Um Framework Metodológico para a Validação de Benchmarking Experimental em Ciência Computacional

1. Fundamentos de Benchmarking de Algoritmos Rigoroso

A comparação de algoritmos computacionais evoluiu de uma arte ad-hoc para uma disciplina científica rigorosa. A validade de qualquer alegação sobre a superioridade de um novo método depende inteiramente da robustez e transparência da metodologia experimental utilizada para a sua avaliação. Esta seção estabelece os princípios fundamentais que sustentam um benchmarking de algoritmos credível: a necessidade imperativa de reprodutibilidade computacional, os pilares que a sustentam e o papel essencial de conjuntos de benchmarks diversificados e representativos.

1.1 A Necessidade Imperativa de Reprodutibilidade na Ciência Computacional

Antes que o desempenho de qualquer algoritmo possa ser medido ou comparado, o próprio experimento deve ser reprodutível. A reprodutibilidade computacional é a capacidade de regenerar os resultados de um estudo, incluindo figuras e tabelas, utilizando os materiais originais (como dados, código e documentação).[1] Este princípio é a base da ciência cumulativa; sem ele, as descobertas não podem ser verificadas, os erros não podem ser corrigidos e o progresso científico é dificultado.[2] Um estudo pode ser reprodutível, mas ainda assim sofrer de falhas analíticas que invalidam os seus resultados. No entanto, se um estudo for reprodutível, essas falhas podem ser mais facilmente identificadas e retificadas pela comunidade em geral.[1] A falta de reprodutibilidade, por outro lado, mina a confiança e a

credibilidade da investigação computacional.

1.2 Os Cinco Pilares da Reprodutibilidade Computacional

Para transformar o ideal da reprodutibilidade numa prática concreta, a investigação moderna baseia-se num framework de cinco pilares interligados, complementados pelas "Dez Regras Simples para a Investigação Computacional Reprodutível".[1] Estes pilares fornecem um guia prático para garantir que os resultados experimentais sejam transparentes, verificáveis e duradouros.

1.2.1 Programação Literária e Fluxos de Trabalho Automatizados

O primeiro pilar exige a eliminação de intervenções manuais no pipeline de análise de dados. Passos manuais, como copiar e colar dados ou ajustar ficheiros em editores de texto, são propensos a erros, ineficientes e inerentemente difíceis de documentar e reproduzir.[6] A prática recomendada é encapsular todo o fluxo de trabalho de análise — desde a obtenção de dados brutos até à geração de resultados finais — numa forma executável. Isto pode ser alcançado através de scripts de shell,

makefiles ou sistemas de gestão de fluxos de trabalho mais sofisticados.[6] Ferramentas como

Make permitem a automação de pipelines complexos, garantindo que toda a análise pode ser reexecutada com um único comando, o que assegura que a especificação documentada do fluxo de trabalho corresponde precisamente à sua execução real.

1.2.2 Controlo de Versão de Código e Partilha

O segundo pilar aborda a natureza dinâmica do desenvolvimento de software. Mesmo pequenas alterações no código podem ter consequências significativas nos resultados. Sem um registo sistemático da evolução do código, torna-se quase impossível retroceder a um estado específico que produziu um resultado particular.[6] Os Sistemas de Controlo de Versão Distribuídos (DVCS), com o

Git a ser o padrão de facto, são a solução para este desafio. O Git permite o rastreio

completo do histórico de todas as alterações de código, facilita a colaboração através de plataformas centralizadas como o GitHub e protege contra a perda de código.[2]

No entanto, projetos de machine learning e de análise de dados em larga escala lidam frequentemente com artefactos grandes, como conjuntos de dados e modelos treinados, que são inadequados para repositórios Git padrão. A extensão **Git Large File Storage (Git LFS)** resolve este problema ao substituir ficheiros grandes por ponteiros de texto leves no repositório Git, enquanto o conteúdo real do ficheiro é armazenado num servidor remoto. Esta abordagem permite o controlo de versão de todo o ecossistema do projeto — código, dados e modelos — de forma coesa e eficiente.[7]

1.2.3 Controlo do Ambiente Computacional

O terceiro pilar foca-se na estabilidade e consistência do ambiente de software. Os resultados computacionais são sensíveis não só ao código, mas também às versões exatas do sistema operativo, compiladores, bibliotecas e pacotes de software utilizados. As atualizações de software, embora geralmente benéficas, podem alterar subtilmente os resultados, tornando a reprodução num sistema diferente ou numa data posterior um desafio significativo.[2]

O controlo rigoroso do ambiente é, portanto, essencial. Ferramentas de gestão de pacotes e ambientes, como o **Poetry** para Python ou o Conda, são cruciais. Estas ferramentas permitem a criação de ambientes isolados e a especificação exata (ou "bloqueio") de todas as dependências do projeto e das suas versões. Ao partilhar o ficheiro de bloqueio (por exemplo, poetry.lock), outros investigadores podem recriar o ambiente computacional idêntico com um único comando, garantindo que o código é executado no mesmo contexto de software em que foi originalmente desenvolvido.[2] Para ambientes mais complexos, tecnologias de contentorização como o Docker fornecem uma solução ainda mais robusta, encapsulando toda a pilha de software, desde as bibliotecas do sistema até à aplicação, num contentor portátil e consistente.[2]

1.2.4 Partilha Persistente de Dados

O quarto pilar estende o princípio da transparência aos dados. Para uma verificação independente completa, todos os dados de entrada, scripts, parâmetros e resultados intermédios devem ser tornados públicos e facilmente acessíveis.[6] Embora plataformas como o GitHub sejam excelentes para partilhar código, não são soluções de arquivo a longo prazo.

Os dados e o código devem ser depositados em repositórios persistentes, como Zenodo, Figshare ou Software Heritage, que fornecem identificadores persistentes (como DOIs) para garantir a acessibilidade a longo prazo.[2]

1.2.5 Documentação Abrangente

O pilar final é a documentação. Código, dados e fluxos de trabalho, por mais bem organizados que estejam, são de pouca utilidade sem uma documentação clara que explique a sua configuração, execução e a interpretação dos resultados.[2] Uma boa documentação serve como um manual para outros investigadores (e para o "eu" futuro do investigador original), guiando-os através do processo de reprodução e permitindo-lhes compreender o raciocínio por detrás das escolhas de análise.

1.3 O Papel dos Conjuntos de Benchmarks de Alta Qualidade

Metodologias de comparação rigorosas são apenas tão fiáveis quanto os dados que analisam. A seleção de um conjunto de benchmarks apropriado é um passo crítico que influencia profundamente a validade e a generalização das conclusões experimentais. Um conjunto de benchmarks de alta qualidade deve ser representativo da prática do mundo real e abranger uma vasta gama de características de problemas para evitar o sobreajuste de um algoritmo a um subconjunto restrito de instâncias.[9] A abordagem mais robusta combina o uso de benchmarks do mundo real e sintéticos, cada um oferecendo vantagens complementares.

