DISSERTAÇÃO DE MESTRADO Nº 930

ANÁLISE EXPERIMENTAL DE OPERADORES DE RECOMBINAÇÃO PARA O ALGORITMO DE EVOLUÇÃO DIFERENCIAL

Moisés de Matos Botelho

DATA DA DEFESA: 15/07/2016

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

ANÁLISE EXPERIMENTAL DE OPERADORES DE RECOMBINAÇÃO PARA O ALGORITMO DE EVOLUÇÃO DIFERENCIAL

Moisés de Matos Botelho

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Felipe Campelo França Pinto

Belo Horizonte - MG

Julho de 2016

Botelho, Moisés de Matos.
B748a Análise experimental de operadores de recombinação para o algoritmo de evolução diferencial [manuscrito] / Moisés de Matos Botelho. – 2016.
xvii, 78 f., enc.: il.

Orientador: Felipe Campelo França Pinto.

Dissertação (mestrado) Universidade Federal de Minas Gerais, Escola de Engenharia.

Anexos: f. 47-72.

Bibliografia: f. 73-78.

1. Engenharia elétrica - Teses. I. Pinto, Felipe Campelo França. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

"Análise Experimental de Operadores de Recombinação para o Algoritmo de Evolução Diferencial"

Moisés de Matos Botelho

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 15 de julho de 2016.

Por:

Prof. Dr. Felipe Campelo França Pinto
DEE (UFMG)

Prof. Dr. Lucas de Souza Batista
DEE (UFMG)

Prof. Dr. Rodney Rezende Saldanha
DEE (UFMG)

Agradecimentos

Um sonho só pode ser realizado se existirem pessoas que acreditam que ele possa se realizar. E é graças a essas pessoas que isto se tornou real: a conclusão do meu mestrado.

Agradeço imensamente ao meu orientador, Felipe Campelo, pela dedicação, apoio e por acreditar que eu seria capaz de concluir este mestrado. Tenho a certeza que Deus coloca as pessoas certas no nosso caminho, e agradeço a Ele por colocar essas pessoas.

Também gostaria de deixar meus agradecimentos à galera do Orcs que além de sempre me apoiarem, tornaram-se pessoas importantes na minha vida. Obrigado a todos.

À minha família, que sempre me apoiou em todos os momentos da minha vida, em especial a minha mãe Zilena (mulher da minha vida). Aos amigos e colegas de trabalho que sempre acreditaram em mim, e à Prodemge pela facilidade que me deu para poder cursar as disciplinas. Ao meu gerente Eduardo Eustáquio, que me deu total apoio juntamente com a superintendente Edna Canabrava.

As amizades que fiz na UFMG, que vou levar pro resto da vida.

Obrigado a todos.

"O mundo tem muitas coisas boas a oferecer para quem tem a ousadia de buscar".

—ZÍBIA GASPARETTO

Resumo

A busca por melhores resultados é um dos objetivos constantes da engenharia, o que leva a um grande interesse no desenvolvimento de técnicas de otimização eficientes, que possibilitem a solução rápida e efetiva de problemas. Dentre os diversos métodos utilizados para a solução de problemas de otimização em engenharia, os algoritmos evolutivos (AEs) são amplamente utilizados. Destes, o método de Evolução Diferencial (DE) tem demonstrado um ótimo desempenho em problemas contínuos de pequena e média escala, oferecendo um balanço interessante de simplicidade de implementação e qualidade das soluções encontradas.

Assim como a maioria dos AEs, o desempenho do DE é fortemente influenciado pelos tipos de operadores de recombinação e mutação utilizados, bem como pelos valores atribuídos às constantes do algoritmo. Este trabalho apresenta uma investigação sistemática dos efeitos de dezesseis operadores de recombinação para espaços de codificação real no desempenho do método de Evolução Diferencial. Uma notação unificada dos operadores em termos de operações matemáticas na forma vetorizada é apresentada, e uma implementação padronizada destes é apresentada na forma de um pacote computacional em linguagem R. Com isto, espera-se simplificar a análise das semelhanças e diferenças entre os operadores, bem como a compreensão do seu efeito sob uma população de soluções candidatas. Pretende-se também proporcionar uma plataforma na qual futuros operadores possam ser incorporados e avaliados.

A comparação experimental dos operadores de recombinação investigados neste trabalho é realizada utilizando vinte e oito problemas de teste, e os resultados são utilizados para discutir possíveis direções promissoras no desenvolvimento de operadores melhorados para o método de evolução diferencial. Os resultados desta comparação sugerem a utilização de operadores de recombinação até o momento não associados ao método de Evolução Diferencial como forma de melhorar o desempenho esperado deste algoritmo em problemas de 5 e 20 dimensões.

Palavras-chave: Evolução Diferencial, operadores de recombinação, análise experimental, pacote R.

Abstract

The search for improved solutions is a constant goal in all fields of engineering. Consequently, there is a great interest in the development of robust and efficient optimization techniques, to enable the solution of a wide range of problems in a fast and effective fashion. Amongst the algorithms commonly employed for the solution of optimization problems in engineering, Evolutionary Algorithms (EAs) are widespread across several disciplines. More specifically, the method known as Differential Evolution (DE) has yielded good performance for continuous optimization problems, particularly in low-dimensional spaces, and represents an interesting balance between simplicity of implementation and quality of the solutions found.

As is the case with the vast majority of EA approaches, the performance of DE is strongly influenced by the specific variation operators (recombination and mutation) employed, as well as by the values used for the algorithm parameters. This work presents a systematic investigation of the effects of sixteen recombination operators for real-coded spaces on the performance of differential evolution. A unified notation is adopted for all operators, based on vector operations, and a modular, standardized implementation is presented in the form of an open-source package in R. The goal is to simplify the analysis of similarities among different operator variants, as well as the understanding of their effects on a population of candidate solutions. Another objective is to provide a standardized platform for the development and evaluation of new operators for Differential Evolution.

To investigate and compare the performance of the recombination operators investigated in this work, a 28-problem benchmark set is used with dimensions 5 and 20. The results are discussed in terms of promising directions for the development of new operators for DE. The results also suggest that some recombination variants so far unused in the DE literature may be recommended in terms of improving the expected performance of the algorithm in problems of the two dimensions tested.

Keywords: Differential evolution, recombination operators, experimental analysis, R packages.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A otimização busca encontrar soluções para problemas onde se deseje alcançar a maximização ou minimização de uma ou mais grandezas, sendo utilizada extensivamente em diversas áreas da ciência e engenharia. Diversos problemas podem ser modelados visando a otimização, como por exemplo: controle de perdas de um sistema elétrico, controle de tensão, projeto de circuitos eletrônicos, planejamento de produção (Pinheiro, 1998).

Na otimização temos a programação contínua, em que os parâmetros do problema assumem quaisquer valores em uma região do espaço, que é habitualmente dividida em programação linear e não-linear (Nocedal e Wright, 2006). Técnicas de programação não-linear podem ser divididas em dois tipos de métodos: determinísticos e estocásticos, podendo ser com ou sem restrições. Métodos determinísticos, a partir de um ponto inicial determinado, possuem como resultado sempre a mesma solução, já os métodos estocásticos, a partir de um determinado ponto inicial, não garantem que a solução seja a mesma para cada execução. Em diversas aplicações de engenharia, para as quais não há modelos específicos ou cuja modelagem envolva não-linearidades, descontinuidade, ou não-diferenciabilidades, métodos estocásticos são usualmente aplicados como estratégias preferenciais (Gomes *et al.*, 2014).

Classificados usualmente em três grupos, os métodos de otimização não-linear são divididos em (Takahashi *et al.*, 2010):

- Métodos de direção de busca;
- Métodos de exclusão de semi-espaços;
- Métodos de busca por populações.

Dentre os métodos de otimização, os de busca por populações, em particular os Algoritmos Evolutivos, trabalham com um conjunto de soluções de forma simultânea, avaliadas sob uma função objetivo, permitindo assim determinar as melhores soluções e consequentemente possuindo um papel importante na solução de problemas aplicados. Baseados em processos biológicos, mais especificamente nas dinâmicas que regem a evolução dos organismos vivos,

os algoritmos evolutivos possuem em sua grande maioria, mecanismos de variação e seleção de modo a alcançar melhores soluções dado um conjunto de possíveis soluções.

Caracterizado como algoritmo evolutivo, o método de evolução diferencial (DE, do inglês *differential evolution*) vem se destacando dentre as metaheurísticas existentes como alternativa para a solução de problemas de otimização, devido a sua capacidade de convergência, precisão e qualidade de solução encontrada. Apesar de ser considerado como método evolutivo, seus operadores de mutação e recombinação não possuem inspiração biológica, mas sim matemática. Uma compreensão adequada do efeito destes operadores no desempenho dos métodos evolucionários é um problema em aberto na literatura, cuja solução potencialmente poderia orientar o desenvolvimento de algoritmos mais eficientes para determinadas classes de problemas. Sabe-se que a função dos operadores nos algoritmos evolutivos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório.

Neste trabalho, é feita uma padronização e análise experimental de diversos operadores de recombinação aplicados ao método DE. São apontados nesta análise o desempenho e comportamento dos operadores sob problemas de otimização existentes na literatura que serão abordados no decorrer deste trabalho.

1.1 Motivação

A busca por melhores resultados é um dos grandes objetivos da engenharia. Para isso, há um grande interesse em se desenvolver métodos de otimização robustos e eficientes, de modo a atender determinadas necessidades. A abordagem *mono-objetivo* ou *escalar* é bastante estudada na otimização e consiste em obter uma solução (maximização ou minimização de uma função objetivo) dado um problema.

Dentre os diversos métodos utilizados para a solução de problemas mono-objetivo, os algoritmos evolutivos (AEs), em especial o método de evolução diferencial, estão entre os mais utilizados e possuem ótimo desempenho em problemas de pequena escala (principalmente em engenharia), sendo um balanço de simplicidade e qualidade de soluções (Chakraborty, 2008). Tal como acontece com a maioria dos AEs, o desempenho do método de evolução diferencial é fortemente influenciada pelos valores atribuídos às constantes definidas pelo usuário (que são dependentes do problema em questão) (Brest *et al.*, 2006), bem como pela escolha da variação dos operadores (mutação, recombinação e seleção) (Birattari, 2009), (Price *et al.*, 2006).

Isto implica na necessidade de especificar como as soluções candidatas serão modificadas para gerar novas soluções (Maruo *et al.*, 2005) através de parâmetros que devem ser fixados nos operadores.

O papel dos operadores em métodos evolutivos é fundamental para a geração de novas soluções diversificando assim a população. O balanço entre convergência e diversidade em algoritmos evolutivos é um importante tópico de estudo, onde se busca evitar a convergência prematura para ótimos locais, e ao mesmo tempo acelerar a convergência para o ótimo global do problema. Este balanço é muitas vezes regulado implicitamente, mediante as escolhas dos tipos e configurações dos operadores de variação e seleção utilizados. Tal fato justifica o estudo destes operadores como forma de possibilitar uma melhor compreensão dos mecanismos de funcionamento e dos efeitos de diferentes operadores no processo de busca do algoritmo.

1.2 Objetivos

O trabalho desenvolvido na presente dissertação possui como objetivo principal realizar uma análise e comparação experimental do efeito de operadores de recombinação disponíveis na literatura de Algoritmos Evolutivos no desempenho médio do método de evolução diferencial, na tentativa de abstrair possíveis características que levam aos melhores desempenhos dos operadores. Para tal, estes operadores são formalizados em uma notação unificada, e um pacote computacional implementa o método de forma padronizada e modular, possibilitando não somente o trabalho atualmente apresentado como também investigações futuras de outros aspectos do método de evolução diferencial. São considerados objetivos específicos deste trabalho:

- apresentar e explorar características comuns dos operadores de recombinação;
- propor uma formulação padronizada e unificada dos operadores de recombinação, de forma a facilitar a análise e compreensão dos mesmos;
- avaliar o desempenho médio dos operadores aplicados ao DE;
- implementação do método DE tradicional e consolidação de operadores de recombinação para espaços contínuos em um pacote computacional aberto que possa servir de base para outras investigações e de possíveis proposições de novos operadores.

1.2.1 Contribuições do trabalho

Neste trabalho é apresentada uma comparação experimental de 16 operadores de recombinação presentes na literatura para espaços contínuos em métodos evolutivos, adaptados e aplicados ao método de evolução diferencial. Também é proposta uma formulação unificada de tais operadores visando uma melhor compreensão e implementação das características matemáticas destes operadores. É apresentada também uma implementação modular padronizada do algoritmo e dos operadores estudados, na forma de um pacote computacional de código aberto em linguagem R.

1.3 Estrutura do trabalho

Os demais capítulos deste trabalho estão organizados da seguinte forma:

O Capítulo 2 apresenta conceitos de otimização, algoritmos evolutivos, o método DE e operadores de recombinação. A partir desses conceitos, é apresentada a relação da otimização com algoritmos evolutivos, que por sua vez, aborda em especial o método DE como sendo um dos mais visados atualmente na solução de problemas de pequena escala de valores contínuos. Após a apresentação destes conceitos, metodologias são apresentadas como alternativa na solução de determinada classe de problemas aplicados na otimização.

O Capítulo 3 apresenta diversos operadores de recombinação que estão presentes na literatura para algoritmos evolutivos, que em sua grande maioria foram adaptados para o método DE. Uma revisão e uma notação matemática padronizada de tais operadores são apresentadas.

O Capítulo 4 apresenta uma codificação do método DE e de operadores de recombinação para espaços contínuos que foram ajustados e aplicados ao método DE e cujo objetivo dessa codificação é analisar o desempenho médio e abstrair possíveis características comuns que auxiliam no resultado obtido pelos operadores. Também é apresentado uma notação simplificada e padronizada dos operadores de modo a facilitar o entendimento e implementação destes. A implementação do método DE em conjunto com os operadores de recombinação foi realizada em linguagem R, que é uma linguagem gratuita e com grande contribuição da comunidade acadêmica no desenvolvimento desta. Com isso, um pacote computacional foi criado e disponibilizado para toda a comunidade para que possa servir de base para outras investigações.

O Capitulo 5 apresenta os experimentos computacionais realizados para avaliação do

desempenho do método DE em conjunto com os operadores de recombinação, comparando os resultados obtidos nos diversos operadores utilizados neste método. Para o experimento foi realizado um ajuste de parâmetros aplicando o método conhecido como corrida iterativa (iterated racing), que é utilizado para determinar configurações de parâmetros para serem empregados nos métodos. Como referência para o ajuste de parâmetros e testes computacionais, foram utilizados os 28 problemas de teste da base CEC2013. Os testes foram realizados em dimensão 5 e 20.

No Capítulo 6 são apresentadas as conclusões e são discutidas possibilidades de estudos de continuidade do trabalho aqui apresentado.

CAPÍTULO 2

Referencial Teórico

Neste capítulo são apresentadas definições relevantes relacionadas a otimização, algoritmos evolutivos e operadores de variação de população. Logo após, é apresentado a relação dos operadores de recombinação com os algoritmos evolutivos e sua importância no contexto da otimização.

Dentre os métodos evolucionários existentes, o algoritmo de evolução diferencial vem sendo um dos mais visados e com ótimo desempenho para problemas de pequena escala. Com isso, o foco deste trabalho é voltado para o método de evolução diferencial no contexto do processo de recombinação.

2.1 Otimização

Com o objetivo de obter o melhor resultado possível dado um problema em função de um objetivo determinado e das restrições que venham a existir, a otimização busca através de técnicas específicas obter a maximização ou minimização de uma função, chamada função objetivo, sobre um determinado domínio (Aarts e Lenstra, 1997). Uma função ou *função objetivo*, é uma formulação matemática que possui uma relação entre variáveis de escolha e a variável a ser maximizada ou minimizada. O conjunto, espaço ou região que contém as possíveis soluções sobre as variáveis de um vetor $\mathbf{x}$ do problema a ser otimizado é conhecido como *espaço de busca*. Entretanto, o espaço de busca é delimitado pelas funções de restrição que venham existir no problema de otimização.

