LUCAS FRANCISCO AMARAL OROSCO PELLICER

OTIMIZAÇÃO DE HIPERPARÂMETROS DE MODELOS MACHINE LEARNING COM BARYSEARCH

São Paulo
2020

LUCAS FRANCISCO AMARAL OROSCO PELLICER

OTIMIZAÇÃO DE HIPERPARÂMETROS DE MODELOS MACHINE LEARNING COM BARYSEARCH

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

São Paulo
2020

LUCAS FRANCISCO AMARAL OROSCO PELLICER

OTIMIZAÇÃO DE HIPERPARÂMETROS DE MODELOS MACHINE LEARNING COM BARYSEARCH

Versão Corrigida

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Área de Concentração:

Engenharia de Sistemas

Orientador:

Prof. Dr. Felipe Miguel Pait

São Paulo
2020

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 12 de Junho de 2020

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Pellicer, Lucas Francisco Amaral Orosco
Otimização de Hiperparâmetros de Modelos Machine Learning com BarySearch / L. F. A. O. Pellicer -- versão corr. -- São Paulo, 2020.
91 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1.Aprendizado de Máquina 2.Hiperparâmetros 3.Otimização 4.Métodos de Busca 5.Hibridização I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II.t.

Dedico esse trabalho à minha mãe, Rosanna e ao meu pai, Francisco, por todo ensinamento, inspiração, auxílio, carinho e orientação. Também para minhas avós (Irene e Ignês) e avôs (Antônio e Walter), além de tias e tios e aos meus amigos Mike e Jimmy que sempre deram alegria.

AGRADECIMENTOS

Agradeço à Universidade de São Paulo (USP) e ao Laboboratório de Automação e Controle (LAC) por toda formação acadêmica. Ao Itaú Unibanco pelo apoio e disponibilidade de horas de estudo para o desenvolvimento desse trabalho. Agradecimento aos colegas André Hermenegildo Costa e Fernando Fernandes pela revisão de textos e artigos. Finalmente, agradecimento especial ao orientador Prof. Dr. Felipe M. Pait por toda orientação e compartilhamento de conhecimento importantíssimos para a realização desse trabalho.

"Epígrafe"

E, se bem que seja obscuro
Tudo pela estrada fora,
E falso, ele vem seguro,
E, vencendo estrada e muro,
Chega onde em sono ela mora.
E, inda tonto do que houvera,
A cabeça, em maresia,
Ergue a mão, e encontra hera,
E vê que ele mesmo era A Princesa
que dormia.
Fernando Pessoa

RESUMO

Obter bons desempenhos de modelos de aprendizado de máquina geralmente requer que os hiperparâmetros sejam ajustados. Entretanto, é complicado encontrar uma função matemática bem definida entre os valores do hiperparâmetro e o desempenho do modelo. A coleta de valores de desempenho do modelo é cara e ainda o comportamento das funções de desempenho tende a ser imprevisível, com muitas regiões de oscilação ou mesmo regiões descontínuas. Além disso os hiperparâmetros podem ser contínuos, discretos, categóricos ou condicionais, o que torna o problema de ajuste de hiperparâmetros complexo. Muitas técnicas foram desenvolvidas para solucionar esse problema. Neste trabalho, é apresentada a técnica BarySearch: um método livre de derivação com baixo custo computacional. Essa técnica apresenta um bom compromisso entre resultado e esforço de execução. O BarySearch utiliza do método do baricentro já utilizado em sintonização de controladores. O método apresenta características de convergências interessantes e pode comportar similar a um gradiente descendente ou métodos evolucionários sob suposições razoáveis.

Palavras-Chave – Aprendizado de Máquina, Hiperparâmetros, Otimização, Métodos de Busca, Hibridização

ABSTRACT

Obtaining good models in machine learning applications often requires hyperparameters to be tuned. However, it is rarely possible to find a well-defined mathematical relationship between the values of the hyperparameters and the performance of the model. Performance evaluation is computationally expensive, the behavior of the objective function tends to be unpredictable and oscillatory, and hyperparameters may not be continuous, which makes the tuning process a challenging optimization problem. Many optimization techniques have been developed to solve this complicated problem. In this work, we present the BarySearch technique: a derivative-free optimization technique with a low computational cost. This technique exhibits a good compromise between performance and execution effort. BarySearch uses the barycenter method, already used in some controller optimization techniques. The method has interesting properties in terms of convergence and may behave in a manner similar to the descending gradient or evolutionary methods under reasonable assumptions.

Keywords – Machine Learning, Hyperparameters, Optimization, Searching Methods, Hybridization

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ML	Machine Learning (Aprendizado de Máquina)
AutoML	Automatic Machine Learning
CASH	Combined Algorithm Selection and Hyperparameter Optimization
AUC	Area Under Curve (métrica de classificação)
PSO	Particle Swarm Optimization (Otimização por Enxame de Partículas)
LPSO	Local Particle Swarm Optimization
UPSO	Unified Particle Swarm Optimization
APSO	Accelerated Particle Swarm Optimization
ChaosPSO	Chaotic Particle Swarm Optimization
CLPSO	Compreheensive Learning Particle Swarm Optimization
AdpPSO	Adaptive Particle Swarm Optimization
COPSO	Cooperative Particle Swarm Optimization
GA	Genetic Algoritmo
GLPSO	Genetic Learning Particle Optimization
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
TPE	Tree Parzen Estimators
SMAC	Sequencial Model-based Algorithm Configuration
TPOT	Tree-based Pipeline Optimization Tool
LHS	Latin Hyperplane Sampling
tol	tolerância
n est	n estimators (quantidade de estimadores)
max feat	max features
MLP	Multi-Layer Perceptron
LightGBM	Light Gradient Boosting Machine
KNN	k-Nearest Neighbor
SVM	Support Vector Machine
L.R.	Learning Rate
ADAM	Adaptive Moment Estimation (Otimizador utilizado no treinamento de redes neurais)
SGD	Stochastic Gradient Descendent (gradiente descendente estocástico)
CNN	Convolutional Neural Network (redes neurais convolucionais)

LISTA DE SÍMBOLOS

ν	parâmetro de convergência
z	parâmetro de curiosidade
σ	desvio padrão do parâmetro de curiosidade
t	iteração
D	dimensão
$bary$	baricentro

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação do Problema

Muitas aplicações atuais utilizam modelos de inteligência artificial, com grande destaque em modelos de Aprendizado de Máquina (Machine Learning - ML). Modelos ML são modelos matemático e estatísticos que utilizam distribuições passadas para identificar padrões e inferir dados futuros (BISHOP, 2006). Esses modelos são treinados em bases passadas conhecidas como base de treino, onde o próprio algoritmo do modelo ajuste seus parâmetros internos em consonância com as distribuições da base de treino (BISHOP, 2006).

Todo modelo ML necessita de passar pelo processo de treinamento em alguma base de dados para definição de parâmetros internos do modelo. Entretanto, os usuários dessas técnicas podem sintonizar parâmetros anteriormente ao processo de treinamento, esses parâmetros são denominados como hiperparâmetros (BISHOP, 2006).

Esses hiperparâmetros são extremamente sensíveis para o treinamento de modelos ML e por consequência o desempenho do modelo (KOCH et al., 2018). Assim, em muitas aplicações, é desejável escolher o melhor conjunto de hiperparâmetros para alcançar o melhor desempenho possível do modelo, como a Figura 1. Esse problema de encontrar o melhor conjunto de hiperparâmetros é conhecido como tuning de modelos ML. O problema de tuning é definido matematicamente como um problema de otimização, onde a função objetivo é a métrica de desempenho que o usuário deseja otimizar.

Contudo, o problema de otimização de hiperparâmetros é bem complexo. Primeiramente, os modelos ML são vistos como caixas pretas, pois é comum não encontrar uma relação bem definida matematicamente entre a função objetivo e os hiperparâmetros em tuning de modelos ML. As métricas de avaliação de modelos são costumeiramente não diferenciáveis e muitas vezes não continuas. Além dessa questão central no problema de tuning, podem ser citadas as dificuldades: os hiperparâmetos podem ser contínuos, discretos, categóricos ou mesmo condicionais; as funções podem apresentar regiões bem

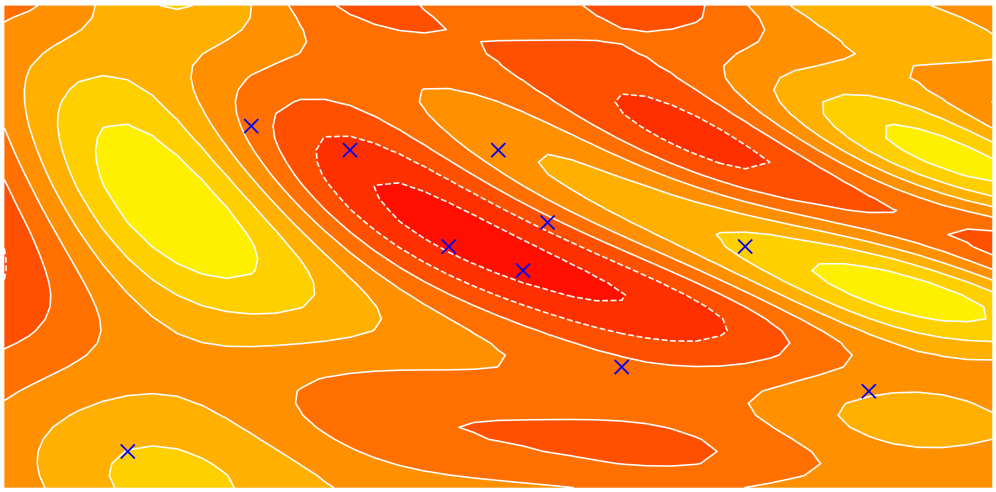

Figura 1: Problemas de Otimização: encontrar melhor combinação de valores de entrada de x em uma função $f(x)$.
Fonte: (BISSUEL, 2019)

oscilatórias ou de grandes platôs que dificultam a otimização (KOCH et al., 2018).

Para contornar esses problemas, diversos algoritmos foram desenvolvidos. Dentre eles, podem ser destacados: métodos de busca aleatória e exaustiva(BERGSTRA; BENGIO, 2012), busca bayesianas (SNOEK; LAROCHELLE; ADAMS, 2012), Sequential Model-based Algorithm Configuration (SMAC) (HUTTER; HOOS; LEYTON-BROWN, 2011), Tree Parzen Estimator (TPE) (BERGSTRA et al., 2011), algoritmos genéticos (LORENA; CARVALHO, 2008; LE; FU; MOORE, 2020) e algoritmos evolucionários (HANSEN; OSTERMEIER, 2001; EBERHART; KENNEDY, 1995). Todos esses algoritmos apresentam uma característica semelhante de não dependerem de calcular o gradiente da função objetivo e dessa forma são denominados como algoritmos livres de derivação (CONN; SCHEINBERG; VICENTE, 2009).

1.2 Objetivo

Embora a grande quantidade de algoritmos desenvolvidos e empregados para resolução de problemas de otimização de hiperparâmetros de modelos ML, ainda existem muitos aspectos que são temas de pesquisas nesse contexto. Dentre esses temas, podem ser destacados: melhores maneiras de inicialização dos algoritmos; formulação matemática de convergência de solução na otimização; bom compromisso entre desempenho e gasto computacional; técnicas para construção e seleção de modelos automáticamente (AutoML).

O trabalho apresenta a nova técnica de otimização BarySearch que é baseado no já proposto método do baricentro (Pait, 2018). O BarySearch será diretamente comparado aos principais algoritmos de otimização já empregados, tanto em questão de desempenho, como eficiência computacional de processamento. Esses testes serão realizados sobre contexto diferentes e com níveis de complexidade muito diferente.

Os testes serão feitos utilizando o Python versão 3, com aplicações de modelos ML retirados diretamente das bibliotecas scikit-learn 0.21 (BUITINCK et al., 2013) e keras 1.3 (CHOLLET et al., 2015). Todos os testes serão realizados sobre a mesma configuração de computador, além de serem submetidos em padrões de testes e validações amplamente encontrados na academia.

1.3 Organização do Trabalho

A estrutura desse trabalho foi planejada para abranger todo o trabalho de pesquisa, desenvolvimento e validação do algoritmo proposto neste texto.

No capítulo 2, é apresentada a pesquisa bibliográfica de problema de tuning ou otimização de hiperparâmetro, bem como algoritmos que foram construídos a fim de solucionar esse problema.

No capítulo 3, está descrita a formulação matemática do BarySearch onde é definido o método do baricentro e está inserido teoremas de convergência. Além disso, este capítulo entre em detalhes da inicialização, dos parâmetros internos e comportamento do BarySearch.

No capítulo 4, são demonstrados diversos testes e resultados práticos para a validação de desempenho do BarySearch como forma de solução do problema de otimização de hiperparâmetros.

Por fim, no capítulo 5, estão dispostas as conclusões e discussões finais sobre esse trabalho e destacando suas contribuições.

2 RESUMO BIBLIOGRÁFICO

Este capítulo é escrito para levantar os contextos, os conceitos e soluções que envolvem os problemas de otimização de hiperparâmetros em modelos ML. Trabalhos anteriores estão citados e explicados para motivar a principal aplicação que o algoritmo proposto no trabalho.

2.1 Problema de Tuning de Hiperparâmetro

Em muitas aplicações de modelos ML, é desejável alcançar o melhor desempenho disponível. Devido a esse requisito, é comum ser realizado o tuning dos hiperparâmetros desses modelos ML.

O problema de tuning de modelo de ML pode ser formulado como: considere os hiperparâmetros de um modelo ML X, onde existe uma função de medição de desempenho do modelo conhecida como $f(X)$. Dessa forma o problema de tuning é encontrar o melhor grupo de valores dos hiperparâmetros X que otimizem a função métrica $f(X)$.

Apesar da formulação matemática do problema de tuning ser relativamente simples, a sua resolução apresenta algumas dificuldades. Primeiramente, os modelos ML são vistos como verdadeiras caixas-pretas, assim não é possível encontrar uma fórmula bem definida da função de desempenho $f(X)$ (CONN; SCHEINBERG; VICENTE, 2009). Sem essa formulação exata, muitas técnicas de otimização não podem solucionar e ainda não possível calcular gradiente de $f(X)$, o que torna técnicas que utilizem derivação inviáveis.

Além dessa dificuldade inicial, os problemas de tuning ainda encontram outras dificuldades que deixam o problema ainda mais complexo de solucionar (KOCH et al., 2018):

- Os hiperparâmetros dos modelos ML podem ser contínuos, discretos, categóricos ou condicionais.
- As funções de desempenho $f(X)$ podem apresentar comportamentos complexos

como áreas ruidosas e/ou oscilatórios e regiões de grandes platôs.

- Existem certas combinações de hiperparâmetros X que são impossíveis, formando restrições do modelo.

2.1.1 Problemas de Seleção de Modelo e AutoML

O projeto de ML é composto por alguns blocos e algoritmos de muita importância. Em geral, esses projetos apresentam tratamento dos dados chamados de pré–processamento, também apresentam tratativas de agregação de variáveis ou seleção de variáveis e finalmente um modelo de ML responsável pela tarefa que será treinado nos dados de treino. Todos os blocos de pré–processamento, agregação e seleção de variáveis e modelos ML são essenciais para a performance do projeto ML, além disso, é esperado que cada um desses blocos tenham hiperparâmetros relevantes para o desempenho.

Considerando a influência de todos os blocos, a tarefa de AutoMl é construir automaticamente a melhor composição de préprocessamento, agregações e seleções de variável, modelo ML e seus hiperparâmetros que otimize a performance do projeto ML. Dessa forma, o AutoML também resolve o problema de tuning do modelo, mas contém um complicador ainda maior que é selecionar os melhores blocos.

Na literatura, o problema de AutoML é conhecido como problema de CASH (*Combined Algorithm Selection and Hyperparameter*) (THORNTON et al., 2013). Esse problema é considerado muito complexo de ser resolvido e atualmente é uma grande área de volume de pesquisas. O problema de CASH é definido matematicamente como:

Onde A é o conjunto de blocos selecionado (pré–processamento, agregações e seleções de variável, modelo ML) e é o conjunto de valores que maximizam o desempenho do projeto ML treinado no *dataset* de treino D_treino e validado no *dataset* D_val.

O método de validação podem ser os populares holdout (separação em treino e teste) ou o cross-validation e a métrica de desempenho $f(x)$ pode ser qualquer métrica de desempenho de projetos ML, como taxa de erro médio ou *logloss* para classificações e RSME para regressões. Como o problema de CASH é um problema mais complexo que o tuning, as mesmas dificuldades encontradas no tuning são iguais ou mais díficeis que no tuning. Destacando a dificuldade em encontrar uma relação matemática entre as entradas e a função $f(x)$ vista como caixa-preta; o mau comportamento da função $f(x)$; parâmetros numéricos, categóricos e entre outros (THORNTON et al., 2013).

A seguir, são discutidos alguns algoritmos propostos para resolver o problema de

tuning e alguns para o problema de AutoML CASH, principalmente os considerados como estado da arte.

2.2 Algoritmos de Otimização de Hiperparâmetros

Como já explicado, os problemas de tuning de modelos são essencialmente complexos. Devido à ausência da formulação matemáticas das métricas de desempenho e os hiperparâmetros, não é possível usar métodos de otimização direta ou mesmo usar métodos que se utilizam de cálculos do gradiente (CONN; SCHEINBERG; VICENTE, 2009).

Para contornar esses empecilhos, diversas técnicas de tuning foram desenvolvidas. Em geral, essas técnicas apresentam uma característica em comum de não precisar do cálculo de derivadas, o que são conhecidas como técnicas livres de derivação (CONN; SCHEINBERG; VICENTE, 2009), e apresentam um comportamento semelhante em buscar conjunto de valores X e colher os resultados de $f(X)$ até alcançar valores promissores em diversas iterações, assim também são conhecidas como técnicas iterativas.

As iterações para esse problema de tuning são custosas computacionalmente, pois o passo de colher o resultado de $f(X)$ requisita o treinamento do modelo com um conjunto de valores de X, o que é custoso e pode ser demorado (KOCH et al., 2018). Dessa forma, o desejável é que não sejam necessárias muitas iterações para o algoritmo alcançar a convergência.

Nos próximos tópicos, são descritas essas técnicas utilizadas para o tuning dos modelos ML.

2.2.1 Algoritmos Exaustivos e de Amostragem Aleatória

As primeiras técnicas desenvolvidas para tuning de hiperparâmetro de modelos ML foram técnicas exaustivas nas quais são definidos todos os valores possíveis de X. Basicamente, a ideia é testar todas as combinações possíveis com esses possíveis valores de X. Essa técnica é conhecida como Grid Search e é uma das mais utilizadas nas aplicações atuais.

Apesar do Grid Search ser simples de compreendido e implementado, a técnica apresenta vários problemas em situações com dimensões maiores e mais valores de testes. A cada valor testado e a cada nova dimensão incluída em X faz com que o número de combinações a serem testadas cresça exponencialmente. Dessa forma, o Grid Search poder

ser muito custoso computacionalmente.

Para tentar contornar isso, foi proposta a busca aleatória. Na busca aleatória, valores de X são sorteados aleatoriamente e são testadas. A busca retorna o melhor conjunto de valores de X. Com a busca aleatória, é possível contornar os problemas de dimensionalidade e ser um algoritmo menos custoso que o Grid Search.

Além de tudo isso, Bersgtra defende que a busca aleatória apresenta melhor desempenho esperado que o Grid Search, principalmente com modelos mais complexos (redes neurais, boostings e outros). A justificativa desse melhor desempenho é que as funções de métricas de desempenho de modelos ML não são funções bem comportadas, com regiões pequenas de pontos promissores. O Grid Search espalha os pontos testados e muitas vezes não realiza busca por essas regiões, contudo a busca aleatória (BERGSTRA; BENGIO, 2012) apresenta uma probabilidade de explorar essas regiões (Figura 2). Assim, quantos mais pontos explorados, melhor é resultado esperado.

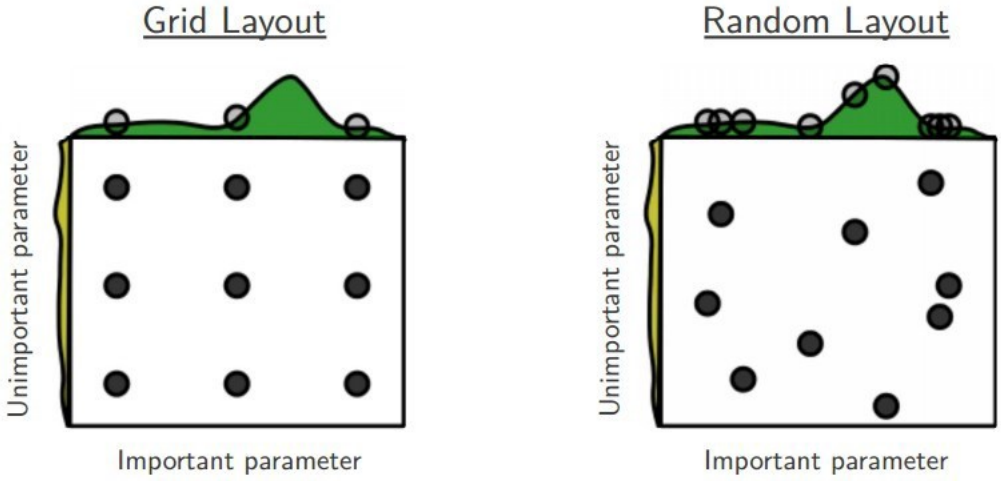

Figura 2: Comparativo entre as buscas exaustivas e busca aleatória. Repare que na busca exaustiva é provável que não explore regiões onde se encontram os resultados mais promissores, contudo a busca aleatória apresenta uma probabilidade de atingir essas mesmas regiões. Dessa forma, a convergência esperada da busca aleatória é melhor.
Fonte: (BERGSTRA; BENGIO, 2012)

Nesse mesmo trabalho, Bersgtra (BERGSTRA; BENGIO, 2012) demonstra que a busca aleatória consegue convergir para o ponto ótimo em modelos ML.

2.2.2 Busca Bayesiana

A Busca Bayesiana é considerada o método bem sucedido em termos de problemas de tuning de modelos ML (SNOEK; LAROCHELLE; ADAMS, 2012). As buscas bayesianas

demonstraram grande sucesso em problemas complexos de otimização de funções caixas-pretas e são totalmente livres de derivação.

Como o próprio nome indica, as buscas bayesianas são fundamentadas na teoria de probabilidade condicional de Bayes. Para isso, elas se valem de dois componentes essenciais: as curvas de *surrogates* e a maximização da esperança esperada (SNOEK; LAROCHELLE; ADAMS, 2012).

As curvas de *surrogates* são curvas para estimar a probabilidade de algum ponto x apresentar um resultado promissor de $f(x)$. As curvas de *surrogates* utilizam das informações anteriores para atualizarem, semelhante ao cálculo de uma probabilidade condicional ilustrado pela Figura 3.

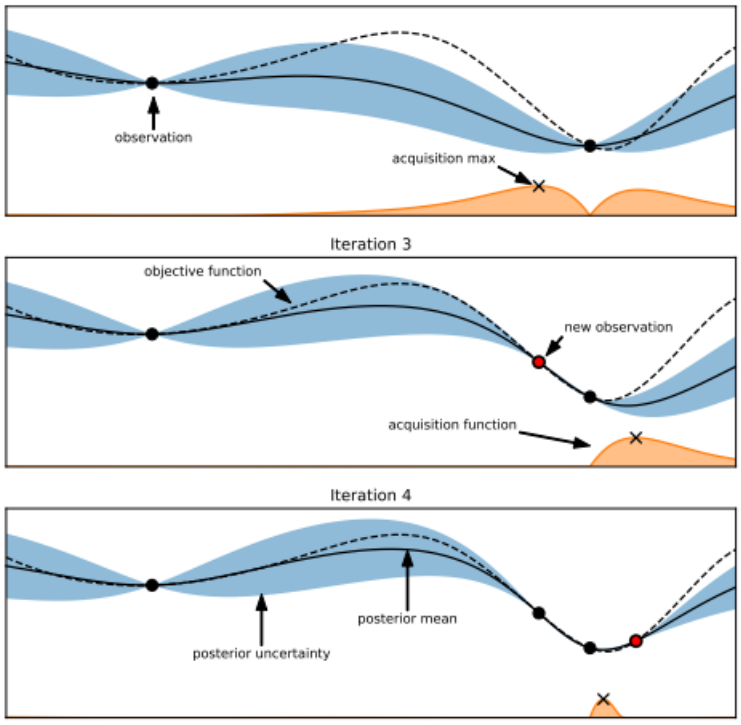

Figura 3: Funcionamento da curva de surrogate. Objetivo é que conforme as iterações passam, a curva consegue ficar com um formato mais semelhante a $f(x)$.
Fonte: (HUTTER; KOTTHOFF; VANSCHOREN, 2019)

As curvas de *surrogates* costumam utilizar de processos Gaussianos para estimarem seus formatos. Esses processos conseguem utilizar pontos observados anteriormente para estimarem a probabilidade de algum resultado de uma função $f(x)$. Conforme o aumento de pontos observados, é de esperar que a curva *surrogate* apresente um formato mais semelhante à função objetivo como Figura 3.