Esta dualidade entre o real e o sintético não é uma escolha de um ou outro, mas uma necessidade para uma avaliação completa. Os grafos do mundo real fornecem relevância inegável e testam o desempenho em estruturas orgânicas e complexas. Os grafos sintéticos, por outro lado, oferecem ambientes controlados para testar sistematicamente os limites algorítmicos, a escalabilidade e a sensibilidade a parâmetros específicos. Uma conclusão robusta sobre o desempenho de um algoritmo só pode ser tirada se este demonstrar um bom desempenho em ambos os tipos de benchmarks, provando tanto a sua aplicabilidade no mundo real como o seu comportamento de escalamento previsível.

1.3.1 Benchmarks do Mundo Real

Benchmarks baseados em dados do mundo real são essenciais para garantir a relevância prática. Estes conjuntos de dados capturam as complexidades, irregularidades e estruturas subtis que os modelos sintéticos muitas vezes não conseguem replicar. Um exemplo de excelência é o benchmark **LDBC Graphalytics**, um padrão de nível industrial para plataformas de análise de grafos.[10] O Graphalytics inclui um conjunto diversificado de grafos do mundo real de vários domínios, como redes sociais (

com-friendster, twitter_mpi), conhecimento (wiki-talk, cit-patents) e jogos (dota-league, kgs). Estes grafos variam drasticamente em tamanho, densidade e características estruturais, fornecendo um teste abrangente e realista para os algoritmos.[10] A qualidade de um benchmark moderno como o LDBC Graphalytics não reside apenas nos seus dados, mas em todo o seu ecossistema. Ao fornecer software de código aberto para geração de dados e monitorização de desempenho, o LDBC implementa diretamente os pilares da reprodutibilidade, tornando o próprio benchmark um artefacto científico reprodutível.[10]

1.3.2 Benchmarks Sintéticos

Embora os benchmarks do mundo real testem a relevância, os benchmarks sintéticos são indispensáveis para experiências controladas. Permitem aos investigadores gerar sistematicamente grafos com propriedades específicas, testar a escalabilidade de um algoritmo em ordens de magnitude e isolar o impacto de parâmetros individuais na performance.

No domínio da deteção de comunidades (uma forma de particionamento de grafos), o benchmark **Lancichinetti–Fortunato–Radicchi (LFR)** tem sido um padrão de longa data.[12] O LFR gera grafos que replicam duas propriedades fundamentais de muitas redes do mundo real: distribuições de grau e de tamanho de comunidade em lei de potência. O seu parâmetro chave, o parâmetro de mistura

μ, controla a fração de arestas de um nó que ligam a nós fora da sua comunidade, permitindo aos investigadores ajustar a "dificuldade" da tarefa de deteção de comunidades.[12]

Mais recentemente, o modelo **Artificial Benchmark for Community Detection (ABCD)** surgiu como uma alternativa mais rápida, escalável e teoricamente mais simples ao LFR.[16] O ABCD pode gerar grafos com propriedades semelhantes às do LFR, mas é 40-100 vezes mais rápido, pode ser analisado teoricamente com maior facilidade e possui um parâmetro de mistura

ξ com uma interpretação mais natural, que varia suavemente entre grafos com comunidades

perfeitamente separadas (ξ=0) e grafos aleatórios puros sem estrutura comunitária (ξ=1).[16]

1.3.3 Geração Imparcial e Conectividade

Um objetivo teórico na geração de benchmarks sintéticos é a imparcialidade, que idealmente seria alcançada através da amostragem uniforme do conjunto de todos os grafos com um determinado conjunto de parâmetros (por exemplo, número de vértices e arestas).[17] Na prática, isto é computacionalmente impraticável para grafos grandes. No entanto, princípios relacionados são cruciais, como garantir a conectividade do grafo gerado. Uma abordagem para garantir a conectividade é começar por gerar uma

árvore de expansão uniforme (uniform spanning tree - UST), que é uma amostra imparcial de todas as árvores de expansão possíveis de um grafo.[17] Após a geração da UST, que garante um "esqueleto" conectado, arestas adicionais podem ser amostradas para atingir a densidade desejada.[17] Algoritmos como o de Wilson, que utiliza passeios aleatórios com eliminação de ciclos, são métodos canónicos para gerar USTs e, assim, garantir uma base conectada e imparcial para a construção de benchmarks de grafos.[18]

2. Perfis de Desempenho: Uma Visão Holística da Eficácia Algorítmica

A comparação do desempenho de múltiplos algoritmos de otimização através de um grande conjunto de problemas de benchmark gera uma quantidade esmagadora de dados. Tabelas que listam métricas como tempo de CPU ou número de avaliações de função para cada par de algoritmo-problema são difíceis de interpretar e podem levar a conclusões ambíguas.[21] Métricas agregadas simples, como o tempo médio de execução, podem ser enganadoras, pois podem ser fortemente influenciadas por um desempenho excecionalmente bom ou mau em alguns problemas atípicos. Para resolver este desafio, Dolan e Moré introduziram os Perfis de Desempenho, uma ferramenta de visualização robusta que fornece uma visão geral abrangente e matizada do desempenho comparativo de um conjunto de solvers.[21]

2.1 Construção Teórica

Um Perfil de Desempenho é essencialmente uma função de distribuição cumulativa que mostra a probabilidade de um solver estar dentro de um certo fator do melhor desempenho observado. A sua construção baseia-se em dois conceitos fundamentais: a métrica de desempenho e o rácio de desempenho.

- **A Métrica de Desempenho ($t_{p,s}$):** Para um conjunto de problemas P e um conjunto de solvers S, primeiro definimos uma métrica de desempenho $t_{p,s}$ que representa o "custo" para o solver $s \in S$ resolver o problema $p \in P$. Este custo é tipicamente o tempo de CPU ou o número de avaliações da função objetivo necessárias para satisfazer um critério de convergência.[25] Valores mais elevados de $t_{p,s}$ indicam um pior desempenho. Se um solver s não conseguir resolver um problema p, o seu custo é considerado infinito ($t_{p,s}=\infty$).
- O Rácio de Desempenho ($r_{p,s}$): Para normalizar o desempenho entre problemas de diferentes dificuldades, o rácio de desempenho é calculado para cada par problema-solver:

  $$r_{p,s}=\min\{t_{p,s'}:s' \in S\}t_{p,s}$$

  Este rácio compara o desempenho do solver s no problema p com o melhor desempenho obtido por qualquer solver nesse mesmo problema. Por definição, $r_{p,s} \geq 1$. Um valor de $r_{p,s}=1$ indica que o solver s foi o melhor (ou um dos melhores) para o problema p. Se um solver não conseguir resolver um problema, o seu rácio de desempenho é ∞.25
- A Função de Distribuição Cumulativa ($\rho_s(\tau)$): O perfil de desempenho para um solver s é então definido como a função de distribuição cumulativa (CDF) dos rácios de desempenho. Para um fator $\tau \geq 1$, $\rho_s(\tau)$ é a fração de problemas para os quais o rácio de desempenho do solver s foi no máximo τ:

  $$\rho_s(\tau)=|P|1|\{p \in P|r_{p,s} \leq \tau\}|$$

  Em termos probabilísticos, $\rho_s(\tau)$ representa a probabilidade de o solver s estar dentro de um fator τ do melhor solver num problema selecionado aleatoriamente do conjunto P.23

