F, the percentage of reduction on fitness approximation quality is compared with the percentages of economy on function evaluations obtained by CSMOn, using as configurations all values of M and the two previously selected relaxations.

For the second test set, the MSG landscape generator is used to create test classes of 50 dimensions (varying the number of Gaussians randomly in the interval $[50, 100]$). Configurations are empirically set to CCPSO2-IP E: $\{p = 30, B_r = 0.6, maxTries = 3\}$, PSO: $\{p = 30, c1 = 0.4, c2 = 1.4, w = 0.9\}$[27] and ABC: $\{foodSource = 30, trials = 1e3\}$:

(4) Every search algorithm is executed 30 times and, for each execution, a new test class is created from which a random test function is instantiated and optimized 30 times independently, with $M = 5e4$. At the end, 900 executions were be performed for each search algorithm.



(a) Fixed relaxation



(b) Quadratic relaxation

Figure 2: Averaged Cost/Benefit of CSMOn for CEC13 Functions. ($M1 = 1e6$, $M2 = 3e6$, $M3 = 6e6$, $M4 = 1e7$)

4.2 Results and Analysis

Experiments described in section 4.1 are discussed below. For the CEC13 functions:

Experiment 1: Figure 2(a) shows the results for fixed relaxation and Figure 2(b) for quadratic decreasing relaxation. Although they look similar, fixed relaxation presents a smoother trade-off matrix (soft transitions between configurations), with better results for small M and high R. However, quadratic decreasing relaxation obtains overall better results for both high M and low R.

Experiment 2: From the previous experiment, $R_{low} = 0.08$ and $R_{high} = 0.7$ are chosen. Figure 3 presents the best possible results (in blue) and CSMOn's results (in red). Confirming the good convergence of CCPSO2-IP E for functions F3, F5, F6, F9 and F10 [23], best results are obtained on F5 and F9, with high relaxation $R = 0.7$, exceeding 90% economy (the red values on X axis), showing that for problems where the search algorithm is well suited, large relaxations should suffice (although some medians seem differ, their relative differences are very small).

Function F2 processed almost up to M evaluations, meaning that CCPSO2-IP E keeps converging throughout the search. For the other 9 functions (comprising 36 combinations of F $\times M$), 28 combinations have their medians and 15 have their quartiles similar to the best possible results, with up to 49% economy on latter 15 combinations. Fully separable F1 and F2 are special cases since for those functions CCPSO2-IP E presents an erratic convergence with frequent transitions between poor logarithmic and moderate exponential decays, meaning that the method would process up to M evaluations if smaller relaxations were used.

Experiment 3: Figure 4 shows the percentages of advantage and drawback for functions F3, F5, F6, F9 and F10 with quadratic decreasing relaxations $R_{low} = 0.08$ and $R_{high} = 0.7$. Results clearly show the advan-