Os processos Gaussianos demonstraram grande valor em muitas aplicações. Entre-

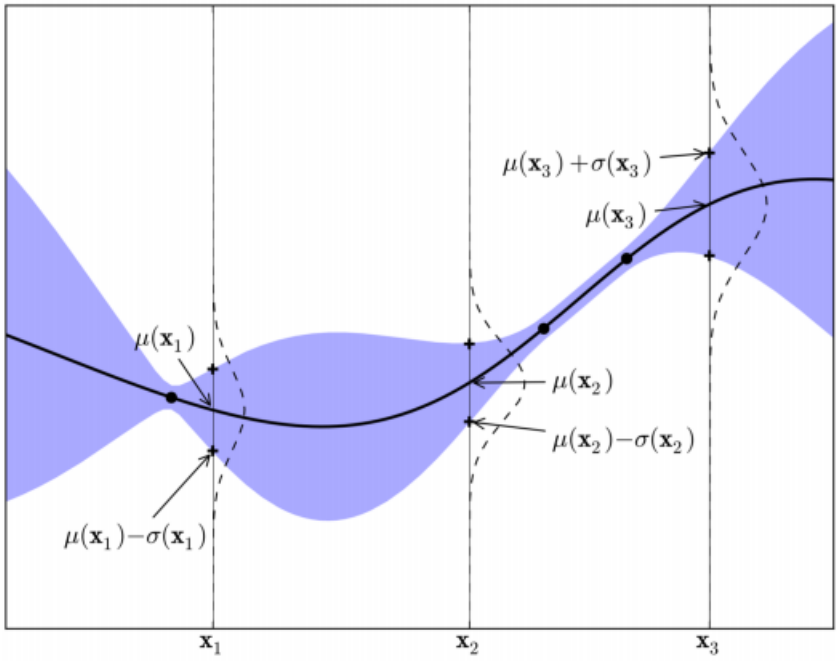

Figura 4: Funcionamento do processo gaussiano. A ideia é que conforme as iterações de x_i ocorrem, o processo consegue calcular uma probabilidade de o próximo ponto ter um resultado bom.
Fonte: (KIM; CHUNG, 2019)

tanto eles apresentam problemas em grande número de dimensões de X, pois os processos Gaussianos não conseguem separar bem vetores de alta dimensionalidade e também aumentam o custo computacional. Em contrapartida, as curvas permitem o uso de diferentes parâmetros de entrada (numéricos ou categóricos).

Em seguida, o algoritmo foca em selecionar os pontos com maior probabilidade de serem promissores. Esses pontos são amostrados com valor de $f(x)$ coletado. Dessa maneira, é possível repetir o processo iterativo de atualizar as curvas de *surrogate* e se aproximarem mais do comportamento de $f(x)$.

Apesar de bons resultados com o processo Gaussiano, muitos outros métodos bayesianos buscaram maneiras de substituir o processo Gaussiano a fim de atingirem melhores desempenho, principalmente em altas dimensões.

2.2.3 Técnicas Evolucionárias

2.2.3.1 Simulated Annealing

O Simulated Annealing é uma das técnicas mais antigas evolucionárias. Ela foi inspirada no processo dos metaleiros em forjar os metais (KIRKPATRICK; GELATT; VEC-

CHI, 1983). Primeiro se esquenta o metal para ficar mais maleável e conforme a diminuição da temperatura, ele fica mais resistente.

O Simulated Annealing é muito semelhante à busca aleatória, pois os candidatos selecionados a serem testados são gerados aleatoriamente. Entretanto, a cada iteração t, o Simulated Annealing pode aceitar um resultado pior que o atual através de uma probabilidade P_t. Caso o resultado já seja melhor que o atual, o candidato é aceito independente da temperatura.

A probabilidade P_t é proporcional a uma "temperatura" (KIRKPATRICK; GELATT; VECCHI, 1983) pela Equação 2.1. A temperatura é alta no início do algoritmo e no final das iterações, a temperatura abaixa. Como consequência, no início da busca o algoritmo está mais sujeito a aceitar resultado piores que o original e conforme as iterações passam, a busca fica mais resistente a resultado ruins (Figura 5).

$$T(t) = \frac{d}{log(t)} \tag{2.1}$$

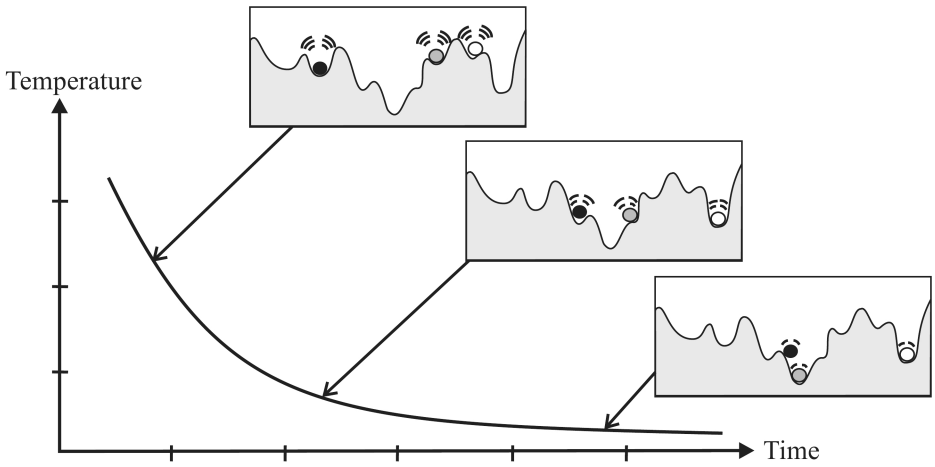

Figura 5: Funcionamento do Simulated Annealing e a importância da temperatura no funcionamento do algoritmo.
Fonte: (LEDESMA; RUIZ-PINALES; GARCIA-HERNANDEZ, 2012)

Pode se concluir que o algoritmo inicialmente tem um comportamento mais exploratório e depois fica mais convergente (KIRKPATRICK; GELATT; VECCHI, 1983). Esse tipo de comportamento é que inspirou muitos outros algoritmos desenvolvidos posteriormente. Em algumas variações do Simulated Anealing propõem um parâmetro de reaquecimento (CHIARANDINI et al., 2003) para aumentar a temperatura no meio das iterações e logo diminuindo a resistência e deixando o comportamento mais exploratório. A ideia é explorar o máximo possível e evitar pontos de mínimos locais.

2.2.3.2 Otimização de Enxame - PSO

Otimização de enxame ou PSO (Particle Swarm Optimization) é uma técnica clássica de técnicas evolucionária, proposta por Kennedy em (EBERHART; KENNEDY, 1995; Kennedy; Eberhart, 1995). Essa técnica é baseada em movimentos coletivos onde a direção de um indivíduo depente tanto de seus movimentos anteriores como do movimento do coletivo. Esse tipo de movimento pode ser visto em grupos de pássaros, colônias de bactérias ou mesmo colônias de insetos. O método clássico é formulado pelas equações de posição X de uma partícula p pela Equação 2.2.

$$x_p(t+1) = x_p(t) + v_p(t) \tag{2.2}$$

Assim, a próxima posição de uma partícula p depende de uma velocidade, definida pela Equação 2.3.

$$v_p(t) = \omega \cdot v_p(t) + c_1 \cdot r_1(t) \cdot (pbest(t) - x_p(t)) + c_2 \cdot r_2(t) \cdot (gbest(t) - x_p(t)) \tag{2.3}$$

Onde *pbest* é a melhor posição anterior da partícula p e *gbest* é a melhor posição do enxame. Os parâmetros ω, c_1 e c_2 são respectivamente os parâmetros de inércia, parâmetro individual e parâmetro social. Finalmente r_1 e r_2 são variáveis aleatórias com distribuição uniforme. Resumidamente, o Algoritmo 1 representa o funcionamento do PSO.

Algorithm 1: Tradicional PSO

Result: Melhor combinação de x que otimiza $f(x)$
Inicialize os parâmetros ω, c_1 and c_2;
Inicialize um enxame de tamanho P com posições iniciais $x_p(0)$ no espaço de busca D;
while *critério de parada não é satisfeito* **do**
  Compute $f(x_p(t))$ para todas as P partículas ;
  if $f(x_p(t)) < f(pbest(t))$ **then**
    $pbest(t) = x_p(t)$;
  end
  if $f(x_p(t)) < f(gbest(t))$ **then**
    $gbest(t) = x_p(t)$;
  end
  Calcule a velocidade de cada partícula conforme a Equação 2.3;
  Atualize a posição de cada partícula conforme Equação 2.2
end

A técnica demonstrou bons resultados em diversas aplicações para engenharia, oti-

mização e mesmo para tuning de modelos ML .

Outras variações do algoritmo PSO foram desenvolvidos. Em (SHI; EBERHART, 1998), foi construído um parâmetro de inércia que variável linearmente conforme o tempo. Outros trabalhos se concentraram em construção de restrição de valores de velocidade como valores de V_{max} para ajudar na convergência do PSO ou mesmo quando em métodos para tratar situações quando a partícula ultrapassa os limites do espaço de busco (Clerc; Kennedy, 2002).

Kennedy em (KENNEDY, 1999; KENNEDY; MENDES, 2002) criou o conceito de topologia que controla a troca de informação entre as partículas de um enxame. A ideia era que sem total visibilidade do enxame, as partículas evitam de ficarem presas em pontos de mínimos locais. Em (SUGANTHAN, 1999) foi produzido o Local PSO onde as partículas estão ligadas a apenas m partículas mais próximas.

Parppolos em (PARSOPOULOS; VRAHATIS, 2004) propôs o Unified PSO, o qual há um parâmetro u que pondera tanto a formulação entre o tradicional PSO e o Local PSO. Além da topologia local, muitas outras topologias foram desenvolvidas (LIU et al., 2016a), como a "roda", na qual a uma partícula focal concentra todas as informações. Outras de destaque são: quatro clusters em que as partículas são divididas em quatro grupos totalmente ligados e Von Neuman onde as partículas são ligadas por quatro vizinhos de direções verticais e horizontais conforme a Figura 6.

Para aproveitar o histórico das posições anteriores e não apenas o melhor resultado, foi desenvolvido o Fully Informed PSO (Mendes; Kennedy; Neves, 2004). Essa técnica pondera todas as posições anteriores através da distância atual e do resultado.

Algumas outras técnicas focaram em acelerar o PSO, como o Accelerated PSO (YANG; DEB; FONG, 2012) que utiliza apenas o melhor valor do enxame *gbest* para o cálculo da velocidade da partícula. Em (HEFNY; AZAB, 2010) foi proposto o Chaotic PSO, na qual os valores de ω ou β são definidos por mapas caóticos, como funções logísticas ou composição de senoidais.

Foram criadas hibridizações do método PSO, pricipalmente usando outras técnicas evolucionárias. O Comprehensive Learning PSO (Liang et al., 2006) é uma técnica que utiliza uma probabilidade P_c que indica a chance de uma partícula ver a posição de outras duas partículas aleatoriamente, caso contrário ela utiliza apenas a sua melhor posição. O Cooperativo PSO (van den Bergh; Engelbrecht, 2004) é uma variação do PSO que otimiza a função objetivo por dimensão e em seguida aplica a mesma técnica para otimização global. O Adaptive PSO (Zhan et al., 2009) é uma técnica que busca adaptar

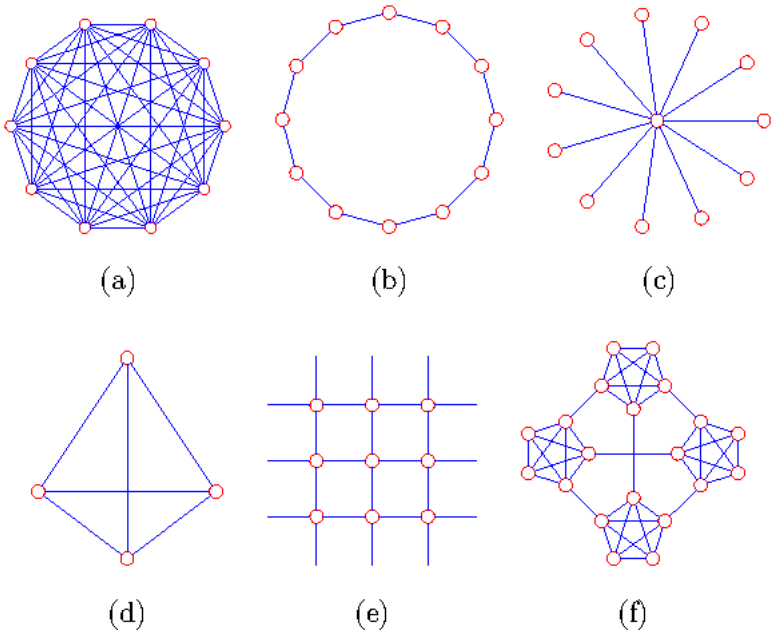

Figura 6: Diagrama de diferentes topologias. a) é a topologia do PSO tradicional ou *gbest*, b) é a topologia Local ou anel, c) é topologia roda, d) é a piramidal, e) é a de Von-Neuman e f) é a topologia 4-clusters.
Fonte: (LIU et al., 2016b)

o comportamento de cada partícula, mais convergente ou mais exploratório, através do fator adaptável. Esse fator adaptável é calculado através das distâncias da posição atual da partícula e a melhor posição do enxame. Assim a depender desse valor, os paramêtros ω, c_1 e c_2 são alterados.

Finalmente, pesquisadores criaram híbridos do PSO com o algoritmo genético tradicional. Mais recentemente, o Genetic Learning PSO (Gong et al., 2016) é uma técnica que realiza as operações de mutação e $cross - over$ através das posições das partículas.

2.2.3.3 CMA-ES

CMA-ES (Covariance Matrix Adaptation Evolution Strategy) é um método evolucionário utilizado para resolução de problemas de otimização não convexos difíceis (HANSEN; OSTERMEIER, 2001). O método é considerado Quasi-Newton com uma abordagem semelhante à de segunda ordem.

O método parte da premissa de aproximar a matriz de Hessiano pela matriz de covariância da função objetivo. Essa abordagem torna o método muito útil para problemas com funções objetivos não separáveis e mal condicionados (HANSEN; OSTERMEIER, 2001). O método também pode ser aplicado para problemas não complexos por não utilizar cálculos de gradientes.

O método se inicia com uma amostragem aleatória do espaço de busca. Em seguida, é calculada a média dos valores de $f(x)$ da função objetiva desses pontos amostrados. Com a média calculada, é possível realizar a conta de atualização da matriz de covariância C e de seus valores de desvios σ. O próximo passo é sortear novamente mais pontos mediante a norma da matriz de covariância para repetir a atualização da matriz de covariância novamente. Esse passo iterativo é repetido até obedecer a algum critério de parada. Conforme essas iterações ocorrem, o espaço de busca pode aumentar ou diminuir proporcionalmente aos resultados da iteração anterior como Figura 7. Assim, o método pode aumentar a o espaço de busca conforme os resultados piorem ou diminuir com a melhora dos resultados obtidos.

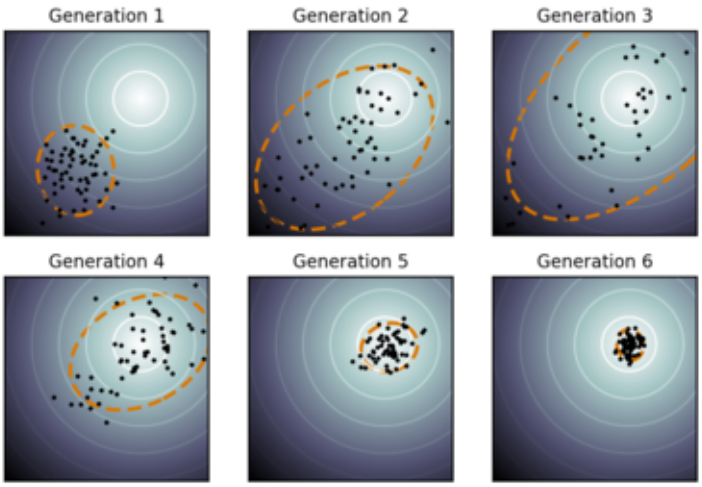

Figura 7: Evolução do método CMA-ES. Conforme o resultado da iteração anterior, o espaço de busca pode ser aumentado e diminuído.
Fonte: (CMA-ES, 2019)

O CMA-ES se destaca por apresentar muitas propriedades de invariância. Ele não é sensível a escala e nem transformações lineares em $f(x)$. Além disso, o algoritmo também é invariante a translações e rotações de $f(x)$ o que o torna muito robusto.

Por fim, CMA-ES é um algoritmo que requisita de poucos parâmetros para funcionar. Os únicos parâmetros de importância são a quantidade de pontos amostrados em cada geração e o valor inicial σ para ser utilizado na formula de atualização da matriz de covariância.

2.2.4 Técnicas Estado da Arte

As técnicas e *frameworks* a seguir são considerados os melhores no contexto de otimização de modelos ML. No meio acadêmico, são os principais baselines para comparação

de algoritmos de tuning.

2.2.4.1 HyperOpt e TPE

HyperOpt é um dos primeiros *frameworks* que foram desenvolvidos para a otimização (BERGSTRA et al., 2015). O HyperOpt necessita de algumas entradas para resolver o problema de otimização:

- Função objetivo com os resultados disponíveis a coleta.
- Espaço de busca de parâmetros .
- Algoritmo de Busca.
- Critério de Parada (geralmente número máximo de iterações).

O HyperOpt consegue solucionar problemas com diferentes distribuições de variáveis (uniforme, normais, categóricos, condicionais e outros). Como algoritmos de busca, ele tem implementados os algoritmos: a busca aleatória e Simulated Annealing citados anteriormente, contudo a principal contribuição é o algoritmo de busca Tree Parzen Estimators (TPE) (BERGSTRA et al., 2011).

O TPE é um algoritmo bayesiano, mas ele apresenta algumas diferenças do algoritmo original bayesiano. O TPE divide os candidatos em bons e ruins (geralmente os bons é o primeiro quartil dos resultados e o restante é ruim). Com esses valores, são calculadas duas curvas *surrogate* de probabilidade do ponto a ser testado pertencer ao grupo bom ou ao grupo ruim.

Com essas duas curvas, o ponto a ser testado é o que maximiza a esperança de ser um ponto bom. Essa diferenciação do algoritmo original agiliza o TPE em relação a busca bayesiana tradicional (BERGSTRA et al., 2011).

O HyperOpt combinado com TPE foi considerado por anos como a principal referência em otimização bayesiana e até hoje é um dos melhores *frameworks* para tuning de modelos ML, com ótimos resultados.

2.2.4.2 SMAC

SMAC é a abreviatura para Sequential Model-based Algorithm Configuration (HUTTER; HOOS; LEYTON-BROWN, 2011). Trata-se de uma variação da busca bayesiana

comum. No SMAC, as funções *surrogates* podem ser geradas por processos Gaussianos, entretanto é mais comum ser utilizados modelos de Random Forest (HUTTER; HOOS; LEYTON-BROWN, 2011).

Random Forest são modelos ML com alta capacidade de generalização (CUTLER; CUTLER; STEVENS, 2011) que são compostos por muitos modelos de Árvores de Classificação ou Regressão. Para SMAC, o Random Forest é utilizado para definir os valores da função custo. A ideia é que o modelo Random Forest é mais veloz para computar os valores da função custo do que coletá-los diretamente. Dessa maneira, o Random Forest consegue fazer uma generalização da função custo que se deseja otimizar.

Além dessa alteração com o Random Forest, o SMAC também utiliza de algoritmos competitivos conhecidos como algoritmos de corrida. No caso, SMAC usa o Randomized Online Aggresive Racing (HUTTER; HOOS; LEYTON-BROWN, 2011). Essa técnica trata-se da geração aleatória de candidatos e em seguida os candidatos são aceitos ou rejeitados de forma "agressiva". Em termos práticos, a seleção dos candidatos é feita de forma a depender apenas dos valores atuais e não de alguma probabilidade como Simulated Annealing.

2.2.4.3 Auto-Sklearn

O pacote Auto-Sklearn é um dos *framework* de maior sucesso em AutoML proposto por (FEURER et al., 2015) em 2015. Ele é um pacote que combina várias técnicas de tuning e construção automática de modelos ML. Ele realiza três grandes etapas Figura 8:

- Meta-Learning.
- Otimização de *Pipelines* com uma busca bayesiana.
- Construção de *Ensemble* de modelos.

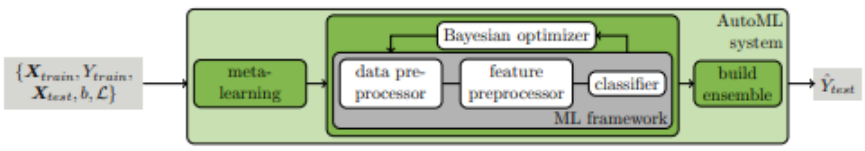

Figura 8: Funcionamento do pacote Auto-Sklearn. Repare como as etapas de Meta-Learning, Otimização de Pipelines e Construção de Ensembles se encaixam no fluxograma do Auto-Sklearn.
Fonte: (FEURER et al., 2015)

A etapa de Meta-Learning do Auto-Sklearn usa de técnicas mais famosas da área. A ideia é transferir o conhecimento em outros *datasets* semelhantes a atual para saber qual é o modelo ML e hiperparâmetros melhores para o base de dados atual (BRAZDIL et al., 2009). O pacote apresenta 140 *datasets* clássicos com mais de 32 estatísticas de representação de banco de dados, como número de linhas, colunas, quantidade de dados faltantes, médias, desvios, entre outros. Dessa forma, é possível verificar o base de dados mais semelhante estatisticamente com a base treino onde o modelo será otimizado.

Com a identificação da base mais semelhante, o Auto-Sklearn se encarrega de verificar os melhores modelos e os melhores hiperparâmetros para aquela base de dados e assim realizar uma inicialização com esses melhores modelos para a etapa de otimização de *Pipelines*.

A ideia central da etapa de Meta-Learning é basicamente encontrar uma inicialização bem promissora para acelerar a otimização da próxima etapa. Assim, seriam necessárias menos iterações para a otimização.

Na etapa de otimização, é realizada uma busca bayesiana dos melhores *Pipelines* e dos melhores hiperparâmetros com elementos da Figura 9. A técnica de bayesiana utilizada é o SMAC já citada anteriormente. Os modelos e processos que são otimizados estão representados na Figura 10.

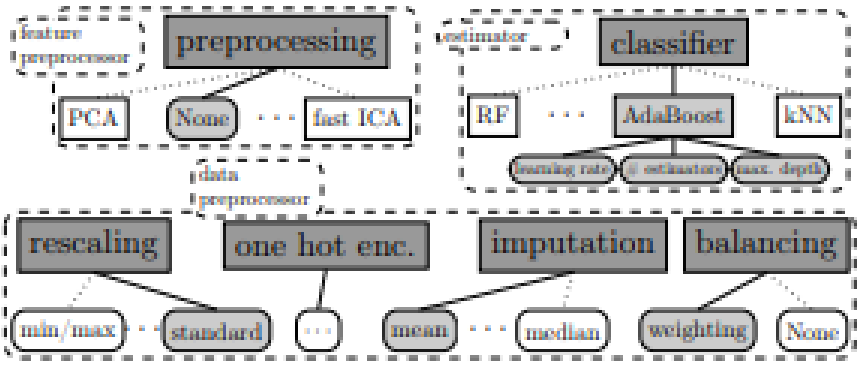

Figura 9: Elementos que compõem os Pipelines do Auto-Sklearn que estão dentro de caixas. Cada elemento apresenta uma classificação diferente no nível superior.
Fonte: (FEURER et al., 2015)

Por fim, o Auto-Sklearn executa uma etapa de criação de *Ensemble* de modelos ML. Para isso, ele realiza técnicas para identificar se há correlações dos erros entre os diferentes modelos (LACOSTE et al., 2014). Caso não haja correlação entre os erros dos modelos, esses modelos são "misturados" através de ponderações simples. Verificam-se as taxas de erros com a técnica de *cross-validation* para identificar os pesos na ponderação do

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
Bernoulli naïve Bayes	2	1 (-)	1 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
Linear Class. (SGD)	10	4 (-)	6 (3)

(a) classification algorithms

name	#λ	cat (cond)	cont (cond)
extreml. rand. trees prepr.	5	2 (-)	3 (-)
fast ICA	4	3 (-)	1 (1)
feature agglomeration	4	3 ()	1 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
PCA	2	1 (-)	1 (-)
polynomial	3	2 (-)	1 (-)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
one-hot encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

(b) preprocessing methods

Figura 10: Algoritmos e pré–processamentos que compõem o Auto-Sklearn
Fonte: (FEURER et al., 2015)

Ensemble.

2.2.4.4 TPOT

TPOT é a abreviatura para *Tree-based Pipeline Optimation Tool* que foi proposto em 2016 por (LE; FU; MOORE, 2020) e ainda estão sendo realizadas melhorias no pacote original. Devido os resultados encontrados, TPOT é considerado atualmente como estado da arte em tuning de modelos ML.