2.2 Um Guia para a Interpretação

Os Perfis de Desempenho são plotados com o fator τ (geralmente numa escala logarítmica) no eixo horizontal e a probabilidade cumulativa $\rho_s(\tau)$ no eixo vertical. A forma e a posição da curva de cada solver revelam informações cruciais sobre as suas características de desempenho.

- **Eficiência (ρs(1)):** O valor da curva no ponto de interceção com o eixo y (onde τ=1) é ρs(1). Este valor representa a fração de problemas para os quais o solver s foi o mais rápido. Um valor mais alto de ρs(1) indica um solver mais eficiente, que ganha a "corrida" com mais frequência.[26]
- **Robustez:** O limite da curva à direita (o valor de ρs(τ) para um τ grande) indica a fração total de problemas que o solver conseguiu resolver. Um solver que atinge um valor de ρs(τ)=1 é considerado totalmente robusto no conjunto de testes, pois resolveu todos os problemas. Um solver cuja curva se estabiliza num valor inferior a 1 falhou numa fração correspondente dos problemas.[23]
- **Desempenho Geral:** Um solver cuja curva se encontra consistentemente acima e à esquerda da curva de outro solver é inequivocamente superior nesse conjunto de benchmarks. Tem uma maior probabilidade de ser o melhor e de estar dentro de qualquer fator de desempenho do melhor.[23] A área sob a curva do perfil de desempenho pode ser usada como uma única métrica agregada que combina eficiência e robustez.

Além de uma simples classificação, a forma da curva de um perfil de desempenho pode revelar o "caráter" de um algoritmo. Uma subida inicial acentuada seguida por uma cauda longa e plana sugere um algoritmo que é muito rápido num subconjunto específico de problemas "fáceis", mas que tem dificuldades com instâncias mais difíceis. Em contraste, uma subida mais lenta e constante para um alto nível de robustez indica um solver de propósito mais geral e consistente. Esta análise da forma do perfil permite que um investigador selecione um algoritmo não apenas com base na velocidade, mas também nas características de desempenho desejadas, como a especialização versus a fiabilidade geral.

2.3 Limitações e Análise Aprofundada

Apesar da sua utilidade, os perfis de desempenho devem ser interpretados com cautela, pois têm limitações importantes.

- **O Problema da "Comparação com o Melhor":** Uma limitação crítica é que os perfis comparam cada solver com o melhor desempenho virtual em cada problema, e não diretamente entre si. Se o Solver A for o melhor em 50% dos problemas e o Solver B nos outros 50%, os seus perfis podem parecer semelhantes. No entanto, isto não nos diz quão pior o Solver A é nos problemas em que B ganha, e vice-versa. Uma comparação direta entre o segundo e o terceiro melhor solver, por exemplo, pode ser enganadora, pois o seu desempenho relativo pode ser ofuscado por um único solver dominante.[27]
- **Prática Recomendada: Perfis Iterativos:** Para mitigar esta limitação, uma técnica de análise avançada consiste em gerar uma série de perfis. Após a análise inicial, o solver com melhor desempenho geral é removido do conjunto de dados, e um novo perfil é gerado para os solvers restantes. Este processo pode ser repetido, permitindo

comparações diretas e justas entre os concorrentes de segundo e terceiro nível, e assim por diante.[27]

- **Sensibilidade ao Conjunto de Benchmarks:** É crucial lembrar que um perfil de desempenho é uma declaração sobre o desempenho num *conjunto específico de problemas*. A sua forma e as conclusões que dele se retiram podem mudar drasticamente com a adição ou remoção de problemas. Não representa uma verdade universal sobre o algoritmo.[23] Esta sensibilidade reforça a importância de selecionar conjuntos de benchmarks que sejam diversificados, representativos e de alta qualidade, como discutido na Seção 1.3.

A utilização de uma escala logarítmica para o eixo τ não é apenas uma conveniência de visualização; revela como os algoritmos lidam com o aumento da dificuldade dos problemas. Uma grande lacuna vertical entre duas curvas num valor elevado de τ indica que, quando um algoritmo é mais lento, tende a ser *muito* mais lento. Esta análise da inclinação do perfil em diferentes regiões do eixo τ fornece uma visão mais profunda sobre a escalabilidade algorítmica e o comportamento em instâncias difíceis.

3. Validação Estatística da Superioridade Algorítmica

Enquanto os Perfis de Desempenho oferecem uma poderosa visão qualitativa e agregada, as alegações de superioridade algorítmica exigem uma validação estatística rigorosa. A natureza estocástica de muitas heurísticas e a variabilidade do desempenho em diferentes instâncias de problemas tornam as comparações diretas de médias ou medianas insuficientes. Esta seção detalha o framework estatístico recomendado para comparar algoritmos, enfatizando por que os testes não paramétricos são a escolha preferida e descrevendo um fluxo de trabalho disciplinado para análises de dois e múltiplos algoritmos.

3.1 A Justificativa para Testes Não Paramétricos na Ciência Computacional

O trabalho seminal de Janez Demšar em 2006 forneceu uma crítica contundente ao uso inadequado de testes estatísticos paramétricos (como o teste t pareado ou ANOVA) em estudos de comparação de algoritmos.[29] Os testes paramétricos baseiam-se em suposições rigorosas sobre os dados subjacentes, que raramente são satisfeitas em cenários de benchmarking do mundo real:

- **Normalidade:** Testes como o teste t assumem que as diferenças de desempenho entre os algoritmos seguem uma distribuição normal. As métricas de desempenho em ciência da computação raramente aderem a esta suposição, e os testes de normalidade têm pouco poder em amostras pequenas (típicas em benchmarking, onde o número de conjuntos de dados pode ser de 10-30), tornando improvável a deteção de violações.[29]
- **Comensurabilidade e Homogeneidade de Variância:** A ANOVA assume que as variâncias são iguais entre os grupos (homocedasticidade) e que as medições em diferentes conjuntos de dados são comensuráveis (comparáveis numa escala comum). Estas suposições são difíceis de justificar quando se comparam, por exemplo, taxas de erro em domínios de problemas completamente diferentes.[29]
- **Sensibilidade a Outliers:** Os testes paramétricos são sensíveis a valores atípicos, que podem distorcer as médias e as variâncias, levando a conclusões erróneas.