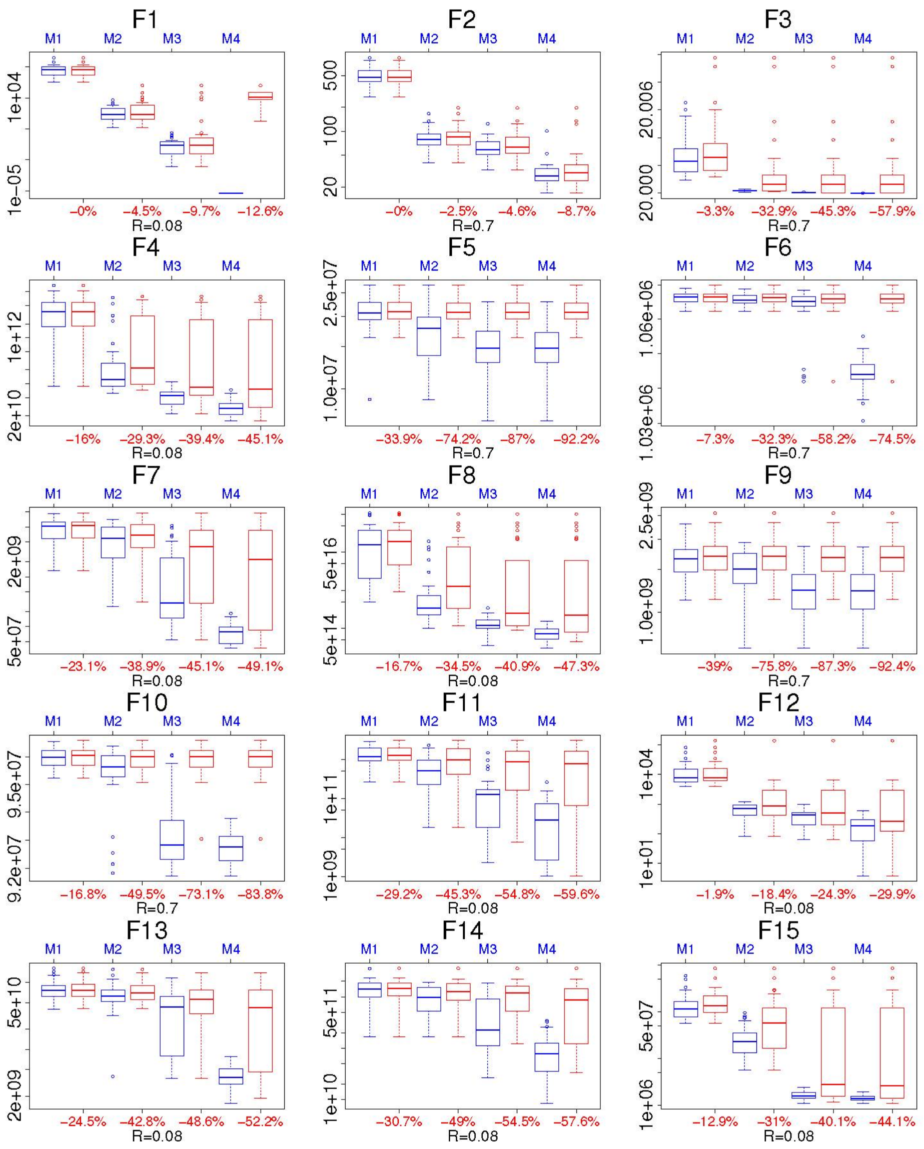


Figure 3: Comparison Between CSMOn stopping point and Processing until M evaluations.

Quadratic decreasing $R = \{0.08, 0.7\}$ (indicated below bottom axis). Red box is CSMOn method and blue box is limited by M. Vertical axis is the fitness value. Actual number of CSMOn's function evaluations is given by M configuration (top axis) minus respective percentage of economy (bottom axis).

tages overcoming the drawbacks in all 40 combinations of F $\times M \times R$, with R_{high} presenting the highest economy, reaching above 80% difference between drawback and advantage on function F10.

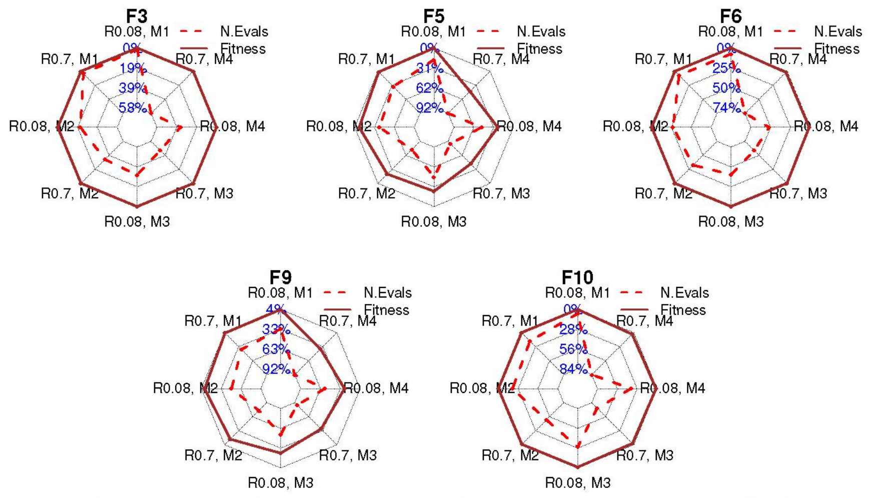


Figure 4: Percentage of Economy on Fitness Evaluations $\times$ Reduction on Fitness due to Quadratic Decreasing Relaxations ($R = \{0.08, 0.7\}$). Best configurations are Fitness on borders and N.Evals on center.