O TPOT é um algoritmo evolucionário, baseado em esquemas de geração de candidatos de forma genética (LANGDON et al., 1970). Inicialmente a técnica se baseia na construção de *Pipelines* de modelo composto por passos separados em:

- **pré–processamento de dados**: são operações realizadas para processar os dados e alcançar melhores resultados, como transformações de escala, transformações de funções (logarítimas por exemplo) e criação de variáveis polinomiais (iteração das variáveis anteriores).
- **Decomposição ou Agregação**: são algoritmos que buscam agregar as variáveis e diminuir a dimensão, como por exemplo PCA.
- **Seleção de Variáveis**: realização de filtros para se retirar variáveis que podem não agregar nos resultados finais.
- **Modelos ML (Classificadores e Regressores)**: Esta é a etapa final dos *Pipelines* que são os modelos ML que serão treinados. Em muitas situações, são realizados

Ensemble dos modelos que são junções de vários modelos ML em busca de melhorar os resultados.

Essas etapas encaixam diretamente no fluxograma de um projeto de modelo ML, Figura 11. Dessa forma, o TPOT além de otimizar modelos ML, servem para a construção e seleção do modelo como um todo.

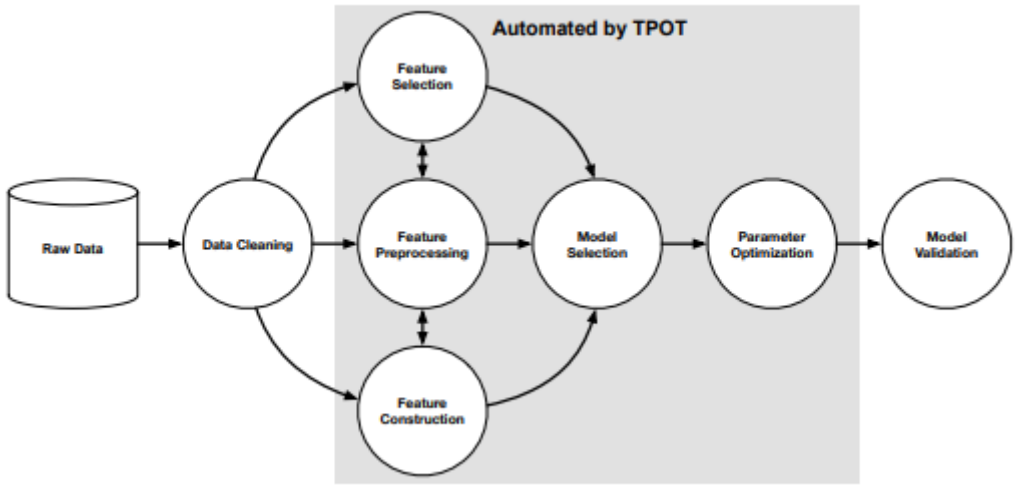

Figura 11: Pacote TPOT como um construtor de modelos ML e onde se encaixam.
Fonte: (LE; FU; MOORE, 2020)

Os *Pipelines* são gerados através de muitas cópias do *dataset* original. Em cada cópia é realizada uma operação diferente de pré–processamento ou de Agregação dos dados. Em seguida, é realizada uma junção dos resultados desses pré–processamentos e Agregações que passa por um bloco de Seleção de Variáveis. Depois de ser realizado um filtro das variáveis, o *dataset* segue para o bloco dos modelos ML que já tem hiperparâmetros definidos. A Figura 12 representa a geração de um *Pipeline*.

Ao final da construção de diversos *Pipelines*, o TPOT realiza uma otimização desses *Pipelines* da mesma forma que um algoritmo genético tradicional (LANGDON et al., 1970): Uma população G de *Pipeline* é gerada, mas apenas P *Pipelines* com os melhores resultados é salva, o restante da geração é descartada. Esse é um passo de seleção natural dos "genes mais bem adaptáveis". Em seguida, os melhores *Pipelines* passam por um processo de recombinação de genes para a geração de uma nova safra de *Pipelines*. Essa recombinação é feita pela técnica de *crossover*, onde os *Pipelines* filhos são composto por partes ou blocos dos *Pipelines* de seus pais. Depois da geração da nova safra de *Pipelines*, essa geração passa novamente pela etapa de seleção natural dos melhores resultados. O processo é repetido até algum critério de parada. Além do TPOT com algoritmo genético

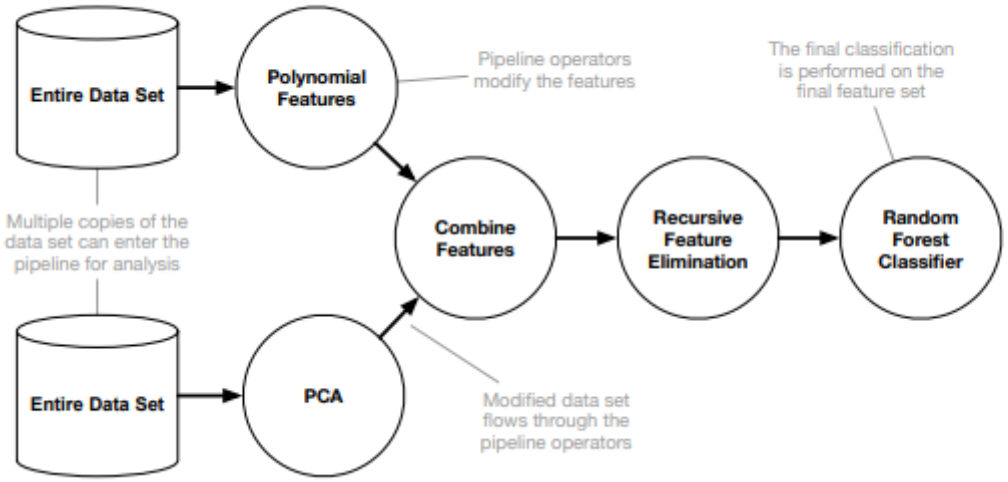

Figura 12: Geração de um Pipeline no algoritmo TPOT. Cada Pipeline é gerado com base em uma cópia do dataset original.
Fonte: (LE; FU; MOORE, 2020)

tradicional, também existem variações com o algoritmo Pareto Genético (OLSON et al., 2016) para acelerar o resultado.

Os resultados de aplicações utilizando TPOT são semelhantes à busca aleatória, contudo o TPOT consegue resultado com *Pipelines* de menor complexidade e mais rapidamente que a busca aleatória. Dessa forma, o TPOT apresenta uma vantagem tanto em tempo de execução como complexidade / facilidade de interpretação (LE; FU; MOORE, 2020).

2.2.4.5 Hyperband

O Hyperband é uma técnica recente e que se destaca por não ser nem uma busca bayesiana e nem uma técnica evolucionária por inteiro (LI et al., 2017). O Hyperband é uma evolução do algoritmo SucessiveHalving (JAMIESON; TALWALKAR, 2015). No SucessiveHalving algoritmo, são gerados B candidatos de soluções do problema, em seguida são separados a metade com os melhores resultados. Depois são geradas uma nova safra de candidata com distribuição semelhante da metade com melhor resultado. O próximo passo é exatamente separar uma nova metade de melhores candidatos. Esse passo é repetido até algum critério de parada.

O Hyperband utiliza essa mesma ideia de B configurações diferentes de hiperparâmetros com n combinações diferentes de hiperparâmetros. O procedimento do algoritmo é gerar combinações dos hiperparâmetros B e depois aplicar alguma busca (geralmente é aplicada

uma busca exaustiva) nessa combinação de hiperparâmetros. Em seguida são descartadas a metade dos piores valores e assim é repetido essa iteração.

O Hyperband lida com *trade-off* B e B/n:

- Com n alto, algumas boas combinações de parâmetros podem apresentar convergência lenta e serem descartados.
- Com B/n alto, combinações ruins podem ter bons resultados iniciais e serem mantidos no algoritmo

Apesar da grande simplicidade do Hyperband, ele demonstrou ter resultados muito competitivos às buscas bayesianas (LI et al., 2017). Nessas situações, a arquitetura da rede neural gera um grande número de parâmetros de camadas e quantidade de neurônios. Com o Hyperband, é possível descartar soluções ruins rapidamente e manter arquiteturas promissoras inicialmente, chegando em bons níveis de resultados com menor quantidade de iterações.

2.3 Técnicas de Validação de Tuning e AutoML

Os processos de tuning e AutoML são muito importantes para se alcançar o melhor desempenho possível em alguma aplicação de ML. A otimização de hiperparâmetros pode ocasionar situações em que a performance do modelo ML na base de treino seja bom, mas, quando aplicado na prática, essa performance se deteriora. Esse fenômeno é muito comum em aplicações ML, sendo conhecido como *overfitting* (CARUANA; LAWRENCE; GILES, 2000), em que é denominado que o modelo ML perde o poder de generalização, prejudicando o desempenho do modelo.

Para combater o *overfitting*, é essencial a utilização de técnicas para validar o poder de generalização do modelo ML. Essas validações servem como forma de avaliação de o quanto o modelo ML pode ser aplicado na prática de forma confiável. Dessa forma, em problemas de tuning ou AutoML, muito mais importante que otimizar o desempenho do modelo em alguma aplicação ML é saber o desempenho desse mesmo modelo na base de validação ou base de teste.

Para os problemas de tuning e AutoML, muitas técnicas de validação podem ser utilizadas:

- **Holdout**: a tradicional técnica de divisão entre uma base de dados de treino e teste, que muitas vezes segue proporções 80%-20% ou 70%-30% de treino-teste, onde o modelo é treinado na base de treino e o desempenho colhido na base de teste. Quando essa técnica é utilizada para tuning, em geral, busca otimizar o desempenho do modelo ML na base de teste.

- **Validação cruzada (cross-validation) ou K-Fold**: a base é dividida em K partes de tamanhos iguais. A cada iteração é realizado testes onde uma dessas divisões é utilizada para treino, enquanto as demais são usadas para treino. Dessa forma, são realizadas K iterações e são colhidos K desempenhos dos modelos. Quando o K-fold é utilizado para tuning, geralmente é otimizada a média do desempenho dos K testes. Essa técnica é mais recomendada para tuning de hiperparâmetros, pois ela consegue avaliar melhor o poder de generalização do modelo.

- **Leave One Out**: basicamente é uma condição especial do K-fold, onde é testado para somente uma observação (linha) da base. Assim, caso a base tenha tamanho N, o K será igual N. Essa técnica é mais indicada para situações onde há um volume muito baixo de dados.

- **Bootstrap**: essa técnica é composta por uma seleção aleatória com reposição dos dados. São selecionados tanto dados para treino e teste para N experimentos. Na maior parte das vezes, o tuning é realizado na média da performance dos N experimentos.

- **Ao longo do tempo**: esse tipo de validação é indicada para dados temporais. Pode ser escolhida uma validação com janela fixa, onde sempre será utilizada uma base de treino com o mesmo intervalo de tempo, ou a janela acumulativa, onde a base de treino acumula mais intervalos de tempo ficando maior.

Como já foi dito, a técnica K-fold é usualmente escolhida para a validação por poder fazer uma boa medição do poder de generalização do modelo (RODRíGUEZ; PéREZ; LOZANO, 2010). Em técnicas de tuning ou AutoML, onde há um risco plausível de ocorrer *overfitting*, o K-fold é o mais utilizado academicamente.

3 PROPOSTA DO ALGORITMO

Nesse capítulo, é discutida a teoria do método do baricentro, explicando suas propriedades que podem ser úteis no contexto de otimização de hiperparâmetros. Com a base no baricentro, o capítulo traz a proposta do algoritmo BarySearch para tuning de modelos ML, além de uma hibridização do método do baricentro com a técnica evolucionária PSO já citada nesse texto.

3.1 Formulação Matemática do Baricentro

O método do baricentro é um algoritmo recursivo livre de derivação: ele não necessita de cálculo de gradientes ou derivadas para o funcionamento. O principal enfoque é encontrar pontos de extrema onde não há uma função objetivo matematicamente definida, conforme é explicado em (CONN; SCHEINBERG; VICENTE, 2009). Em muitas situações de otimização onde não há um conhecimento a priori da função, pode ser utilizados métodos de busca totalmente aleatórios para selecionar os pontos de forma aleatória e testar esses pontos.

Em situações assim totalmente desfavoráveis, o método do baricentro pode ser uma boa alternativa. A formulação do baricentro é uma ponderação dos pontos x selecionados no histórico com pesos que dependem respectivamente dos resultados da função objetivo $f(x)$ (Pait, 2018). A Equação 3.1 representa a fórmula do baricentro para alguma função $f : X \to \mathbb{R}$.

$$\hat{x}_n = \frac{\sum_{i=1}^{n} x_i e^{-\nu f(x_i)}}{\sum_{i=1}^{n} e^{-\nu f(x_i)}}, \tag{3.1}$$

Essa equação representa um centro de massa da função $f(x)$, onde os pesos são ponderações dos resultados da função objetivo com uma equação exponencial. Nessa fórmula, ν é uma constante que indica a velocidade de convergência, quanto maior em módulo essa constante (Pait, 2018), maior será o peso para os melhores resultados.

Conforme indicado na Equação 3.1, o método do baricentro não depende do cálculo de derivadas ou de gradientes. Como já foi dito, o livro (CONN; SCHEINBERG; VICENTE, 2009) explica que métodos que utilizam derivadas de primeira ordem (como gradiente descendente) ou derivadas de segunda ordem (como métodos de Newton) necessitam do cálculo de derivadas para a convergência e busca de pontos ótimos no espaço de busca. Entretanto, o cálculo de derivada nem sempre é possível, seja pelo alto custo computacional de calcular derivadas, seja pela ausência de fórmula matemática definida de $f(x)$.

A Equação 3.1 pode ser reescrita como uma equação recursiva conforme:

$$m_n = m_{n-1} + e^{-\nu f(x_n)} \tag{3.2}$$

$$\hat{x}_n = \frac{1}{m_n}\left(m_{n-1}\hat{x}_{n-1} + e^{-\nu f(x_n)}x_n\right). \tag{3.3}$$

Para esse ponto de vista de algoritmo recursivo, pode ser interessante adicionar uma perturbação aleatória de forma a introduzir um comportamento exploratório z_n no método do baricentro (Pait, 2018). Essa perturbação aleatória z_n é chamada de taxa de curiosidade (Pait, 2018) nos resultado do baricentro $\hat{x}_{n-1}$. A Equação 3.4 demonstra como é introduzida essa perturbação no método.

$$x_n = \hat{x}_{n-1} + z_n. \tag{3.4}$$

Assim, pode se obter uma nova relação:

$$x_n - x_{n-1} = \frac{e^{-\nu f(x_n)}}{m_{n-1} + e^{-\nu f(x_n)}} z_n. \tag{3.5}$$

Apesar de que o passo x_n - x_{n-1} tende a decrescer quando n cresce e o denominador aumenta, não é possível confirmar que os pontos x_n formam uma sequência de Cauchy (Pait, 2018). Assim, a convergência do baricentro pode não ocorrer, mas sim sobre a escolha criteriosa de sequência de teste ou de parâmetros.

Caso $f(x)$ seja diferenciável e z_n tenha uma distribuição gaussiana, o valor esperado de $\Delta x_n = x_n - x_{n-1}$ é proporcional a média do valor do gradiente de $f(x)$ (Pait, 2018).

Esse resultado é interessante, pois o baricentro consegue se diferenciar da busca aleatória, já que os próximos pontos selecionados são através de pontos das direções

de maior taxa de variação de $f(x)$. Dessa forma, o método do baricentro pode combinar busca aleatória e introduzir uma orientação pelo decrescimento ou crescimento da função.

Essas propriedades podem fazer com que o método do baricentro apresente uma velocidade de encontrar pontos promissores mais rapidamente que através de busca totalmente aleatórias. Assim, apesar de não ser possível calcular o gradiente de $f(x)$ quando essa função é desconhecida, o baricentro pode aproximar a direção do gradiente e colocar um comportamento randômico no comportamento do algoritmo recursivo.

Por fim, o baricentro pode ser aplicável para funções ruidosas e não tão bem comportadas, o que pode ser representado por $f(x)+\sigma$. Assumindo que σ é independente de x (o que em geral é uma hipótese razoável em muitas aplicações com ruídos e incertezas), pode se provar que o algoritmo do baricentro não sofre influência do ruído de $f(x)$ caso esse ruído tenha esperança igual a 0 com ruídos que sejam baixos perante a função objetivo $f(x)$ (cerca de 30% alcance do espaço de busca).

Todas essas propriedades tornam o método do baricentro promissor ao uso em aplicações em tuning de modelos ML. Podem ser citados:

- O método do baricentro não necessita da fórmula da função objetivo $f(x)$. Em problemas de tuning, a função $f(x)$ é desconhecida, pois modelos ML são vistos como caixas pretas.
- O método do baricentro nem sempre é convergente o que pode evitar mínimos locais ruins. Em problemas de tuning, a existência de muitos mínimos locais ruins é comum o que complica o problema.
- O baricentro pode combinar a busca aleatória com orientação da direção de $f(x)$ para acelerar a busca. Como já citado, a busca aleatória demonstra ter bons resultados em tuning de ML e poder acelerar essa busca é uma característica muito desejada.
- O baricentro não é influenciado por perturbações baixas a moderadas com média igual a zero. Em muitas aplicações de ML, é frequente encontrar métricas de performance de modelos com ruídos, mas em geral, são ruídos mais bem comportados que poderiam ser contornados pela equação do baricentro (Pait, 2018) .

As próximas seções demonstram desenvolvimento de algoritmos que utilizam o baricentro a fim de aproveitar essas características desejadas nas aplicações de contexto de tuning de ML.

3.2 BarySearch

Como demonstrado anteriormente, o método do baricentro apresenta algumas propriedades interessantes para resolução de problemas de otimização. A equação de baricentro pode ser utilizada para otimizar uma função $f(x)$ com os valores de entrada x, sem necessidade alguma de cálculo de derivadas ou de gradiente. Assim, o método pode encontrar aplicações onde não há uma formulação matemática bem definida da função objetivo $f(x)$.

Nesse trabalho, é proposta a técnica BarySearch. A técnica combina elementos da equação do baricentro com algumas poucas variações para ser utilizada em muitos problemas de otimização como tuning de modelos ML. Para que haja uma boa convergência do baricentro, uma inicialização promissora pode agilizar esse processo. O BarySearch utiliza uma forma de inicialização para melhorar o desempenho do baricentro tradicional.

3.2.1 Inicialização

A inicialização do BarySearch é executada pelo método de amostragem Latin Hypercube Sample (LHS) (MCKAY, 1992). LHS se destaca por realizar uma amostragem aleatória, mas que tenta retirar amostras bem distribuídas do espaço de busca. Essa técnica busca coletar pontos que estejam em todos os subespaços ortogonais do espaço de busca. Esse tipo de inicialização permite que o modelo tenha mais entropia que as forma aleatória padrão, Figura 13.

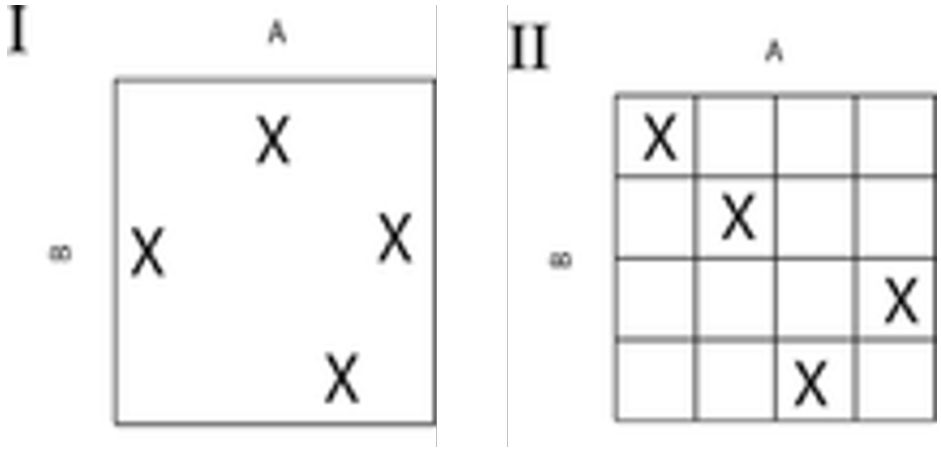

Figura 13: Comparativo entre a amostragem aleatória (I) e amostragem LHS (II) em um espaço de busca de duas dimensões. Repare que a amostragem LHS (II) cria uma restrição de amostrar um ponto por "coluna" e "linha" na divisão de espaço de busca, já na amostragem aleatória (I) não é obrigado seguir essa restrição.
Fonte: (LHS, 2019)

A inicialização com LHS se demonstra promissora por poder fazer um mapeamento

melhor do espaço de busca. Uma amostragem aleatória apresenta certa probabilidade de retirar amostras bem "próximas" ou não tão distribuídas, assim a equação do baricentro pode ficar míope no espaço de busca. Entretanto, o LHS consegue retirar pontos muito mais bem distribuídos (quase equidistantes), o que permite que a equação do baricentro busque e explore por mais regiões.

Ao utilizar a técnica do BarySearch, deve-se escolher um valor de quantidade de pontos para serem utilizados como inicialização. Um critério interessante é separar uma porcentagem da quantidade máxima de iterações. Por exemplo, caso deseja realizar T iterações, podem ser separadas 20% de T para inicilização com LHS e os restante 80% seriam calculadas através da Equação 3.1.

3.2.2 Parâmetros de Velocidade e Curiosidade

A quantidade de pontos de inicialização é o primeiro parâmetro a ser escolhido, contudo não é o único. O parâmetro de velocidade de convergência ν é o parâmetro que controla os pesos que são dados para as parcelas na soma do baricentro. Simplificando, o parâmetro ν é vital para ponderar a importância que cada valor de x terá na fórmula do baricentro. Quando o módulo de ν é mais alto, o peso para os resultados melhores será maior que os demais resultados e assim a fórmula do baricentro fica muito mais convergente ao melhor resultado obtido no histórico. Com módulos de ν menores, o baricentro apresenta um comportamento mais ponderador e menos convergente. Nesse caso, o baricentro faz uma conta mais ponderada e não tem uma convergência tão forte.

O módulo de ν é muito importante para o comportamento do baricentro e por consequência o BarySearch. Módulos altos implicam em um comportamento mais convergente e acelerado, contudo mais sujeito a ficar travado em pontos de mínimos locais. Módulos menores significam um comportamento mais ponderado e lento na convergência, mas mais robusto contra pontos de mínimos locais.

O módulo não é a única parte importante de ν. O sinal de ν indica qual tipo de otimização está sendo realizado. Sinal positivo indicam que a busca é maximizar a função objetivo. Caso sinal seja negativo, logo a operação é minimizar a função objetivo.

Por fim, a escala da função objetivo pode ser vital na escolha do parâmetro ν, pois com funções objetivos de alta escala, valores de ν com módulos altos podem causar erros computacionais no cálculo do expoente. Entretanto, funções com escalas menores, valores de ν baixos podem deixar o comportamento muito lento e não desejável. Existem maneiras

de contornar esses problemas, como padronizar a função objetivo para valores entre 0 a 1, conforme Equação 3.6.

$$f_z(x_N) = \frac{f(x_N) - \min_{i=1}^{N} f(x_i)}{\max_{i=1}^{N} f(x_i) - \min_{i=1}^{N} f(x_i)} \tag{3.6}$$

Dessa forma, a fórmula do baricentro fica invariante tanto a escala da função objetivo, evitando erros computacionais, mas também a torna invariante a transformações lineares da função objetivo, como demonstra em 3.7.

$$\begin{aligned} \frac{f_l(x_N) - \min_{i=1}^{N} f_l(x_i)}{\max_{i=1}^{N} f_l(x_i) - \min_{i=1}^{N} f_l(x_i)} &= \frac{a \cdot f(x_N) + b - a \cdot \min_{i=1}^{N} f(x_i) - b}{a \cdot \max_{i=1}^{N} f(x_i) + b - a \cdot \min_{i=1}^{N} f(x_i) - b} \\ &= \frac{f(x_N) - \min_{i=1}^{N} f(x_i)}{\max_{i=1}^{N} f(x_i) - \min_{i=1}^{N} f(x_i)} \end{aligned} \tag{3.7}$$

Outro parâmetro importante é o grau de curiosidade. O grau de curiosidade é uma perturbação aleatória introduzida na fórmula do baricentro. Geralmente, é escolhida uma distribuição normal para o grau de curiosidade, o que traz as propriedades interessantes. O grau de curiosidade apresenta uma normal de média 0 e um desvio padrão escolhido multiplicado por uma constante, que no caso do BarySearch é igual a metade de da amplitude do espaço de busca em x.

O desvio padrão inserido no grau de curiosidade também tem o poder de alterar comportamento do método do baricentro. Valores baixos de desvio tornam o comportamento mais determinístico e valores alto aumentam a parcela randômica do algoritmo. Um comportamento mais determinístico do BarySearch faz com que o método apresente um comportamento mais similar ao gradiente descendente ou mesmo ao Nelder-Mead , o que significa dizer que o BarySearch faz uma busca mais orientada à variação da função objetivo e logo muito mais convergente e veloz.