Devido a estas violações frequentes das suposições, os **testes não paramétricos** são a alternativa mais segura, robusta e estatisticamente sólida. Estes testes não assumem uma distribuição subjacente específica para os dados. Em vez disso, operam sobre os *ranks* dos dados, o que os torna inerentemente robustos a outliers e a distribuições não normais.[29]

3.2 Comparando Dois Algoritmos: O Teste de Postos Sinalizados de Wilcoxon

Para a comparação direta e focada de dois algoritmos (Algoritmo A vs. Algoritmo B) através de múltiplos conjuntos de dados, o **Teste de Postos Sinalizados de Wilcoxon** é o método não paramétrico recomendado, servindo como alternativa ao teste t pareado.[30]

- **Caso de Uso:** Utilizado quando se pretende determinar se existe uma diferença de desempenho consistente e estatisticamente significativa entre dois algoritmos que foram testados nos mesmos N problemas.
- **Procedimento:**
  1. Para cada problema i, calcule a diferença de desempenho, $d_i = performance_{A,i} - performance_{B,i}$.
  2. Ignore as diferenças nulas ($d_i = 0$).
  3. Ordene os valores absolutos das diferenças restantes, $|d_i|$, do menor para o maior, e atribua-lhes postos (ranks). Em caso de empates, atribua o posto médio.
  4. Calcule a soma dos postos para as diferenças positivas ($R+$) e a soma dos postos para as diferenças negativas ($R-$).
  5. A estatística de teste, W, é o mínimo entre $R+$ e $R-$.
  6. Compare W com um valor crítico de uma tabela de Wilcoxon (para $N \leq 25$) ou use uma aproximação normal (para $N > 25$) para obter um p-valor.[36]
- **Interpretação:** A hipótese nula (H0) é que a mediana das diferenças é zero (ou seja, não

há diferença entre os algoritmos). Um p-valor abaixo do nível de significância escolhido (tipicamente $\alpha=0.05$) permite-nos rejeitar H0 e concluir que existe uma diferença estatisticamente significativa entre os dois algoritmos.

3.3 Comparando Múltiplos (k > 2) Algoritmos: Um Fluxo de Trabalho em Duas Etapas

Quando se comparam três ou mais algoritmos, realizar múltiplos testes de Wilcoxon par a par é estatisticamente incorreto. Esta abordagem inflaciona a taxa de erro Tipo I (o erro de detetar uma diferença que não existe), um problema conhecido como o problema das comparações múltiplas. A abordagem estatisticamente disciplinada é um fluxo de trabalho em duas etapas que controla o erro familiar (a probabilidade de cometer pelo menos um erro Tipo I em toda a família de testes).[29]

3.3.1 Etapa 1: Teste Omnibus com o Teste de Friedman

O **Teste de Friedman** é um teste omnibus não paramétrico, análogo à ANOVA de medidas repetidas. O seu objetivo é testar a hipótese nula global de que todos os k algoritmos têm um desempenho equivalente em todos os N conjuntos de dados.[29]

- **Procedimento:**
  1. Para cada um dos N conjuntos de dados, classifique (rank) os k algoritmos do melhor (posto 1) ao pior (posto k). Em caso de empates, atribua postos médios.
  2. Calcule o posto médio R_j para cada algoritmo j em todos os conjuntos de dados.
  3. Calcule a estatística de Friedman, χ_{F2}, ou, preferencialmente, a estatística de Iman-Davenport, F_F, que é menos conservadora e mais robusta para valores pequenos de k e N.[29]
  4. Compare a estatística de teste com a distribuição apropriada (χ^2 com $k-1$ graus de liberdade para χ_{F2}, ou a distribuição F com $k-1$ e $(k-1)(N-1)$ graus de liberdade para F_F) para obter um p-valor.
- **Interpretação:** Se o p-valor for significativo (p. ex., $p<0.05$), rejeitamos a hipótese nula global. Isto indica que existe pelo menos uma diferença significativa no desempenho *entre algum par* de algoritmos. Este resultado serve como um "porteiro", justificando a continuação para uma análise post-hoc para identificar onde residem essas diferenças. Se o teste de Friedman não for significativo, a análise termina aqui.

3.3.2 Etapa 2: Análise Post-Hoc Par a Par com o Teste de Nemenyi

Após um resultado significativo no teste de Friedman, um teste **post-hoc** é necessário para realizar comparações par a par de forma controlada. O **Teste de Nemenyi** é o procedimento recomendado para comparar todos os algoritmos entre si.[38]

- Procedimento: O teste de Nemenyi calcula uma única Diferença Crítica (CD). O desempenho de dois algoritmos é considerado significativamente diferente se a diferença entre os seus postos médios ($|Ri−Rj|$) for maior que a CD. A CD é calculada como:

  $CD=qα6Nk(k+1)$

  onde $qα$ é um valor crítico baseado na distribuição de amplitude Studentizada para o nível de significância $α$ e k algoritmos.29
- **Visualização com Diagramas de Diferença Crítica:** Os resultados do teste de Nemenyi são melhor apresentados visualmente através de **Diagramas de Diferença Crítica**. Nestes diagramas, os algoritmos são plotados numa linha numérica de acordo com os seus postos médios. Uma barra horizontal é desenhada para conectar grupos de algoritmos cuja diferença de posto médio é *menor* que a CD. Quaisquer dois algoritmos que não estejam conectados por uma barra são considerados como tendo um desempenho estatisticamente diferente.[33]

3.4 Para Além da Significância: Medindo o Tamanho do Efeito

Um p-valor, embora essencial, apenas nos informa se um efeito (uma diferença) é estatisticamente improvável de ter ocorrido por acaso; não nos diz quão *grande* ou *praticamente importante* é esse efeito. Uma análise verdadeiramente completa deve complementar os testes de significância com uma medida do **tamanho do efeito**.

A **Estatística A12 de Vargha-Delaney** é uma medida de tamanho de efeito não paramétrica, robusta e intuitiva, ideal para a comparação de algoritmos.[42]

- **Interpretação:** A estatística A12 mede a probabilidade de um algoritmo (A) obter um resultado melhor do que outro algoritmo (B) numa instância de problema escolhida aleatoriamente.
  - $AA,B=0.5$ implica que ambos os algoritmos têm a mesma probabilidade de ganhar.