Um conceito importante na otimização é a abordagem mono-objetivo, ou escalar. Essa abordagem consiste em obter uma solução dado um problema cujo objetivo é a maximização ou minimização de uma única função objetivo. Com base nisso, uma formulação genérica de um problema de otimização mono-objetivo irrestrita de minimização pode ser expresso por:

$$\text{Encontrar } \mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.1}$$

$$\text{Sujeito a } \mathbf{x} \in \mathcal{X} \tag{2.2}$$

onde $\mathbf{x}$ é um vetor de variáveis de otimização e $\mathcal{X} \subset \mathbb{R}^n$ é o espaço de busca viável, isto é, o conjunto de todos os pontos que representam soluções candidatas válidas para o problema. A função objetivo do problema é representada por $f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$, que quantifica cada possível solução $\mathbf{x}$ (Takahashi, 2004). $\mathbf{x}^*$ é a solução encontrada que atenda ao critério de eficiência definido pelo problema em questão.

2.1.1 Métodos de Otimização

Em geral, os métodos de otimização podem ser classificados em dois tipos: Métodos Determinísticos e Métodos Probabilísticos.

2.1.1.1 Métodos Determinísticos

Os métodos determinísticos em otimização geram uma sequência determinística de possíveis soluções necessitando, em sua grande maioria, o uso da primeira derivada da função objetivo em relação às variáveis do problema (Martha, 2005). A solução encontrada por estes métodos é extremamente dependente do ponto inicial que é fornecido para a localização da solução, podendo assim, convergir para uma solução local. Portanto, tais métodos possuem desempenho limitado em funções multimodais(que possuem várias soluções ótimas locais).

Para a otimização não-linear em espaços contínuos, os métodos determinísticos geralmente são divididos entre técnicas de direção de busca e métodos de exclusão de semi-espaços. As técnicas de direção de busca consistem em fazer o algoritmo evoluir encontrando novas soluções candidatas situadas em direções cuja função objetivo decresça em relação a solução candidata corrente (Takahashi *et al.*, 2010). São exemplos de técnicas de direção de busca mais comuns:

- Método do Gradiente (Hestenes e Stiefel, 1952): que através do gradiente da função objetivo, consiste em achar uma direção de decréscimo da função, que é a direção do gradiente negativo, e um tamanho máximo de passo a ser tomado naquela direção;

- Método de Newton: é um tipo de técnica de aproximação quadrática que realiza uma correção na direção de busca sugerida pelo vetor gradiente através da incorporação de informações obtidas a partir das derivadas parciais de segunda ordem da função objetivo (Linden, 2008).

Já os métodos de exclusão de semi-espaços utilizam aproximações do gradiente da função objetivo para definir um plano que divide o espaço de objetivos em dois semi-espaços, sendo que o gradiente deve necessariamente decrescer em um dos semi-espaços (Ávila *et al.*, 2006). São exemplos de métodos de exclusão de semi-espaços mais comuns:

- Métodos de pontos interiores: dado um ponto viável, o mesmo move-se em direção ao ótimo através do interior de uma região factível (Gaspar-Cunha *et al.*, 2012).

- Método elipsoidal (Ecker e Kupferschmid, 1983): baseado na informação do gradiente da função objetivo, delimita uma região elipsoidal na qual se encontra o ponto ótimo do problema, e iterativamente reduz o hipervolume deste elipsóide a partir de informações relativas ao gradiente da função objetivo ou de funções de restrição, em ambos os casos assumidas como sendo convexas.

2.1.1.2 Métodos Probabilísticos

Os métodos probabilísticos em otimização utilizam a avaliação da função objetivo para encontrar no espaço de busca soluções ótimas. Neste tipo de método, dados e parâmetros utilizados no processo de otimização são obtidos de forma estocástica (Borges, 2013).

Apesar de não garantirem a exatidão do resultado, os métodos probabilísticos realizam buscas simultâneas no espaço de possíveis soluções através de uma população de soluções candidatas, possibilitando um maior alcance de soluções ótimas (Qiu *et al.*, 2012). São exemplos de métodos representativos dos métodos probabilísticos de populações (Takahashi *et al.*, 2010):

- *Simulated Annealing*: é uma técnica de busca local probabilística inspirada na termodinâmica. Este método gera novos pontos de forma aleatória a partir de um ponto inicial. Caso o ponto gerado possua melhor desempenho que o ponto atual, a regra de transição é determinística, caso contrário a regra é probabilística A probabilidade de aceitação de pontos inferiores é decrescida monotonicamente ao longo das iterações. (Beasley *et al.*, 1993).

- *Hill Climber*: é uma técnica simples de busca local que avalia as soluções candidatas na região atual, optando pela que tiver melhor avaliação de função objetivo para gerar novas soluções. O método finaliza quando não se encontra nenhuma melhora possível em uma iteração, mas isso não garante que há a otimalidade da solução (Wang *et al.*, 2006).

- *Particle Swarm Optimization* (PSO): a partir de um vetor de velocidade e outro de posição, a posição de cada solução candidata é atualizada de acordo com a velocidade atual, simulando assim um tipo de otimização social inspirado no comportamento de enxames biológicos e em aspectos sociais, utilizando uma estratégia de colaboração para a evolução da população de soluções-candidatas. (Waintraub, 2009).

Dentre os diversos métodos probabilísticos utilizados para a solução de problemas mono-objetivo destacam-se os algoritmos evolutivos, que serão abordados na próxima seção.

2.2 Algoritmos Evolutivos

A computação evolutiva teve sua origem na década de 50, nos experimentos sobre evolução artificial pelo matemático ítalo-norueguês Nils Barricelli (Barricelli *et al.*, 1954) e nos trabalhos do geneticista britânico Alex Fraser sobre a simulação de processos evolutivos (Fraser, 1957). Posteriormente outros pesquisadores realizaram diversos trabalhos dando sequência no desenvolvimento da área da computação evolutiva. Em meados da década de 60 nos Estados Unidos, o pesquisador Lawrence J. Fogel desenvolvia técnicas para a geração de inteligência artificial através da utilização de mecanismos evolutivos (Fogel, 1964) em que se faziam uso de mutações em uma população de soluções sujeita a uma pressão seletiva, e com isso surgiu o conceito de *Programação Evolutiva*. Na década de 80 a publicação do livro *Genetic Algorithms in Search, Optimization, and Machine Learning* por David E. Goldberg (Goldberg *et al.*, 1989) ganhou destaque na área de otimização com a utilização dos Algoritmos Genéticos (AGs) gerando uma tendência de unificação de técnicas baseadas na evolução de soluções sob a definição de *algoritmos evolutivos* (Campelo, 2016).

Algoritmos evolutivos (AEs) recebem este nome devido a sua inspiração original no processo de evolução natural proposto pro Darwin, e possuem como principais componentes:

- População: conjunto de soluções-candidatas (também chamadas de indivíduos) existentes em uma dada iteração do algoritmo.

- Operadores de variação: funções executadas sobre a população de forma a gerar novas soluções-candidatas que amostrem iterativamente o espaço de variáveis de decisão, buscando soluções de melhor desempenho. Os operadores mais comuns recebem os nomes específicos de recombinação e mutação, e serão explicados adiante.

- Função objetivo (*fitness*): função que quantifica a qualidade relativa de uma dada solução-candidata, refletindo sua capacidade de sobrevivência e seu potencial de reprodução. Em problemas irrestritos a função-objetivo é usualmente conhecida como função de desempenho.

- Operadores de seleção: funções que determinam quais soluções candidatas serão utilizadas para a construção de novos pontos, ou para a composição da população na próxima iteração. Usualmente são baseadas na comparação dos valores de desempenho das soluções da população corrente e daquelas geradas através dos operadores de variação.

- Geração: refere-se a uma iteração do algoritmo.

Os AEs são baseados em populações na qual os indivíduos podem ser afetados por outros indivíduos e pelo próprio ambiente. Quanto mais adequado for o indivíduo nestas condições, maior é a chance dele sobreviver e gerar descendentes na população, uma vez que herdam informações genéticas dos pais. Um algoritmo evolutivo básico pode ser expresso como indicado no Algoritmo 1. Neste algoritmo $P(t)$ simboliza a população de soluções candidatas na iteração t. $P'(t)$ é a população de descendentes gerada por meio da variação, que é o conjunto de operadores mutação e/ou recombinação aplicados à população $P(t)$.

Algoritmo 1: Algoritmo Evolutivo Básico

$t \leftarrow 1$
Inicializa $P(t)$
avalia $P(t)$
enquanto *Condição_de_parada* **faça**
  $P'(t) \leftarrow$ variação $[P(t)]$
  avalia $[P'(t)]$
  $P(t+1) \leftarrow$ seleciona $[P'(t) \cup P(t)]$
  $t \leftarrow t+1$
fim

Portanto as etapas básicas de um algoritmo evolutivo são:

- Geração da população inicial
  Normalmente uma população é formada por indivíduos gerados aleatoriamente.

- Avaliação dos indivíduos da população
  Avalia-se todos os indivíduos segundo algum critério determinado pela função-objetivo e pelas restrições do problema, de forma a quantificar o desempenho de cada solução-candidata para o problema de otimização que está sendo tratado.

- Aplicação dos operadores de variação na população
  Os operadores de variação possuem a finalidade de diversificar a população de indivíduos através do processo de mutação, que modifica o indivíduo segundo algum critério, e do processo de recombinação, que herdam características presentes dos indivíduos que irão se recombinar para gerar novos indivíduos.

- Seleção dos melhores indivíduos em comparação com: população original e população que sofreu variação. Nesta etapa, o algoritmo seleciona mediante algum critério as soluções-candidatas que serão utilizadas para compor a população na iteração $t+1$.

2.3 Algoritmo de Evolução Diferencial

Originalmente apresentado por Storn e Price em meados da década de 1990 (Storn e Price, 1997) o algoritmo de evolução diferencial(DE, do inglês *differential evolution*) é uma técnica de otimização para problemas não lineares de espaço contínuo. A principal inovação deste método é a utilização da diferença de dois vetores escalares retiradas da população para a perturbação de um terceiro vetor, o que resultou em um mecanismo de variação de auto-adaptação que permitiu o método aprender implicitamente determinadas características no espaço de busca e se auto-adaptar (Goulart *et al.*, 2011). Similar em diversos aspectos aos Algoritmos Genéticos, o DE utiliza-se de operadores de variação e seleção aplicados sobre uma população, com a finalidade de encontrar uma solução ótima dada uma função objetivo, ao longo de sucessivas gerações.

Apesar de sua simplicidade, o DE é bastante eficiente em problemas não lineares contínuos. Embora seja classificado como método evolutivo, os operadores do DE não possuem inspiração biológica, mas sim matemática, possuindo similaridades com alguns aspectos da técnica conhecida no método Nelder-Mead(Nelder e Mead, 1965). Devido a essas características o

DE tem atraído a atenção de pesquisadores em diversas variedades de problemas (Plagianakos *et al.*, 2008)

De modo semelhante a outras metaheurísticas evolutivas, o DE utiliza através de amostragem, um número finito de vetores do espaço de soluções candidatas viáveis como tentativa para a solução de problemas. Em cada iteração, a função de amostragem (representada pelos operadores de variação) é modificada com base nos valores observados de desempenho das soluções candidatas ao longo das iterações. Os mecanismos básicos da iteração do DE são: dado um conjunto de soluções candidatas na iteração t, $\mathcal{P}(t) = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x}_\mu\}$ o algoritmo gera um segundo conjunto de amostras $\tilde{\mathcal{P}}(t) = \{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}_\mu}\}$ por aplicação sequencial de dois operadores de variação. O primeiro é comumente chamado de *mutação diferencial* , e funciona através da adição de um vetor de perturbação composto das diferenças entre duas ou mais soluções candidatas amostradas a partir de $P(t)$. Para este operador existem diversas variações (Price *et al.*, 2006) sendo duas as mais comum: **/rand/1** e **/best/1** que seguem a seguinte estrutura:

$$\mathbf{v}_i = \mathbf{x}_{base} + \phi\,(\mathbf{x}_{r1} - \mathbf{x}_{r2}), \quad i = 1, \ldots, \mu \tag{2.3}$$

onde $\mathbf{x_{base}}$, $\mathbf{x_{r1}}$ e $\mathbf{x_{r2}}$ são três soluções mutuamente exclusivas a partir de $\mathcal{P}(t)$ e ϕ é um fator de escala. As estratégias */rand/1* e */best/1* diferem na seleção do $\mathbf{x_{base}}$: o primeiro é selecionado aleatoriamente de $\mathcal{P}(t)$, enquanto o segundo é o que obtiver melhor desempenho na população. As soluções $\mathbf{x_{r1}}$ e $\mathbf{x_{r2}}$ são selecionadas aleatoriamente em ambos os casos. O segundo operador de variação é chamado de recombinação, e, geralmente envolve a geração de uma solução com base em operações realizadas na mutação $\mathbf{v_i}$ e em uma solução corrente $\mathbf{x_i}$. Dentre os operadores de recombinação existentes, dois são comumente utilizados no DE: o binomial e o exponencial.

Independentemente da escolha dos operadores de recombinação a serem utilizados no DE, a saída sempre será um conjunto de soluções $\tilde{\mathcal{P}}(t) = \{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}}_\mu\}$. Essas novas soluções são avaliadas pela função objetivo e cada solução é comparada com a solução corrente $\mathbf{x}_i$. Um processo de seleção elitista é realizado e reúne $\mathcal{P}(t+1)$ que substitui todas as soluções para as quais a condição $f(\tilde{\mathbf{x}}_i) \leq f(\mathbf{x}_i)$ for verdadeira. O ciclo iterativo é realizado até um critério de terminação, sendo geralmente o número máximo de chamadas da função objetivo que é executada. O algoritmo 2 resume a estrutura base do DE.

Neste trabalho é realizada uma análise experimental de operados de recombinação para o DE que serão apresentados e discutidos no próximo Capítulo.

Algoritmo 2: Estrutura básica do algoritmo de evolução diferencial.

Entrada: Parâmetros de controle: μ, t_{max}
Entrada: Operadores de variação
Entrada: Parâmetros dos operadores
Saída: População final de soluções: $\mathcal{P}(t_{max})$
$t \leftarrow 0$;
$\mathcal{P}(t) \leftarrow \{\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_\mu\}$; `// Soluções aleatórias`
enquanto $t < t_{max}$ **faça**
  para $i = 1, \ldots, \mu$ **faça**
    Geração de soluções $\mathbf{v}_i$; `// Eq.` (2.3)
    Geração de soluções $\tilde{\mathbf{x}}_i$; `// Sec.` 3.1
    se $f(\tilde{x}_i) \leq f(\vec{x}_i)$ **então**
      $\vec{\mathbf{x}}_i \leftarrow \tilde{\mathbf{x}}_i$;
    fim
  fim
  $\mathcal{P}(t+1) \leftarrow \{\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_\mu\}$;
  $t \leftarrow t + 1$;
fim

CAPÍTULO 3

Operadores de Recombinação para Evolução Diferencial

Neste capítulo são apresentados diversos operadores de recombinação encontrados na literatura para algoritmos evolutivos, mais especificamente algoritmos genéticos e DE, sendo expressos utilizando uma notação matemática padronizada.

3.1 Operadores de Recombinação

Propostos com o intuito de transformar a população através de sucessivas gerações, os operadores de recombinação combinam as informações contidas em duas ou mais soluções candidatas a fim de gerar novas soluções, podendo diversificar ou contrair a população ao longo do processo de busca pela solução ótima.

Na literatura geral de Algoritmos Evolutivos (AEs), operadores de recombinação (às vezes também chamados de operadores de cruzamento) são geralmente interpretados como técnicas de variação, que geram uma ou mais novas soluções candidatas ao combinar as informações contidas em duas ou os mais soluções existentes. [1]

Esta seção apresenta uma formulação matemática dos operadores de recombinação investigados na seção experimental. Enquanto alguns deles têm sido aplicados ao DE, com variados graus de sucesso (incluindo os dois mais populares, ou seja, as variantes *exponencial* e *binomial*), a maioria nunca foram utilizados com esta metaheurística, e vêm da literatura dos algoritmos genéticos de codificação real (RCGA). Como tal, as versões originais destas variantes foram concebidos para devolver vários vetores (soluções candidatas) enquanto o DE requer apenas um. Sempre que necessário, os operadores foram adaptados para voltar um ponto único

[1] Note que esta definição também abrange o operador *mutação diferencial* do DE, que na verdade é também um tipo de recombinação multi-pais. Aqui simplesmente é sugerida uma convenção da literatura do DE e chamou-se de mutação.

no final da operação, de forma a atender as premissas do DE.