Já maiores valores de desvio mudam o comportamento do método do baricentro para algo mais próximo de técnicas evolucionárias. Com a componente randômica mais forte, o BarySearch mostra um comportamento mais aleatório e sujeito a explorar mais pontos. Isso pode diminuir a velocidade, porém pode auxiliar a "fugir" de mínimos locais ruins ou mesmo tornar o valor do baricentro mais abrangente. O BarySearch mais propenso a explorar faz com que o baricentro tenha que ponderar mais pontos e sujeito a realizar mais testes.

Com essas características, é possível realizar recomendações de valores de parâmetros

conforme o contexto da otimização. Problemas mais simples, com funções objetivos mais comportáveis, unimodais e separáveis (Xin Yao; Yong Liu; Guangming Lin, 1999), pode fazer uma inicialização menor com alto módulo de ν e menor desvio do grau de curiosidade para convergência rápida. Já para problemas mais complexos, funções multimodais, não separáveis e não bem condicionadas é desejável um comportamento mais exploratório, ou seja, uma inicialização maior, com módulo de ν menor e um desvio maior do grau de curiosidade.

Assim, o BarySearch realiza o cálculo para T iterações que é número máximo de iterações e ao final retorna o melhor valor alcançado. O Algoritmo 2 apresenta o passo a passo de como é o funcionamento do BarySearch na prática.

Algorithm 2: Algoritmo BarySearch

Input: Limites dos Parâmetros x
Máximo de Iterações T; Velocidade Convergência ν; Taxa Curiosidade σ
Output: Melhor combinação de valores x
Geração de I pontos iniciais e é calculado o baricentro desses pontos;
while *até t ser menor que o número máximo de iterações T* **do**
  $x'_{t+1} = \frac{\sum_{j=1}^{t} x_j * ^{-\nu f(x_j)}}{\sum_{j=1}^{t} ^{-\nu f(x_j)}}$;
  `/* cálculo baricentro */`
  $x_{t+1} = x'_{t+1} + N(0, \sigma)$;
  Calcule $f(x_{t+1})$;
  `/* passo custoso */`
end

3.3 Barycentric Memory Particle Swarm Optimization (BPSO)

Além da construção do método BarySearch para otimização de funções caixas-pretas sem a derivada, esse trabalho também propõem uma modificação de métodos evolucionários que utilizam enxame de partículas, PSO já explicado na seção anterior. O método realiza a otimização de uma função através de partículas cujos os movimentos são influenciados pelo movimento individual e pelo comportamento do enxame.

Usualmente o PSO usa da melhor posição da partícula p, conhecido como *pbest*, e da melhor posição do enxame, chamado de *gbest*, para o cálculo da velocidade da partícula. Muitas outras variações do PSO também utilizam as melhores posições da partícula, a melhor posição do enxame ou a melhor posição das partículas vizinhas a depender da topologia definida. Vários desses métodos alteram a formulação da velocidade, mas em

geral utilizam as melhores posições. O uso da melhor posição ajuda na convergência, mas é possível que perder todo o histórico das posições anteriores possa não ser desejável, já que se perde todos possíveis pontos promissores. Além disso, o uso da melhor posição em contrapondo todo o histórico pode fazer com que o método fique preso em pontos de mínimos locais ruins.

Para contornar esses problemas, esse trabalho traz a proposta de utilizar os o baricentro do histórico das posições anteriores em contrapartida da melhor posição. Usando como exemplo o PSO tradicional, pode-se trocar as melhores posições da partícula e do enxame pelo baricentro da partícula e pelo baricentro do enxame respectivamente, como a Equação 3.8.

$$v_p(t) = \omega \cdot v_p(t) + c_1 \cdot r_1(t) \cdot (bary_p(t) - x_p(t)) \quad + c_2 \cdot r_2(t) \cdot (bary_g(t) - x_p(t)) \quad (3.8)$$

Entretanto, o baricentro pode apresentar o problema da formulação do expoente com o parâmetro ν o que implica que o parâmetro ν ser dependente da escala da função objetivo ou mesmo na ocorrência de erros computacionais. Para evitar isso, é aderido a recomendação de padronizar os valores das funções objetivos entre 0 e 1.

Contudo, a padronização da função objetivo pode ocasionar uma grande diferença entre as piores posições e as melhores. No início do algoritmo, é esperado que ocorra mais posições ruins que ao longo das iterações ficam com resultados melhores. Essa grande diferença entre as funções acarreta que o baricentro não consegue separar pontos com resultados muitos próximos. Assim, esse texto propõe que não se utiliza todos os pontos, mas parte deles mais recentes usando um parâmetro de memória m. Em termos práticos, o baricentro só é calculado nas últimas m iterações do algoritmo.

Sendo assim a formula do baricentro das partículas e do baricentro do enxame como as Equações 3.9 e 3.10.

$$bary_p(t) = \frac{\sum_{i=t-m}^{t} x_p(i) e^{-\nu f_z(x_p(i))}}{\sum_{i=t-m}^{t} e^{-\nu f_z(x_p(i))}} \quad (3.9)$$

$$bary_g(t) = \frac{\sum_{j=1}^{P} \sum_{i=t-m}^{t} x_j(i) e^{-\nu f_z(x_j(i))}}{\sum_{j=1}^{P} \sum_{i=t-m}^{t} e^{-\nu f_z(x_j(i))}} \quad (3.10)$$

Dessa forma o híbrido do PSO com baricentro pode ser descrito conforme o Algoritmo 3.

Algorithm 3: Barycentric PSO

Result: Melhor combinação de x que otimiza $f(x)$
Inicialize os parâmetros ω, c_1, c_2, ν, m;
Inicialize um enxame de tamanho P com partículas com posições iniciais $x_p(0)$ no espaço de busca D;
while *o critério de parada não é satisfeito* **do**
  Compute $f(x_p(t))$ para todas as partículas ;
  Calcule baricentro da partícula p ;
  $bary_p(t) = \frac{\sum_{i=t-m}^{t} x_p(i) e^{-\nu f_z(x_p(i))}}{\sum_{i=t-m}^{t} e^{-\nu f_z(x_p(i))}}$;
  Calcule o baricentro do enxame ;
  $bary_g(t) = \frac{\sum_{j=1}^{P} \sum_{i=t-m}^{t} x_j(i) e^{-\nu f_z(x_j(i))}}{\sum_{j=1}^{P} \sum_{i=t-m}^{t} e^{-\nu f_z(x_j(i))}}$;
  Calcule a velocidade de cada partícula conforme Equação 3.8; Atualize a posição de cada partícula conforme Equação 2.2
end

Finalmente, como é possível construir esse híbrido entre o baricentro e o PSO tradicional, pode-se realizar o mesmo para outras variações do PSO. Para PSO Local, basta calcular o baricentro das partículas da vizinhança, assim como qualquer outra topologia. Para CLPSO, o híbrido com baricentro é usar o baricentro das partículas sorteadas para serem aprendidas na iteração. Para GLPSO, basta usar o baricentro na definição dos genes e assim manter a mesma técnica de *cross-over*. Como é possível concluir, fazer um híbrido com o baricentro é relativamente simples para diferentes variações do PSO.

3.3.1 Análises de Convergência

Inicialmente, analisando a padronização da função objetivo $f(x)$, ela traz características interessantes para aplicação de problemas de otimização. Considerando agora que a função objetivo está em uma amplitude conhecida, o parâmetro de velocidade de convergência ν não é mais dependente da escala de $f(x)$, além disso, o algoritmo fica mais robusto a erros computacionais causados pela conta da exponencial do baricentro.

Assim, o usuário tem uma liberdade muito maior na escolha do valor de ν, sem se preocupar com eventuais erros. A sua escolha pode alterar o comportamento do algoritmo mais convergente com ν maiores em módulo ou mais ponderados com ν menores em módulo.

Entretanto, essa não é a única característica que a padronização traz para o algoritmo. A padronização deixa o algoritmo invariante a qualquer transformação linear de $f(x)$ conforme a equação 3.7.

Sabendo que o algoritmo é invariante a transformações lineares, é essencial saber qual é a convergência da velocidade das partículas e da posição das partículas. Começando com as partículas, pode se fazer de forma semelhante que realizado em (BERGH; ENGELBRECHT, 2006). Usando Equações 2.2 e 2.3, pode se obter a expressão da Equação 3.11.

$$x_p(t+1) = x_p(t)\cdot(1+\omega-c_1\cdot r_1(t)-c_2\cdot r_2(t))-\omega\cdot x_p(t-1)+c_1\cdot r_1(t)\cdot bary_p(t)+c_2\cdot r_2(t)\cdot bary_g(t) \tag{3.11}$$

A Equação 3.11 pode ser representada por um sistema de transição de estados dado pela Equação Matricial 3.12.

$$\begin{bmatrix} x_p(t+1) \\ x_p(t) \\ 1 \end{bmatrix} = \begin{bmatrix} 1+\omega-r_1c_1-r_2c_2 & -\omega & c_1r_1bary_p(t)+c_2r_2bary_g(t) \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p(t) \\ x_p(t-1) \\ 1 \end{bmatrix} \tag{3.12}$$

O sistema da Equação 3.12 apresenta autovetores que são iguais a 1 e escalares definidos por $\omega, c_1, c_2, r_1, r_2$, Equações 3.13 e 3.14.

$$\gamma_1 = \frac{1+\omega-c_1-c2+\epsilon}{2} \tag{3.13}$$

$$\gamma_2 = \frac{1+\omega-c_1-c2-\epsilon}{2} \tag{3.14}$$

Onde a variável ϵ é definida pela Equação 3.15.

$$\epsilon = \sqrt{(1+\omega-c_1-c2)^2-4\omega} \tag{3.15}$$

Esses autovetores indicam que a posição x_p com t tendendo ao infinito, a posição indica uma relação de recorrência Equação 3.16.

$$x_p(t) = k_1 + k_2\cdot\gamma_1^t + k_3\cdot\gamma_2^t \tag{3.16}$$

Onde k_1, k_2, k_3 são constantes definidas por $\omega, c_1, c_2, r_1, r_2$. Com a Equação 3.16 pode

se calcular a convergência tanto de x_p como v_p. Para v_p, pode se mostrar que a há uma relação de recorrência substituindo a Equação 3.16 em 2.2.

$$v_p(t) = h_1 \cdot \alpha_1^t + h_2 \cdot \alpha_2^t \tag{3.17}$$

Com $\alpha_1 = k_2 \cdot (1 - 1/\gamma_1)$ e $\alpha_2 = k_3 \cdot (1 - 1/\gamma_2)$. Dessa forma, colocando t tendendo a infinito, obtêm-se duas situações:

- Caso os módulos de α_1, α_2 tenham módulos menores que 1, v_p tende a 0.
- Caso o módulos de α_1 ou módulo de α_2 igual 1, v_p tende a um valor constante igual a h_1, h_2 ou $h_1 + h_2$ caso os dois módulos sejam iguais a 1 .

Pode se concluir que a convergência de v_p independe do cálculo do baricentro, depende apenas das constantes ω, c_1, c_2 que são definidas inicialmente. Logo, a hibridização com o baricentro não altera a convergência de v_p.

Agora, analisando x_p, calculando o limite da Equação 3.16, pode se obter:

$$\lim_{t \to \infty} x(t) = \frac{c_1}{c_1 + c_2} bary_p + \frac{c_2}{c_1 + c_2} bary_g \tag{3.18}$$

Logo, a posição de cada partícula se aproxima de uma ponderação entre os baricentros da própria partícula e do enxame. Essa modificação pode ser interessante principalmente em funções multimodais, pois o baricentro consegue ponderar bem os pontos locais e evitar de se travar ou oscilar em pontos de mínimos locais ruins, conforme a Figura 14 demonstra.

Por fim, o parâmetro ν também apresenta uma característica importante para o híbrido entre o PSO e baricentro. Em altas dimensões, os algoritmos tradicionais costumam sofrer de problemas de maldição de dimensionalidade: os vetores em alta dimensões ficam mais próximos e as noções de distâncias ficam distorcidas, além de aumentar os custos computacionais de cálculo de movimentação do enxame. Nas altas dimensões, uma grande variedade de vetores de entrada x apresentam valores de função objetivo $f(x)$ muito similiar, o que dificulta a otimização.

O parâmetro ν é importante em situações de grandes dimensões, pois ele pode alterar o comportamento do algoritmo para melhor ponderar as posições com resultados de função objetivo semelhantes . Dessa maneira, o parâmetro ν fazer que o baricentro pondere as posições da partícula do enxame e chegar em resultados mais promissores que a utilização

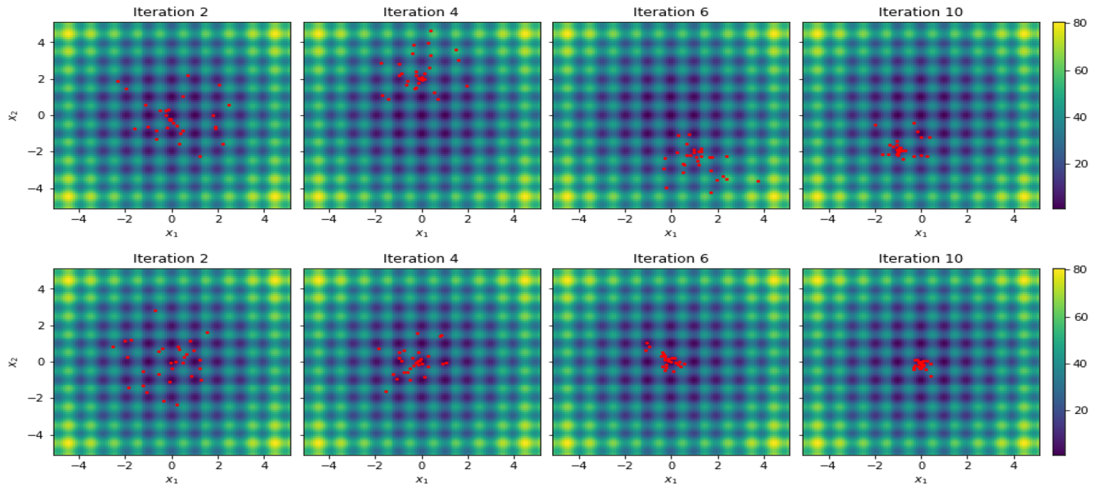

Figura 14: Comparativo entre PSO tradicional (cima) e PSO híbrido com o baricentro (baixo). Repare que o PSO original apresenta comportamento oscilatório por causa dos pontos ruins de mínimos locais, já o híbrido com o baricentro pondera melhor os mínimos e apresenta comportamento mais comportado e tendendo ao mínimo global.
Fonte: Autor

da melhor posição. Essa ponderação auxilia que o algoritmo não fique travado em pontos de mínimos locais ruins como é comum em variações tradicionais que utilizam a melhor posição da partícula e do enxame para cálculo da posição.

Nesse trabalho, foi identificado empiricamente uma relação matemática entre a dimensão do espaço de busca D e o parâmetro de convergência do baricentro ν dada pela Equação 3.19.

$$\nu = \frac{-2000}{D^2} \tag{3.19}$$

Essas propriedades matemáticas demonstram que a hibridização do PSO e suas variações com o baricentro pode trazer boas contribuições para o algoritmo, principalmente em situações mais complexas: funções objetivos multimodais, não separáveis, não bem comportadas e em altas dimensões de x.

3.4 BarySearch e BPSO para Resolução do Problema AutoML

Como visto no capítulo anterior, o problema de AutoML é uma evolução do problema de tuning de modelos ML. O problema de AutoML, geralmente chamado de CASH, necessita de otimização de muitos parâmetros e de diferentes naturezas e escalas.

O problema de CASH inclui não só otimizar os hiperparâmetros de modelos ML (tuning), mas também inclui a seleção do melhor modelo ML, o melhor pré–processamento e a melhor seleção/agregação de variáveis (FEURER et al., 2015). Isso implica que o problema de CASH necessariamente está otimizando parâmetros que são condicionais, pois a otimização de algum parâmetro só ocorrerá se houver a escolha de outro, como o número de neurônios só será otimizado caso for escolhido rede neural, ao contrário não será realizado a otimização. Além disso, parte desses parâmetros podem ser categóricos e ainda existem parâmetros contínuos e discretos com diferentes distribuições e escalas. Otimizar esses modelos com tantas restrições e variações é considerado um problema muito complexo e não tem uma resolução matemática bem definida.

Como é conhecido, os algoritmos PSO não são construídos para otimizar parâmetros categóricos e muito menos condicionais, até mesmo para parâmetros discretos, o PSO não funciona bem naturalmente. O BarySearch usando o cálculo do baricentro também enfrenta a mesma dificuldade, já que não foi desenvolvido para parâmetros dessa natureza categórica ou condicional.

O que é apresentado nessa seção é uma abordagem para adaptar tanto PSO, como Barysearch e até mesmo o híbrido entre PSO e baricentro, o BPSO. Essa abordagem é a mesma utilizada em para resolução dos problemas de AutoML (ESCALANTE; MONTES; SUCAR, 2010). A abordagem transforma as posições em vetores com muitas dimensões as quais cada dimensão tem um significado. Algumas dimensões do vetor significam qual tipo de pré–processamento é utilizado, outra dimensão traz qual é seleção de variável utilizada e por fim ainda há a dimensão de seleção do modelo ML. Além dessas dimensões, existem também dimensões que são proporcionais aos valores dos hiperparâmetros do modelo ML.

Todos as etapas escolhidas de pré–processamento, seleção de variável e modelo ML está apresentada na Tabela 1 e também estão apresentados todos os hiperparâmetros dos modelos ML e cada natureza desse hiperparâmetro (contínuo, discreto, categórico).

O vetor de posição de partícula ou do cálculo do baricentro apresenta as seguintes dimensões: 1 para escolha de préprocessamento, 1 para seleção de variável e 1 para seleção de modelo ML, além de mais Y dimensões para os hiperparâmetros do modelo ML. Para agrupar os hiperparâmetros em cada dimensão do vetor X, foram analisadas as naturezas de cada hiperparâmetro: se o hiperparâmetro é categórico, discreto ou contínuo; quantos níveis o parâmetro categórico tem; qual é o tamanho de espaço de busca. Esse cálculo foi denominado como cálculo de “sinergia” entre os hiperparâmetros. Dessa forma, puderam ser agrupados hiperparâmetros como o peso de classse que é categórico com dois níveis

Tabela 1: Tipo e Espaço de busca de hiperparâmetros para o problema de AutoML.

Parâmetro	Tipo	Espaço de Busca
tol	contínuo	10^{-6} a 10^{-3}
norma	categórico	l2 ou l1
peso	categórico	sem ou balanceado
k vizinhos	discreto	3 a 50
P	discreto	1 a 5
max dep	discreto	3 a 10
min split	discreto	5 a 50
min child	discreto	1 a 50
critério (árvores)	categórico	entropia ou gini
max feat	categórico	sqrt, log2 ou sem
C	discreto	1 a 10
alpha	contínuo	10^{-4} a 10^{-3}
n_est	discreto	10 a 100
learning rate	contínuo	10^{-3} a 0,05
kernel	categórico	linear ou RBF (parâmetros padrões de cada kernel)

Tabela 2: Mapeamento dos hiperparâmetros no vetor de entrada X.

Algoritmo	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
Regressão Logística	tol							norma	peso
KNN			P			k			peso
SGD	tol				alpha			norma	
Naive-Bayes									
Árvore de Decisão		max dep	min sp	min child			crit	max feat	peso
SVM	tol		C					kernel	
Random Forest		max dep	min sp			n est	crit	max feat	peso
Gradient Boosting	tol	max dep	min sp		lr	n est	crit	max feat	

(nenhum peso ou peso balanceado) presente em muitos algoritmos em uma dimensão, ou learning rate (taxa de aprendizagem) com parâmetro de regularização alpha que são contínuos com espaços de busca semelhantes. A Tabela 2 mostra o mapeamento das dimensões do vetor x de entrada com o respectivo hiperparâmetro do modelo.

Todas as dimensões de x estão em um alcance de 0 a 1. Para parâmetros contínuos e discretos, o valor 0 da dimensão significa que será escolhido o valor mínimo no espaço de busca para esse hiperparâmetro, já 1 será o valor máximo. Para as variáveis categóricas, o valor da dimensão será dividido uniformemente com o número de níveis possíveis desse hiperparâmetro. Por exemplo, caso do peso da classe com 2 níveis, caso o valor de X_y seja menor que 0,5, será aplicado um modelo sem peso nas classes, caso seja maior, será aplicado um modelo com peso de classe balanceado. Da mesma forma para outros casos de parâmetros categóricos.

Para a representação do vetor x foi utilizado um vetor de espaço de busca com 9 di-

mensões para os hiperparâmetros dos modelos ML conforme a Tabela 2 e mais 1 parâmetro de aplicação de pré–processamento simples de padronização de dados e 2 parâmetros de seleção de variáveis, 1 para a aplicação ou não de algoritmos de seleção e outro para definir o limiar das variáveis a serem filtradas.

Esse tipo de abordagem permite que algoritmos como PSO, BarySearch e BPSO sejam adaptados facilmente para parâmetros categóricos e/ou condicionais. Entretanto, essa abordagem traz desvantagens principalmente se tratando na questão de tratar os hiperparâmetros e as etapas de pré–processamento e seleção de variáveis como independentes do modelo ML, o que nem sempre é verdade.

Finalmente, essa abordagem apresenta uma clara facilidade de ser simples de ser implementada e com baixo custo computacional. Assim, as aplicações de problemas de AutoML com essa abordagem tendem a serem mais rápidos por iteração que os métodos tradicionais, porém o desempenho pode não ser comparável aos métodos mais clássicos de AutoML.

O próximo capítulo trata de testes práticos da performance dos nossos algoritmos perante aos métodos mais clássicos de tuning e AutoML.

4 TESTES E VALIDAÇÃO DO BARYSEARCH E BPSO

No capítulo a seguir, são apresentados testes de desempenho dos algoritmos propostos com as técnicas já sugeridas e largamente utilizadas nas literaturas. Os testes realizados seguem os principais padrões de comparações de desempenho em tuning de modelos ML, seleção de modelos e otimização usados na academia.

4.1 Testes com Função Benchmark com BPSO

Como descrito na seção anterior, a hibridização do PSO com o método do baricentro pode apresentar propriedades interessantes para resolução de problemas mais complexos, principalmente se tratando de funções multimodais, pouco comportadas, não separáveis e altas dimensões de entrada. Logo, essa seção foca em testes para demonstrar a melhora dessa hibridização perante às variantes de PSO tradicionais.

Para a realização dessa comparação, foram selecionadas funções de benchmark de otimização. Essas funções de benchmark são utilizadas para simular diversas situações e poder testar a robustez do algoritmo de otimização. Essas funções são caracterizadas conforme as seguintes propriedades:

- Unimodais ou multimodais: funções unimodais são funções que não apresentam vários mínimos locais, mas um mínimo global que se destaca. Multimodais são funções com muitos pontos de mínimos locais que dificultam o problema de algoritmos de otimização.
- Bem ou mal comportadas: funções bem comportadas são funções com formato e curvas totalmente deriváveis em todos os pontos. Já mal comportadas são deriváveis em apenas alguns pontos ou apresentam oscilações.
- Separáveis ou não: uma função é dita separável se é possível encontrar o ponto de

mínimo global minimizando a função objetivo por cada dimensão separadamente, caso contrária, a função é dita não separável.

Essas propriedades são interessantes, pois conseguem simular e testar os algoritmos em várias situações que são comuns nas aplicações de otimização. Além disso, cada função pode trazer perspectivas de cada algoritmo e ser utilizada como identificadores de pontos fortes ou fracos dos algoritmos. Nas próximas seções, são explicadas cada uma dessas funções.