    - $A_{A,B}>0.5$ implica que o algoritmo A é estocasticamente superior ao B.
    - $A_{A,B}=0.8$ significa que, se escolhermos uma execução aleatória de A e uma de B, há uma probabilidade de 80% de A ter um desempenho melhor.
- **Cálculo:** Pode ser calculado a partir dos postos usados no teste de Mann-Whitney U (a contrapartida não pareada do teste de Wilcoxon).[42]
- **Magnitude:** Existem limiares estabelecidos para interpretar a magnitude do efeito: um valor de A12 entre 0.5 e 0.56 é considerado "insignificante", entre 0.56 e 0.64 é "pequeno", entre 0.64 e 0.71 é "médio", e acima de 0.71 é "grande".[42]

A combinação de um teste de significância (Friedman/Nemenyi) e uma medida de tamanho de efeito (A12) fornece uma imagem completa. Por exemplo, uma diferença estatisticamente significativa ($p<0.05$) com um tamanho de efeito insignificante ($A_{12}=0.53$) sugere que, embora uma diferença exista, pode não ser praticamente relevante. Por outro lado, um grande tamanho de efeito ($A_{12}=0.8$) com um p-valor marginal ($p=0.07$) pode indicar uma tendência forte que justifica uma investigação mais aprofundada com mais dados.

Tabela 1: Resumo dos Testes Estatísticos Recomendados para Comparação de Algoritmos

Nome do Teste	Número de Algoritmos Comparados	Caso de Uso	Hipótese Nula (H0)
Teste de Postos Sinalizados de Wilcoxon	2	Comparação direta par a par em múltiplos conjuntos de dados.	A mediana das diferenças de desempenho entre os dois algoritmos é zero.
Teste de Friedman	$k\geq2$	Teste omnibus para detetar se existe *alguma* diferença significativa entre um grupo de k algoritmos.	Todos os k algoritmos têm o mesmo desempenho (os seus postos médios são iguais).
Teste de Nemenyi	$k>2$	Teste post-hoc para identificar *quais pares* de algoritmos diferem	O desempenho de dois algoritmos específicos é o mesmo (a diferença

		significativamente, após uma rejeição de H0 pelo Teste de Friedman.	dos seus postos médios é zero).

Tabela 2: Dados de Desempenho de Amostra e Classificação para Ilustração do Teste de Friedman

Instância do Benchmark	Algo A (Tempo)	Algo A (Posto)	Algo B (Tempo)	Algo B (Posto)	Algo C (Tempo)	Algo C (Posto)
bench_01	10.5	1	12.3	2	15.1	3
bench_02	25.2	2	22.1	1	28.4	3
bench_03	8.9	1	10.1	2	10.3	3
bench_04	45.6	3	40.2	1	42.5	2
bench_05	18.3	1	20.5	3	19.9	2
bench_06	33.1	2	30.5	1	35.0	3
bench_07	12.0	1	12.0	1.5	14.2	3
bench_08	5.4	1	6.8	2	7.1	3
bench_09	60.1	3	55.8	2	54.2	1

bench_10	22.5	2	20.1	1	24.6	3
bench_11	14.8	1	16.2	2	18.9	3
bench_12	38.2	3	35.5	2	34.1	1
bench_13	9.5	1	11.3	3	10.8	2
bench_14	17.4	2	16.9	1	19.2	3
bench_15	29.8	3	25.1	1	27.7	2
Soma dos Postos		**30.0**		**25.5**		**39.5**
Posto Médio (Rj)		**2.00**		**1.70**		**2.63**

Nota: Os dados de tempo são hipotéticos para fins de ilustração. O posto 1 é atribuído ao menor tempo (melhor desempenho). O empate na instância bench_07 resulta em postos médios de (1+2)/2 = 1.5.

---

4. Caracterização do Desempenho Estocástico com Análise de Tempo para o Alvo

Algoritmos estocásticos, como algoritmos genéticos, simulated annealing ou otimização por enxame de partículas, introduzem aleatoriedade no seu processo de busca. Consequentemente, execuções repetidas do mesmo algoritmo na mesma instância do problema podem produzir soluções de qualidade variável em tempos de execução diferentes. Esta variabilidade torna as métricas de ponto único, como o tempo médio ou o melhor resultado de N execuções, insuficientes e potencialmente enganadoras para uma caracterização completa do desempenho.[45] Para capturar verdadeiramente o comportamento de um algoritmo estocástico, é necessário analisar a sua

distribuição de tempo de execução completa. Os gráficos de Tempo para o Alvo (Time-to-Target - TTT) são a ferramenta padrão para esta análise.[46]

4.1 A Lógica das Distribuições de Tempo de Execução

Em vez de perguntar "Qual o tempo médio que o algoritmo leva?", uma abordagem mais informativa é perguntar "Qual é a probabilidade de o algoritmo encontrar uma solução de uma certa qualidade dentro de um determinado orçamento de tempo?". Os gráficos TTT respondem a esta última pergunta, fornecendo uma imagem muito mais rica do desempenho de um algoritmo, incluindo a sua velocidade, fiabilidade e consistência.

4.2 Construindo Gráficos de Tempo para o Alvo

Um gráfico TTT é uma visualização da **Função de Distribuição Cumulativa Empírica (ECDF)** do tempo de execução necessário para atingir um valor-alvo de qualidade de solução pré-especificado.[46] A sua construção segue um procedimento experimental rigoroso:

1. **Definir o Cenário:** Para uma única instância de problema e um único valor-alvo de qualidade de solução (por exemplo, um corte de grafo de tamanho ≤100), realize N execuções independentes do algoritmo estocástico. Cada execução deve ser iniciada com uma semente de gerador de números aleatórios diferente para garantir a independência.[46]
2. **Recolher os Tempos de Sucesso:** Para cada execução i que atinge o valor-alvo, registe o tempo (ou número de avaliações) necessário, ti.
3. **Gerir as Execuções Sem Sucesso:** Se uma execução não atingir o valor-alvo dentro de um orçamento de tempo máximo pré-definido (por exemplo, 1 hora de CPU), a execução é terminada. Estas são execuções sem sucesso, e os dados resultantes são conhecidos como **dados censurados**.
4. **Construir a ECDF:**
    - Seja k o número de execuções bem-sucedidas (onde $k \leq N$).
    - Ordene os tempos de sucesso em ordem crescente: $t(1) \leq t(2) \leq \cdots \leq t(k)$.
    - A ECDF é então um gráfico de degraus onde, para cada tempo de sucesso t(i), a probabilidade cumulativa é plotada como i/N. A utilização de N (o número total de execuções) no denominador, em vez de k (o número de sucessos), é o passo crucial para lidar corretamente com os dados censurados.[49]

A forma de um gráfico TTT fornece uma "impressão digital" rica do comportamento de um

algoritmo estocástico. Uma curva muito íngreme indica baixa variância no tempo de execução, significando um desempenho consistente. Uma curva longa e pouco inclinada indica alta variância, ou seja, um desempenho imprevisível. Uma curva que sobe rapidamente mas se estabiliza a um nível baixo sugere um algoritmo que encontra soluções fáceis rapidamente mas fica facilmente preso em ótimos locais.