A notação de todas as variantes de recombinação descritas abaixo foi padronizada, a fim de simplificar a interpretação e implementação, bem como para destacar as diferenças e semelhanças de cada abordagem. Foram utilizadas as seguintes convenções:

- $\mathbf{x}_i$: i-ésima solução corrente da população;
- $\mathbf{v}_i$: i-ésima solução que sofreu mutação (Eq. (2.3)) na iteração corrente;
- $\tilde{\mathbf{x}}_i$: i-ésima solução recombinada na iteração corrente;
- $y^{(j)}$: j-ésima componente de um vetor $\mathbf{y}$
- u: uma variável uniforme independente distribuída aleatoriamente em $(0,1)$.

Além disso, as seguintes equações auxiliares são definidas:

$$\begin{aligned} c_{ij}^{min} &= \min\left(x_i^{(j)}, v_i^{(j)}\right) \\ c_{ij}^{max} &= \max\left(x_i^{(j)}, v_i^{(j)}\right); \quad \begin{matrix} i = 1,\ldots,\mu \\ j = 1,\ldots,n \end{matrix} \\ \ell_{ij} &= c_{ij}^{max} - c_{ij}^{min} \end{aligned} \tag{3.1}$$

Sempre que necessário, as equações seguintes serão utilizadas para garantir diferentes aspectos da recombinação:

- A *equação de escolha aleatória* (Eq. (3.2)) é utilizada antes da aplicação de determinados operadores, para garantir que o mesmo não apresente qualquer viés em relação às soluções candidatas utilizadas no processo de recombinação:

$$\begin{aligned} &\mathbf{a}_i = \mathbf{x}_i \text{ e } \mathbf{b}_i = \mathbf{v}_i, \text{ se } u \leq 0.5 \\ &\mathbf{a}_i = \mathbf{v}_i \text{ e } \mathbf{b}_i = \mathbf{x}_i, \text{ caso contrário} \end{aligned} \tag{3.2}$$

- A *equação de escolha por desempenho* (Eq. (3.3)) é utilizada quando o operador de recombinação necessitar de informações sobre o desempenho relativo de $\mathbf{x}_i$ e $\mathbf{v}_i$:

$$\begin{aligned} &\mathbf{p}_{i,1} = \mathbf{x}_i \text{ e } \mathbf{p}_{i,2} = \mathbf{v}_i, \text{ se } f(\mathbf{x}_i) \leq f(\mathbf{v}_i) \\ &\mathbf{p}_{i,1} = \mathbf{v}_i \text{ e } \mathbf{p}_{i,2} = \mathbf{x}_i, \text{ caso contrário} \end{aligned} \tag{3.3}$$

Finalmente, exceto quando indicado, *inteiros selecionados aleatoriamente* são selecionados com base em uma distribuição aleatória uniforme.

3.1.1 Arithmetic Recombination (/arith) (Herrera *et al.*, 2003; Price *et al.*, 2006)

O operador do tipo *Arithmetic* realiza uma recombinação que é definida como uma combinação linear de dois vetores e que possui dois grandes benefícios: reduzir a complexidade do algoritmo e prover grande flexibilidade na realização de uma distribuição de mutação (Price *et al.*, 2006). Ele possui um coeficiente λ, que pode ser uma constante ou um valor aleatório. Caso λ seja uma constante ou um valor aleatório, que é utilizado para toda a população, temos que a recombinação é linear. Caso o valor de λ seja amostrado para cada solução candidata, a recombinação é dita intermediária.

Suponha $\lambda_i \in (0,1), i = 1,\ldots,\mu$, um conjunto uniforme de valores distribuídos aleatoriamente. Então:

$$\tilde{\mathbf{x}}_i = (1-\lambda_i)\mathbf{x}_i + \lambda_i \mathbf{v}_i; \quad i = 1,\ldots,\mu \tag{3.4}$$

– Parâmetros livres: nenhum.

3.1.2 One-Point Recombination (/onepoint) (Holland, 1992)

Também conhecido como operador de recombinação de um corte, o operador *One-Point* (PX) consiste em selecionar um ponto de recombinação para duas soluções candidatas, de forma aleatória.

Suponha $k_i \in \{1,\ldots,n-1\}, i = 1,\ldots,\mu$, um conjunto de valores inteiros. Aplicando a Eq. (3.2), temos:

$$\tilde{\mathbf{x}}_i = [a_i^{(1)},\ldots,a_i^{(k_i)},b_i^{(k_i+1)},\ldots,b_i^{(n)}] \tag{3.5}$$

– Parâmetros livres: pontos de corte k_i (valor fixo ou selecionado aleatoriamente para cada i).

3.1.3 N-Point Recombination (/npoint) (Eshelman *et al.*, 1989)

Também conhecido como recombinação multipontos (MPX), é uma generalização do operador *One-Point* que subdivide aleatoriamente duas soluções candidatas em $k+1$ trechos,

e gera uma nova solução candidata através da composição alternada de trechos retirados de cada um dos pontos originais. Suponha $k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(m_i)} \in \{1, \ldots, n-1\}$ selecionados aleatoriamente, com inteiros mutuamente exclusivos em ordem crescente, para cada $i = 1, \ldots, \mu$. Aplicando a Eq. (3.2), temos:

$$\tilde{\mathbf{x}}_i = [a_i^{(1)}, \ldots, a_i^{(k_i^{(1)})}, b_i^{(k_i^{(1)}+1)}, \ldots, b_i^{(k_i^{(2)})}, a_i^{(k_i^{(2)}+1)}, \ldots] \tag{3.6}$$

– Parâmetros livres: número de pontos de corte m_i (valor fixo ou selecionado aleatoriamente para cada i).

3.1.4 Binomial Recombination (/bin) (Storn e Price, 1997; Syswerda, 1991)

Também conhecido como recombinação uniforme na literatura de AEs, esta recombinação é um processo independente que determina a herança de cada parâmetro experimental para uma solução candidata filha, obtida de forma aleatória de soluções candidatas pais. Supondo $\rho \in (0,1)$ denotando um parâmetro de probabilidade, e $k_i \in \{1, \ldots, n\}, i = 1 \ldots, \mu$, denotando um conjunto de inteiros selecionados aleatoriamente. Esta variante pode, então, ser expressa pela aplicação consecutiva das duas operações abaixo.

$$\tilde{x}_i^{(j)} = \begin{cases} v_i^{(j)} & \text{, se } u \leq \rho \\ x_i^{(j)} & \text{, caso contrário} \end{cases}; \quad \begin{matrix} i = 1, \ldots, \mu \\ j = 1, \ldots, n \end{matrix} \tag{3.7}$$

e

$$\tilde{x}_i^{(k_i)} = \begin{cases} v_i^{(k_i)} & \text{, se } \tilde{\mathbf{x}}_i = \mathbf{x}_i \\ \tilde{x}_i^{(k_i)} & \text{, caso contrário} \end{cases}; \quad i = 1 \ldots, \mu \tag{3.8}$$

– Parâmetros livres: probabilidade de recombinação ρ.

3.1.5 Exponential Recombination (/exp) (Price *et al.*, 2006)

O operador *Exponential*, realiza a recombinação através de um ponto de corte selecionado aleatoriamente no intervalo $1, ..., n$, onde n é o tamanho das variáveis da solução. Suponha $\rho \in (0,1)$ um dado parâmetro probabilidade e $k_{i,1} \in \{0, \ldots, (n-1)\}, i = 1, \ldots, \mu$, um conjunto de inteiros selecionados aleatoriamente. Supondo $d_i \in \{1, \ldots, n\}$ um segundo conjunto de in-

teiros aleatoriamente gerados usando a distribuição exponencial:

$$P(d_i = x) = \frac{\rho^{x-1} - \rho^x}{\sum_{y=1}^{n} (\rho^{y-1} - \rho^y)} \tag{3.9}$$

Sejam também $k_{i,2} = k_{i,1} + d_i$, e $\mathbf{r}_i$ um vetor com elementos indicadores, definidos como:

$$r_i^{(j)} = \begin{cases} 1 & \text{se } k_{i,2} \leq n \wedge (k_{i,1} \leq j \leq k_{i,2}) \\ 1 & \text{se } k_{i,2} > n \wedge (j \leq (k_{i,2} \bmod n) \vee j \geq k_{i,1}) \\ 0 & \text{caso contrário} \end{cases} \tag{3.10}$$

para $j = 1, \ldots, n$. Utilizando estas definições, a recombinação exponencial pode ser definida por:

$$\tilde{x}_i^{(j)} = r_i^{(j)} v_i^{(j)} + \left(1 - r_i^{(j)}\right) x_i^{(j)} \tag{3.11}$$

– Parâmetros livres: probabilidade de recombinação ρ.

3.1.6 Geometric Recombination (/geo) (Herrera *et al.*, 2003; Michalewicz e Schoenauer, 1996)

O operador *Geometric* utiliza uma função da distância associada no espaço de busca das soluções, sendo independente do problema a ser analisado. Com isso a recombinação geométrica pode ser aplicada apenas para problemas em que cada parâmetro da solução candidata toma valores não negativos apenas, portanto, seu uso é bastante restrito, a menos que o espaço de busca seja transformado para um intervalo de valores estritamente positivos.

Assuma que $\alpha_i \in (0,1), i = 1, \ldots, \mu$, seja um dado conjunto de coeficientes. Utilizando a Eq. (3.2), temos:

$$\tilde{x}_i^{(j)} = \left(a_i^{(j)}\right)^{\alpha_i} \left(b_i^{(j)}\right)^{1-\alpha_i}; \quad \begin{matrix} i = 1, \ldots, \mu \\ j = 1, \ldots, n \end{matrix} \tag{3.12}$$

– Parâmetros livres: expoente de recombinação α_i.

3.1.7 Linear Recombination (/linear) (Wright, 1991)

O operador *Linear* gera três soluções candidatas de acordo com (3.13), que são em seguida avaliadas em relação a seu desempenho. Apenas a melhor das três é retornada pelo operador.

$$\begin{aligned} \tilde{\mathbf{x}}_{i,a} &= 0.5\,(\mathbf{x}_i + \mathbf{v}_i) \\ \tilde{\mathbf{x}}_{i,b} &= 1.5\mathbf{x}_i - 0.5\mathbf{v}_i \\ \tilde{\mathbf{x}}_{i,c} &= -0.5\mathbf{x}_i + 1.5\mathbf{v}_i \end{aligned} \tag{3.13}$$

Este operador de recombinação resulta em 3μ avaliações adicionais de função objetivo para cada iteração, uma vez que todas as três soluções geradas na em (3.13) devem ser avaliadas para se definir a melhor.

– Parâmetros livres: nenhum.

3.1.8 BLX-α-β Recombination (/blxAlphaBeta) (Herrera *et al.*, 2003)

O operador *BLX-α-β* gera soluções candidatas em um dado intervalo de valores regulado pelos parâmetros α e β. Sejam $\alpha \in (0, 0.5)$ e $\beta \in (0, 0.5)$ os coeficientes de extrapolação, e $\lambda_{ij} \in (0,1), i = 1,\ldots,\mu;\ j = 1,\ldots,n$, um conjunto uniforme de valores distribuídos aleatoriamente. Então, utilizando (3.1) e (3.3), temos:

$$\tilde{x}_i^{(j)} = \begin{cases} c_{ij}^{min} - \alpha\ell_{ij} + \lambda_{ij}\ell_{ij}\,(1+\alpha+\beta) & \text{if } p_{i,1}^{(j)} \leq p_{i,2}^{(j)} \\ c_{ij}^{min} - \beta\ell_{ij} + \lambda_{ij}\ell_{ij}\,(1+\alpha+\beta) & \text{caso contrário} \end{cases} \tag{3.14}$$

para todo $i = 1,\ldots,\mu$ e $j = 1,\ldots,n$. Essa operação de recombinação resulta em μ avaliações de função objetivo para cada iteração, uma vez que as soluções que sofreram mutação $\mathbf{v}_i$ devem ser avaliadas para se definir a ordem (3.3) da recombinação.

Este operador possui dois casos particulares que receberam nomes específicos na literatura: O primeiro, chamado de $BLX-\alpha$ (/blxAlpha) (Eshelman e Schaffer, 1992), ocorre quando $\alpha = \beta$. Já o segundo, conhecido como recombinação *Flat* (/flat) (Radcliffe, 1991), corresponde à situação onde $\alpha = \beta = 0$.

– Parâmetros livres: coeficientes de extrapolação $\alpha,\ \beta$.

3.1.9 Linear BGA Recombination (/lbga) (Schlierkamp-Voosen e Mühlenbein, 1994)

O operador *Linear BGA* cria novos parâmetros de direção que compõe a solução candidata filha produzidos a partir de parâmetros de duas soluções candidatas pais. Suponha $\alpha_{ijk} \in \{0,1\}$ um conjunto inteiros aleatórios gerados a partir de uma distribuição de Bernoulli com $P(\alpha_{ijk} = 1) = 1/16$. Utilizando (3.3), temos:

$$\begin{aligned} \xi_{ij} &= \frac{p_{i,2}^{(j)} - p_{i,1}^{(j)}}{\| \mathbf{p}_{i,1} - \mathbf{p}_{i,2} \|_2} \\ \gamma_{ij} &= \sum_{k=0}^{15} \left(\alpha_{ijk} 2^{-k} \right) \end{aligned} \tag{3.15}$$

A recombinação é então realizada através de (3.16):

$$\tilde{x}_i^{(j)} = \begin{cases} p_{i,1}^{(j)} - 0.5 \ell_{ij} \gamma_{ij} \xi_{ij} & \text{se } u \leq 0.9 \\ p_{i,1}^{(j)} + 0.5 \ell_{ij} \gamma_{ij} \xi_{ij} & \text{caso contrário} \end{cases} \tag{3.16}$$

para $i = 1, \ldots, \mu$ e $j = 1, \ldots, n$. Este operador de recombinação resulta em μ avaliações adicionais de função objetivo para cada iteração, uma vez que as soluções mutantes $\mathbf{v}_i$ devem ser avaliadas para definir a ordem em (3.3).

– Parâmetros livres: nenhum.

3.1.10 Max-Min-arithmetical Recombination (/mmax) (Herrera *et al.*, 2003, 1995)

Considere que $\lambda_{ij} \in (0,1), i = 1, \ldots, \mu;\ j = 1, \ldots, n$, seja um conjunto de pesos. Utilizando (3.3), este operador gera quatro soluções candidatas de acordo com (3.17), e retorna aquela que apresenta o melhor desempenho.

$$\begin{aligned} \tilde{x}_{i,a}^{(j)} &= \lambda_{ij} x_i + (1 - \lambda) v_i \\ \tilde{x}_{i,b}^{(j)} &= (1 - \lambda) x_i + \lambda_{ij} v_i \\ \tilde{x}_{i,c}^{(j)} &= c_{ij}^{min} \\ \tilde{x}_{i,d}^{(j)} &= c_{ij}^{max} \end{aligned} \tag{3.17}$$

Este operador resulta em 4μ avaliações adicionais de função objetivo em cada iteração, uma vez que as soluções candidatas $\tilde{\mathbf{x}}_{i,a}$, $\tilde{\mathbf{x}}_{i,b}$, $\tilde{\mathbf{x}}_{i,c}$ e $\tilde{\mathbf{x}}_{i,d}$ devem ser avaliadas para definir a melhor solução.

– Parâmetros livres: pesos λ_{ij} (se não for fornecido um valor, é utilizado para cada par i, j um valor selecionado aleatoriamente).

3.1.11 p-Best Recombination (/pbest) (Islam *et al.*, 2012)

Seja $[\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\mu]$ um conjunto μ de soluções candidatas amostradas com repetição das $p(t)$ soluções com melhores desempenho de $\mathcal{P}(t)$, com:

$$p(t) = \left\lceil 0.5\mu \left(1 - \frac{t-1}{t_{max}}\right) \right\rceil \tag{3.18}$$

A recombinação p-best aplica o operador binomial `/bin` (Sec.3.1.4), utilizando soluções candidatas $\hat{\mathbf{x}}_i$ no lugar de $\mathbf{x}_i$.

– Parâmetros livres: probabilidade elemento a elemento ρ.