A primeira função testada é a função Esfera dada pela Equação 4.1. Essa é a função mais simples não linear que um algoritmo de otimização pode resolver. Ela é uma função unimodal, muito bem comportada e totalmente separável, todas características que a tornam um problema fácil de ser solucionado. A principal característica de se usar essa função é o fato dela ser uma medida da velocidade de convergência do algoritmo.

$$f_1(x) = \sum_{i=1}^{D} (x_i - 0.2)^2 \tag{4.1}$$

A segunda função testada é a função de Rosenbrock definida pela Equação 4.2. Essa função também é unimodal, bem comportada e separável, mas apresenta curvas mais íngremes e mais complexas para encontrar o ponto ótimo em comparação à função Esfera. Essa função pode indicar o quanto o algoritmo pode sofrer a sensibilidade e alteração de direção, pois a função altera muito o valor em relação a uma dimensão em comparação a outras.

$$f_2(x) = \sum_{i=1}^{D-1} 100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \tag{4.2}$$

A próxima Equação 4.3 é a equação da definição da última função unimodal utilizada conhecida como função de diferentes expoentes. Essa função de diferentes expoentes é conhecida pela complexidade em otimizar, pois ela apresenta o ótimo em uma região onde a sensibilidade aumenta muito a cada alteração da entrada em x. Em outras palavras, muitas regiões são como platôs em que não há grandes alterações em $f_3(X)$ e só uma pequena região é que há uma alta queda na função objetivo para o ponto ótimo. Essa função é um bom teste para ver a robustez do algoritmo de otimização frente a platôs.

$$f_3(x) = \sqrt{\sum_{i=1}^{D} |x|^{2+4\frac{i-1}{D-1}}} \tag{4.3}$$

Depois das funções unimodais mais simples, também foram testadas diversas funções multimodais, todas explicitadas a seguir.

$$f_4(x) = -20e^{-0.2\cdot\sqrt{\frac{1}{D}\cdot\sum_{i=1}^{D} x_i^2}} - e^{\frac{1}{D}\cdot\sum_{i=1}^{D}\cos(2\pi\cdot x_i)} + 20 + e \tag{4.4}$$

$$f_5(x) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{4.5}$$

$$\begin{aligned} f_6(x) = &\sum_{i=1}^{D}\sum_{k=0}^{20}[0.5^k \cdot \cos(2\pi 3^k(x_i + 0.5))] \\ &- \sum_{k=0}^{20}[0.5^k \cdot \cos(2\pi 3^k \cdot 0.5)] \end{aligned} \tag{4.6}$$

$$f_7(x) = \sum_{i=1}^{D} x_i^2 - 10\cos(2\pi \cdot x_i) + 10 \tag{4.7}$$

$$f_8(x) = 418.9829 \cdot D - \sum_{i=1}^{D} x_i \cdot \sin(|x_i|^{\frac{1}{2}}) \tag{4.8}$$

A função $f_3(x)$ da Equação 4.4 trata-se da função de Ackley, essa função e multimodal (com alguns mínimos locais), mas que apresenta bom comportamento. O principal desafio dessa função é evitar os pontos de mínimos locais para convergir. A função $f_4(x)$ da Equação 4.5 é a equação de Griewank. A função de Griewank é multimodal, mas que apresenta links entre as dimensões da entrada de x que complicam a otimização, além disso, também não é uma função bem comportada.

A Equação 4.6 é a função multimodal de Weierstrass. Essa é considerada uma função complexa que apesar de contínua, apresenta muitos pontos não deriváveis. A penúltima Equação 4.7 é a função de Rastrigin. Essa função é clássica para medir robustez de algoritmos de otimização, pois é uma função com tantos pontos de mínimos locais ruins que é complexo o algoritmo não travar nessas regiões e, portanto, não convergir para o ponto ótimo global. Finalmente, a Equação 4.8 é a função de Schwefel que apresenta um profundo mínimo local que pode atrapalhar a convergência dos algoritmos para o ponto mínimo global.

Além das funções multimodais, era desejável realizar testes com funções não se-

paráveis. Para criar funções não separáveis, foi gerada uma matriz de rotação que serve para rotacionar as funções de modo que o resultado da função não pode ser calculado separadamente de cada dimensão e assim retirando a propriedade de separabilidade das funções. A matriz de rotação é dada pela Equação 4.10.

$$M = \begin{bmatrix} m_{11} & m_{12} & ... & m_{1D} \\ ... & ... & ... & ... \\ m_{D1} & m_{D2} & ... & m_{DD} \end{bmatrix} \tag{4.9}$$

$$y = M \cdot x \tag{4.10}$$

Dessa forma, cada dimensão de y é uma combinação linear das dimensões de x, retirando a separabilidade das funções. Foi utilizada a matriz de rotação em todas as funções multimodais, o que implica que foram testadas no total 13 funções de benchmark (3 funções unimodais, 5 funções multimodais e mais 5 funções multimodais rotacionadas).

Para validar a melhora da performance da hibridização das variantes do PSO com o método do baricentro, são escolhidas 9 variações diferentes do PSO para a comparação. As variações selecionadas são:

- Global PSO (PSO) (Kennedy; Eberhart, 1995)
- Local PSO (LPSO) (SUGANTHAN, 1999)
- Wheel (Roda) PSO (LIU et al., 2016a)
- Unified PSO (UPSO) (PARSOPOULOS; VRAHATIS, 2004)
- Accelerated PSO (APSO) (YANG; DEB; FONG, 2012)
- Chaotic PSO (PSO) (HEFNY; AZAB, 2010)
- Comprehensive Learning PSO (CLPSO) (Liang et al., 2006)
- Adaptive PSO (AdpPSO) (Zhan et al., 2009)
- Genetic Learning PSO (GLPSO) (Gong et al., 2016)

Em todas essas variações, são fixados os demais parâmetros de uso, com o intuito de eventuais diferenças de desempenho ocorrerem apenas por causa das modificações com o

Tabela 3: Parâmetros utilizados nos algoritmos PSO nos testes realizados.

Algorithm	Parameters
Global PSO	$c_1 = 0.5$; c_2=0.3; $\omega = 0.4$
Local PSO	$c_1 = 0.5$; c_2=0.3; $\omega = 0.4$; k =3
Unified PSO	$c_1 = 0.5$; c_2=0.3; $\omega = 0.4$; k=3; u =0.3
Accelerated PSO	$c = 0.4$; β=0.8
Chaotic PSO	$c = 0.4$; β=0.8
CL PSO	$c = 0.5$; ω=0.4; $P_c = 0.5$
Wheel PSO	$c_1 = 0.5$; c_2=0.3; $\omega = 0.4$
Adaptive PSO	Depende da iteração t
GL PSO	$c = 0.5$; ω=0.4; $Prob_cross = 0.5$

Tabela 4: Resultado das Funções Benchmark: A tabela contém o número de testes em que o resultado é estatisticamente igual / tradicional é estatisticamente superior / hibrído com baricentro é melhor estatitiscamente nas 13 funções testadas com **5** gerações.

Algorithm	**D=10**	**D=30**	**D=50**	**D=100**
PSO	5/0/8	0/2/11	0/2/11	0/2/11
LPSO	11/0/2	2/0/11	2/0/11	2/0/11
UPSO	12/0/0	3/0/9	3/0/10	3/0/11
WheelPSO	7/0/6	2/0/11	1/1/11	1/1/11
APSO	11/0/2	3/1/9	3/1/9	3/1/9
ChaosPSO	12/1/0	12/0/1	11/0/2	11/0/2
CLPSO	13/0/0	6/0/7	3/0/10	2/0/11
AdpPSO	7/0/6	2/0/11	2/0/11	2/0/11
GLPSO	10/0/3	2/0/11	2/0/11	2/0/11
Total	88/1/28	32/3/82	27/4/86	27/4/86

híbrido do PSO como baricentro. A Tabela 3 demonstra os parâmetros que são utilizados para os testes.

Além destes parâmetros, para todas as hibridizações com o baricentro, foram fixados parâmetros ν definido pela Equação 3.19 e com o parâmetro de memória $m = 2$, além de realizar os testes com todas as 13 funções com dimensão D de entrada iguais a 10, 30, 50 e 100, com o número de iterações 5, 10, 25 e 30, fixando um enxame com 10 partículas.

Em todos os testes com alguma variação de PSO, com alguma dimensão D e número de iterações t, são realizados cada teste com 30 repetições com inicializações diferentes.

Nos Anexos A, é possível ver todos os gráficos com os resultados em box-plot dos testes das funções benchmark. Nessa seção, estão disponibilizados as Tabelas de comparação de desempenho entre as variantes do PSO usando a melhor posição contra os híbridos dessas variações com o método do baricentro. Nessas tabelas, é realizado um testes estatístico de Willcoxon Signed Rank com nível de confiança $\alpha = 5\%$ para ver se há diferença estatística entre o desempenho das variações tradicionais com os híbridos do PSO.

Tabela 5: Resultado das Funções Benchmark: A tabela contém o número de testes em que o resultado é estatisticamente igual / tradicional é estatisticamente superior / hibrído com baricentro é melhor estatitiscamente nas 13 funções testadas com **10** gerações.

Algorithm	**D=10**	**D=30**	**D=50**	**D=100**
PSO	7/2/4	0/2/11	0/2/11	0/2/11
LPSO	9/0/4	3/1/9	2/0/11	2/0/11
UPSO	11/0/2	2/0/11	2/0/11	2/0/11
WheelPSO	9/1/3	0/2/11	1/1/11	0/2/11
APSO	8/0/5	2/2/9	2/2/9	2/2/9
ChaosPSO	11/0/2	13/0/0	11/0/2	11/1/1
CLPSO	11/0/2	7/0/6	4/0/9	3/0/10
AdpPSO	8/0/5	2/0/11	2/0/11	2/1/10
GLPSO	11/0/2	1/1/11	2/0/11	2/0/11
Total	85/3/29	30/8/79	26/5/86	23/8/86

Tabela 6: Resultado das Funções Benchmark: A tabela contém o número de testes em que o resultado é estatisticamente igual / tradicional é estatisticamente superior / hibrído com baricentro é melhor estatitiscamente nas 13 funções testadas com **15** gerações.

Algorithm	**D=10**	**D=30**	**D=50**	**D=100**
PSO	8/1/4	0/2/11	0/2/11	0/2/11
LPSO	11/1/1	1/2/10	1/1/11	1/1/11
UPSO	12/0/1	2/1/10	2/0/11	1/1/11
WheelPSO	8/0/5	0/2/11	0/2/11	0/2/11
APSO	10/1/2	3/1/9	2/2/9	2/2/9
ChaosPSO	13/0/0	8/1/4	8/1/4	9/0/4
CLPSO	11/2/0	9/0/4	3/0/10	2/0/11
AdpPSO	8/0/5	2/0/11	2/1/11	2/1/11
GLPSO	12/0/1	2/0/11	2/0/11	2/0/11
Total	93/5/19	27/9/81	19/9/89	18/9/90

Tabela 7: Resultado das Funções Benchmark: A tabela contém o número de testes em que o resultado é estatisticamente igual / tradicional é estatisticamente superior / hibrído com baricentro é melhor estatitiscamente nas 13 funções testadas com **25** gerações.

Algorithm	**D=10**	**D=30**	**D=50**	**D=100**
PSO	9/0/4	0/2/11	0/2/11	0/2/11
LPSO	11/2/0	4/0/9	3/1/9	1/1/11
UPSO	13/0/0	4/0/9	3/0/10	2/0/11
WheelPSO	10/0/3	0/2/11	0/2/11	0/2/11
APSO	8/1/4	2/2/9	3/1/9	2/2/9
ChaosPSO	13/0/0	5/1/7	3/1/9	5/1/7
CLPSO	11/2/0	10/0/3	3/0/10	5/0/8
AdpPSO	7/0/6	2/0/11	2/0/11	2/0/11
GLPSO	12/0/1	2/0/11	2/0/11	2/0/11
Total	94/5/18	29/7/81	19/7/91	19/8/90

Tabela 8: Resultado das Funções Benchmark: A tabela contém o número de testes em que o resultado é estatisticamente igual / tradicional é estatisticamente superior / hibrído com baricentro é melhor estatitiscamente nas 13 funções testadas com **30** gerações.

Algorithm	**D=10**	**D=30**	**D=50**	**D=100**
PSO	6/2/5	0/2/11	0/2/11	0/2/11
LPSO	11/1/1	2/2/9	2/2/9	3/0/10
UPSO	13/0/0	2/1/10	3/0/10	2/0/11
WheelPSO	9/1/3	0/2/11	0/2/11	0/2/11
APSO	8/2/3	2/2/9	0/4/9	3/1/9
ChaosPSO	12/1/0	6/1/6	5/0/8	3/2/8
CLPSO	11/2/0	11/0/2	6/0/7	6/1/6
AdpPSO	6/0/7	2/0/11	1/0/11	1/1/11
GLPSO	13/0/0	3/0/10	2/0/11	1/0/12
Total	89/9/19	28/10/79	20/10/87	19/9/89

Repare que em todas as Tabelas anteriores de resultados demonstram que em altas dimensões, o híbrido das variantes PSO com o baricentro tiveram melhores desempenho que as variações tradicionais. Nas baixas dimensões ($D = 5$), é possível observar que as variações de PSO tradicionais e os híbridos apresentam desempenhos estatisticamente iguais. Também é fácil notar que o número de gerações ou iterações do algoritmo pouco influenciou na diferença do desempenho comparativo entre os tradicionais e os híbridos. Dessa forma, pode ser dito que os algoritmos híbridos do PSO com baricentro obtêm resultados superiores que os algoritmos tradicionais conforme a dimensão do espaço de busca é maior.

Esse resultado é muito promissor e traz propriedades interessantes para aplicações de otimização. Problemas com altas dimensões são considerados mais complexos de serem solucionados e utilizando os híbridos do PSO, é provável ter um resultado melhor na otimização. Sabendo que muitas aplicações de otimização necessitam trabalhar em espaços de grandes dimensões, como o tuning dos modelos ML.

4.2 Testes de Classificação Benchmark

Nesta seção, estão dispostos testes de classificação de modelos ML. Para a realização destes testes, foram adotadas as principais práticas de tuning de modelos ML. Uma base de treino é utilizada para a realização de um cross-validation de $K = 5$ folds para encontrar o melhor conjunto de hiperparâmetros que otimizam os resultados da validação da classificação. Depois, com os valores de hiperparâmetros escolhidos, o modelo ML é testado em uma base de teste para verificar o poder de generalização. Em todos algoritmos foram utilizadas configurações padrões e para BarySearch foi definido $\nu = -50$ e com 10%

Tabela 9: Espaço de busca dos hiperparâmetros a serem tunados nos testes.

Modelo	Parâmetros	Espaço de Busca
LightGBM	Número de folhas	30 a 150
	Número de estimadores	100 a 1000
	Learning Rate	0.01 a 0.2
	Mínimo de observações no nó fiho	20 a 500
	Regularização α	0 a 1
	Regularização λ	0 a 1
	Colsample	0.6 to 1
MLP	Regularização α	2 a 100
	Learning Rate	0.001 a 0.01
	β_1 do otimizador ADAM	0.6 a 0.99
	β_1 do otimizador ADAM	0.85 a 0.999
	ϵ do otimizador ADAM	10^-10 a 10^-5
	Quantidade de Neurônios na primeira camada	2 a 20
	Quantidade de Neurônios na segunda camada	2 a 20

de iterações para inicialização. A Figura representa o fluxograma dos testes realizados.

Para esses testes, foram selecionadas algumas bases benchmark de problemas de classificação que estão disponíveis nos *site* da UCI (DUA; GRAFF, 2017). Essas bases são de problemas clássicos de ML que foram muito pesquisados pela academia.

O principal objetivo desse teste é verificar se o desempenho do BarySearch é competitivo frente às técnicas tradicionais de tuning de ML e às técnicas consideradas estado da arte. Para as técnicas tradicionais foram escolhidas: busca aleatória (BERGSTRA et al., 2011), Simulated Annealing (KIRKPATRICK; GELATT; VECCHI, 1983), CMA-ES (HANSEN; OSTERMEIER, 2001) e TPE do Hyperopt (BERGSTRA et al., 2015). Para as técnicas estado da arte, estão representando o TPOT (LE; FU; MOORE, 2020) e Hyperband (LI et al., 2017).

Existem diversos modelos ML que poderiam ser utilizados para o teste de tuning, mas foram escolhidos modelos com muitos hiperparâmetros e que são complexos de serem otimizados: Redes Neurais com arquitetura MLP de duas camadas escondidas e LightGBM, uma técnica de boosting. Para ambos os modelos, são otimizados 7 valores de hiperparâmetros conforme a Tabela 9. Os hiperparâmetros selecionados são todos parâmetros numéricos (contínuos ou discretos) devido às limitações iniciais do BarySearch em trabalhar com parâmetros categóricos.

Para os testes, cada algoritmo foi compilado com t iterações de 10, 25, 50 repetidos por 30 vezes com inicializações diferentes e a métrica de desempenho dos modelos ML selecionada é o erro médio de classificação (a métrica mais utilizada em problemas de tuning para classificação). A Tabela 10 dispõem os resultados desses testes.

Os resultados demonstram que o BarySearch apresenta uma perfomance melhor ou pelo menos competitiva em relação aos demais algoritmos. Na maior parte das bases e iterações, o BarySearch é o melhor resultado e nas demais ele é segundo melhor resultado. Isso demonstra o quanto a técnica pode ser competitiva e apresentar um desempenho bom, considerando o contexto de otimizar parâmetros numéricos. Devido a simplicidade do BarySearch, ele requer menos capacidade computacional que outros métodos e isso pode ser uma vantagem importante para o algoritmo, considerando que o BarySearch consegue bons resultados com baixo número de iterações.

4.3 Testes com Modelos de Aprendizagem Profunda

Muitas aplicações utilizam Aprendizado Profundo, onde se destaca muito para problemas com dados não estruturados (imagens, textos, áudios, vídeos e etc). Modelos de Aprendizado Profundo são considerados estado da arte nessas aplicações com bases de dados não estruturados e alcançam desempenhos muito bons, contudo também precisam ter hiperparâmetros otimizados.

Para essa seção, realizaram-se testes de tuning de modelos de Aprendizado Profundo. Algumas bases típicas de problemas de classificação de imagens foram selecionadas para estes testes: MNIST, CIFAR e Fashion-MNIST. Cada uma dessas bases teve uma divisão de 70% de imagens para a realização de treino e as demais 30% foram utilizadas para o teste.

Para esses testes de Aprendizado Profundos, foram construídas arquiteturas de redes CNN, onde os parâmetros construtivos dessas redes seriam hiperparâmetros otimizados. Os parâmetros escolhidos são diversos como quantidade de neurônios e taxa de dropout. A Tabela 11 demonstra o espaço de busca de todos esses parâmetros da CNN. Ao final, a rede construída apresenta uma arquitetura semelhante a Figura 15.

A Tabela 12 demonstra os resultados dos modelos de Aprendizado Profundo na base de teste das imagens com diferentes algoritmos de otimização. Em todos os testes, foi fixado um número de iterações $t = 10$, pois modelos de Aprendizado Profundo são custosos para ser treinados, logo é essencial alcançar bons resultados com baixo número de iterações, todos os testes rodados por 20 vezes com inicializações diferentes. Novamente, foram utilizadas as configurações padrões de cada algoritmo e para o BarySearch, ν foi fixado em -50 e com 10% de iterações para inicialização.

Conforme demonstram os resultados, o BarySearch mostra que ele pode alcançar

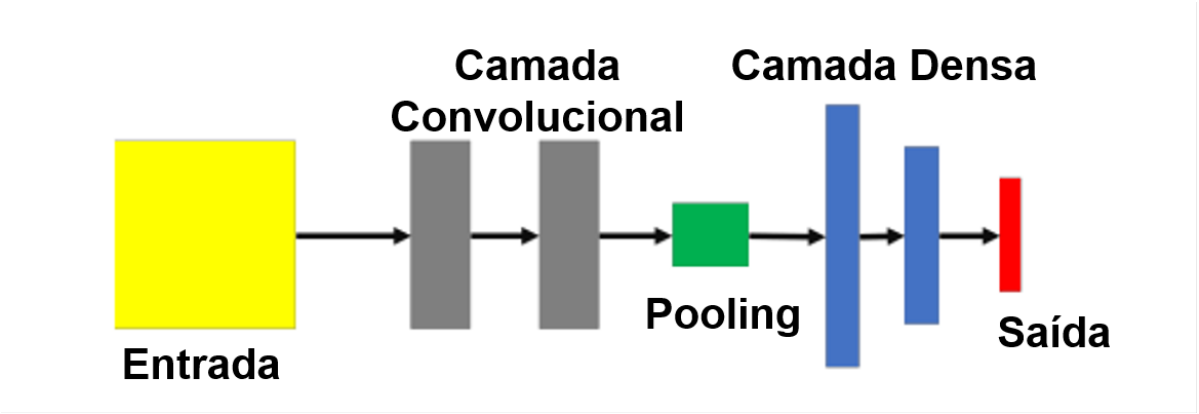

Figura 15: Arquitetura da Rede Neural Convolucional, na qual há duas camadas convolucionais, uma camada de pooling e mais duas camadas densas.
Fonte: Autor

resultados melhores que os demais algoritmos. Repare que ao contrário dos problemas de classificação de Benchmark, os algoritmos mais simples de tuning apresentaram resultados melhores (Simulated Anealing e busca aleatória) que algoritmos mais complexos (TPE). Dessa forma, pode se concluir que o BarySearch mostra ser algoritmo bem flexível, pois funciona bem em contexto onde tanto algoritmos simples e complexos falham.

4.4 Testes de AutoML

Para os problemas de AutoML, é necessário realizar adaptações tanto no BarySearch, quanto BPSO e suas variações para aceitarem parâmetros categóricos e condicionais. Como já explicado anteriormente, os problemas de AutoML ou CASH obrigatoriamente apresentam parâmetros categóricos e condicionais, devido aos requisitos de seleção de pré–processamento, de seleção de variáveis e seleção de modelo.

Para contornar essa limitação dos métodos propostos nesse trabalho, foi utilizada a abordagem do capítulo anterior para problemas AutoML na qual os parâmetros a serem otimizados são representados por um vetor x de dimensão $D = 13$, com algumas dessas dimensões representando a aplicação ou não de pré–processamento, seleção de variáveis ou hiperparâmetros de modelos. Claramente essa abordagem pode trazer limitações por considerar que os hiperparâmetros sejam independentes de si, mas será aceito essa limitação para verificar se o desempenho será impactado.

Para os testes, foram selecionadas as mesmas bases de classificação do UCI (DUA; GRAFF, 2017) como os testes de benchmark de classificação, contudo ao invés dos algoritmos tenham que apenas fazer um tuning de algum modelo, eles devem construir *Pipeline*

do modelo ML e fazer o tuning desse modelo ML. Os algoritmos testados foram a busca aleatória, Simulated Annealing (KIRKPATRICK; GELATT; VECCHI, 1983), TPE do Hyperopt (BERGSTRA et al., 2015), TPOT (LE; FU; MOORE, 2020), Hyperband (LI et al., 2017) e Auto-Sklearn (FEURER et al., 2015), além dos algoritmos propostos de BarySearch e BPSO Global. Em todos os casos, foram utilizadas as configurações padrões de cada algoritmo e para o BarySearch, ν está igual a -50 e com 10% de iterações para inicialização.

Para os testes, variou-se a quantidade de iterações para os algoritmos em t iguais a 10, 25, 50. Cada teste foi repetido por 30 vezes com inicializações diferentes. Novamente foi utilizada a métrica de taxa de erro de classificação média como função objetivo que se deseja otimizar.

Finalmente, todas as bases foram divididas em treino e teste, sendo que no treino é realizado o problema de tuning com uma validação de cross-validation com 5 repartições. O melhor *Pipeline* gerado no treino é testado na base de teste, onde é recolhida a taxa de erro de classificação. Os resultados estão dispostos na Tabela 13 adiante.

Como é possível verificar na Tabela 13, os algoritmos propostos, mesmo tendo as limitações de dados categórico e condicionais, bem como usar uma abordagem que não é ideal para esse problema, apresentam resultados comparáveis aos algoritmos de AutoML tradicionais. Esse resultado é muito promissor, pois é possível que, com abordagens melhores com parâmetros categóricos e condicionais, alcançar resultado ainda melhores que os algoritmos estado da arte.

Tabela 10: Média da taxa de erro de classificação. Os valores estão marcados com •ou ∗se a média for a melhor ou a secunda melhor respectivamente.

Base de Dados	Algoritmo	LGBM $t = 25$	MLP $t = 25$
Ionsphere	Random Search	9,0%	16,2%
	Simulated Annealing	19,3%	16,9%
	TPE	8,5% ∗	12,1%
	CMA-ES	21,1%	20,9%
	Hyperband	12,1%	15,2%
	TPOT	13,0%	12,1% ∗
	BarySearch	7,8% •	11,6% •
Credit	Random Search	9,7% ∗	29,9%
	Simulated Annealing	11,7%	29,0%
	TPE	9,9%	29,6%
	CMA-ES	12,9%	29,8%
	Hyperband	11,2%	25,3% ∗
	TPOT	12,1%	23,5% •
	BarySearch	9,4% •	27,9%
SPAM	Random Search	4,6%	17,2%
	Simulated Annealing	4,6%	17,4%
	TPE	4,6% ∗	15,3%
	CMA-ES	4,6% ∗	15,5%
	Hyperband	4,8%	11,0% •
	TPOT	4,5% •	15,1%
	BarySearch	4,5% •	13,6% ∗
Breast-Cancer	Random Search	0,6%	9,8%
	Simulated Annealing	10,0%	10,3%
	TPE	0,1% ∗	9,0%
	CMA-ES	12,4%	8,7%
	Hyperband	0,6%	3,0% •
	TPOT	0,4%	11,4%
	BarySearch	0,0% •	8,3% ∗
Dermatology	Random Search	5,9%	9,5%
	Simulated Annealing	30,9%	32,8%
	TPE	5,3% •	8,8% ∗
	CMA-ES	37,8%	28,4%
	Hyperband	8,2%	10,1%
	TPOT	6,7%	9,7%
	BarySearch	5,6% ∗	8,3% •

Tabela 11: Espaço de busca dos hiperparâmetros a serem tunados nos testes de Aprendizado Profundo.