4.3 A Questão Crítica das Execuções Sem Sucesso (Dados Censurados)

A forma correta de lidar com as execuções sem sucesso é fundamental para uma interpretação precisa dos gráficos TTT. Uma abordagem ingénua que simplesmente ignora as falhas e constrói a ECDF apenas a partir das execuções bem-sucedidas (ou seja, plotando i/k) inflacionaria artificialmente o desempenho percebido do algoritmo, pois mascara a sua taxa de falha.

A metodologia padrão da indústria, exemplificada pelo framework de benchmarking **COCO (Comparing Continuous Optimizers)**, fornece a abordagem correta.[49] O tratamento de dados censurados nos gráficos TTT/ECDF é o seguinte:

- **As execuções sem sucesso determinam a altura final da curva:** A ECDF é construída usando apenas os tempos das k execuções bem-sucedidas. No entanto, a função cumulativa é normalizada pelo número total de execuções, N.
- **Interpretação do Nível de Saturação:** Como resultado, a altura máxima (ou assíntota direita) que a curva ECDF atinge é k/N. Este valor representa diretamente a **taxa de sucesso** do algoritmo para aquele par específico de problema-alvo. Por exemplo, se uma curva se estabiliza (satura) a uma altura de y=0.8, isso significa que o algoritmo conseguiu atingir o alvo em 80% das N execuções dentro do orçamento de tempo alocado.[52]

Esta abordagem integra elegantemente duas dimensões críticas do desempenho estocástico num único gráfico: a distribuição de velocidade (a forma da curva ascendente) e a robustez ou fiabilidade (a altura final da curva). Formalmente, esta abordagem é uma aplicação prática do **estimador de Kaplan-Meier**, o método estatístico canónico para estimar funções de sobrevivência (o complemento da ECDF) a partir de dados com censura.[53]

4.4 Interpretação Comparativa dos Gráficos TTT

A principal utilidade dos gráficos TTT reside na comparação de múltiplos algoritmos. Ao sobrepor as ECDFs de diferentes algoritmos no mesmo gráfico, podem ser tiradas conclusões matizadas:

- **Dominância:** Um algoritmo cuja curva está consistentemente acima e à esquerda de outro é superior em todos os aspetos: tem uma maior probabilidade de encontrar a solução e fá-lo mais rapidamente.
- **Compromisso Velocidade vs. Robustez:** É comum observar compromissos. O Algoritmo A pode ter uma subida inicial mais íngreme (sendo mais rápido para encontrar soluções "fáceis"), mas estabilizar-se a uma altura mais baixa (sendo menos fiável). O Algoritmo B pode ser mais lento no início, mas atingir uma taxa de sucesso final mais elevada (sendo mais robusto). A escolha entre A e B dependeria então dos requisitos da aplicação: velocidade ou garantia de solução.

O valor-alvo é um parâmetro livre que pode ser explorado para uma análise mais profunda. Em vez de gerar um único gráfico TTT para um alvo fácil, uma análise completa envolve a geração de uma série de gráficos para alvos progressivamente mais difíceis (por exemplo, soluções que estão a 10%, 5%, 1% e 0.1% do ótimo conhecido). Esta análise multi-alvo revela como o desempenho e a robustez de um algoritmo se degradam à medida que o problema se torna mais difícil, fornecendo uma caracterização muito mais completa do que um único gráfico TTT.

5. Um Framework Integrado para Benchmarking Reprodutível

As metodologias discutidas nas seções anteriores — os pilares da reprodutibilidade, as ferramentas de visualização agregada como os Perfis de Desempenho, a validação estatística rigorosa e a caracterização detalhada do desempenho estocástico através de gráficos TTT — não são técnicas isoladas. Elas formam os componentes de um framework integrado e coeso para a condução de investigações experimentais de ponta em ciência computacional. Esta seção final sintetiza estes componentes num fluxo de trabalho unificado e acionável, demonstrando como uma abordagem holística eleva uma simples comparação de desempenho a uma contribuição científica robusta, defensável e valiosa.

5.1 Um Fluxo de Trabalho Unificado

Um projeto de benchmarking exemplar pode ser estruturado em quatro fases distintas, cada uma baseada nos princípios estabelecidos anteriormente.

Fase 1: Configuração e Desenho Experimental
Esta fase inicial estabelece as bases para todo o projeto e é a mais crítica para garantir a reprodutibilidade.

- **Adotar os Cinco Pilares:** O projeto deve começar com a implementação dos Cinco Pilares da Reprodutibilidade (Seção 1.2). Isto envolve:
  - Inicializar um repositório **Git** para controlo de versão de todo o código e documentação.
  - Configurar o **Git LFS** para gerir ficheiros de dados grandes, modelos ou outros artefactos binários.[7]
  - Definir o ambiente computacional usando uma ferramenta como **Poetry**, que cria um ficheiro pyproject.toml e poetry.lock para garantir que as dependências exatas possam ser recriadas.[5,6]
  - Estruturar o projeto para usar um sistema de automação de fluxo de trabalho como o **Make**, criando um Makefile que define alvos para cada etapa da análise (por exemplo, make data, make train, make plots).[6]
- **Selecionar Benchmarks:** Escolher um conjunto de benchmarks diversificado e representativo que inclua tanto instâncias do mundo real (por exemplo, da coleção LDBC Graphalytics) como instâncias sintéticas (por exemplo, geradas com os modelos LFR ou ABCD) para testar tanto a relevância prática como a escalabilidade controlada (Seção 1.3).[10]

Fase 2: Execução e Recolha de Dados
Com a estrutura do projeto estabelecida, os experimentos são executados de forma automatizada.

- **Execução Automatizada:** Utilize o Makefile para executar todo o pipeline experimental. Isto garante que o processo é documentado em código e pode ser repetido com um único comando.
- **Recolha de Dados Abrangente:** Para algoritmos estocásticos, não registe apenas a melhor ou a média das execuções. Guarde os dados brutos de cada execução individual, incluindo o tempo para atingir o alvo (ou a falha em fazê-lo) e a semente do gerador de números aleatórios utilizada.[6] Isto é essencial para a análise TTT. Os resultados devem ser guardados em formatos de dados padronizados (por exemplo, CSV, JSON) para facilitar o pós-processamento.

Fase 3: Análise e Visualização
Esta fase foca-se na transformação dos dados brutos em insights compreensíveis.