3.1.12 SBX Recombination (/sbx) (Deb e Agrawal, 1994)

O *Simulated Binary Crossover* (SBX) possui como ideia básica criar um comportamento similar ao observado no operador *One-Point* (ver Sec. 3.5) no qual, através da modificação de um parâmetro, é possível afetar todas as variáveis da solução candidata simultaneamente, gerando soluções candidatas filhas distribuídas num volume centrado em cada solução candidata pai (Takahashi *et al.*, 2010). Suponha $\eta \in \mathbb{R}^+$ um dado parâmetro estritamente positivo, e $\lambda_{ij} \in (0,1), i = 1, \ldots, \mu;\ j = 1, \ldots, n$, um conjunto de valores distribuídos aleatoriamente . Além disso, seja:

$$\beta_{ij} = \begin{cases} (2\lambda_{ij})^{\frac{1}{\eta+1}} & \text{se } \lambda_{ij} \leq 0.5 \\ [2(1-\lambda_{ij})]^{\frac{1}{\eta+1}} & \text{caso contrário} \end{cases} \tag{3.19}$$

Então, utilizando (3.2):

$$\tilde{x}_i^{(j)} = \frac{\left(1+\beta_{ij}\right) a_i^{(j)} + \left(1-\beta_{ij}\right) b_i^{(j)}}{2}; \quad \begin{matrix} i = 1, \ldots, \mu \\ j = 1, \ldots, n \end{matrix} \tag{3.20}$$

– Parâmetros livres: coeficiente de recombinação η.

3.1.13 Wright Heuristic Recombination(/wright) (Wright, 1991)

O *Wright Heuristic Recombination* (HX) seleciona duas soluções candidatas da população tais que o valor de função objetivo das soluções selecionadas possuem valores distintos. Seja $\lambda_{ij} \in (0,1), i = 1,\ldots,\mu;\ j = 1,\ldots,n$, um conjunto de valores aleatórios. Então, utilizando (3.3):

$$\tilde{x}_i^{(j)} = \left(\lambda_{ij}+1\right) p_{i,1}^{(j)} - \lambda_{ij} p_{i,2}^{(j)}; \quad \begin{matrix} i = 1,\ldots,\mu \\ j = 1,\ldots,n \end{matrix} \tag{3.21}$$

Esta recombinação resulta em μ avaliações adicionais de função objetivo por iteração, uma vez que as soluções que sofreram mutação $\mathbf{v}_i$ devem ser avaliadas para se definir a ordem (3.3) da recombinação.

– Parâmetros livres: nenhum.

3.1.14 Eigenvector-based recombination(/eigen/⋆) (Guo e Yang, 2015)

O operador *Eigenvector-based* é, na verdade, uma transformação que pode ser aplicada a qualquer uma das recombinações citadas anteriormente. A cada iteração, esta estratégia estima a matriz de covariância da população atual, $C(\mathcal{P}(t))$. Antes da aplicação de uma recombinação, todas as soluções candidatas $\mathbf{x}_i$ e $\mathbf{v}_i$ são projetadas sobre uma base composta pelos autovetores de $C(\mathcal{P}(t))$. Uma variante de recombinação então é aplicada utilizando os vetores projetados, e posteriormente as soluções candidatas $\tilde{\mathbf{x}}_i$ são obtidas por projeção através do resultado da recombinação anterior em direção a base original.

Neste trabalho, somente a variante `/eigen/bin` foi utilizada.

– Parâmetros livres: condicionado ao operador a ser utilizado. Para `/eigen/bin`, probabilidade elemento a elemento ρ.

Na literatura existem diversas outras técnicas de recombinação que não foram abordadas neste estudo como, por exemplo, *Orthogonal* que utiliza uma estratégia de evolução na adaptação de uma matriz de covariância prevendo eficácia no processo de busca da solução

ótima (Leung e Wang, 2001), o *Hybrid Linkage* que utiliza um método de perturbação base para extrair automaticamente informações ligadas de um problema específico e com essa informação guia-se o processo de recombinação (Cai e Wang, 2015), dentre outras técnicas de recombinação.

Dos métodos apresentados nesta seção, sua grande maioria foi uma adaptação dos algoritmos genéticos de codificação real para o DE com a finalidade de realizar uma investigação experimental e com isso obter o desempenho médio dos mesmos através de um pacote computacional que será apresentado na próxima seção.

CAPÍTULO 4

O pacote ExpDE

O *ExpDE* é um pacote computacional desenvolvido em linguagem R que implementa o método DE de forma padronizada e modular, possibilitando investigações de diferentes operadores (recombinação, mutação e seleção) presentes no método. Este pacote implementa diferentes versões (compostas por diferentes combinações de operadores de mutação, recombinação e seleção) assumindo a otimização de problemas irrestritos (a menos de restrições do tipo caixa) de minimização. Mais especificamente, o presente trabalho se concentrou na implementação e teste de operadores de recombinação, conforme descrito no capítulo anterior.

R é um ambiente de desenvolvimento para cálculos estatísticos e gráficos sendo extensível e oferece funcionalidades para criação de ferramentas e métodos para a análise de dados. R possui uma ampla comunidade de colaboradores, que alimentam a plataforma através de diversos pacotes (frameworks), que são bibliotecas de funções específicas, para inúmeras finalidades (Crawley, 2012).

4.1 Metodologia

O desenvolvimento do *ExpDE* foi realizado de forma modular para a experimentação com diferentes tipos de variantes. A figura 4.1 apresentada a estrutura do pacote computacional representada através de modelagem UML(*Unified Modeling Language*).

A chamada do método *ExpDE* segue a seguinte estrutura:

```
ExpDE(popsize, mutpars, recpars, selpars,
      stopcrit, probpars, seed, showpars)
```

Essa rotina é utilizada para iniciar o algoritmo DE para minimização irrestrita dada uma instância do problema, podendo utilizar diferentes variações de operadores de recombinação, mutação e seleção. Os parâmetros específicos da função ExpDE são:

- popsize: tamanho da população de soluções candidatas (escalar inteiro)
- mutpars: parâmetros do processo de mutação (lista)
- recpars: parâmetros do processo de recombinação (lista)
- selpars: parâmetros do processo de seleção (lista)
- stopcrit: parâmetros relativos aos critérios de parada do método (lista)
- probpars: parâmetros da instância de problema a ser resolvida (lista)
- seed: valor de semente para o gerador de números aleatórios (escalar inteiro)
- showpars: parâmetros de controle da visualização do progresso da otimização (lista)

4.2 Parâmetros de Mutação - mutpars

O *mutpars* é um parâmetro usado para informar a rotina de mutação a ser utilizada no método DE, bem como quaisquer valores de parâmetros relacionados com a mutação. Na versão atual do pacote são aceitas as seguintes opções:

- `mutpars$name = "mutation_rand"`: é o tipo padrão de mutação do método DE em que vetores são selecionados aleatoriamente do espaço de soluções para a realização da mutação (ref. Seção 2.3).
- `mutpars$name = "mutation_best"`: este método seleciona a melhor solução candidata presente no espaço de soluções para a realização da mutação (ref. Seção 2.3).

  Os seguintes campos são compreendidos para ambos os métodos:

  f: fator de escala para o(s) vetor(es) de diferenças. São aceitos vetores numéricos de tamanho 1 ou *nvecs*.

  nvecs: número de vetores de diferenças a serem utilizados. Aceita $1 <= nvecs <= (nrow(X)/2-2)$, onde nrow(X) é a quantidade de soluções candidatas presentes em $\mathbf{X}$. O padrão é tamanho 1.

O *mutpars* recebe uma lista de objetos contendo :

- `mutpars$name`: contém o nome da função de mutação a ser chamada pelo método, por exemplo: `name = "mutation_rand"`;
- parâmetros requeridos para a função de mutação, por exemplo: `mutpars$f = 0.7`, `mutpars$nvecs = 2`

4.3 Parâmetros de Recombinação - recpars

Tal como acontece com os parâmetros de mutação, *recpars* é usado para definir a estratégia de recombinação desejada. Através do campo "name" é informado o nome da função de recombinação a ser utilizada no processo de recombinação e demais campos específicos que venham a existir de acordo com a função de recombinação a ser utilizada, por exemplo: `recpars$name = "recombination_bin"`, `recpars$cr = 0.8`. No *ExpDE* foram implementados os operadores de recombinação descritos no Capítulo 3, e listados abaixo para referência. Os parâmetros específicos de cada variante deste operador são aqueles explicitados no Capítulo 3. Maiores detalhes estão disponíveis na documentação das funções específicas do pacote *ExpDE*[1], disponibilizado como apêndice.

- *recombination_arith*
- *recombination_bin*
- *recombination_blxAlpha*
- *recombination_blxAlphaBeta*
- *recombination_eigen*
- *recombination_exp*
- *recombination_flat*
- *recombination_geo*
- *recombination_lbga*
- *recombination_linear*

[1] https://cran.r-project.org/web/packages/ExpDE/ExpDE.pdf

- *recombination_mmax*
- *recombination_npoint*
- *recombination_onepoint*
- *recombination_pbest*
- *recombination_sbx*
- *recombination_wright*

4.4 Parâmetros de Seleção - selpars

Seguindo a mesma ideia do *mutpars* e do *recpars*, o *selpars* é utilizado para definir os operadores de seleção. Atualmente no *ExpDE* está disponível apenas a implementação da seleção elitista padrão do método de evolução diferencial, conforme descrito na Seção 2.3. Esta opção é atribuída através de `selpars$name = "selection_standard"`.

4.5 Critério de parada - stopcrit

Com similaridade ao *recpars* e outros parâmetros, o *stopcrit* possui como diferença a escolha de múltiplos critérios de parada para o *ExpDE*. Os critérios de parada a serem utilizados são passados através do campo stopcrit$names que deve conter uma cadeia de caracteres. Outros parâmetros podem ser utilizados para parar o algoritmo. São critérios atualmente implementados no *ExpDE*:

- `stopcrit$maxiter`: número máximo de iterações
- `stocrit$maxeval`: número máximo de chamadas de função objetivo

4.6 Parâmetros do Problema - probpars

O *probpars* recebe uma lista com todas as definições relacionadas ao problema a ser otimizado. Ele possui três campos obrigatórios:

- `probpars$name`: nome da função que representa o problema a ser otimizado
- `probpars$xmin`: um vetor contendo os limites inferiores de todas as variáveis de otimização
- `probpars$xmax`: um vetor contendo os limites superiores de todas as variáveis de otimização

4.7 Semente do gerador de números aleatórios - seed

O *seed* recebe um valor chamado de "semente"que pode ser utilizado para a reprodutibilidade das soluções candidatas. O valor padrão deste parâmetro é `NULL`, que faz com que o sistema defina automaticamente um valor de *seed* com base na hora do sistema.

4.8 Parâmetros de impressão - showpars

O *showpars* contém os parâmetros que controlam a saída impressa na tela de comando do *ExpDE*. Este parâmetro aceita os seguintes campos:

- `showpars$show.iters`: que possui como alternativas "*point*", "*numbers*"ou "*none*", sendo *numbers* o valor padrão.
- `showpars$showevery`: inteiro positivo que determina quantas iterações o *ExpDE* deverá imprimir o indicador referenciado em `showpars$show.iters` para a tela de comando. O padrão é 1, ou seja, indicar a passagem de cada iteração.

4.9 Estrutura de saída

Como resultado o *ExpDE* disponibiliza um objeto do tipo lista contendo as seguintes informações:

- *X*: população de soluções candidatas classificada pelo desempenho de cada solução
- *Fx*: vetor de desempenho das soluções candidatas
- *Xbest*: melhor solução encontrada
- *Fbest*: valor de desempenho da melhor solução encontrada
- *nfe*: número de avaliações da função objetivo
- *iter*: quantidade de iterações realizada pelo método

4.10 Exemplo de uso

Utilizando a versão padrão (`rand/1/bin`) do DE, os seguintes parâmetros são ajustados para a utilização do *ExpDE* para a minimização de uma função esférica em 10 dimensões:

```
#DE/rand/1/bin
popsize  <- 100
mutpars  <- list(name = "mutation_rand", f = 0.8)
recpars  <- list(name = "recombination_bin", cr = 0.5,
                 minchange = TRUE)
selpars  <- list(name = "selection_standard")
stopcrit <- list(names = "stop_maxiter", maxiter = 100)
probpars <- list(name  = "sphere",
                 xmin = rep(-5.12,10), xmax = rep(5.12,10))
seed <- NULL
showpars <- list(show.iters = "numbers", showevery = 1)
output <- ExpDE(popsize, mutpars, recpars, selpars, stopcrit,
                probpars, seed, showpars)
```

No exemplo acima uma população de 100 soluções candidatas foi utilizada sob as seguintes condições:

- mutação padrão (rand) com a constante F no valor de 0.8
- recombinação binomial, com taxa de cruzamento no valor de 0.5
- seleção elitista, que seleciona a solução com melhor valor de função objetivo
- critério de parada sendo o número máximo de iterações no valor de 100
- função objetivo sendo o problema utilizado o esfera (sphere) sob os limites de -5.12 e 5.12.

A saída do método é representado da seguinte forma:

```
output$Xbest # melhor solução
[1]  0.1182979726  0.0367085782
    -0.1336290753  0.0379303622
     0.0006389787  0.0849878071
     0.0682196852  0.1497953866
    -0.0079158279  0.1286907679
-----------------------------------------------------
output$Fbest # valor de desempenho da melhor solução
[1] 0.2925359
-----------------------------------------------------
output$nfe # número de avaliações da função objetivo
[1] 10100
-----------------------------------------------------
output$iter # número de iterações do método
[1] 100
```

Outros exemplos de utilização são disponibilizados na documentação do pacote, disponível online [2] e reproduzida como Apêndice da presente dissertação.

[2] http://cran.r-project.org/web/packages/ExpDE/ExpDE.pdf

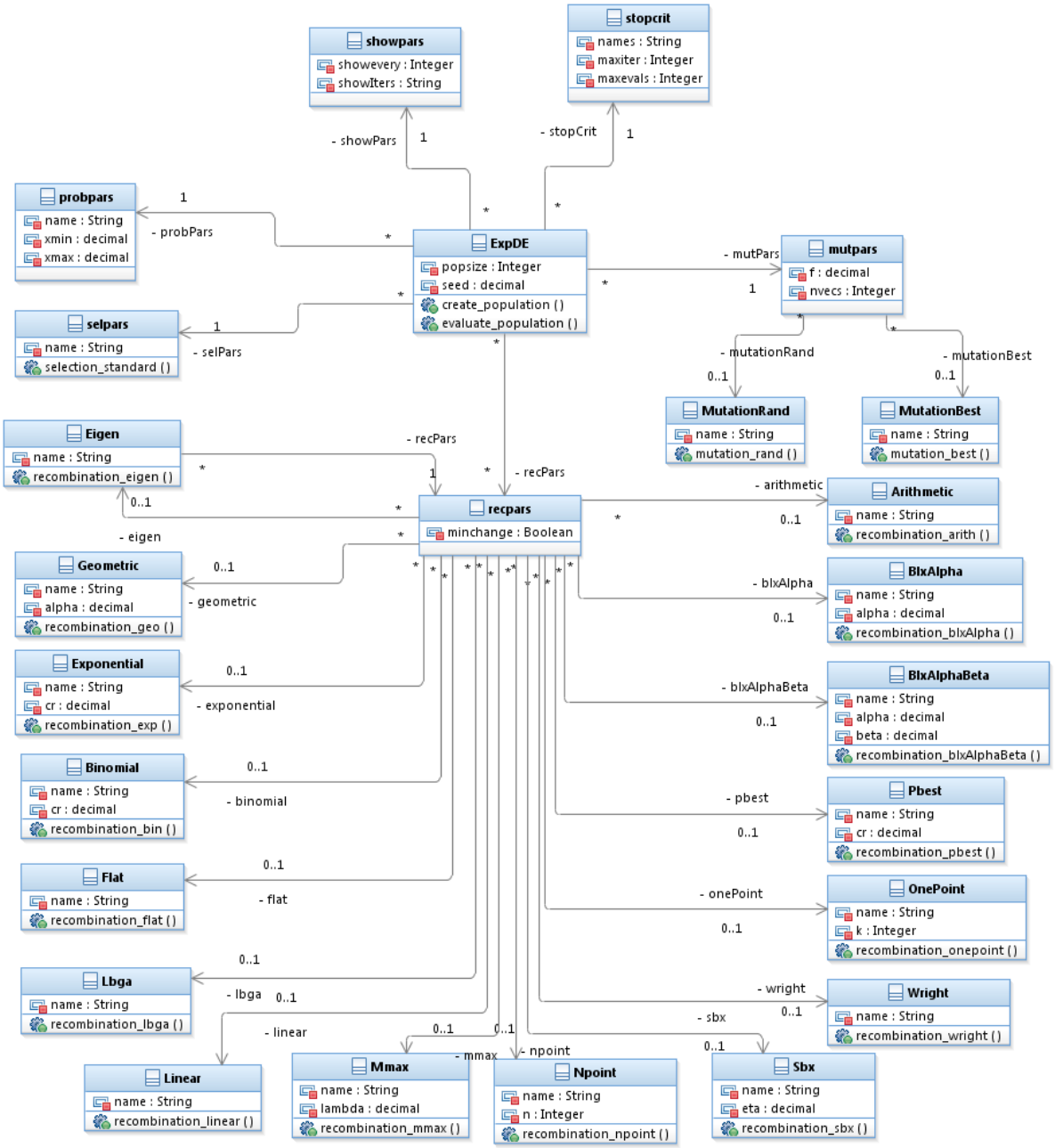


Figura 4.1 Diagrama UML do ExpDE

CAPÍTULO 5

Experimentos Computacionais e Resultados

Este capítulo apresenta o protocolo experimental utilizado na comparação dos efeitos de cada um dos operadores de recombinação no desempenho médio do DE e os resultados obtidos. Para isso, foi utilizada a ferramenta computacional *irace* para o ajuste dos parâmetros livres do método para cada variante do operador de recombinação. A partir disso, um conjunto de testes foi aplicado na comparação dos operadores, que será discutido ao longo do capítulo.