Parâmetros	Espaço de Busca
Neurônios na 1ª Camada Convolucional	16 a 64
Neurônios na 2ª Camada Convolucional	16 a 64
Dropout na 1ª Camada Densa	0 a 1
Dropout na 2ª Camada Densa	0 a 1
Neurônios na 1ª Camada Densa	16 a 128
Neurônios na 2ª Camada Densa	16 a 128
Tamanho do kernel na horizontal (1ª e 2ª Camadas)	1 a 5
Tamanho do kernel na vertical (1ª e 2ª Camadas)	1 a 5
Tamanho do pooling na horizontal	1 a 5
Tamanho do pooling na vertical	1 a 5
Learning Rate	0.001 a 0.1
β_1 do otimizador ADAM	0.6 to 0.99
β_1 do otimizador ADAM	0.85 to 0.999
Tamanho de *batch*	500 a 200
Épocas	2 a 10

Tabela 12: Taxa de Erro Médio de Classificação das Imagens.

Algoritmo	**MNIST**	**Fashion-MNIST**	**CIFAR10**
TPE	1.02%	9.76%	42.16%
Simulated Annealing	0.97%	9.11%	40.70%
Random Search	0.93%	9.07%	38.57%
BarySearch	0.90%	8.85%	36.33%

Tabela 13: Média da taxa de erro de classificação para os testes do AutoML.

Base	Algoritmo	$t = 20$	$t = 50$	$t = 100$	$t = 200$
Ionsphere	BarySearch	6,3%	6,4%	6,7%	6,2%
	TPE	6,5%	6,5%	6,6%	7,7%
	Random Search	6,7%	6,7%	7,1%	7,4%
	TPOT	9,3%	8,2%	7,5%	7,3%
	AutoSklearn	5,2%	5,2%	5,2%	4,5%
Credito	BarySearch	9,8%	9,5%	9,4%	9,0%
	TPE	11,1%	10,9%	11,2%	11,7%
	Random Search	10,9%	11,2%	10,9%	10,5%
	TPOT	9,0%	7,8%	7,4%	7,0%
	AutoSklearn	8,3%	8,9%	8,5%	8,9%
Spam	BarySearch	6,3%	4,2%	4,3%	4,0%
	TPE	6,2%	4,4%	4,4%	4,1%
	Random Search	6,2%	5,1%	5,0%	4,3%
	TPOT	7,8%	7,5%	7,2%	7,3%
	AutoSklearn	8,9%	11,7%	11,1%	10,3%
Cancer	BarySearch	2,5%	2,5%	2,5%	2,1%
	TPE	2,7%	2,7%	2,7%	2,5%
	Random Search	3,1%	2,7%	2,7%	2,6%
	TPOT	8,7%	7,7%	8,0%	7,3%
	AutoSklearn	3,9%	3,7%	4,4%	4,5%
Dermatology	BarySearch	5,0%	5,0%	4,6%	4,4%
	TPE	5,6%	5,2%	5,1%	4,9%
	Random Search	5,0%	5,2%	5,0%	4,9%
	TPOT	9,1%	7,6%	7,0%	6,7%
	AutoSklearn	6,3%	5,4%	4,6%	5,1%

5 DISCUSSÃO E COMENTÁRIOS FINAIS

Essa dissertação de mestrado foi focada nos problemas de tuning de modelos ML e resolução de problemas AutoML. Inicialmente, foi explicado o que são modelos ML e onde estão sendo aplicados. Em seguida, foi demonstrada a diferença de parâmetros internos de modelos ML que são definidos no treinamento desses modelos com hiperparâmetros dos modelos ML que são parâmetros definidos antes do treinamento que podem alterar o comportamento do modelo e influenciar nos seu desempenho. Sobre esse contexto, encontrar o melhor conjunto de valores de hiperparâmetros pode ser importante para aplicações que se desejam o melhor desempenho. Esse problema de busca de melhores valores é conhecido como tuning de modelos ML.

No texto, foi apresentado que o problema de tuning de modelos ML é um modelo de otimização, entretanto o tuning não pode ser resolvido diretamente como muitos outros problemas de otimização, pois os modelos ML são vistos como caixas pretas e não é possível encontrar formulação matemática bem definida entre o desempenho do modelo e seus hiperparâmetros. Essa ausência de formulação matemática também impede abordagens usando cálculos de derivadas ou gradientes da função objetivo. Somando a esses problemas, também são encontradas complexidades no tuning como parâmetros categóricos e condicionais, além de regiões com muitas oscilações de desempenho ou regiões de grandes platôs. Dessa forma, as abordagens utilizadas para resolver esses problemas em geral são métodos de buscas que não necessitam do cálculo de derivadas ou gradientes, nomeados na academia como livre de derivadas.

No trabalho, foi citado algumas das principais técnicas utilizadas para resolução desse problema de tuning de hiperparâmetros. Primeiramente, foram citadas as técnicas mais simples e tradicionais como a busca exaustiva e a busca aleatória. A busca exaustiva em geral não é recomendada pelo custo computacional e apresenta pior desempenho esperado que a busca aleatória, contudo outros métodos foram desenvolvidos a fim de alcançar melhores desempenhos. Alguns desses métodos foram técnicas evolucionárias que são técnicas inspiradas em fenômenos naturais e biológicos vistos na natureza. Nessa cate-

goria, podem ser lembrados o Simulated Annealing, PSO, CMAES e técnicas genéticas. Todos os métodos guardam certas semelhanças por tentar criar gerações de resultados melhores que o histórico das resoluções atuais. Em outra grande categoria estão os métodos de busca bayesianas que são baseadas na formula de Bayes de probabilidade condicionais. Esses métodos estão apoiados na construção da curva surrogate que calcula a probabilidade de algum outro ponto ter um resultado promissor mediante a todo outro histórico anterior. Algumas variações das buscas bayesianas foram desenvolvidas: com processos gaussianos, TPE ou SMAC por exemplo.

Além do problema de tuning, também há o problema de AutoML que busca a construção do *Pipeline* que obtenha o melhor resultado na tarefa. Esses problemas selecionam pré–processamento, variável e modelo ML, além de um conjunto de hiperparâmetros. O problema de AutoML é considerado um problema mais complexo, pois além de englobar o problema de tuning de modelos ML, também realiza o problema de seleção de modelos. Para solucionar o problema de AutoML, vários trabalhos propuseram técnicas ou sistemas para solucionar o problema. Destacam-se como estado da arte: TPOT, Auto-Sklearn e Hyperband. Cada um dos três apresenta uma abordagem diferente, seja genética dada pelo TPOT, bayesiana pelo Auto-Sklearn ou outra abordagem mais simples do Hyperband.

Finalmente, o trabalho demonstra o método do baricentro e explica um pouco de suas propriedades de convergências. O método do baricentro é definido por uma ponderação ponderada pelo resultado da função objetivo, onde os pesos da ponderação são exponenciais. O método do baricentro também introduz variáveis aleatórias para realização da exploração do espaço de busca. O baricentro pode ser considerado como um método evolucionário graças em grande parte por semelhanças aos demais métodos evolucionários já proposto. A formula do baricentro basicamente é uma maneira de selecionar os melhores resultados e combiná-los em busca de resultado ainda melhores o que lembra muito as técnicas evolucionárias.

Com a definição do método do baricentro, foi possível propor o algoritmo BarySearch que é basicamente o baricentro aplicado para tuning com uma inicialização conhecida como LHS. Essa inicialização traz algumas vantagens por fazer uma amostragem de forma a "varrer" por várias regiões do espaço de busca e fazer um bom mapeamento da função objetivo. Essa inicialização combinada com o método do baricentro pode trazer bons resultados na busca dos melhores parâmetros. Além disso, foi discutido a importância dos parâmetros de convergência ν e do parâmetro de curiosidade no comportamento do algoritmo do BarySearch. Esses parâmetros podem ser escolhidos de forma a ter um algoritmo

mais convergente, mais determinístico e mais semelhante a um método do gradiente ou então um algoritmo mais exploratório, mais aleatório e com comportamento mais similar ao evolucionário.

Além do BarySearch, o trabalho encontrou oportunidades de realização de hibridização do método do baricentro com muitas variações do PSO. Em geral, os métodos do PSO usam a melhor posição da partícula e do enxame para atualização das posições das partículas, contudo todo um potencial histórico pode ser perdido. Em face disso, o trabalho faz a proposta de substituir as melhores posições pelo o baricentro das m posições anteriores, assim pode ser possível fazer ponderações das demais posições e evitar por exemplo que o algoritmo fique preso em regiões de mínimos locais ruins. Essa hibridização com o baricentro apresenta características muito interessantes e que podem ser úteis em vários contextos de otimização.

Por fim, os testes realizados demonstraram o quanto promissor os algoritmos propostos tem de desempenho. Para os testes de comparação dos híbridos de PSO e baricentro frente aos PSO tradicionais, os híbridos apresentam resultados estatisticamente superiores aos tradicionais em várias funções clássicas de otimização em sua maioria. Assim, os híbridos podem ter propriedades de grande interesse, principalmente em espaços de busca com dimensões maiores.

Para os problemas de tuning e AutoML, o BarySearch e os híbridos do PSO e baricentro demonstram resultados muito promissores, mesmo considerando algumas limitações com parâmetros categóricos e condicionais. Apesar dessas limitações, os algoritmos propostos mostraram desempenho comparável ou mesmo superior em relação aos algoritmos clássicos ou estado da arte. Considerando o AutoML, foi necessária uma abordagem simplificadora para poder tratar parâmetros categóricos e condicionais e mesmo assim a performance foi muito promissora comparada aos demais algoritmos.

Com esses resultados, é possível concluir que os métodos propostos apresentam por exemplo vantagens em trabalhar com espaços de busca em altas dimensões, são flexíveis ao contexto de aplicação (seja tuning de modelos ML, tuning de Aprendizado Profundo ou AutoML) o que muitos outros algoritmos não são e também são algoritmos que apresentam resultados superiores, principalmente em um número mais baixo de iterações o que indica uma convergência mais rápida.

Para trabalhos futuros, é desejável realizar uma abordagem melhor para parâmetros categóricos e condicionais, como híbridos com algoritmos genéticos. Podem ser realizados testes de hibridização do método do baricentro com outros algoritmos evolucionários, como

Colônia de Formigas, Tabu Search, Gravitacional ou Cocu Search. Também é considerado a construção de pacotes e serviços no estilo de HyperOpt, Auto-Sklearn ou TPOT com BarySearch ou os híbridos com o baricentro disponíveis para a comunidade acadêmica para pesquisas e contribuições.

REFERÊNCIAS

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

KOCH, P. et al. Autotune: A derivative-free optimization framework for hyperparameter tuning. ***KDD***, p. 443–452, 07 2018. ISBN: 978-1-4503-4887-4.

BISSUEL, A. *Hyper-parameter optimization algorithms: a short review*. 2019. Acessado em: 2019-08-20. Disponível em: ⟨https://medium.com/criteo-labs/hyper-parameter-optimization-algorithms-2fe447525903⟩.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, JMLR.org, v. 13, p. 281–305, fev. 2012. ISSN 1532-4435. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2188385.2188395⟩.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In: PEREIRA, F. et al. (Ed.). ***Advances in Neural Information Processing Systems 25***. Curran Associates, Inc., 2012. p. 2951–2959. Disponível em: ⟨http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf⟩.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: COELLO, C. A. C. (Ed.). ***Learning and Intelligent Optimization***. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 507–523. ISBN 978-3-642-25566-3.

BERGSTRA, J. S. et al. Algorithms for hyper-parameter optimization. In: SHAWE-TAYLOR, J. et al. (Ed.). *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011. p. 2546–2554. Disponível em: ⟨http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf⟩.

LORENA, A.; CARVALHO, A. Evolutionary tuning of svm parameter values in multiclass problems. ***Neurocomputing - IJON***, v. 71, p. 3326–3334, 10 2008.

LE, T. T.; FU, W.; MOORE, J. H. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. ***Bioinformatics***, Oxford University Press, v. 36, n. 1, p. 250–256, 2020.

HANSEN, N.; OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 2, p. 159–195, jun. 2001. ISSN 1063-6560. Disponível em: ⟨https://doi.org/10.1162/106365601750190398⟩.

EBERHART, R.; KENNEDY, J. S. A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, p. 39–43, 1995.

CONN, A. R.; SCHEINBERG, K.; VICENTE, L. N. ***Introduction to derivative–free optimization***. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2009. v. 8. (MPS/SIAM Series on Optimization, v. 8).

Pait, F. M. The Barycenter Method for Direct Optimization. ***ArXiv e-prints***, jan. 2018. Disponível em: ⟨https://arxiv.org/abs/1801.10533⟩.

BUITINCK, L. et al. API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. [S.l.: s.n.], 2013. p. 108–122.

CHOLLET, F. et al. *Keras*. 2015. "⟨https://keras.io⟩".

THORNTON, C. et al. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: ***Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining***. New York, NY, USA: Association for Computing Machinery, 2013. (KDD '13), p. 847–855. ISBN 9781450321747. Disponível em: ⟨https://doi.org/10.1145/2487575.2487629⟩.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). *Automatic machine learning: methods, systems, challenges*. Germany: Springer, 2019. (Challenges in Machine Learning). ISBN 978-3-030-05317-8.

KIM, Y.; CHUNG, M. An approach to hyperparameter optimization for the objective function in machine learning. *Electronics*, v. 8, p. 1267, 2019. Disponível em: ⟨https://www.mdpi.com/2079-9292/8/11/1267⟩.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. ***Science***, v. 220 4598, p. 671–80, 1983.

LEDESMA, S.; RUIZ-PINALES, J.; GARCIA-HERNANDEZ, M. Simulated annealing evolution. *Simulated Annealing - Advances, Applications and Hybridizations*, p. 210–218, 08 2012.

CHIARANDINI, M. et al. An effective hybrid approach for the university course timetabling problem. In: . [S.l.: s.n.], 2003.

Kennedy, J.; Eberhart, R. Particle swarm optimization. In: ***Proceedings of ICNN'95 - International Conference on Neural Networks***. Boston, MA: Springer US, 1995. v. 4, p. 1942–1948 vol.4. ISSN null.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. ***1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*** *(Cat. No.98TH8360)*, p. 69–73, 1998.

Clerc, M.; Kennedy, J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. ***IEEE Transactions on Evolutionary Computation***, v. 6, n. 1, p. 58–73, Feb 2002. ISSN 1941-0026.

KENNEDY, J. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. ***Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*** *(Cat. No. 99TH8406)*, v. 3, p. 1931–1938 Vol. 3, 1999.

KENNEDY, J.; MENDES, R. Population structure and particle swarm performance. In: ***Proceedings of the 2002 Congress on Evolutionary Computation***. *CEC'02 (Cat. No.02TH8600)*. [S.l.: s.n.], 2002. v. 2, p. 1671 – 1676. ISBN 0-7803-7282-4.

SUGANTHAN, P. N. Particle swarm optimiser with neighbourhood operator. ***Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*** *(Cat. No. 99TH8406)*, v. 3, p. 1958–1962 Vol. 3, 1999.

PARSOPOULOS, K.; VRAHATIS, M. A unified particle swarm optimization scheme. *Proceedings of 2004 International Conference of Computational Methods in Sciences and Engineering (ICCMSE'04)*, p. 874–879, 01 2004.

LIU, Q. et al. Topology selection for particle swarm optimization. ***Information Sciences***, v. 363, 05 2016.

LIU, Q. et al. Dynamic small world network topology for particle swarm optimization. *Int. J. Pattern Recognit. Artif. Intell.*, v. 30, p. 1660009:1–1660009:20, 2016.

Mendes, R.; Kennedy, J.; Neves, J. The fully informed particle swarm: simpler, maybe better. ***IEEE Transactions on Evolutionary Computation***, v. 8, n. 3, p. 204–210, June 2004. ISSN 1941-0026.

YANG, X.-S.; DEB, S.; FONG, S. Accelerated particle swarm optimization and support vector machine for business optimization and applications. In: *NDT*. [S.l.: s.n.], 2012. v. 136.

HEFNY, H. A.; AZAB, S. S. Chaotic particle swarm optimization. ***2010 The 7th International Conference on Informatics and Systems (INFOS)***, p. 1–8, 2010.

Liang, J. J. et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. ***IEEE Transactions on Evolutionary Computation***, v. 10, n. 3, p. 281–295, June 2006. ISSN 1941-0026.

van den Bergh, F.; Engelbrecht, A. P. A cooperative approach to particle swarm optimization. ***IEEE Transactions on Evolutionary Computation***, v. 8, n. 3, p. 225–239, June 2004. ISSN 1941-0026.

Zhan, Z. et al. Adaptive particle swarm optimization. ***IEEE Transactions on Systems, Man, and Cybernetics****, Part B (Cybernetics)*, v. 39, n. 6, p. 1362–1381, Dec 2009. ISSN 1941-0492.

Gong, Y. et al. Genetic learning particle swarm optimization. ***IEEE Transactions on Cybernetics***, v. 46, n. 10, p. 2277–2290, Oct 2016. ISSN 2168-2275.

CMA-ES. In: WIKIPEDIA, the free encyclopedia. Wikimedia, 2019. Disponível em: ⟨https://en.wikipedia.org/wiki/CMA-ES⟩. Acesso em: 2019-05-04.

BERGSTRA, J. et al. Hyperopt: A python library for model selection and hyperparameter optimization. ***Computational Science & Discovery***, v. 8, p. 014008, 07 2015.

CUTLER, A.; CUTLER, D.; STEVENS, J. Random forests. In: _____. [S.l.: s.n.], 2011. v. 45, p. 157–176.

FEURER, M. et al. Efficient and robust automated machine learning. In: CORTES, C. et al. (Ed.). ***Advances in Neural Information Processing Systems 28***. Curran Associates, Inc., 2015. p. 2962–2970. Disponível em: ⟨http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf⟩.

BRAZDIL, P. et al. ***Metalearning - Applications to Data Mining.*** [S.l.: s.n.], 2009. ISBN 978-3-540-73262-4.

LACOSTE, A. et al. Agnostic bayesian learning of ensembles. In: XING, E. P.; JEBARA, T. (Ed.). ***Proceedings of the 31st International Conference on Machine Learning***. Bejing, China: PMLR, 2014. (Proceedings of Machine Learning Research, 1), p. 611–619. Disponível em: ⟨http://proceedings.mlr.press/v32/lacoste14.html⟩.

LANGDON, W. et al. Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In: _____. [S.l.: s.n.], 1970. v. 115, p. 927–1028.

OLSON, R. S. et al. Automating biomedical data science through tree-based pipeline optimization. In: *EvoApplications*. [S.l.: s.n.], 2016.

LI, L. et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, JMLR.org, v. 18, n. 1, p. 6765–6816, jan. 2017. ISSN 1532-4435.

JAMIESON, K. G.; TALWALKAR, A. Non-stochastic best arm identification and hyperparameter optimization. In: *AISTATS*. [S.l.: s.n.], 2015.

CARUANA, R.; LAWRENCE, S.; GILES, C. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: . [S.l.: s.n.], 2000. v. 13, p. 402–408.

RODRíGUEZ, J.; PéREZ, A.; LOZANO, J. Sensitivity analysis of k-fold cross validation in prediction error estimation. ***Pattern Analysis and Machine Intelligence***, *IEEE Transactions on*, v. 32, p. 569 – 575, 04 2010.

MCKAY, M. D. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In: ***Proceedings of the 24th Conference on Winter Simulation***. New York, NY, USA: ACM, 1992. (WSC '92), p. 557–564. ISBN 0-7803-0798-4. Disponível em: ⟨http://doi.acm.org/10.1145/167293.167637⟩.

LHS. In: WIKIPEDIA, the free encyclopedia. Wikimedia, 2019. Disponível em: ⟨https://en.wikipedia.org/wiki/Latin_hypercube_sampling⟩. Acesso em: 2019-06-17.

Xin Yao; Yong Liu; Guangming Lin. Evolutionary programming made faster. ***IEEE Transactions on Evolutionary Computation***, v. 3, n. 2, p. 82–102, July 1999. ISSN 1941-0026.

BERGH, F.; ENGELBRECHT, A. A study of particle swarm optimization particle trajectories. ***Information Sciences***, v. 176, p. 937–971, 04 2006.

ESCALANTE, H. J.; MONTES, M.; SUCAR, L. Particle swarm model selection. In: . [S.l.: s.n.], 2010. v. 10, p. 1 – 8.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: ⟨http://archive.ics.uci.edu/ml⟩.

ANEXO A – ANEXO A - GRÁFICOS DOS RESULTADOS DAS FUNÇÕES BENCHMARK

As Figuras adiante são os gráficos com os resultados das funções benchmark com 5, 10, 15, 25 e 30 iterações para as funções unimodais, multimodais e multimodais rotacionadas. No eixo vertical, estão os valores das funções objetivos e no eixo horizontal, estão os algoritmos testados. Em cada iteração e algoritmo, é realizado um teste de comparação entre PSO clássico (azul) e a hibridização do PSO com o baricentro (laranja). Como o objetivo é minimizar a função objetivo, a PSO com o melhor resultado é aquele cujas distribuições estão em níveis inferiores. Repare que na maioria dos cenários, a distribuição do híbrido (laranja), está em nível menor que o tradicional (azul).

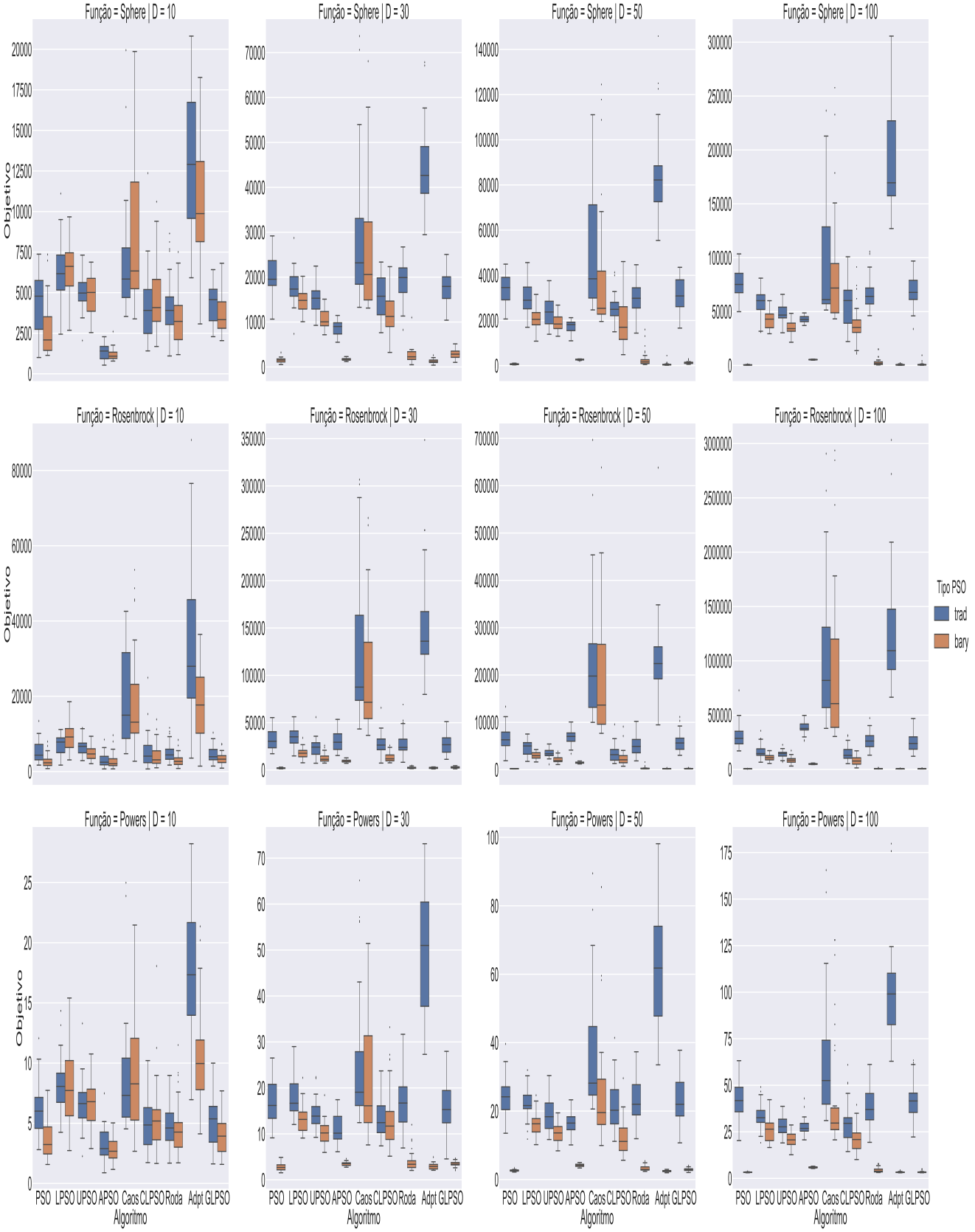

Figura 16: Resultado das funções Unimodais com 5 gerações dos algoritmos.
Fonte: Autor