For the MSG functions:

Experiment 4: In the paired comparison plot (Figure 5), overlapping intervals show that the means are not significantly different, leading to the conclusion that there is no significant difference between results with and without CSMOn for CCPSO2-IP E. Despite the fact that the figure shows that PSO with and without CSMOn are not similar, the normalized fitness difference is within a small percentage (3%), with minor importance when compared to the economy of 27% on fitness evaluations. The small economy obtained by CCPSO2-IP E (6%) and ABC (6.7%) occurred because these algorithms were still converging for most of the runs, reaching M evaluations. The 4% significance difference for ABC is due to the "scout bees" phase, that reset the position from current to a different local minimum, suggesting that CSMOn could be used to detect upfront a better moment for the scout phase to occur.

Although not included here, additional tests with fixed relaxations have shown that they are more conservative on convergence stabilization analysis than decreasing relaxation, tending to produce smaller drawbacks with moderate savings on erratic optimization.

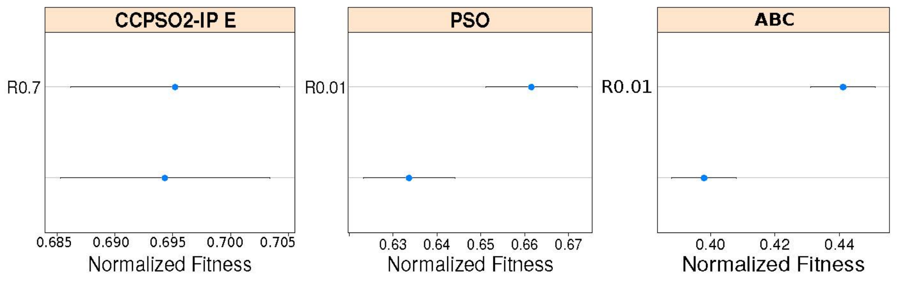


Figure 5: Paired comparison of CCPSO2-IP E, PSO and ABC search algorithms, with and without CSMOn.
MSG landscape generator used to create 30 test classes per algorithm, one test function instantiated per class and 30 executions per function using $M = 5e4$. Overall fitness evaluations economy: CCPSO2-IP E=6%, PSO=27%, ABC=6.7%. Decreasing relaxation values (R) shown on labels.

5 Conclusion

Tracking convergence trends for stochastic algorithms is not a trivial task, however if successfully done could benefit the overall optimization by controlling drawbacks resulting from fitness approximation due to evaluation savings. This work has presented a method to track this trend for swarm based stochastic metaheuristic.

The proposed approach (CSMOn) has been tested with CCPSO2-IP E search algorithm on all 15 CEC13 benchmark functions. Results show that the process presented to model the convergence stabilization behavior is able to effectively adapt to each optimization in progress (online), obtaining a good trade-off between the drawback resulting from the fitness approximation and the advantage of saving fitness function evaluations. The best results are obtained with functions where CCPSO2-IP E has a stable convergence behavior, presenting a great economy with the fitness consistently approaching the best possible values (i.e. values achieved considering the restriction of maximum number of evaluations M). At some cases, CSMOn obtains above 90% economy with moderate drawbacks on the fitness estimation, and in other cases it obtains above 70% positive difference between advantages and drawbacks. Tests with Max-Set random functions show that CSMOn is able to detect, for metaheuristics CCPSO2-IP E, PSO and ABC, when the search is no longer converging, suggesting that it should be either stopped or some additional action should be taken by the search algorithm to reactivate the convergence.

Future work would include: the test to take into consideration long stagnation periods of the search algorithm; different update mechanisms for the relaxation parameter aiming to react based on the distance to the maximum number of evaluations; a relaxation update mechanism to consider the remaining budget in a multiple run scenario; and the test of the CSMOn method with non-swarm memory-based metaheuristics.

References

[1] J. Sun, X. Wu, V. Palade, W. Fang, C.-H. Lai, and W. Xu, "Convergence analysis and improvements of quantum-behaved particle swarm optimization," *Information Sciences*, vol. 193, pp. 81–103, 2012.

[2] "Chapter 21 metaheuristics," in *Simulation*, ser. Handbooks in Operations Research and Management Science, S. G. Henderson and B. L. Nelson, Eds. Elsevier, 2006, vol. 13, pp. 633 – 654.