5.1 Instâncias de Problemas

Vinte e oito instâncias obtidas da competição do CEC 2013 (Liang *et al.*, 2013) foram utilizadas no experimento. Todas a funções foram tanto utilizadas para o ajuste de parâmetros quanto para a comparação das execuções do método DE. Dois valores de dimensão foram utilizados no experimento, sendo N = 5 e N = 20, onde N é a dimensão do problema. Todos os problemas utilizados são apresentados na tabela 5.1.

Em linhas gerais, todos os problemas podem ser definidos como:

$$\begin{aligned} &\text{Encontrar } \mathbf{x}^* = \arg\min_{\mathbf{x}} f_i(\mathbf{x}) \\ &\text{Sujeito a } x_i^{(j)} \in (-100, 100), \quad j = 1, \ldots, n \end{aligned} \tag{5.1}$$

Trucamento de valores foi utilizado para manter as soluções candidatas dentro dos limites definidos em (5.1). O critério de parada para todas as execuções foi definido em $10000N$, como sugerido na definição original dos problemas (Zambrano-Bigiarini e Gonzalez-Fernandez, 2015).

Tabela 5.1 Resumo das 28 funções de teste CEC'13

	No.	Função	$f_i^* = f_i(x^*)$
Funções Unimodais	1	*Sphere Function*	-1400
	2	*Rotated High Conditioned Elliptic Function*	-1300
	3	*Rotated Bent Cigar Function*	-1200
	4	*Rotated Discus Function*	-1100
	5	*Different Powers Functions*	-1000
Funções Multimodais Básicas	6	*Rotated Rosenbrock's Function*	-900
	7	*Rotated Schaffers F7 Function*	-800
	8	*Rotated Ackley's Function*	-700
	9	*Rotated Weierstrass Function*	-600
	10	*Rotated Griewank's Function*	-500
	11	*Rastrigin's Function*	-400
	12	*Rotated Rastrigin's Function*	-300
	13	*Non-Continous Rotated Rastrigin's Function*	-200
	14	*Schwefel's Function*	-100
	15	*Rotated Schwefel's Function*	100
	16	*Rotated Katsuura Function*	200
	17	*Lunacek Bi_Rastrigin Function*	300
	18	*Rotated Lunacek Bi_Rastrigin Function*	400
	19	*Expanded Griewank's plus Rosenbrock's Function*	500
	20	*Expanded Scaffer's F6 Function*	600
Funções Compostas	21	*Composition Function 1 (n=5,Rotated)*	700
	22	*Composition Function 2 (n=3,Unrotated)*	800
	23	*Composition Function 3 (n=3,Rotated)*	900
	24	*Composition Function 4 (n=3,Rotated)*	1000
	25	*Composition Function 5 (n=3,Rotated)*	1100
	26	*Composition Function 6 (n=5,Rotated)*	1200
	27	*Composition Function 7 (n=5,Rotated)*	1300
	28	*Composition Function 8 (n=5,Rotated)*	1400
Espaço de busca $[-100, 100]^n$			

5.2 Ajuste de Parâmetros

Foram avaliadas todas as instâncias aqui utilizadas através de um esforço de ajuste de parâmetros. Foi utilizado uma implementação padrão do método *Iterated F-Racing* (pacote R `irace`, versão 1.0.7) (López-Ibáñez *et al.*, 2011).

O DE com cada operador de recombinação foi ajustado independentemente para problemas com 5 e 20 dimensões, resultando em um conjunto de 32 parâmetros ajustados. Os parâmetros padrão do `irace` versão 1.0.7 foram utilizados em todas as execuções do processo de ajuste. A tabela 5.2 apresenta os parâmetros ajustados para cada estratégia de recombinação em cada dimensionalidade[1]

Os resultados do ajuste de parâmetros foram necessários para a escolha dos valores a serem utilizados no DE, no que diz respeito a classe de problemas do CEC2013 que foram utilizados na avaliação comparativa dos operadores de recombinação. É interessante notar que os valores ajustados do parâmetro ϕ assumem de forma consistente valores maiores que o intervalo normalmente recomendado de [0,2]. Este fenômeno, em conjunto com o mecanismo de truncamento utilizado para reparar as soluções candidatas fora do espaço definido por restrição de caixa, sugere uma exploração de preferência mais intensiva dos limites do espaço de busca, pelo menos nas fases iniciais da pesquisa.

[1]Para a recombinação *npoint*, um valor de $n = 0$ significa que um número aleatório de pontos é utilizado em cada operação e a *onepoint*, com $k = 0$ indica que uma posição aleatória de corte foi utilizada em cada operação.

Tabela 5.2: Valores dos parâmetros ajustados.

Operador	Param.	Variação	Valores ajustados	
			$(5N)$	$(20N)$
arith	μ	$(2N, 20N)$	11	297
	ϕ	$(0.01, 5)$	1.56	4.18
	Mut.	$\{rand, best\}$	*best*	*rand*
bin	μ	$(2N, 20N)$	42	322
	ϕ	$(0.01, 5)$	2.88	3.09
	Mut.	$\{rand, best\}$	*best*	*best*
	ρ	$(0, 1)$	0.51	0.72
blx-α	μ	$(2N, 20N)$	78	228
	ϕ	$(0.01, 5)$	3.93	3.85
	Mut.	$\{rand, best\}$	*rand*	*rand*
	α	$(0, 0.5)$	0.38	0.38
blx-α*-*β	μ	$(2N, 20N)$	60	82
	ϕ	$(0.01, 5)$	3.47	3.09
	Mut.	$\{rand, best\}$	*best*	*rand*
	α	$(0, 0.5)$	0.23	0.11
	β	$(0, 0.5)$	0.04	0.39
eigen/*bin*	μ	$(2N, 20N)$	66	86
	ϕ	$(0.01, 5)$	4.78	2.82
	Mut.	$\{rand, best\}$	*rand*	*best*
	ρ	$(0, 1)$	0.54	0.92
exp	μ	$(2N, 20N)$	72	130
	ϕ	$(0.01, 5)$	4.80	1.95
	Mut.	$\{rand, best\}$	*best*	*best*
	ρ	$(0, 1)$	0.30	0.57
flat	μ	$(2N, 20N)$	98	204
	ϕ	$(0.01, 5)$	4.10	4.08
	Mut.	$\{rand, best\}$	*rand*	*rand*
geo	μ	$(2N, 20N)$	59	70
	ϕ	$(0.01, 5)$	0.71	1.17
	Mut.	$\{rand, best\}$	*best*	*rand*
	α	$(0, 1)$	0.10	0.59

lbga	μ	$(2N, 20N)$	87	286
	ϕ	$(0.01, 5)$	3.13	4.64
	Mut.	$\{rand, best\}$	*best*	*rand*
linear	μ	$(2N, 20N)$	78	256
	ϕ	$(0.01, 5)$	3.14	1.44
	Mut.	$\{rand, best\}$	*rand*	*rand*
mmax	μ	$(2N, 20N)$	66	373
	ϕ	$(0.01, 5)$	4.58	4.08
	Mut.	$\{rand, best\}$	*best*	*best*
	λ	$(0, 1)$	0.43	0.24
npoint	μ	$(2N, 20N)$	39	225
	ϕ	$(0.01, 5)$	3.02	2.22
	Mut.	$\{rand, best\}$	*rand*	*rand*
	n	$\{0, \ldots, n-1\}$	4	17
onepoint	μ	$(2N, 20N)$	60	223
	ϕ	$(0.01, 5)$	1.86	2.40
	Mut.	$\{rand, best\}$	*best*	*best*
	k	$\{0, \ldots, n-1\}$	3	17
pbest	μ	$(2N, 20N)$	21	328
	ϕ	$(0.01, 5)$	4.01	3.49
	Mut.	$\{rand, best\}$	*rand*	*rand*
	ρ	$(0, 1)$	0.58	0.26
	$t_{\max}$	Fixo	1000	1000
sbx	μ	$(2N, 20N)$	56	213
	ϕ	$(0.01, 5)$	4.27	4.49
	Mut.	$\{rand, best\}$	*rand*	*best*
	η	$(0, 100)$	96.53	89.07
wright	μ	$(2N, 20N)$	45	113
	ϕ	$(0.01, 5)$	4.01	4.81
	Mut.	$\{rand, best\}$	*rand*	*best*

5.3 Planejamento Experimental

Um planejamento completamente independente foi utilizado para cada dimensão, tal como é descrito a seguir.

Dentro de cada instância (problema), trinta execuções independentes do DE com cada variante de recombinação foram realizadas e resumidas pelo valor de desempenho mediano, resultando em um planejamento completo em blocos (Montgomery, 2012). Dada a grande variabilidade entre as instâncias e o fato de que o objetivo do experimento é avaliar o desempenho relativo ao longo do conjunto de teste (ao invés de, por exemplo, quantificar o desempenho absoluto), foi realizada uma transformação dos dados em *ranks*, com o menor (melhor) valor mediano em cada instância recebendo o *rank* 1, o segundo melhor 2, e assim por diante.

O teste de *Friedman* foi utilizado para testar a existência de diferenças significativas no desempenho da classificação mediana dos operadores de recombinação. Em caso de resultados significativos no teste de *Friedman*, testes pareados todos-contra-todos foram realizados para identificar as diferenças específicas entre os métodos.

Se resultados significativos forem detectados, testes todos-contra-todos, teste de *post-hoc Conover* (Conover, 1999) com correção do nível de significância do tipo FDR (Shaffer, 1995) foram empregados para identificar as diferenças e inicializar os intervalos de confiança na classificação média que foram gerados para permitir uma discussão gráfica dos resultados. Todos os testes e intervalos foram gerados ao nível de confiança de 95%.

5.4 Resultados

A figura 5.1 ilustra a distribuição da classificação obtida por cada operador sobre as instâncias de teste, para as dimensões $n = 5$ e $n = 20$. Para ambas as dimensões consideradas, o teste de *Friedman* detectou resultados altamente significativos ($p < 2 \times 10^{-16}$), sugerindo a presença de pelo menos um par de operadores com uma diferença considerável no desempenho mediano. O teste de *post-hoc Conover* indicou várias diferenças significativas entre pares de operadores, que são fornecidos nas tabelas 5.3 e 5.4, e confirmados pela análise gráfica apresentado na Figura 5.2, que mostra os intervalos de confiança sobre a classificação média para cada operador.

Tabela 5.3: p-valores para os testes de Conover 5D

	arith	bin	blx-α	blx-$\alpha-\beta$	eigen	exp	flat	geo	lbga	linear	mmax	npoint	onepoint	pbest	sbx
bin	0.000														
blx-α	0.000	0.000													
blx-$\alpha-\beta$	0.001	0.000	0.002												
eigen	0.014	0.000	0.000	0.342											
exp	0.000	0.000	0.085	0.205	0.026										
flat	0.000	0.000	0.094	0.192	0.023	0.960									
geo	0.301	0.000	0.000	0.018	0.161	0.000	0.000								
lbga	0.280	0.000	0.000	0.000	0.000	0.000	0.000	0.033							
linear	0.000	0.896	0.000	0.000	0.000	0.000	0.000	0.000	0.000						
mmax	0.000	0.000	0.838	0.005	0.000	0.137	0.148	0.000	0.000	0.000					
npoint	0.000	0.000	0.137	0.137	0.014	0.838	0.871	0.000	0.000	0.000	0.205				
onepoint	0.000	0.000	0.003	0.960	0.323	0.220	0.205	0.016	0.000	0.000	0.006	0.148			
pbest	0.000	0.012	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.018	0.000	0.000	0.000		
sbx	0.220	0.000	0.000	0.000	0.000	0.000	0.000	0.023	0.896	0.000	0.000	0.000	0.000	0.000	
wright	0.000	0.342	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.422	0.000	0.000	0.000	0.127	0.000

Tabela 5.4: p-valores para os testes de Conover 20D

	arith	bin	blx-α	blx-$\alpha-\beta$	eigen	exp	flat	geo	lbga	linear	mmax	npoint	onepoint	pbest	sbx
bin	0.000														
blx-α	0.000	0.008													
blx-$\alpha-\beta$	0.000	0.754	0.003												
eigen	0.001	0.000	0.000	0.000											
exp	0.251	0.000	0.000	0.000	0.041										
flat	0.000	0.105	0.000	0.192	0.000	0.000									
geo	0.000	0.059	0.000	0.117	0.000	0.000	0.791								
lbga	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000							
linear	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.192						
mmax	0.000	0.230	0.142	0.129	0.000	0.000	0.005	0.002	0.000	0.000					
npoint	0.000	0.000	0.000	0.000	0.000	0.000	0.029	0.055	0.070	0.002	0.000				
onepoint	0.000	0.066	0.000	0.129	0.000	0.000	0.828	0.955	0.000	0.000	0.002	0.049			
pbest	0.000	0.000	0.083	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.000		
sbx	0.452	0.000	0.000	0.000	0.015	0.717	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
wright	0.000	0.002	0.659	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.055	0.000	0.000	0.200	0.000

Os primeiros resultados interessantes observados são o desempenho relativo das variantes padrão `/bin` e `/exp`. O `/bin` não obteve um desempenho favorável em relação as demais variantes em ambas as dimensões, enquanto o `/exp` obteve um bom resultado, mas apenas no cenário de $n = 20$. Dado que o `/bin` é aparentemente uma das variantes de recombinação mais utilizadas em trabalhos de aplicação de evolução diferencial, este resultado sugere que pesquisadores e profissionais aplicando o DE básico para a solução de problemas aplicados poderiam adotar escolhas mais eficientes de operador de recombinação padrão.

O resultado para a variante `/exp` é interessante: embora produzindo um desempenho médio inferior para $n = 5$, alcançou um dos melhores resultados médios para $n = 20$, o que poderia indicar uma melhoria de desempenho quando a dimensão do problema é aumentado. Esta conclusão não pode, contudo, ser embasada apenas utilizando evidências de duas dimensões, mas utilizando uma abordagem baseada em princípios para modelagem dos efeitos de aumento de dimensão do problema sobre o desempenho dos operadores.

Uma outra variante cuja tendência é semelhante à do `/exp` é a `/eigen/bin`. Este método apresenta não só o melhor, como também possui o desempenho mais consistente para $n = 20$. Para $n = 5$ o `/eigen/bin` proporcionou resultados melhores que o `/bin` puro, embora de qualidade menos impressionante. Esta melhoria era esperada pois a variante `/eigen/*` tenta fazer com que o problema seja separável, e o método `/bin` é conhecido por ter um bom funcionamento em problemas separáveis (Price *et al.*, 2006). Uma investigação mais aprofundada deste efeito é a exploração da variante `/eigen` com diferentes operadores secundários, que são conhecidos por não ser de rotação invariável (por exemplo, `/eigen/exp`) e continua a ser uma questão em aberto, a ser investigada em trabalhos futuros.