- **Análise ao Nível da Instância (para Algoritmos Estocásticos):** Comece por gerar **gráficos TTT** para instâncias e alvos representativos (Seção 4). Isto fornece uma compreensão profunda do comportamento estocástico, incluindo a distribuição do tempo de execução e as taxas de sucesso para cada algoritmo.

- **Análise Agregada:** Calcule as métricas de desempenho agregadas (por exemplo, tempo médio de sucesso, qualidade média da solução) em todos os benchmarks.
- **Visualização Holística:** Crie **Perfis de Desempenho Dolan-Moré** para obter uma visão geral de alto nível da eficiência e robustez comparativas de todos os algoritmos em todo o conjunto de benchmarks (Seção 2). Use estes perfis para identificar visualmente os principais concorrentes e formular hipóteses sobre o seu desempenho relativo.

Fase 4: Confirmação Estatística e Relato
A fase final utiliza testes estatísticos formais para validar as observações da fase de visualização.

- **Teste de Hipóteses:** Aplique o fluxo de trabalho estatístico não paramétrico apropriado (Seção 3):
  - Se comparar mais de dois algoritmos, comece com o **Teste de Friedman** para verificar se existem diferenças significativas em geral.
  - Se o Teste de Friedman for significativo, prossiga com o **Teste de Nemenyi** post-hoc para identificar quais pares de algoritmos diferem significativamente. Visualize os resultados com um **Diagrama de Diferença Crítica**.
  - Se comparar apenas dois algoritmos, utilize o **Teste de Postos Sinalizados de Wilcoxon**.
- **Quantificar a Magnitude:** Para cada comparação par a par, calcule a **estatística A12 de Vargha-Delaney** para medir o tamanho do efeito, fornecendo uma medida intuitiva da magnitude da diferença de desempenho.
- **Relato Transparente:** O relatório final ou publicação deve incluir todas as visualizações (Perfis de Desempenho, gráficos TTT, Diagramas de Diferença Crítica), os resultados dos testes estatísticos (p-valores) e as medidas de tamanho do efeito (valores A12). Mais importante, deve fornecer um link para o repositório público contendo todo o código, dados (ou scripts para os obter), definições de ambiente e fluxos de trabalho automatizados, permitindo que a comunidade científica verifique e construa sobre o trabalho.

5.2 Observações Finais

A era de alegações de desempenho baseadas em tabelas de médias ou em gráficos anedóticos chegou ao fim. O padrão moderno para a ciência computacional experimental exige uma abordagem multifacetada que integre práticas de reprodutibilidade desde o início, utilize ferramentas de visualização sofisticadas para explorar dados de desempenho complexos e, finalmente, valide as conclusões com métodos estatísticos robustos e apropriados. A combinação de Perfis de Desempenho para uma visão geral, gráficos de Tempo para o Alvo para uma análise estocástica profunda e o fluxo de trabalho de testes não paramétricos de Friedman-Nemenyi para uma validação rigorosa, tudo dentro de um

framework de investigação totalmente reprodutível, não é apenas uma boa prática — é a essência do método científico aplicado ao domínio computacional. Adotar este framework integrado é o que distingue uma comparação de desempenho superficial de uma contribuição científica duradoura e credível.

Referências citadas

1. The five pillars of computational reproducibility: Bioinformatics and beyond - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/371671830_The_five_pillars_of_computational_reproducibility_Bioinformatics_and_beyond
2. five pillars of computational reproducibility: bioinformatics and ..., acessado em agosto 29, 2025, https://academic.oup.com/bib/article/24/6/bbad375/7326135
3. The five pillars of computational reproducibility: bioinformatics and beyond - OUCI, acessado em agosto 29, 2025, https://ouci.dntb.gov.ua/en/works/7BKpE5Z4/
4. The five pillars of computational reproducibility. - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/figure/The-five-pillars-of-computational-reproducibility_fig1_374922562
5. The five pillars of computational reproducibility: bioinformatics and beyond - PubMed, acessado em agosto 29, 2025, https://pubmed.ncbi.nlm.nih.gov/37870287/
6. Ten Simple Rules for Reproducible Computational Research - PMC, acessado em agosto 29, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC3812051/
7. DVC vs Git vs Git LFS: ML Reproducibility - Censius, acessado em agosto 29, 2025, https://censius.ai/blogs/dvc-vs-git-and-git-lfs-in-machine-learning-reproducibility
8. Git LFS and DVC: The Ultimate Guide to Managing Large Artifacts in MLOps - Medium, acessado em agosto 29, 2025, https://medium.com/@pablojusue/git-lfs-and-dvc-the-ultimate-guide-to-managing-large-artifacts-in-mlops-c1c926e6c5f4
9. Best Practices for Comparing Optimization Algorithms - arXiv, acessado em agosto 29, 2025, https://arxiv.org/pdf/1709.08242
10. LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis ..., acessado em agosto 29, 2025, https://www.vldb.org/pvldb/vol9/p1317-iosup.pdf
11. Stochastic block model - Wikipedia, acessado em agosto 29, 2025, https://en.wikipedia.org/wiki/Stochastic_block_model
12. Lancichinetti–Fortunato–Radicchi benchmark - Wikipedia, acessado em agosto 29, 2025, https://en.wikipedia.org/wiki/Lancichinetti%E2%80%93Fortunato%E2%80%93Radicchi_benchmark
13. LFRBenchmarkGraphs.jl - Julia Packages, acessado em agosto 29, 2025, https://juliapackages.com/p/lfrbenchmarkgraphs