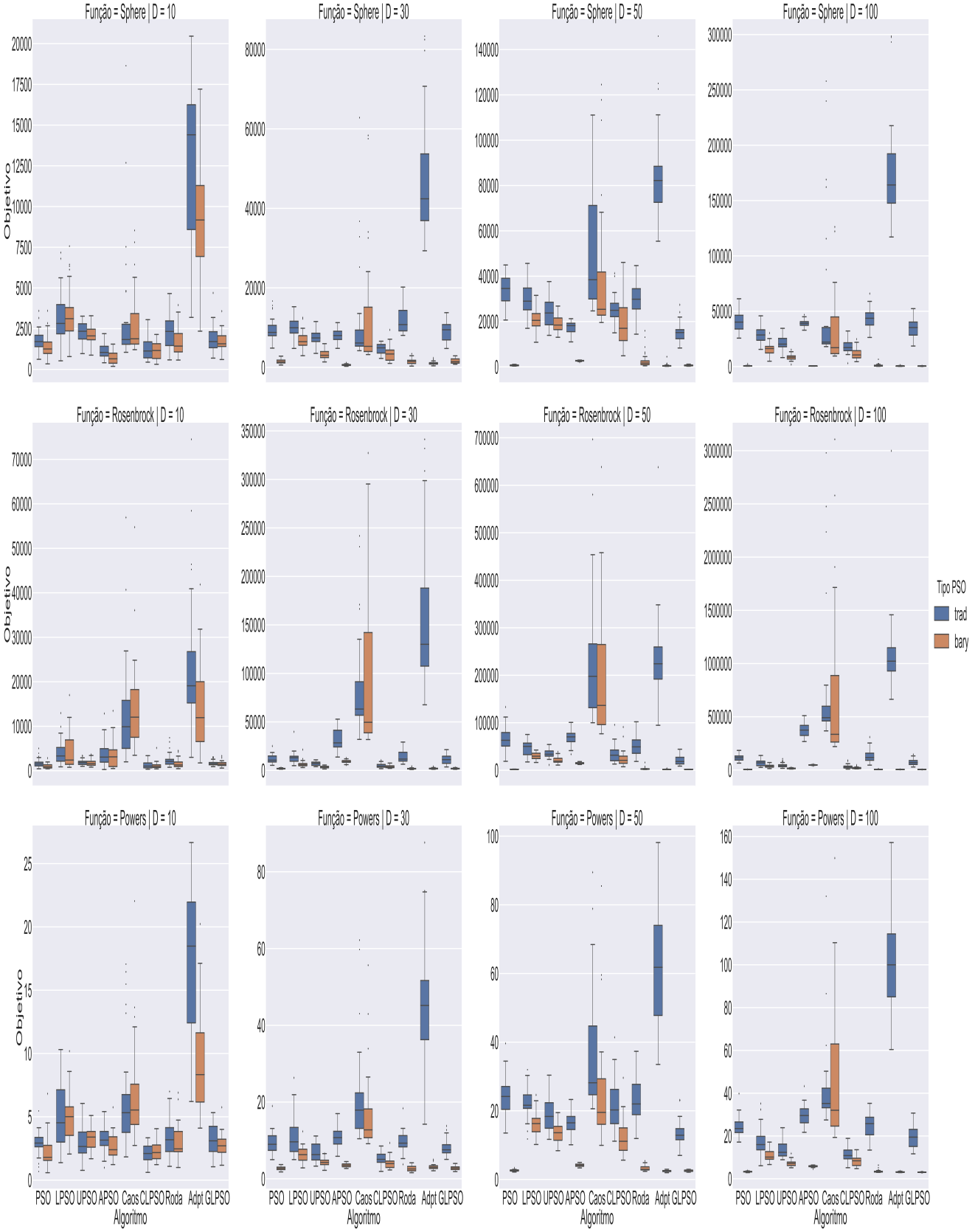

Figura 17: Resultado das funções Unimodais com 10 gerações dos algoritmos.
Fonte: Autor

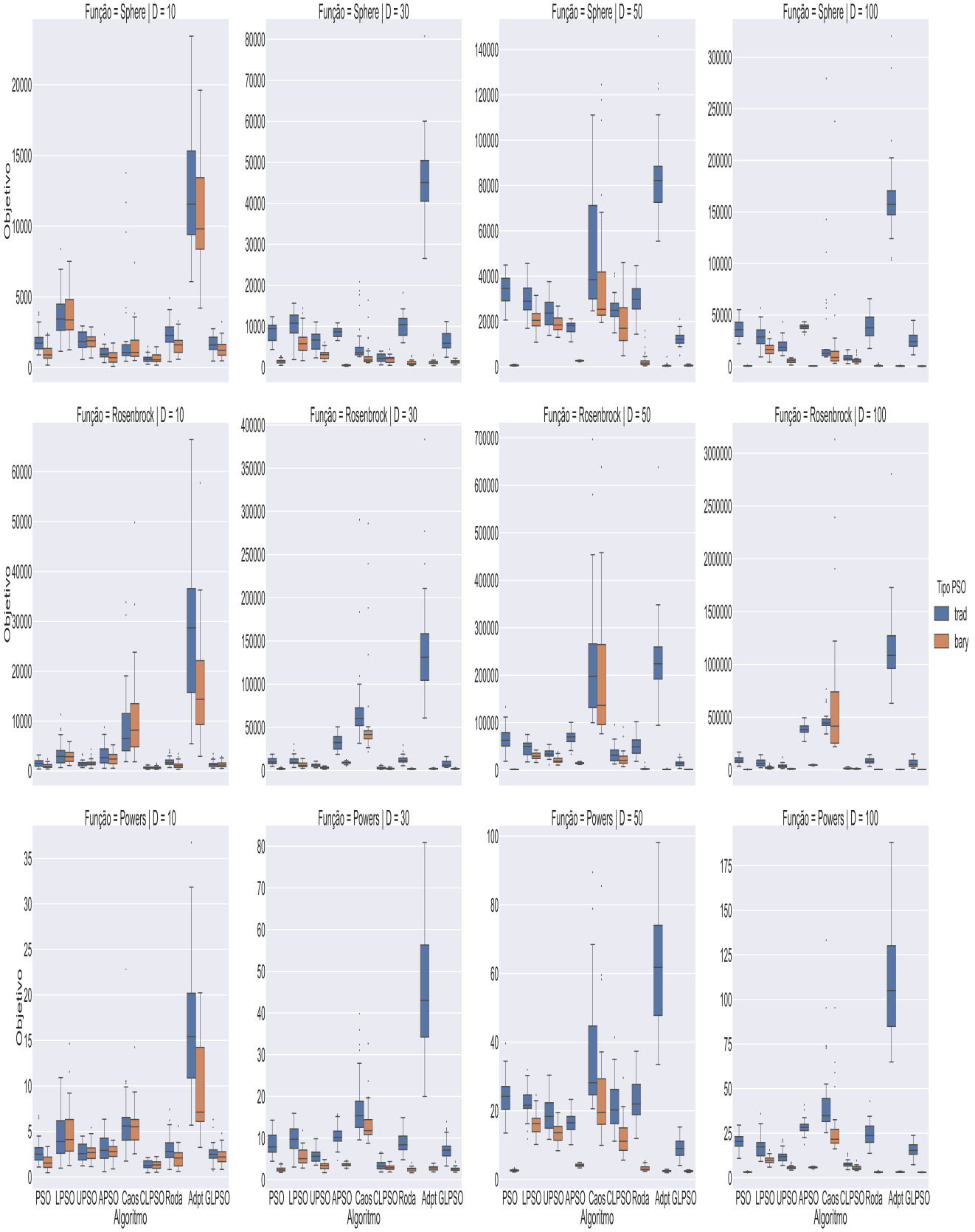

Figura 18: Resultado das funções Unimodais com 15 gerações dos algoritmos.
Fonte: Autor

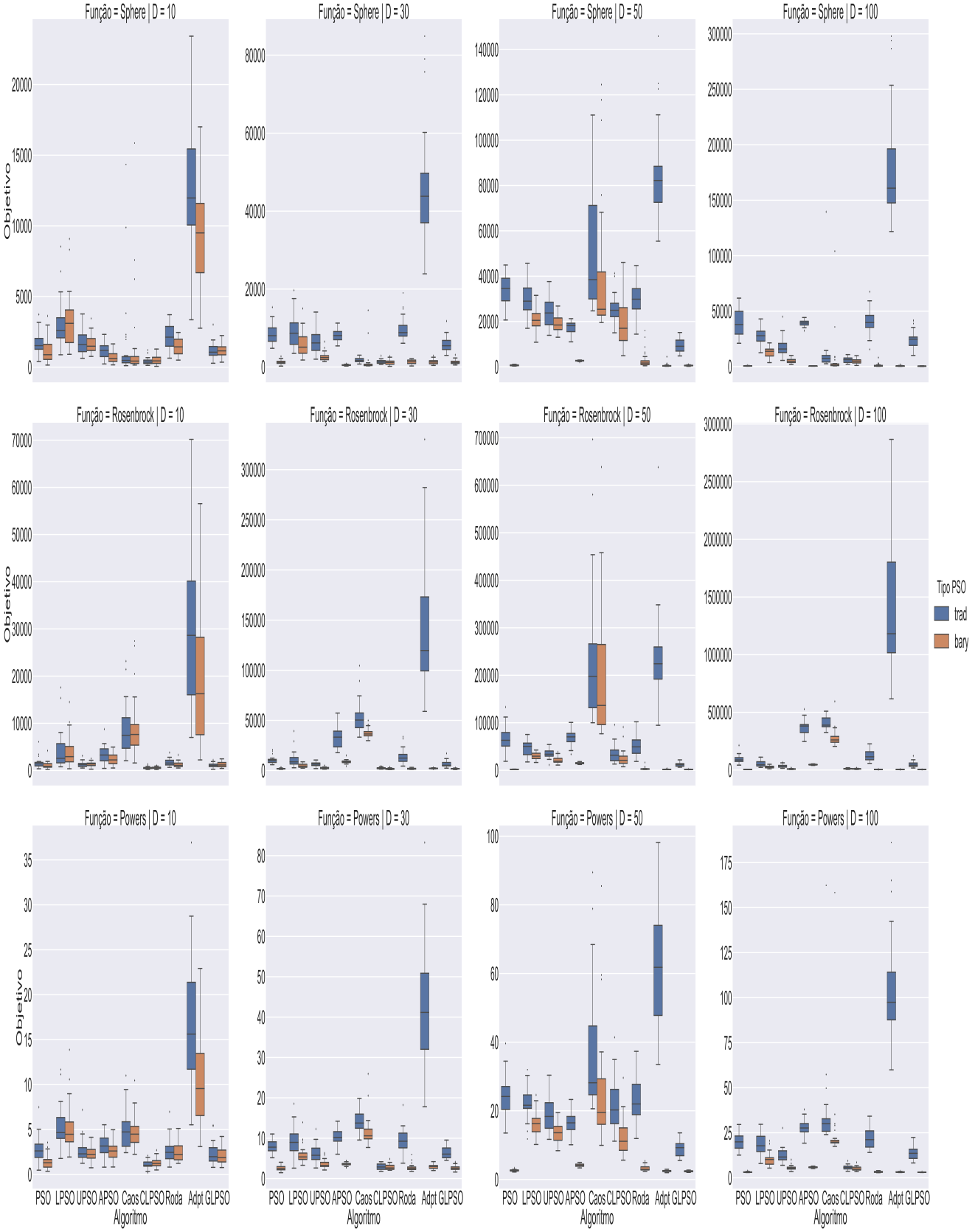

Figura 19: Resultado das funções Unimodais com 25 gerações dos algoritmos.
Fonte: Autor

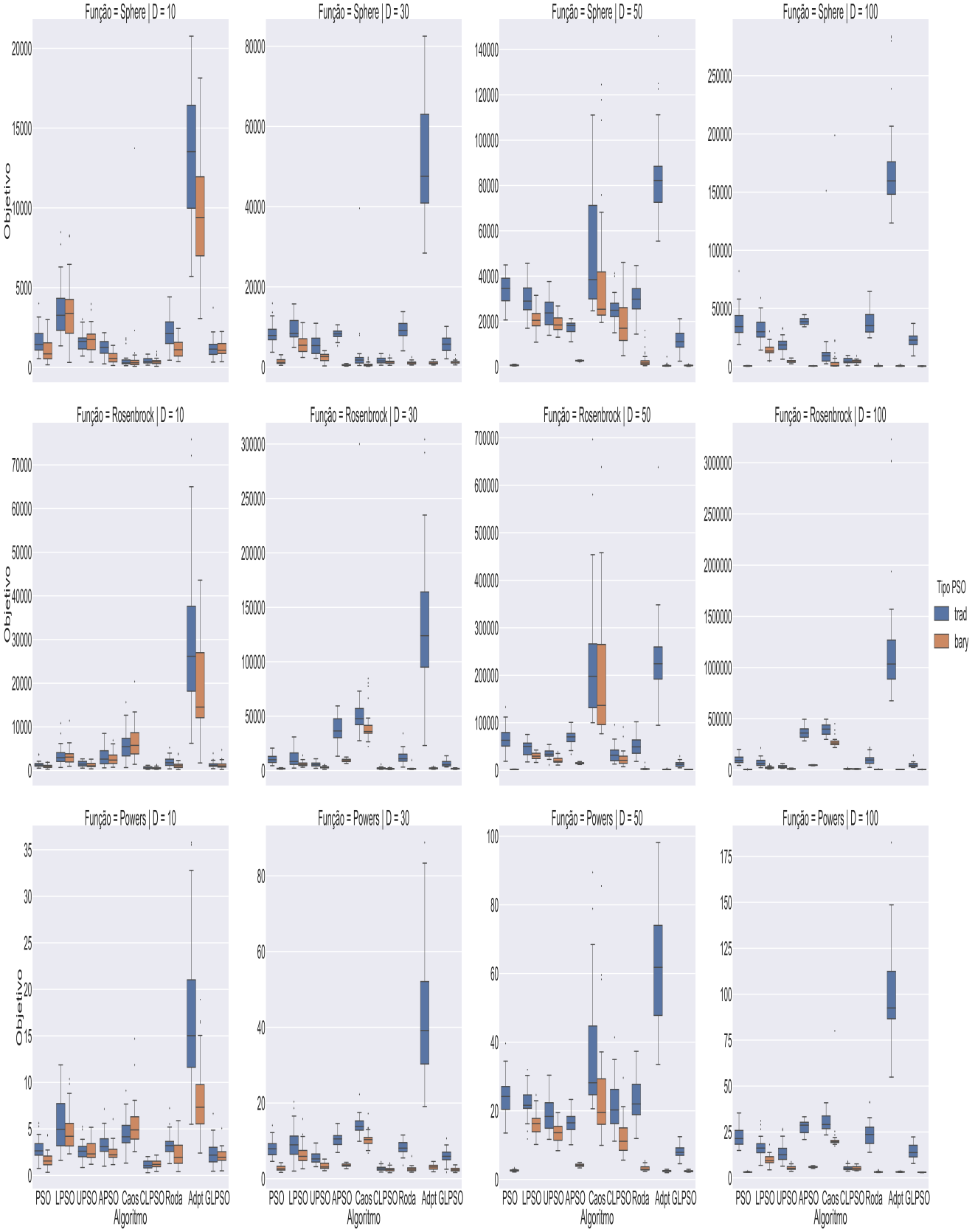

Figura 20: Resultado das funções Unimodais com 30 gerações dos algoritmos.
Fonte: Autor

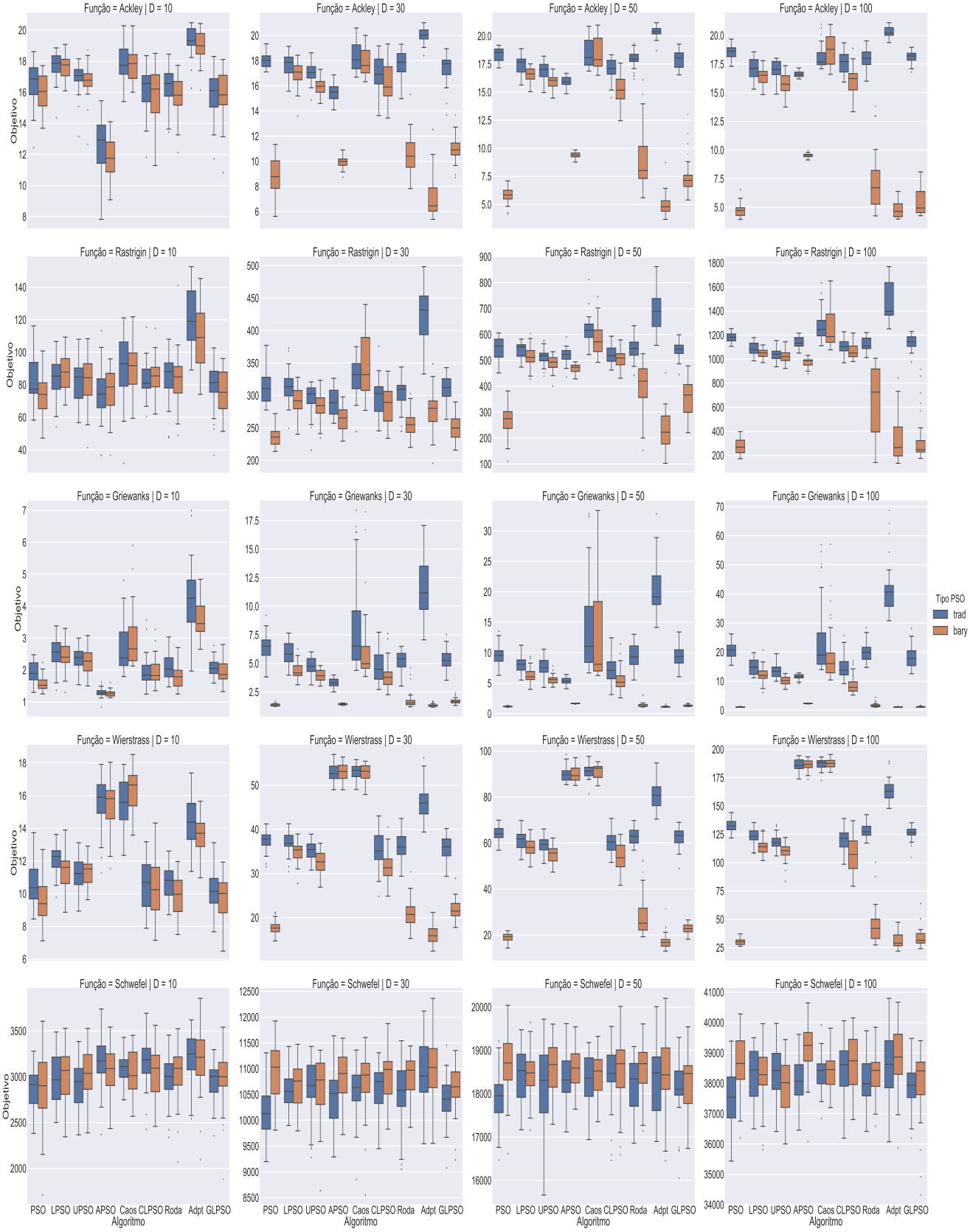

Figura 21: Resultado das funções Multimodais com 5 gerações dos algoritmos.
Fonte: Autor

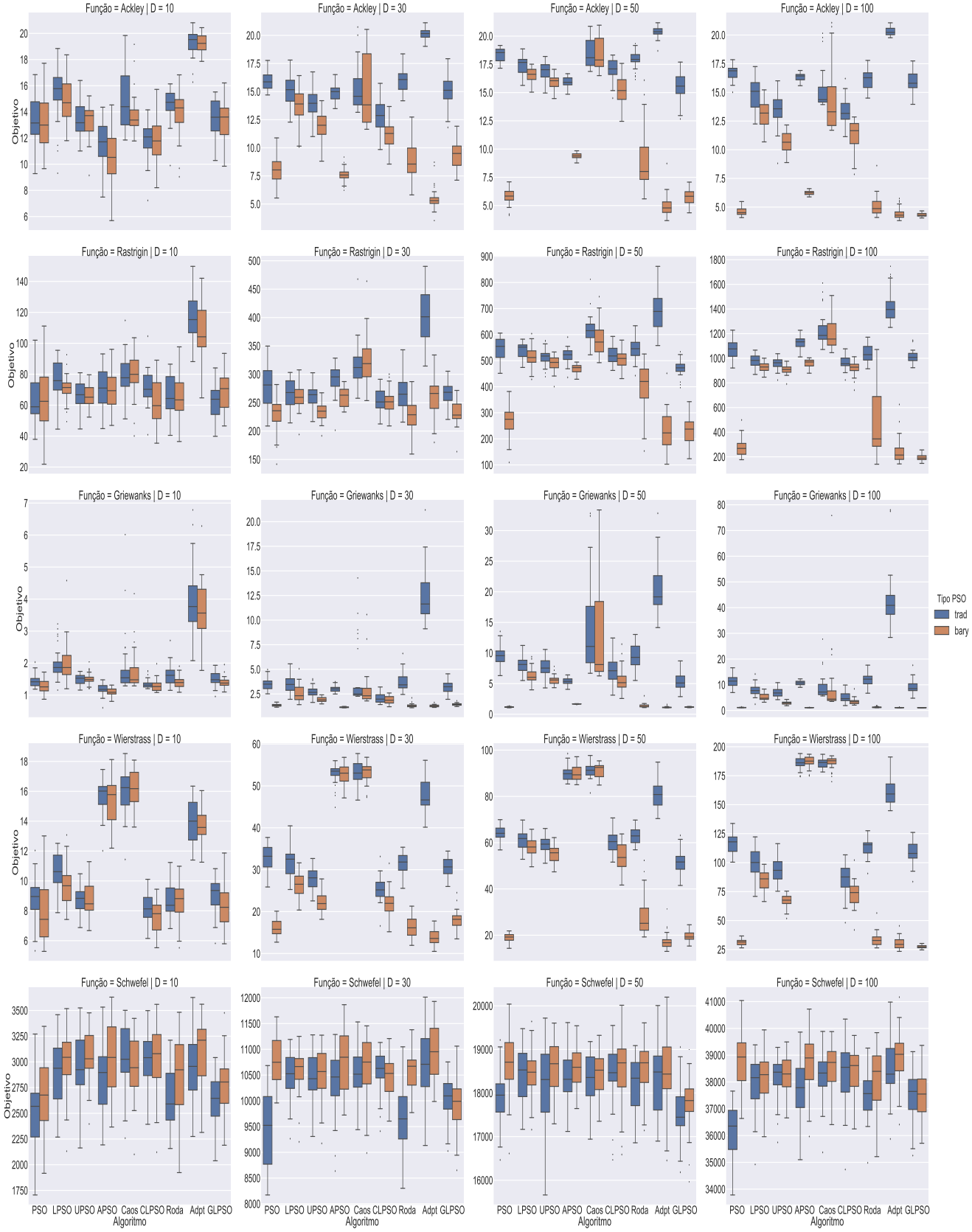

Figura 22: Resultado das funções Multimodais com 10 gerações dos algoritmos.
Fonte: Autor

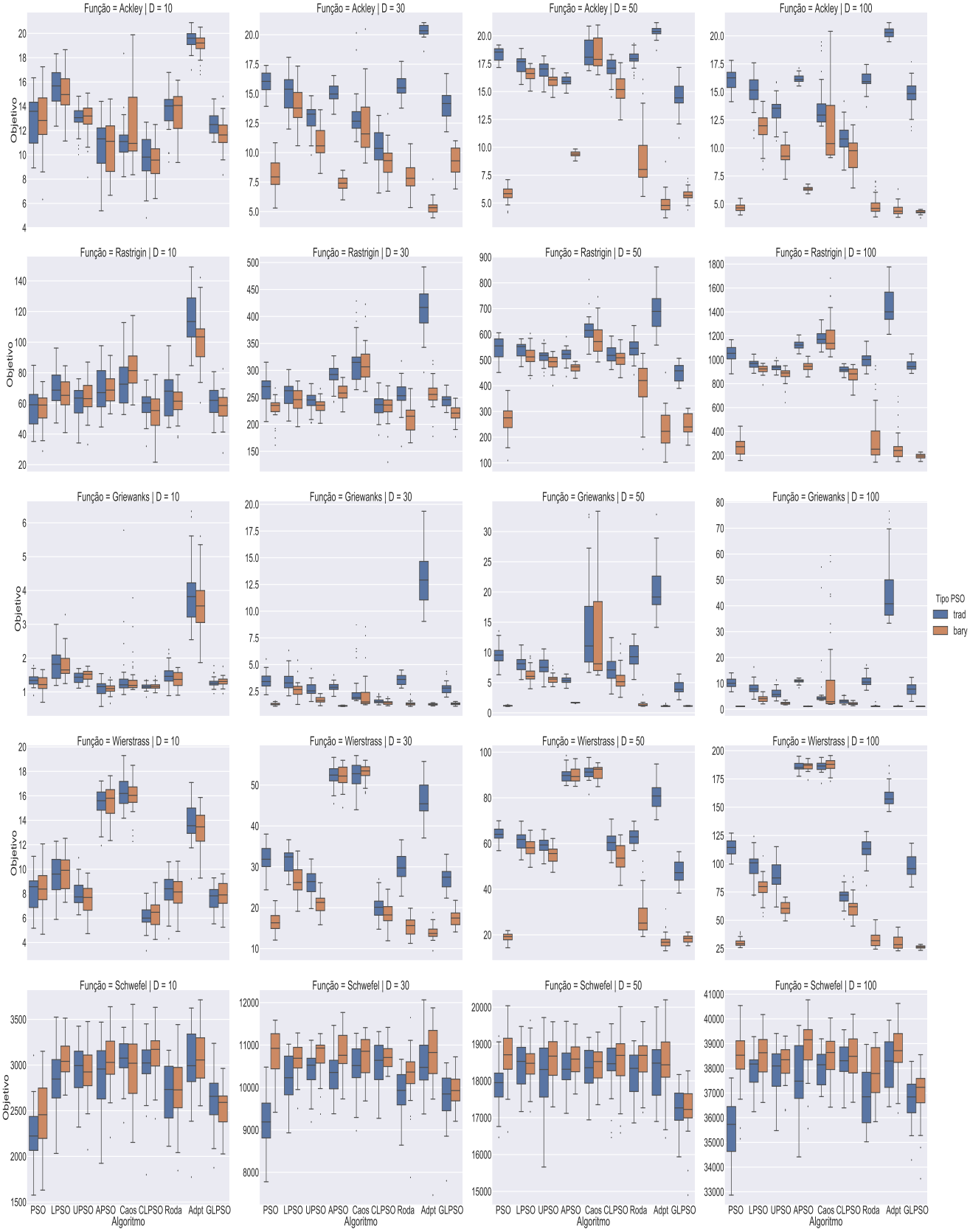

Figura 23: Resultado das funções Multimodais com 15 gerações dos algoritmos.
Fonte: Autor