A partir dos resultados mostrados nas Figuras 5.1–5.2, dois outros métodos de recombinação merecem uma atenção especial por sua consistência no bom desempenho em ambas dimensionalidades: as variantes `/arith` e `/sbx`. A última é atualmente uns dos operadores padrão na literatura dos algoritmos genéticos para codificação real tanto para otimização simples quanto multiobjetivo, e os resultados obtidos nesse experimento parecem sugerir que a capacidade de exploração pode melhorar o desempenho da evolução diferencial. É interessante também notar que os valores definidos pelo procedimento de ajuste para o parâmetro η em ambas as dimensões são de cerca de 90, muito maiores do que o valor padrão encontrado na literatura dos algoritmos genéticos.

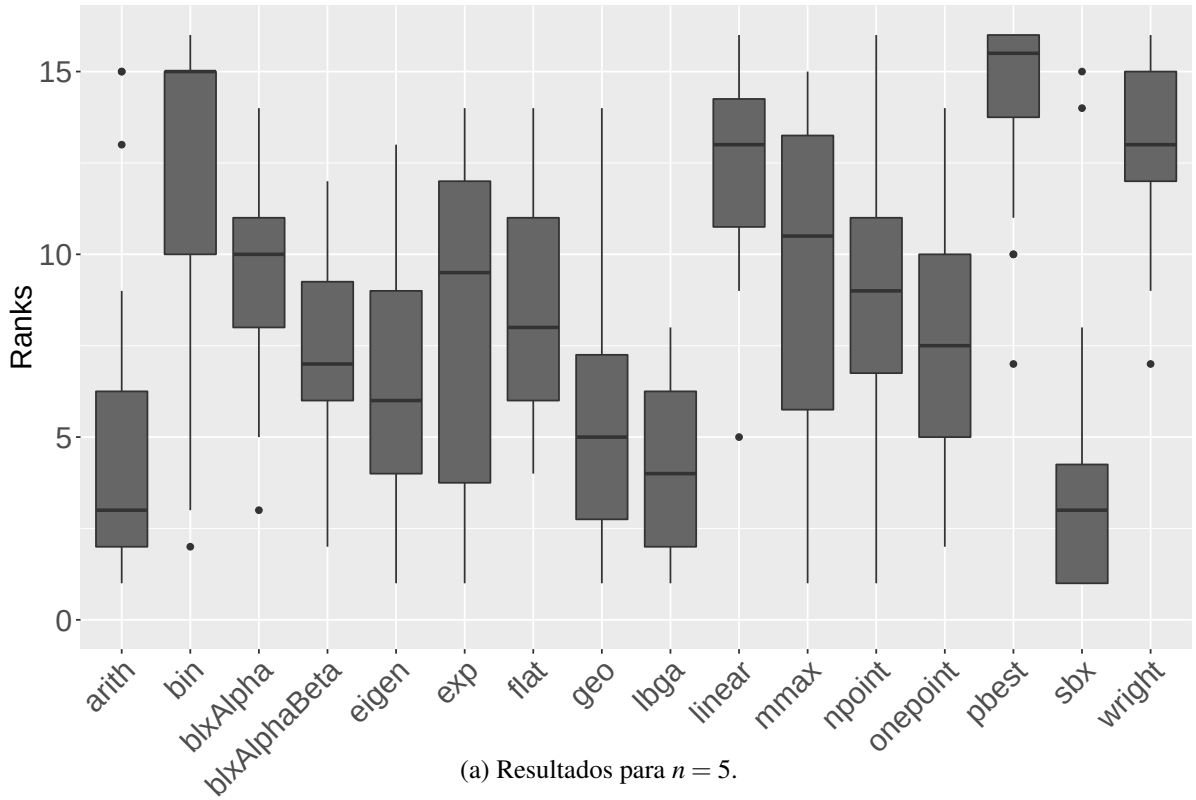
Em relação ao operador `/arith`, é muito importante observar que esta variante de recombinação é muito simples - o seu funcionamento corresponde nada mais do que uma in-

terpolação linear entre dois pontos - é capaz de produzir um desempenho tão bom quando comparado com outra variante mais complexa, incluídas nesta experiência.

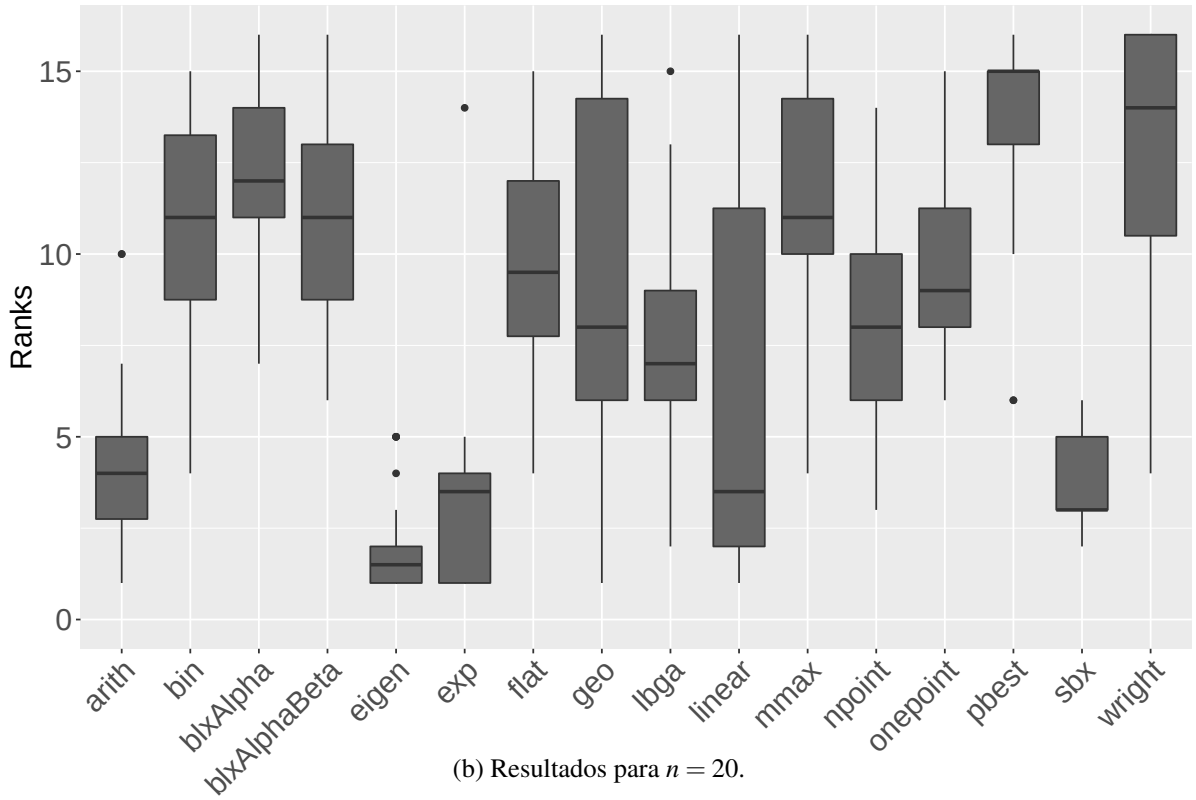
Uma outra observação interessante é que parece não haver nenhuma indicação da superioridade sistemática de qualquer variante de mutação: `mutation_rand` ou `mutation_best`, sendo a opção escolhida dependente da recombinação utilizada e, em muitos casos, da dimensão do problema. Uma investigação mais aprofundada dos efeitos e adequação das variantes de mutação, no entanto, estão fora do escopo deste trabalho, e isso é deixado como uma possibilidade para futuras pesquisas.

Finalmente, uma indicação da eficácia do processo de ajuste empregue pode ser observada através da comparação do desempenho relativo da variante `/blxAlphaBeta` com seus dois casos particulares `/blxAlpha` e `/flat`. Se o procedimento de ajuste converge para um bom conjunto de parâmetros, seria de se esperar que a variante mais geral supere ou, pelo menos, não seja inferior aos seus dois casos particulares. Além disso, é de se esperar que a variante `/blxAlpha` deve ser pelo menos tão boa quanto a `/flat`, uma vez que a anterior generaliza a última.

Este comportamento esperado é visto comparando pares entre `/blxAlphaBeta` e seus dois casos particulares: para $n = 5$, o `/blxAlphaBeta` é significativamente superior a ambos `/blxAlpha` e `/flat`, com estes dois apresentando diferenças estatisticamente significativas entre eles. Estes resultados supõem que o processo de ajuste gere valores que são geralmente consistentes com o que se espera em termos de resultados médios destas variantes, mas que não podem ter tido poder computacional suficiente para convergir para valores ótimos de parâmetros (uma vez que a recombinação `/flat` apresentou um valor médio de rank melhor que o `/blxAlpha` para $n = 20$, o que não deve acontecer dada a formulação mais geral deste último).

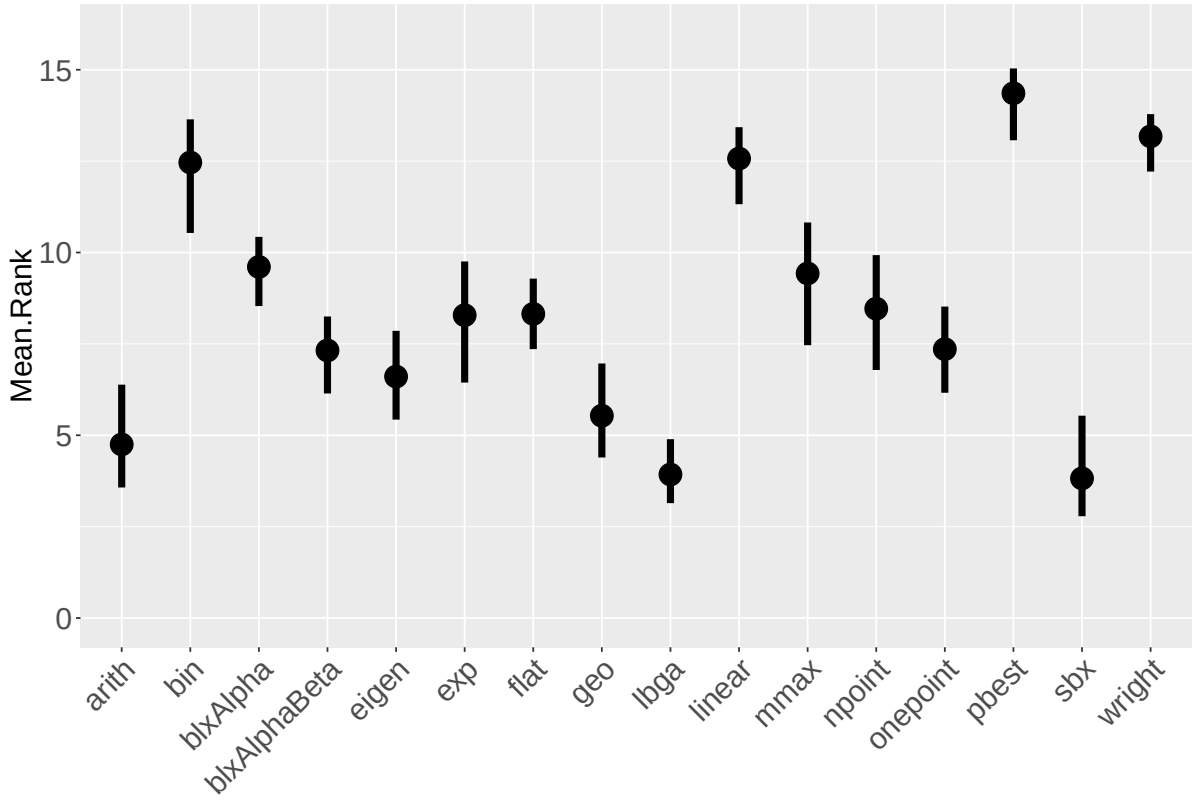


(a) Resultados para $n = 5$.

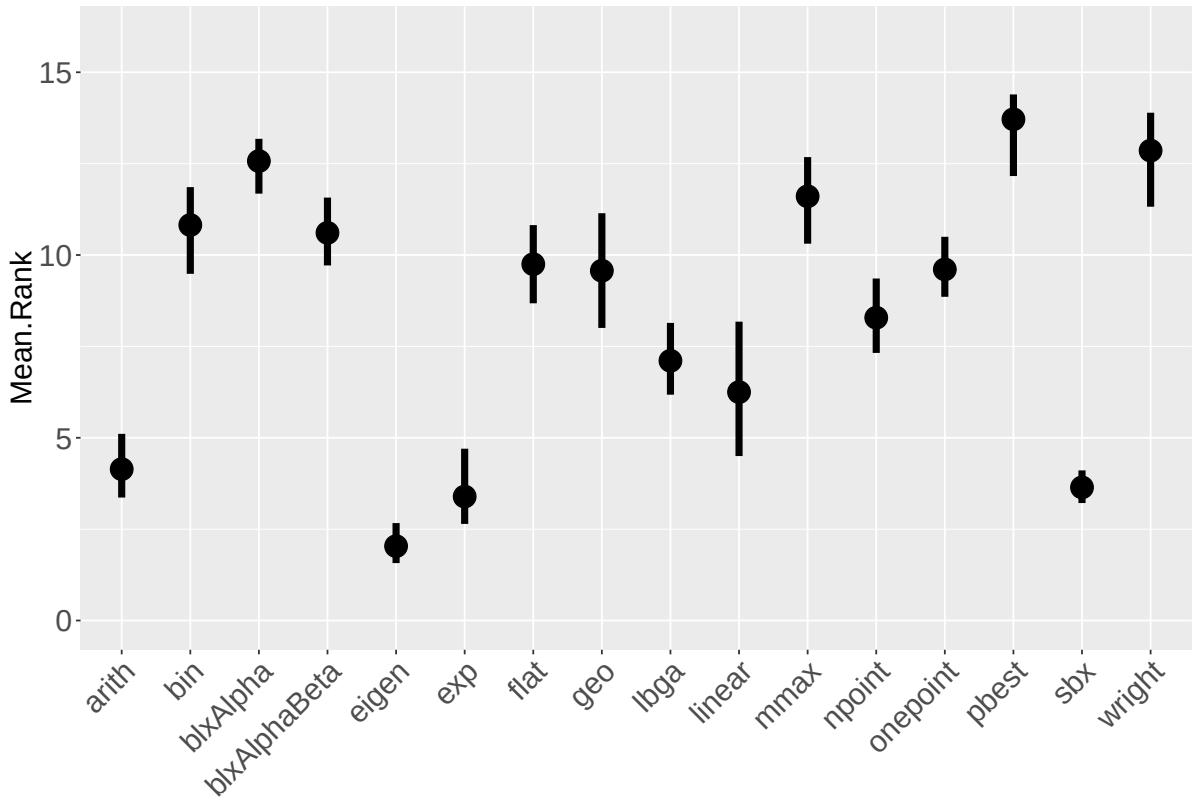


(b) Resultados para $n = 20$.

Figura 5.1 Distribuição dos resultados convertidos em rank.



(a) Intervalo de confiança para $n = 5$.



(b) Intervalo de confiança para $n = 20$.

Figura 5.2 Intervalo de confiança para o rank médio de cada variante de recombinação para $n = 5$ e $n = 20$. Observe o bom desempenho consistente das variantes $/arith$ e $/sbx$, bem como os resultados interessantes da $/lbga$ para $n = 5$ e da $/eigen$ e $/exp$ para $n = 20$.

CAPÍTULO 6

Conclusões e Trabalhos Futuros.

Uma compreensão apropriada do efeito dos operadores de recombinação no desempenho dos métodos evolucionários é um problema em aberto na literatura, e a partir disso, este trabalho apresentou uma avaliação experimental dos operadores de recombinação para a metaheurística evolução diferencial. Dezesseis variantes foram incluídas neste estudo, a maioria deles utilizados pela primeira vez no âmbito do DE. Todas as variantes foram redefinidas usando uma notação matemática unificada, que pretende ser uma contribuição para a padronização na literatura de metaheurísticas de forma a facilitar a compreensão e análise das características matemáticas destes operadores. Uma implementação modular no formato pacote `ExpDE` também foi desenvolvida com os objetivos de simplificar a replicabilidade de estudos e facilitar avaliações experimentais de diferentes aspectos do DE.