[3] J. Pichitlamken and B. L. Nelson, "A combined procedure for optimization via simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 13, no. 2, pp. 155–179, 2003.

[4] W. J. Gutjahr, "Convergence analysis of metaheuristics," in *Matheuristics*. Springer, 2009, pp. 159–187.

[5] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[6] T. Bartz-Beielstein and M. Zaefferer, "Model-based methods for continuous and discrete global optimization," *Applied Soft Computing*, vol. 55, pp. 154–167, 2017.

[7] S. Zhang, J. Xu, L. H. Lee, E. P. Chew, W. P. Wong, and C.-H. Chen, "Optimal computing budget allocation for particle swarm optimization in stochastic optimization," *IEEE Transactions on Evolutionary Computation*, 2016.

[8] J. Rada-Vilela, M. Zhang, and M. Johnston, "Optimal computing budget allocation in particle swarm optimization," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation.* ACM, 2013, pp. 81–88.

[9] P. F. Perroni, *CSMOn*, Federal University of Paraná, Curitiba, Brazil, 2017. [Online]. Available: http://web.inf.ufpr.br/vri/software/csmon

[10] J. L. Devore, *Probability and Statistics for Engineering and the Sciences, Eighth Edition, Chapter 13.* Cengage Learning, 2010.

[11] X. Li, K. Tang, M. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the cec'2013 special session and competition on large scale global optimization," Evolutionary Computation and Machine, 2013.

[12] M. Gallagher and B. Yuan, “A general-purpose tunable landscape generator,” *IEEE transactions on evolutionary computation*, vol. 10, no. 5, pp. 590–603, 2006.

[13] T. Bartz-Beielstein, “How to create generalizable results,” in *Springer Handbook of Computational Intelligence.* Springer, 2015, pp. 1127–1142.

[14] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, “Metaheuristics in large-scale global continues optimization: a survey,” *Information Sciences*, vol. 295, pp. 407–428, 2015.

[15] F. Caraffini, F. Neri, M. Gongora, and B. N. Passow, “Re-sampling search: A seriously simple memetic approach with a high performance,” in *2013 IEEE Workshop on Memetic Computing (MC).* IEEE, 2013, pp. 52–59.

[16] F. Caraffini, F. Neri, B. N. Passow, and G. Iacca, “Re-sampled inheritance search: high performance despite the simplicity,” *Soft Computing*, vol. 17, no. 12, pp. 2235–2256, 2013.

[17] X. Li and X. Yao, “Cooperatively coevolving particle swarms for large scale optimization,” *Evolutionary Computation, IEEE*, vol. 16, no. 2, pp. 210–224, 2012.

[18] R. C. Eberhart, J. Kennedy *et al.*, “A new optimizer using particle swarm theory,” in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[19] X. Li and X. Yao, “Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms,” in *IEEE Congress on Evolutionary Computation, 2009. CEC’09.* IEEE, 2009, pp. 1546–1553.

[20] C. K. Goh, K. C. Tan, D. Liu, and S. C. Chiam, “A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design,” *EJOR*, vol. 202, no. 1, pp. 42–54, 2010.

[21] I. Poikolainen, G. Iacca, F. Caraffini, and F. Neri, “Focusing the search: a progressively shrinking memetic computing framework,” *International Journal of Innovative Computing and Applications*, vol. 5, pp. 127–142, 2013.

[22] S. S. Poorjandaghi and A. Afshar, “A robust evolutionary algorithm for large scale optimization,” in *World Congress*, vol. 19, no. 1, 2014.

[23] P. F. Perroni, D. Weingaertner, and M. R. Delgado, "Automated iterative partitioning for cooperatively coevolving particle swarms in large scale optimization," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, 2015, pp. 19–24.

[24] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE transactions on evolutionary computation*, vol. 8, no. 3, pp. 225–239, 2004.

[25] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, Tech. Rep., 2005.

[26] M. Chiarandini and Y. Goegebeur, "Mixed models for the analysis of optimization algorithms," in *Experimental Methods for the Analysis of Optimization Algorithms.* Springer, 2010, pp. 225–264.

[27] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters*, vol. 85, no. 6, pp. 317–325, 2003.