14. Class LFRGenerator - NetworKit, acessado em agosto 29, 2025, https://networkit.github.io/dev-docs/cpp_api/classNetworKit_1_1LFRGenerator.html
15. LFR_benchmark_graph — NetworkX 3.5 documentation, acessado em agosto 29, 2025, https://networkx.org/documentation/stable/reference/generated/networkx.generators.community.LFR_benchmark_graph.html
16. Artificial Benchmark for Community Detection (ABCD)—Fast random ..., acessado em agosto 29, 2025, https://www.cambridge.org/core/services/aop-cambridge-core/content/view/453FE29A1FA3C1798B0EC116587FE422/S2050124220000454a.pdf/artificial-benchmark-for-community-detection-abcdfast-random-graph-model-with-community-structure.pdf
17. DAGmar: Library for DAGs - Infosun, acessado em agosto 29, 2025, https://www.infosun.fim.uni-passau.de/~chris/down/MIP-1202.pdf
18. Wilson's Algorithm · GitHub, acessado em agosto 29, 2025, https://gist.github.com/mbostock/11357811
19. 1 Generating Uniform Spanning Trees, acessado em agosto 29, 2025, https://n.ethz.ch/~ywigderson/math/static/UniformSpanningTrees.pdf
20. Loop-erased random walk - Wikipedia, acessado em agosto 29, 2025, https://en.wikipedia.org/wiki/Loop-erased_random_walk
21. Benchmarking optimization software with performance profiles. (Journal Article) | OSTI.GOV, acessado em agosto 29, 2025, https://www.osti.gov/biblio/943229
22. Benchmarking Optimization Software with Performance Profiles, acessado em agosto 29, 2025, https://www.researchgate.net/publication/1853672_Benchmarking_Optimization_Software_with_Performance_Profiles
23. Performance Profile Benchmarking Tool - Tangi Migot, acessado em agosto 29, 2025, https://tmigot.github.io/posts/2024/06/teaching/
24. Dolan, E.D. and Moré, J.J. (2002) Benchmarking Optimization Software with Performance Profiles. Mathematical Programming, 91, 201-213. - References, acessado em agosto 29, 2025, https://www.scirp.org/reference/referencespapers?referenceid=2663487
25. benchmarking derivative-free optimization algorithms - CiteSeerX, acessado em agosto 29, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=90319b0b2a3c700015303fa2f30dcdc22f081ac1
26. Introduction to Performance Profile | Abel Soares Siqueira, acessado em agosto 29, 2025, https://abelsiqueira.com/blog/2017-05-09-introduction-to-performance-profile/
27. A note on performance profiles for benchmarking software - ePubs, acessado em agosto 29, 2025, https://epubs.stfc.ac.uk/manifestation/20477017/RAL-P-2015-004.pdf
28. A Note on Performance Profiles for Benchmarking Software | Request PDF - ResearchGate, acessado em agosto 29, 2025,

https://www.researchgate.net/publication/310824952_A_Note_on_Performance_Profiles_for_Benchmarking_Software
29. Statistical Comparisons of Classifiers over Multiple Data Sets, acessado em agosto 29, 2025, https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf
30. (PDF) Statistical Comparisons of Classifiers over Multiple Data Sets (2006) | Janez Demšar | 12256 Citations - SciSpace, acessado em agosto 29, 2025, https://scispace.com/papers/statistical-comparisons-of-classifiers-over-multiple-data-18bm1e3eph
31. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons - Journal of Machine Learning Research, acessado em agosto 29, 2025, https://jmlr.csail.mit.edu/papers/volume9/garcia08a/garcia08a.pdf
32. Statistical Comparisons of Classifiers over Multiple Data Sets - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/220320196_Statistical_Comparisons_of_Classifiers_over_Multiple_Data_Sets
33. Statistical Comparisons of Classifiers over Multiple Data Sets, acessado em agosto 29, 2025, https://jmlr.org/papers/v7/demsar06a.html
34. Understanding the Wilcoxon Sign Test - Statistics Solutions, acessado em agosto 29, 2025, https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/wilcoxon-sign-test/
35. Wilcoxon signed-rank test - Wikipedia, acessado em agosto 29, 2025, https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test
36. Wilcoxon Signed Rank Test - GeeksforGeeks, acessado em agosto 29, 2025, https://www.geeksforgeeks.org/machine-learning/wilcoxon-signed-rank-test/
37. Statistics: 2.2 The Wilcoxon signed rank sum test - Statstutor, acessado em agosto 29, 2025, https://www.statstutor.ac.uk/resources/uploaded/wilcoxonsignedranktest.pdf
38. A Graphical Approach for Friedman Test: Moments Approach - arXiv, acessado em agosto 29, 2025, https://arxiv.org/pdf/2202.09131
39. How to Show that Your Model is Better: A Basic Guide to Statistical Hypothesis Testing, acessado em agosto 29, 2025, https://lab.rivas.ai/?p=2665
40. Nemenyi test - Wikipedia, acessado em agosto 29, 2025, https://en.wikipedia.org/wiki/Nemenyi_test
41. Friedman test and Nemenyi post-hoc test when comparing the time spent... - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/figure/Friedman-test-and-Nemenyi-post-hoc-test-when-comparing-the-time-spent-by-the-different-AO_fig5_346555784
42. Common Language Effect Size - PS, acessado em agosto 29, 2025, https://peterstatistics.com/Terms/EffectSizes/CommonLanguageEffectSize.html
43. Wilcoxon Rank Sum Test and Vargha-Delaney Â12 Measure for the Traditional Metrics-based Models. - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/figure/Wilcoxon-Rank-Sum-Test-and-Vargha-Delaney-A12-Measure-for-the-Traditional-Metrics-based_tbl3_328210662

44. Average Vargha-Delaney effect sizes b A12 of each algorithm (in a row)... - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/figure/Average-Vargha-Delaney-effect-sizes-b-A12-of-each-algorithm-in-a-row-with-another_tbl3_349080081
45. How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/226547420_How_to_assess_and_report_the_performance_of_a_stochastic_algorithm_on_a_benchmark_problem_Mean_or_best_result_on_a_number_of_runs
46. Extending time-to-target plots to multiple instances and targets ... - UFF, acessado em agosto 29, 2025, http://www.dcc.ic.uff.br/~celso/artigos/mtttplot-mic2017.pdf
47. tttplots-compare: a perl program to compare time-to-target plots or general runtime distributions of randomized algorithms | Request PDF - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/271741508_tttplots-compare_a_perl_program_to_compare_time-to-target_plots_or_general_runtime_distributions_of_randomized_algorithms
48. tttplots: a perl program to create time-to-target plots - UFF, acessado em agosto 29, 2025, http://profs.ic.uff.br/~celso/artigos/tttplots.pdf
49. (PDF) COCO – COmparing Continuous Optimizers : The Documentation - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/267853289_COCO_-_COmparing_Continuous_Optimizers_The_Documentation
50. COCO: Performance Assessment, acessado em agosto 29, 2025, https://numbbo.github.io/coco-doc/perf-assessment/
51. (PDF) COCO: Performance Assessment - ResearchGate, acessado em agosto 29, 2025, https://www.researchgate.net/publication/302951878_COCO_Performance_Assessment
52. Volume 13, Issue 4, acessado em agosto 29, 2025, https://evolution.sigevo.org/issues/HTML/sigevolution-13-4/index.htm
53. ecdf - Empirical cumulative distribution function - MATLAB - MathWorks, acessado em agosto 29, 2025, https://www.mathworks.com/help/stats/ecdf.html
54. ecdf — SciPy v1.16.1 Manual, acessado em agosto 29, 2025, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ecdf.html
55. Compute an ECDF for Censored Data - R, acessado em agosto 29, 2025, https://search.r-project.org/CRAN/refmans/NADA/html/cenfit.html
56. Support for Git Repository Packages which have LFS · Issue #8723 · python-poetry/poetry, acessado em agosto 29, 2025, https://github.com/python-poetry/poetry/issues/8723