A análise experimental empregou uma abordagem de ajuste de parâmetros para a configuração do algoritmo DE, de forma a obter valores bem afinados para cada variante de recombinação. Além dos parâmetros específicos de cada método, os valores de tamanho da população, variante mutação (`/rand` or `/best`) e parâmetro de escala ϕ foram ajustados para todos os casos. Os experimentos foram realizados utilizando funções de benchmark com dimensionalidades $N = 5$ e $N = 20$ para investigar os efeitos das variantes de recombinação sobre o desempenho do DE.

Os resultados obtidos sugerem o uso de duas variantes de recombinação que são comuns na literatura GA de codificação real como alternativas interessantes para o DE: a recombinação *simulated binary recombination* (`/sbx`) e a recombinação *arithmetic* (`/arith`) apresentaram desempenho de boa qualidade em termos de média *rank* para ambas as dimensionalidades consideradas. A aplicação da recombinação *eigenvector-based* (`/eigen/`$\star$) também demonstrou ganhos de desempenho consideráveis para a variante secundária `/bin`, e merece uma investigação mais aprofundada com diferentes operadores secundários.

A partir dos resultados experimentais apresentados na Seção 5.4, uma possível recomendação para uma configuração do DE que funciona razoavelmente bem para ambas as dimensões testadas seria o **DE/$\star$/1/sbx**, com o $\star$ denotando tanto o `rand` quanto o `best` como

variantes de mutação. Esta variante parece funcionar bem com os valores dos parâmetros $\phi \approx 4$ e $\eta \approx 90$. A regra geral para o tamanho da população, $\mu \approx 10N$, parece ser adequada neste caso.

Além de uma investigação mais abrangente dos operadores de recombinação para o DE com a modelagem dos efeitos de dimensionalidade (particularmente para dimensões superiores), trabalhos futuros incluem: (i) um estudo semelhante de alternativas de mutação e de auto-adaptação de parâmetros de controle, juntamente com as extensões correspondentes no pacote `ExpDE`; e (ii) a investigação das correlações entre as características de desempenho do operador e modalidade de problemas como, por exemplo, problemas mal-condicionados, separáveis ou não-separáveis, e uni ou multi-modais.

CAPÍTULO 7

Apêndice I: Documentação do Pacote `ExpDE`

Package 'ExpDE'

March 28, 2016

Type Package

Title Modular Differential Evolution for Experimenting with Operators

Version 0.1.2

Depends R (>= 3.2.0)

Date 2016-03-25

Maintainer Felipe Campelo <fcampelo@ufmg.br>

Description Modular implementation of the Differential Evolution algorithm for experimenting with different types of operators.

License GPL-2

RoxygenNote 5.0.1

NeedsCompilation no

Author Felipe Campelo [aut, cre],
Moises Botelho [aut]

Repository CRAN

Date/Publication 2016-03-28 11:03:58

R topics documented:

`check_stop_criteria` *Stop criteria for DE*

Description

Implements different stop criteria for the ExpDE framework

Usage

```
check_stop_criteria()
```

Value

logical flag indicating whether any stop condition has been reached.

Warning

This routine accesses the parent environment used in the main function `ExpDE()`, which means that changes made in the variables contained in `env` WILL change the original values. DO NOT change anything unless you're absolutely sure of what you're doing.

`create_population` *Create population*

Description

Creates a new population for the ExpDE framework

Usage

```
create_population(popsize, probpars)
```

Arguments

popsize	population size
probpars	list of named problem parameters (see ExpDE).

Value

A matrix containing the population for the ExpDE

---

evaluate_population *Evaluate DE population*

Description

Evaluates the DE population on a given objective function.

Usage

```
evaluate_population(probpars, Pop)
```

Arguments

probpars	problem parameters (see ExpDE for details).
Pop	population matrix (each row is a candidate solution, normalized to the [0, 1] interval,)

Value

numeric vector (with length nrow(Pop)) containing the function values of each point in the population.

ExpDE *Experimental Differential Evolution - ExpDE*

Description

Modular implementation of the Differential Evolution Algorithm for the experimental investigation of the effects of different operators on the performance of the algorithm.

Usage

```
ExpDE(popsize, mutpars = list(name = "mutation_rand", f = 0.2),
  recpars = list(name = "recombination_bin", cr = 0.8, nvecs = 1),
  selpars = list(name = "standard"), stopcrit, probpars, seed = NULL,
  showpars = list(show.iters = "none"))
```

Arguments

popsize	population size
mutpars	list of named mutation parameters. See Mutation parameters for details.
recpars	list of named recombination parameters. See Recombination parameters for details.
selpars	list of named selection parameters. See Selection parameters for details.
stopcrit	list of named stop criteria parameters. See Stop criteria for details.
probpars	list of named problem parameters. See Problem Description for details.
seed	seed for the random number generator. See Random Seed for details.
showpars	parameters that regulate the echoing of progress indicators See Showpars for details.

Details

This routine is used to launch a differential evolution algorithm for the **minimization** of a given problem instance using different variants of the recombination, mutation and selection operators. The input parameters that describe those operators receive list objects describing the operator variants to be used in a given optimization procedure.

Value

A list object containing the final population (sorted by performance) , the performance vector, and some run statistics.

Mutation Parameters

mutpars is used to inform the routine the type of differential mutation to use, as well as any mutation-related parameter values. The current version accepts the following options:

- mutation_rand
- mutation_best

mutpars receives a list object with name field mutpars$name (containing the name of the function to be called, e.g., name = "mutation_rand") as well as whatever parameters that function may require/accept (e.g., mutpars$f = 0.7, mutpars$nvecs = 2, etc.). See the specific documentation of each function for details.

Some examples are provided in the Examples section below.

Recombination parameters

As with the mutation parameters, recpars is used to define the desired recombination strategy. The current version accepts the following options:

- recombination_arith
- recombination_bin
- recombination_blxAlpha

- recombination_blxAlphaBeta
- recombination_eigen
- recombination_exp
- recombination_flat
- recombination_geo
- recombination_lbga
- recombination_linear
- recombination_mmax
- recombination_npoint
- recombination_onepoint
- recombination_pbest
- recombination_sbx
- recombination_wright

recpars receives a list object with name field recpars$name (containing the name of the function to be called, e.g., name = "recombination_bin") as well as whatever parameters that function may require/accept (e.g., recpars$cr = 0.8, recpars$minchange = TRUE, etc.). See the specific documentation of each function for details.

Some examples are provided in the Examples section below.

Selection parameters

selpars follows the same idea as mutpars and recpars, and is used to define the selection operators. Currently, only the standard DE selection, selection_standard, is implemented.

Stop criteria

stopcrit is similar to recpar and the other list arguments, but with the difference that multiple stop criteria can be defined for the algorithm. The names of the stop criteria to be used are passed in the stopcrit$names field, which must contain a character vector. Other parameters to be used for stopping the algorithm (e.g., the maximum number of iterations stopcrit$maxiter) can also be included as stopcrit fields. Currently implemented criteria are:

- "stop_maxiter" (requires additional field stopcrit$maxiter = ? with the maximum number of iterations).
- "stop_maxeval" (requires additional field stopcrit$maxevals = ? with the maximum number of function calls).

See check_stop_criteria for details.

Problem description

The probpars parameter receives a list with all definitions related to the problem instance to be optimized. There are three required fields in this parameter:

- probpars$name, the name of the function that represents the problem to be solved.
- probpars$xmin, a vector containing the lower bounds of all optimization variables (i.e., a vector of length M, where M is the dimension of the problem).
- probpars$xmax, a vector containing the upper bounds of all optimization variables.

Important: the objective function routine must receive a matrix of row vectors to be evaluated in the form of an input parameter named either "x" or "X" or "Pop" (any one of the three is allowed).

Random Seed

The seed argument receives the desired seed for the PRNG. This value can be set for reproducibility purposes. The value of this parameter defaults to NULL, in which case the seed is arbitrarily set using as.numeric(Sys.time()).

Showpars

showpars is a list containing parameters that control the printed output of ExpDE. Parameter showpars can have the following fields:

- showpars$show.iters = c("dots", "numbers", "none"): type of output. Defaults to "numbers".
- showpars$showevery: positive integer that determines how frequently the routine echoes something to the terminal. Defaults to 1.

Author(s)

Felipe Campelo (<fcampelo@ufmg.br>) and Moises Botelho (<moisesufop@gmail.com>)

References

F. Campelo, M. Botelho, "Experimental Investigation of Recombination Operators for Differential Evolution", Genetic and Evolutionary Computation Conference, Denver/CO, July 2016 (Accepted).

Examples

```
# DE/rand/1/bin with population 40, F = 0.8 and CR = 0.5
popsize  <- 100
mutpars  <- list(name = "mutation_rand", f = 0.8)
recpars  <- list(name = "recombination_bin", cr = 0.5, minchange = TRUE)
selpars  <- list(name = "selection_standard")
stopcrit <- list(names = "stop_maxiter", maxiter = 100)
probpars <- list(name  = "sphere",
                 xmin = rep(-5.12,10), xmax = rep(5.12,10))
seed <- NULL
showpars <- list(show.iters = "numbers", showevery = 1)
ExpDE(popsize, mutpars, recpars, selpars, stopcrit, probpars, seed, showpars)
```

```
# DE/rand/2/blxAlpha
recpars  <- list(name = "recombination_blxAlpha", alpha = 0.1)
mutpars  <- list(name = "mutation_rand", f = 0.8, nvecs = 2)
ExpDE(popsize, mutpars, recpars, selpars, stopcrit, probpars)

# DE/best/1/sbx
recpars  <- list(name = "recombination_sbx", eta = 10)
mutpars  <- list(name = "mutation_best", f = 0.6, nvecs = 1)
ExpDE(popsize, mutpars, recpars, selpars, stopcrit, probpars)

# DE/best/1/eigen/bin
recpars  <- list(name = "recombination_eigen",
                 othername = "recombination_bin",
                 cr = 0.5, minchange = TRUE)
showpars <- list(show.iters = "dots", showevery = 10)
stopcrit <- list(names = "stop_maxeval", maxevals = 10000)
ExpDE(popsize, mutpars, recpars, selpars, stopcrit, probpars, seed = 1234)
```

`mutation_best` */best mutation for DE*

Description

Implements the "/best/nvecs" mutation for the ExpDE framework

Usage

```
mutation_best(X, mutpars)
```

Arguments

`X`	population matrix
`mutpars`	mutation parameters (see `Mutation parameters` for details)

Value

Matrix `M` containing the mutated population

Mutation Parameters

The `mutpars` parameter contains all parameters required to define the mutation. `mutation_best()` understands the following fields in `mutpars`:

- `f` : scaling factor for difference vector(s).
  Accepts numeric vectors of size 1 or `nvecs`.
- `nvecs` : number of difference vectors to use.
  Accepts `1 <= nvecs <= (nrow(X)/2 - 2)`
  Defaults to 1.

References

K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization", Springer 2005

`mutation_rand` */rand mutation for DE*

Description

Implements the "/rand/nvecs" mutation for the ExpDE framework

Usage

```
mutation_rand(X, mutpars)
```

Arguments

`X`	population matrix
`mutpars`	mutation parameters (see `Mutation parameters` for details)

Value

Matrix `M` containing the mutated population

Mutation Parameters

The `mutpars` parameter contains all parameters required to define the mutation. `mutation_rand()` understands the following fields in `mutpars`:

- `f` : scaling factor for difference vector(s).
  Accepts numeric vectors of size 1 or `nvecs`.
- `nvecs` : number of difference vectors to use.
  Accepts `1 <= nvecs <= (nrow(X)/2 - 2)`
  Defaults to 1.

References

K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization", Springer 2005

print_progress *Print progress of DE*

Description

Echoes the progress of DE to the terminal

Usage

```
print_progress()
```

Parameters

This routine accesses all variables defined in the calling environment using `parent.frame()`, so it does not require any explicit input parameters. However, the calling environment must contain:

- `showpars`: list containing parameters that control the printed output of `moead()`. Parameter `showpars` can have the following fields:
  - `$show.iters = c("dots", "numbers", "none")`: type of output. Defaults to `"numbers"`.
  - `$showevery`: positive integer that determines how frequently the routine echoes something to the terminal. Defaults to `1`.
- `iters()` : counter function that registers the iteration number

recombination_arith *Arithmetic recombination for DE*

Description

Implements the "/arith" (arithmetic) recombination for the ExpDE framework

Usage

```
recombination_arith(X, M, ...)
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`...`	optional parameters (unused)

Value

Matrix `U` containing the recombined population

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.

`recombination_bin` */bin recombination for DE*

Description

Implements the "/bin" (binomial) recombination for the ExpDE framework

Usage

```
recombination_bin(X, M, recpars)
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_bin()` understands the following fields in `recpars`:

- `cr` : component-wise probability of using the value in `M`.
  Accepts numeric value `0 < cr <= 1`.
- `minchange` : logical flag to force each new candidate solution to inherit at least one component from its mutated 'parent'.
  Defaults to TRUE

References

K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization", Springer 2005

recombination_blxAlpha *Blend Alpha recombination for DE*

Description

Implements the "/blxAlpha" (Blend Alpha) recombination for the ExpDE framework

Usage

```
recombination_blxAlpha(X, M, recpars)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
recpars	recombination parameters (see Recombination parameters for details)

Value

Matrix U containing the recombined population

Recombination Parameters

The recpars parameter contains all parameters required to define the recombination. recombination_blxAlpha() understands the following fields in recpars:

- alpha : extrapolation parameter.
  Accepts real value 0 <= alpha <= 0.5.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.

`recombination_blxAlphaBeta` *Blend Alpha Beta recombination for DE*

Description

Implements the "/blxAlphaBeta" (Blend Alpha Beta) recombination for the ExpDE framework

Usage

```
recombination_blxAlphaBeta(X, M, recpars)
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_blxAlpha()` understands the following fields in `recpars`:

- `alpha` : extrapolation parameter for 'best' parent vector.
  Accepts real value `0 <= alpha <= 0.5`.
- `beta` : extrapolation parameter for 'worst' parent vector.
  Accepts real value `0 <= beta <= 0.5`.

@section Warning: This recombination operator evaluates the candidate solutions in `M`, which adds an extra `popsize` evaluations per iteration.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.

recombination_eigen */eigen recombination for DE*

Description

Implements the "/eigen" (eigenvector-based) recombination for the ExpDE framework

Usage

```
recombination_eigen(X, M, recpars)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
recpars	recombination parameters (see Recombination parameters for details)

Value

Matrix U containing the recombined population

Recombination Parameters

The recpars parameter contains all parameters required to define the recombination. recombination_eigen() understands the following fields in recpars:

- othername: name of the recombination operator to be applied after the projection in the eigenvector basis
- ... : parameters required (or optional) to the operator defined by recpars$othername

References

Shu-Mei Guo e Chin-Chang Yang, "Enhancing differential evolution utilizing eigenvector-based crossover operator", IEEE Transactions on Evolutionary Computation 19(1):31-49, 2015.

recombination_exp *Exponential recombination for DE*

Description

Implements the "/exp" (exponential) recombination for the ExpDE framework

Usage

```
recombination_exp(X, M, recpars)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
recpars	recombination parameters (see Recombination parameters for details)

Value

Matrix U containing the recombined population

Recombination Parameters

The recpars parameter contains all parameters required to define the recombination. recombination_exp() understands the following fields in recpars:

- cr : component-wise probability of selection as a cut-point.
  Accepts numeric value 0 < cr <= 1.