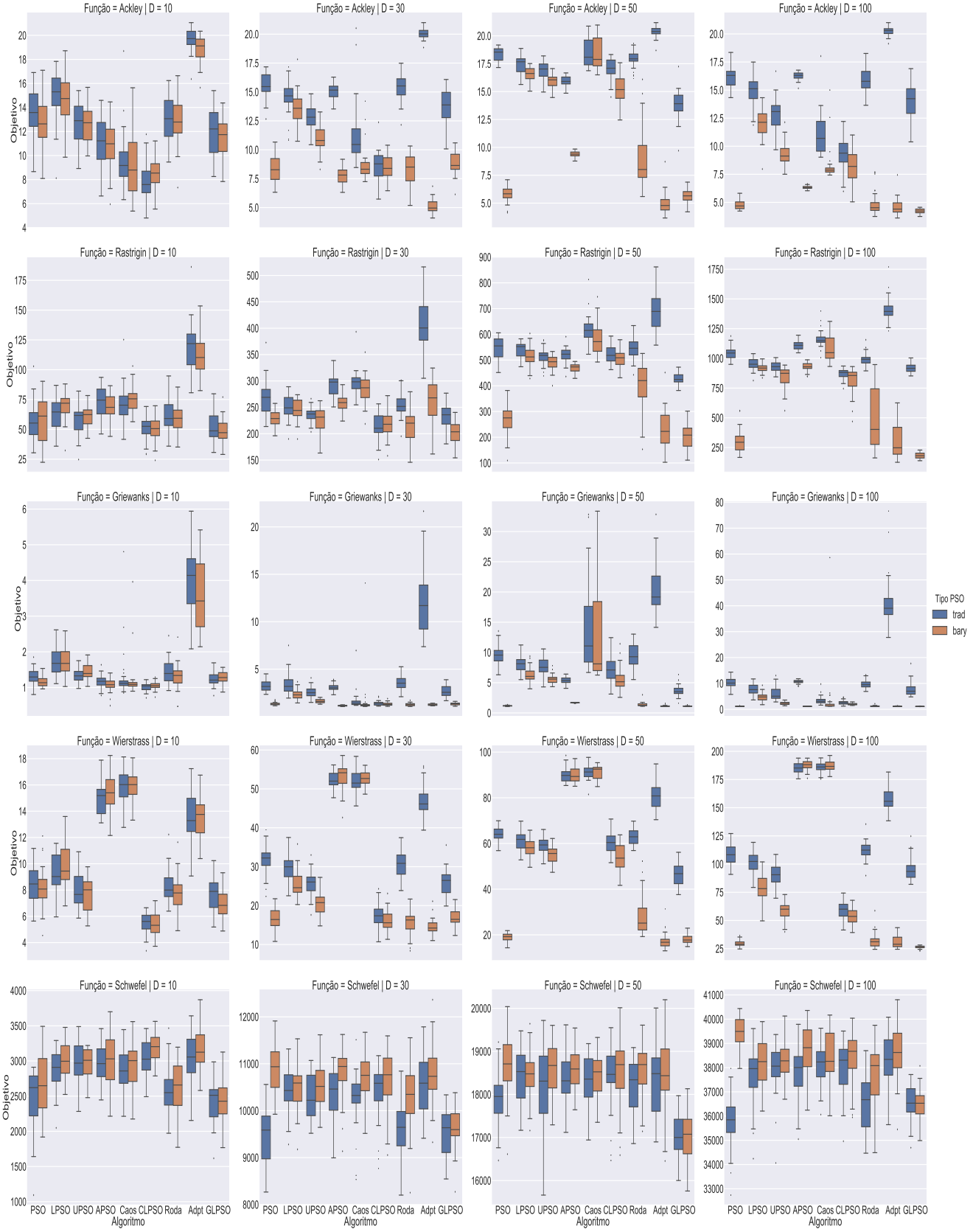

Figura 24: Resultado das funções Multimodais com 25 gerações dos algoritmos.
Fonte: Autor

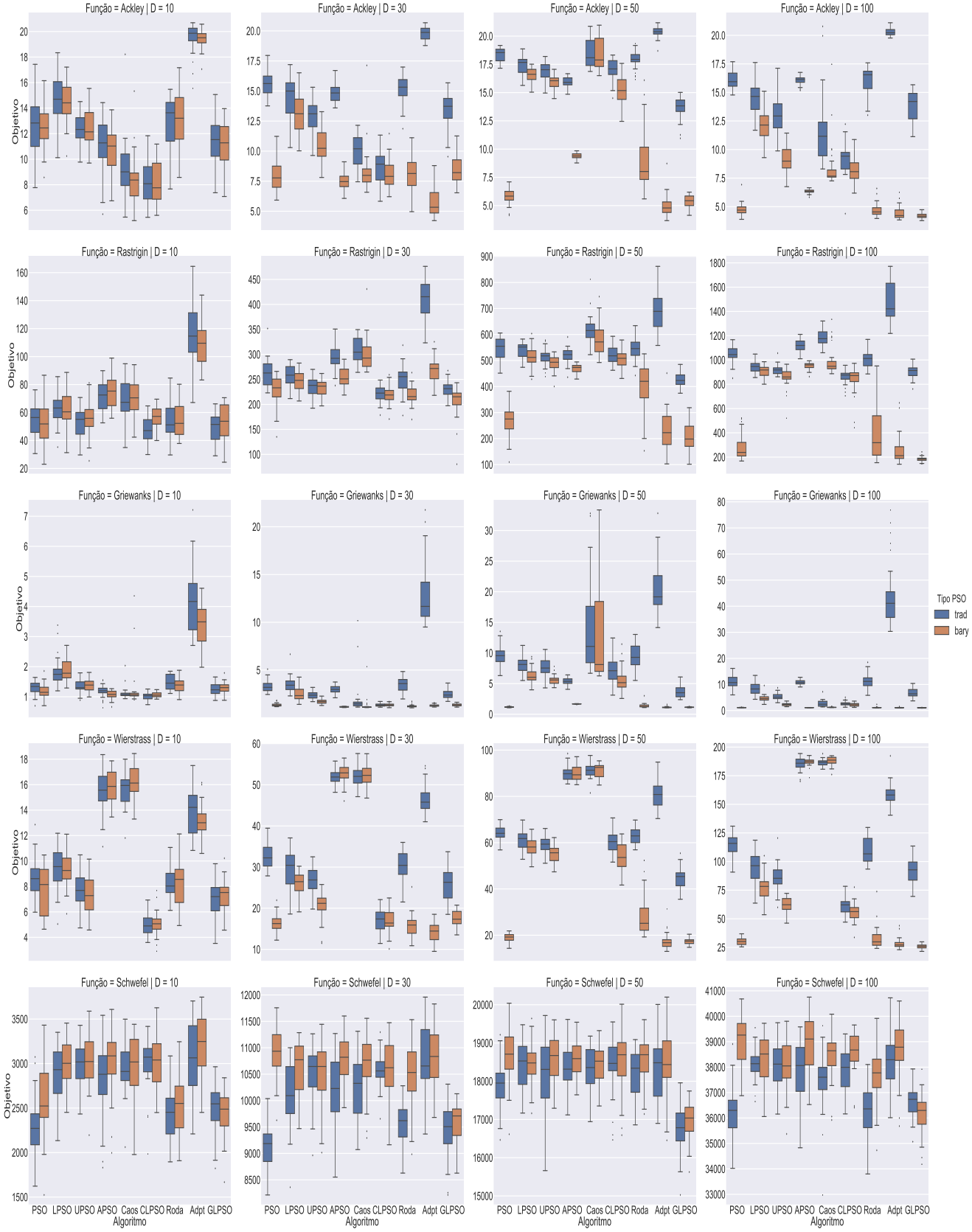

Figura 25: Resultado das funções Multimodais com 30 gerações dos algoritmos.
Fonte: Autor

Figura 26: Resultado das funções Multimodais Rotacionadas com 5 gerações dos algoritmos.
Fonte: Autor

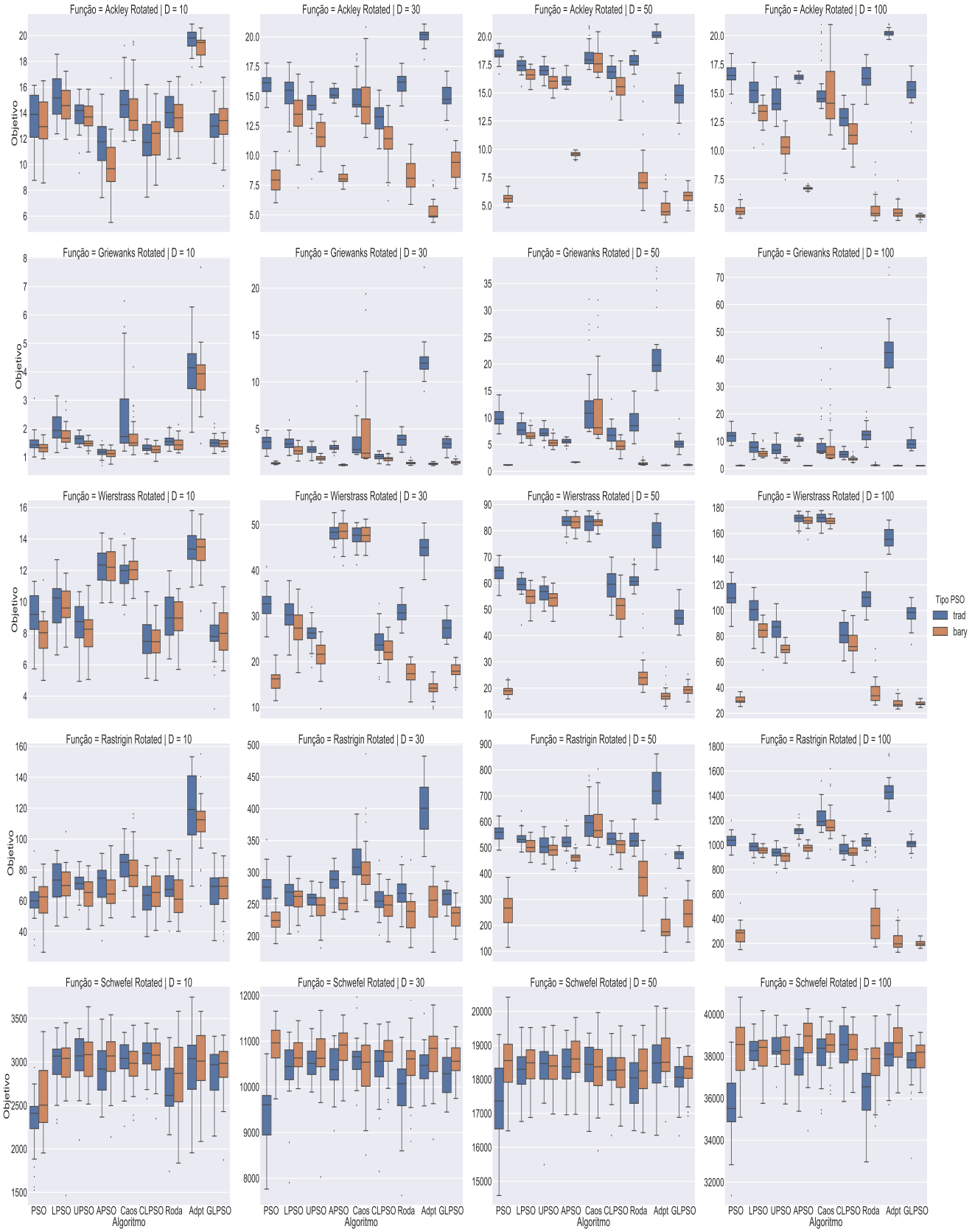

Figura 27: Resultado das funções Multimodais Rotacionadas com 10 gerações dos algoritmos.
Fonte: Autor

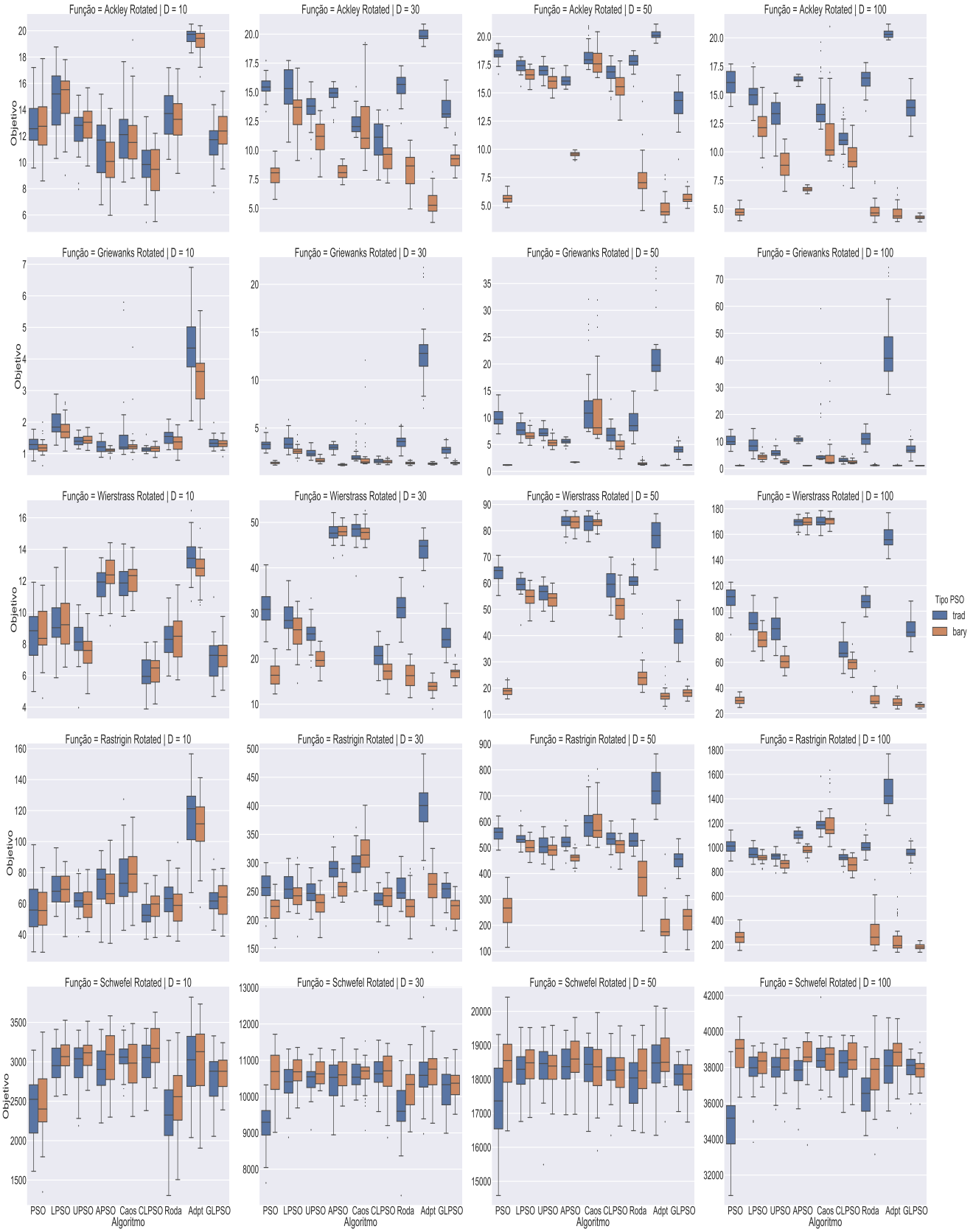

Figura 28: Resultado das funções Multimodais Rotacionadas com 15 gerações dos algoritmos.
Fonte: Autor

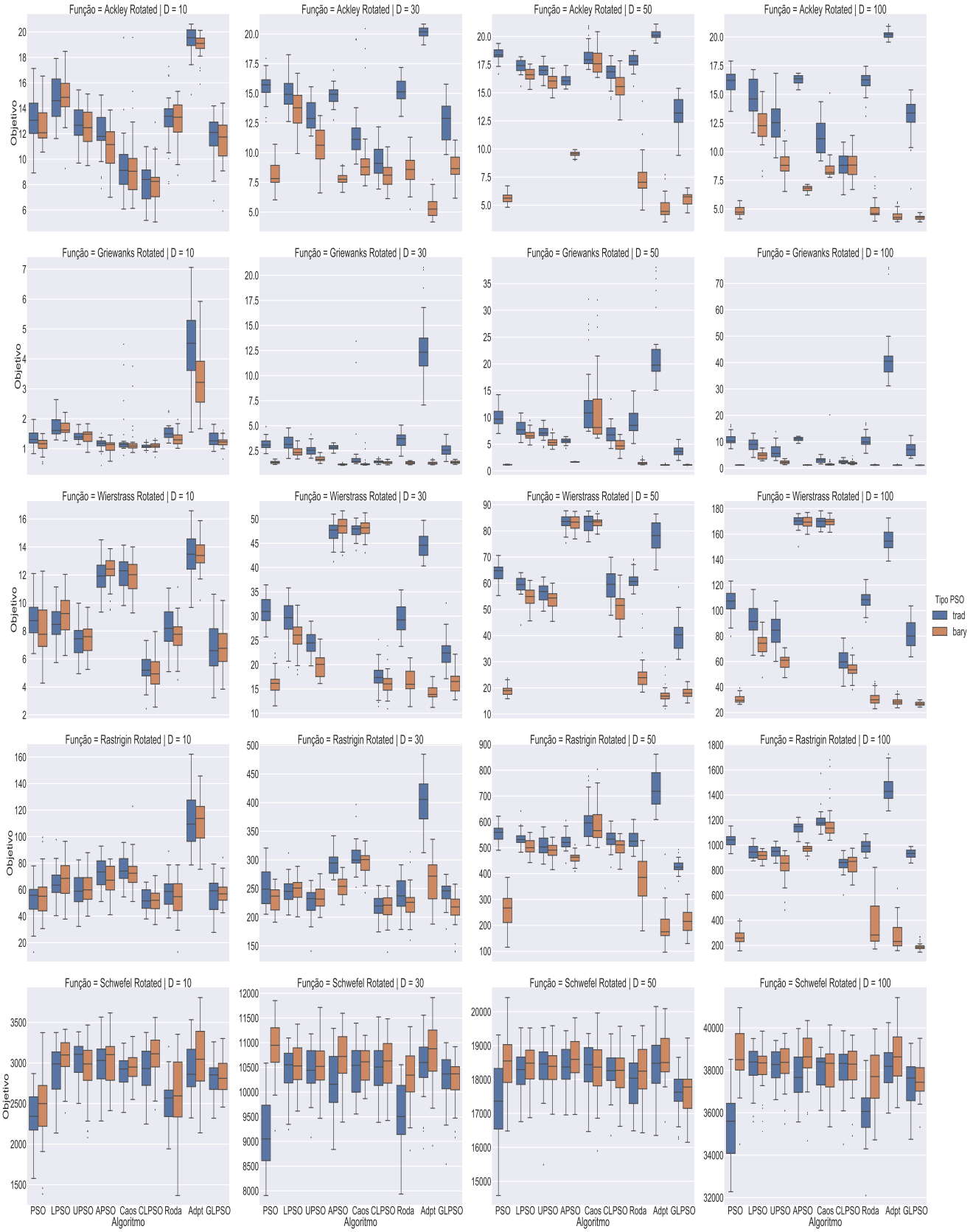

Figura 29: Resultado das funções Multimodais Rotacionadas com 25 gerações dos algoritmos.
Fonte: Autor

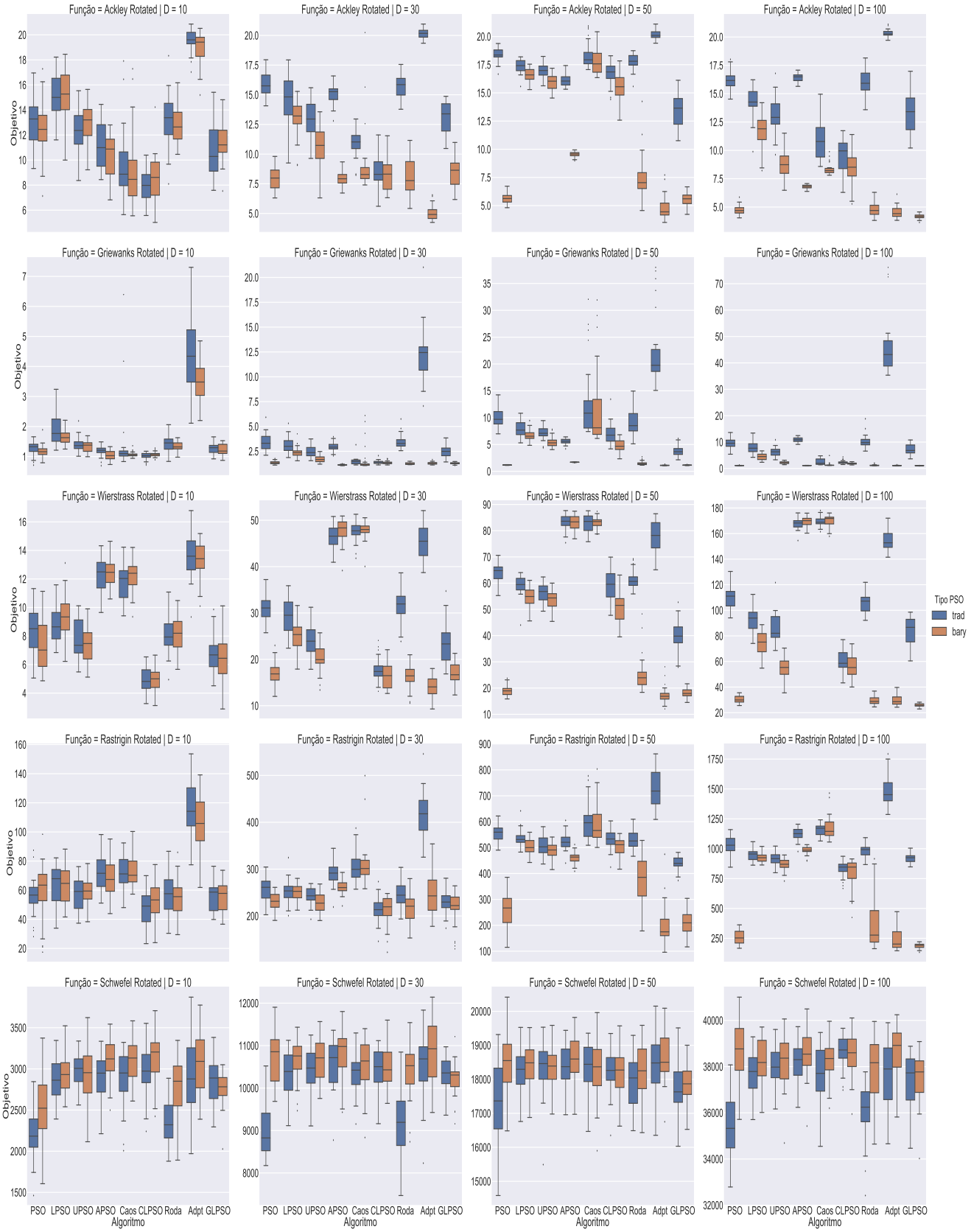

Figura 30: Resultado das funções Multimodais Rotacionadas com 30 gerações dos algoritmos.
Fonte: Autor