References

K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization", Springer 2005

---

recombination_flat *Flat recombination for DE*

Description

Implements the "/flat" (Flat) recombination for the ExpDE framework

Usage

```
recombination_flat(X, M, ...)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
...	optional parameters (unused)

Value

Matrix U containing the recombined population

References

Picek, S.; Jakobovic, D.; Golub, M., "On the recombination operator in the real-coded genetic algorithms," CEC'2013, pp.3103-3110, 2013
F. Herrera, M. Lozano, J.L. Verdegay, "Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis", Artificial Intelligence Review 12 265-319, 1998.

`recombination_geo` *Geometric recombination for DE*

Description

Implements the "/geo" (geometric) recombination for the ExpDE framework

Usage

```
recombination_geo(X, M, recpars = list(alpha = 0.5))
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_geo()` understands the following fields in `recpars`:

- `alpha` : exponent for geometrical recombination.
  Accepts numeric value `0 <= alpha <= 1` or `NULL` (in which case a random value is chosen for each recombination).
  Defaults to `alpha = 0.5`.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.

recombination_lbga *Linear BGA recombination for DE*

Description

Implements the "/lbga" (Linear Breeder Genetic Algorithm) recombination for the ExpDE framework

Usage

```
recombination_lbga(X, M, ...)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
...	optional parameters (unused)

Value

Matrix U containing the recombined population

Warning

This recombination operator evaluates the candidate solutions in M, which adds an extra popsize evaluations per iteration.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.
D. Schlierkamp-voosen , H. Muhlenbein, "Strategy Adaptation by Competing Subpopulations", Proc. Parallel Problem Solving from Nature (PPSN III), 199-208, 1994.

recombination_linear *Linear recombination for DE*

Description

Implements the "/linear" recombination for the ExpDE framework

Usage

```
recombination_linear(X, M, ...)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
...	optional parameters (unused)

Value

Matrix U containing the recombined population

Warning

This recombination operator evaluates 3*popsize candidate solutions per iteration of the algorithm. The value of the nfe counter and the vector of performance values G are updated in the calling environment.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.
A.H. Wright, "Genetic Algorithms for Real Parameter Optimization", Proc. Foundations of Genetic Algorithms, 205-218, 1991.

---

recombination_mmax *Min Max Arithmetical recombination for DE*

Description

Implements the "/mmax" (min-max-arithmetical) recombination for the ExpDE framework

Usage

```
recombination_mmax(X, M, recpars = list(lambda = NULL))
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
recpars	recombination parameters (see Recombination parameters for details)

Value

Matrix U containing the recombined population

Warning

This recombination operator evaluates `4*popsize` candidate solutions per iteration of the algorithm. The value of the `nfe` counter and the vector of performance values `G` are updated in the calling environment.

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_pbest()` understands the following fields in `recpars`:

- `lambda` : Recombination multiplier.
  Optional. Defaults to `NULL` Accepts numeric value `0 < lambda < 1` or `NULL` (in which case a random value is independently used for each variable of each recombination pair).

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3):309-338, 2003.
F Herrera, M. Lozano, J.L. Verdegay, "Tuning fuzzy logic controllers by genetic algorithms.", International Journal of Approximate Reasoning 12(3):299-315, 1995.

`recombination_npoint` *n-point recombination for DE*

Description

Implements the "/npoint" (n-point) recombination for the ExpDE (as used in the Simple GA).

Usage

```
recombination_npoint(X, M, recpars = list(N = NULL))
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The recpars parameter contains all parameters required to define the recombination. recombination_npoint() understands the following fields in recpars:

- N : cut number points for crossover.
  Accepts integer value 0 <= N < n, where n is the dimension of the problem; Use N = 0 or N = NULL for randomly choosing a number of cut points.
  Defaults to NULL.

References

L.J. Eshelman, R.A. Caruana, J.D. Schaffer (1989), "Biases in the crossover landscape. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 10-19, San Francisco, CA, USA.

---

recombination_onepoint

One-point recombination for DE

Description

Implements the one-point recombination (as used in the Simple GA).

Usage

```
recombination_onepoint(X, M, recpars = list(K = NULL))
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
recpars	recombination parameters (see Recombination parameters for details)

Value

Matrix U containing the recombined population

Recombination Parameters

The recpars parameter contains all parameters required to define the recombination. recombination_onepoint() understands the following fields in recpars:

- K : cut point for crossover.
  Accepts integer value 0 <= K < n, where n is the dimension of the problem; Use K = 0 or K = NULL for randomly choosing a position for each pair of points.
  Defaults to NULL.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.

`recombination_pbest` *p-Best recombination for DE*

Description

Implements the "/pbest" (p-Best) recombination for the ExpDE framework

Usage

```
recombination_pbest(X, M, recpars)
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_pbest()` understands the following fields in `recpars`:

- `cr` : component-wise probability of using the value in `M`.
  Accepts numeric value `0 < cr <= 1`.

Warning

This routine will search for the iterations counter (`t`), the maximum number of iterations (`stopcrit$maxiter`), and the performance vector of population `X` (`J`) in the parent environment (using `parent.frame()`.
These variables must be defined for `recombination_pbest()` to work.

References

S.M. Islam, S. Das, S. Ghosh, S. Roy, P.N. Suganthan, "An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization", IEEE. Trans. Systems, Man and Cybernetics - Part B 42(2), 482-500, 2012

`recombination_sbx` */sbx recombination for DE*

Description

Implements the "/sbx" (Simulated Binary) recombination for the ExpDE framework

Usage

```
recombination_sbx(X, M, recpars)
```

Arguments

`X`	population matrix (original)
`M`	population matrix (mutated)
`recpars`	recombination parameters (see `Recombination parameters` for details)

Value

Matrix `U` containing the recombined population

Recombination Parameters

The `recpars` parameter contains all parameters required to define the recombination. `recombination_sbx()` understands the following field in `recpars`:

- `eta` : spread factor.
  Accepts numeric value `eta > 0`.

References

K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization", Springer 2005
F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.
K. Deb, R.B. Agrawal, "Simulated binary crossover for continuous search space", Complex Systems (9):115-148, 1995.

recombination_wright *Heuristic Wright recombination for DE*

Description

Implements the "/wright" (Heuristic Wright) recombination for the ExpDE framework.

Usage

```
recombination_wright(X, M, ...)
```

Arguments

X	population matrix (original)
M	population matrix (mutated)
...	optional parameters (unused)

Value

Matrix U containing the recombined population

Warning

This recombination operator evaluates the candidate solutions in M, which adds an extra popsize evaluations per iteration.

References

F. Herrera, M. Lozano, A. M. Sanchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study", International Journal of Intelligent Systems 18(3) 309-338, 2003.
A.H. Wright, "Genetic Algorithms for Real Parameter Optimization", Proc. Foundations of Genetic Algorithms, 205-218, 1991.

selection_standard *Standard selection for DE*

Description

Implements the standard selection (greedy) for the ExpDE framework

Usage

```
selection_standard(X, U, J, G)
```

Arguments

X	population matrix (original)
U	population matrix (recombined)
J	performance vector for population X
G	performance vector for population U

Value

list object containing the selected population (Xsel) and its corresponding performance values (Jsel).

Index

Referências Bibliográficas

Aarts e Lenstra(1997) Emile Aarts e Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edição. ISBN 0471948225. Citado na pág. 7

Ávila *et al.*(2006) Sérgio Luciano Ávila *et al.* Otimização multiobjetivo e análise de sensibilidade para concepção de dispositivos: aplicação: síntese de antenas refletoras para comunicação via satélite. Citado na pág. 9

Barricelli *et al.*(1954) Nils Aall Barricelli *et al.* Esempi numerici di processi di evoluzione. *Methodos*, 6(21-22):45–68. Citado na pág. 10

Beasley *et al.*(1993) David Beasley, RR Martin, e DR Bull. An overview of genetic algorithms: Part 1. fundamentals. *University computing*, 15:58–58. Citado na pág. 9

Birattari(2009) M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer. Citado na pág. 2

Borges(2013) André de Ávila Borges. Otimização de forma e paramétrica de estruturas treliçadas através dos métodos meta-heurísticos harmony search e firefly algorithm. Citado na pág. 9

Brest *et al.*(2006) Janez Brest, Sašo Greiner, Borko Bošković, Marjan Mernik, e Viljem Zumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657. Citado na pág. 2

Cai e Wang(2015) Yiqiao Cai e Jiahai Wang. Differential evolution with hybrid linkage crossover. *Information Sciences*, 320:244–287. Citado na pág. 24

Campelo(2016) Felipe Campelo. Computação evolutiva. https://github.com/fcampelo/Intro-Computacao-Evolutiva, 2016. Versão preliminar 0.1.0; Creative Commons BY-NC-ND 4.0. Citado na pág. 10

Chakraborty(2008) Uday K. Chakraborty. *Avanços na Evolução Diferencial*. ISBN 3540688277, 9783540688273. Citado na pág. 2

Conover(1999) W.J. Conover. *Practical Nonparametric Statistics*. Wiley, 3rd ed. edição. Citado na pág. 38

Crawley(2012) Michael J Crawley. *The R book*. John Wiley & Sons. Citado na pág. 25

Deb e Agrawal(1994) Kalyanmoy Deb e Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(3):1–15. Citado na pág. x, 22

Ecker e Kupferschmid(1983) Joseph G Ecker e Michael Kupferschmid. An ellipsoid algorithm for nonlinear programming. *Mathematical programming*, 27(1):83–106. Citado na pág. 9

Eshelman e Schaffer(1992) Larry J. Eshelman e J. David Schaffer. Real-Coded Genetic Algorithms and Interval-Schemata. Em *Foundations of Genetic Algorithms*, páginas 187–202. Citado na pág. 20

Eshelman *et al.***(1989)** Larry J. Eshelman, Richard A. Caruana, e J. David Schaffer. Biases in the crossover landscape. Em *Proceedings of the Third International Conference on Genetic Algorithms*, páginas 10–19, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3. Citado na pág. x, 17

Fogel(1964) Lawrence Jerome Fogel. *On the organization of intellect*. Tese de Doutorado. Citado na pág. 10

Fraser(1957) AS Fraser. Simulation of genetic systems by automatic digital computers. i. introduction. *Australian Journal of Biological Sciences*, 10:484–491. Citado na pág. 10

Gaspar-Cunha *et al.***(2012)** A. Gaspar-Cunha, R. Takahashi, e C.H. Antunes. *Manual de computação evolutiva e metaheurística*. Ensino. ISBN 9789892601502. Citado na pág. 9

Goldberg *et al.***(1989)** David E Goldberg *et al. Genetic algorithms in search optimization and machine learning*, volume 412. Addison-wesley Reading Menlo Park. Citado na pág. 10

Gomes *et al.***(2014)** Marília Miranda Forte Gomes, DA PESSOA, Luciana Alves Fernandes, JC SANTOS, e Ana Maria Nogales Vasconcelos. Estatística aplicada à engenharia e áreas afins incentivando meninas do ensino médio nas carreiras de ciências exatas, engenharias e computação. Em *COBENGE–CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA*, volume 42. Citado na pág. 1

Goulart *et al.***(2011)** F. Goulart, F. Campelo, e J.A. Ramírez. Estudo dos mecanismos de funcionamento do algoritmo de evolução diferencial. Em *Anais do X Congresso Brasileiro de Inteligência Computacional*. CD-ROM, paper ST26:2. In Portuguese. Citado na pág. 12

Guo e Yang(2015) Shu-Mei Guo e Chin-Chang Yang. Enhancing differential evolution utilizing eigenvector-based crossover operator. *Evolutionary Computation, IEEE Transactions on*, 19(1):31–49. Citado na pág. x, 23

Herrera *et al.***(2003)** F. Herrera, M. Lozano, e A. M. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338. ISSN 1098-111X. Citado na pág. x, 17, 19, 20, 21

Herrera *et al.***(1995)** Francisco Herrera, Manuel Lozano, e Jose L Verdegay. Tuning fuzzy logic controllers by genetic algorithms. *International Journal of Approximate Reasoning*, 12 (3):299–315. Citado na pág. x, 21

Hestenes e Stiefel(1952) Magnus Rudolph Hestenes e Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS. Citado na pág. 8

Holland(1992) John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Bradford book. MIT Press. ISBN 9780585038445. Citado na pág. x, 17

Islam *et al.***(2012)** Sk Minhazul Islam, Swagatam Das, Saurav Ghosh, Subhrajit Roy, e Ponnuthurai Nagaratnam Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(2):482–500. Citado na pág. x, 22

Leung e Wang(2001) Yiu-Wing Leung e Yuping Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 5(1):41–53. Citado na pág. 24

Liang *et al.***(2013)** JJ Liang, BY Qu, PN Suganthan, e Alfredo G Hernández-Díaz. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. Relatório técnico. Citado na pág. 33

Linden(2008) R. Linden. *Algoritmos Genéticos (2a edição)*. BRASPORT. ISBN 9788574523736. Citado na pág. 9

López-Ibáñez *et al.*(2011) Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, e Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Relatório Técnico TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium. URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf. Citado na pág. 35

Martha(2005) Luiz Fernando Campos Ramos Martha. *Traçado automático de envoltórias de esforços em estruturas planas utilizando um algoritmo evolucionário*. Tese de Doutorado, PUC-Rio. Citado na pág. 8

Maruo *et al.*(2005) Marcos H Maruo, Heitor S Lopes, e Myriam R Delgado. Self-adapting evolutionary parameters: encoding aspects for combinatorial optimization problems. Em *Evolutionary Computation in Combinatorial Optimization*, páginas 154–165. Springer. Citado na pág. 3

Michalewicz e Schoenauer(1996) Zbigniew Michalewicz e Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, 4 (1):1–32. Citado na pág. x, 19

Montgomery(2012) Douglas C. Montgomery. *Design and Analysis of Experiments*. Wiley, 8 edição. Citado na pág. 38

Nelder e Mead(1965) John A Nelder e Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313. Citado na pág. 12

Nocedal e Wright(2006) J. Nocedal e S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York. ISBN 9780387400655. Citado na pág. 1

Pinheiro(1998) Frederico Antônio Pinheiro. Aplicação de um algoritmo genético no estudo das perdas e do controle de tensão em sistemas elétricos de potência. *UFMG, Minas Gerais, Brasil. Dissertação de mestrado*. Citado na pág. 1

Plagianakos *et al.*(2008) VP Plagianakos, DK Tasoulis, e MN Vrahatis. A review of major application areas of differential evolution. Em *Advances in differential evolution*, páginas 197–238. Springer. Citado na pág. 13

Price *et al.*(2006) K. Price, R.M. Storn, e J.A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer Berlin Heidelberg. ISBN 9783540313069. Citado na pág. x, 2, 13, 17, 18, 41

Qiu *et al.*(**2012**) R.C. Qiu, Z. Hu, H. Li, e M.C. Wicks. *Cognitive Radio Communication and Networking: Principles and Practice*. Wiley. ISBN 9781118376294. Citado na pág. 9

Radcliffe(1991) Nicholas J Radcliffe. Equivalence class analysis of genetic algorithms. *Complex systems*, 5(2):183–205. Citado na pág. 20

Schlierkamp-Voosen e Mühlenbein(1994) Dirk Schlierkamp-Voosen e Heinz Mühlenbein. Strategy adaptation by competing subpopulations. Em *Parallel Problem Solving from Nature—PPSN III*, páginas 199–208. Springer. Citado na pág. x, 21

Shaffer(1995) Juliet Popper Shaffer. Multiple hypothesis testing. *Annual review of psychology*, 46:561. Citado na pág. 38

Storn e Price(1997) Rainer Storn e Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11 (4):341–359. Citado na pág. x, 12, 18

Syswerda(1991) Gilbert Syswerda. A study of reproduction in generational and steady state genetic algorithms. *Foundations of genetic algorithms*, 2:94–101. Citado na pág. x, 18

Takahashi(2004) Ricardo HC Takahashi. Otimização escalar e vetorial. *Notas de aula*. URL `http://www.mat.ufmg.br/~taka`. Citado na pág. 8

Takahashi *et al.*(**2010**) Ricardo Hiroshi Caldeira Takahashi, Jaime Arturo Ramırez, Felipe Campelo, e Frederico Gadelha Guimaraes. Notas de aula de otimizaçao. Citado na pág. 1, 8, 9, 22

Waintraub(2009) Marcel Waintraub. *Algoritmos Paralelos de Otimização por Enxame de Partículas em Problemas Nucleares*. Tese de Doutorado, Universidade Federal do Rio de Janeiro. Citado na pág. 10

Wang *et al.*(**2006**) Qingjiang Wang, Yun Gao, e Peishun Liu. Hill climbing-based decentralized job scheduling on computational grids. Em *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*, volume 1, páginas 705–708. IEEE. Citado na pág. 10

Wright(1991) Alden H Wright. Genetic algorithms for real parameter optimization. *Foundations of genetic algorithms*, 1:205–218. Citado na pág. x, 20, 23

Zambrano-Bigiarini e Gonzalez-Fernandez(2015) Mauricio Zambrano-Bigiarini e Yasser Gonzalez-Fernandez. *cec2013: Benchmark Functions for the Special Session and Competition on Real-Parameter Single Objective Optimization at CEC-2013*, 2015. URL http://CRAN.R-project.org/package=cec2013. R package v. 0.1-5. Citado na pág. 33