UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS

Populações Baseadas em *Multisets* para Algoritmos Evolutivos

"Documento Definitivo"

Doutoramento em Informática
Especialidade de Engenharia Informática

António Manuel Rodrigues Manso

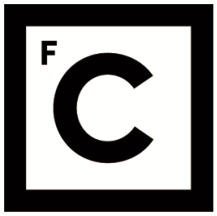
Tese orientada por:
Luís Miguel Parreira e Correia

Documento especialmente elaborado para a obtenção do grau de doutor

2020

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS



Ciências
ULisboa

Populações Baseadas em *Multisets* para Algoritmos Evolutivos

Doutoramento em Informática

Especialidade de Engenharia Informática

António Manuel Rodrigues Manso

Tese orientada por:

Luís Miguel Parreira e Correia

Presidente:

- Doutor Nuno Fuentecilla Maia Ferreira Neves, Professor Catedrático e Presidente do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa

Vogais:

- Doutor Ernesto Jorge Fernandes Costa, Professor Catedrático
  Faculdade de Ciências e Tecnologia da Universidade de Coimbra
- Doutor Francisco José Baptista Pereira, Professor Adjunto
  Instituto Superior de Engenharia de Coimbra do Instituto Politécnico de Coimbra
- Doutor Luís Miguel Parreira e Correia, Professor Associado com Agregação
  Faculdade de Ciências da Universidade de Lisboa
- Doutora Ana Luísa do Carmo Correia Respício, Professora Auxiliar
  Faculdade de Ciências da Universidade de Lisboa
- Doutora Sara Guilherme Oliveira da Silva Vanneschi, Investigadora Principal
  Faculdade de Ciências da Universidade de Lisboa

Documento especialmente elaborado para a obtenção do grau de doutor

2020

Dedico esta tese a todos os que todos aqueles que permitiram que este trabalho fosse realizado.

Resumo

Os algoritmos evolutivos simulam a evolução natural de uma população de indivíduos aplicando iterativamente operadores genéticos, recombinação, mutação e seleção dos mais aptos. O processo evolutivo pode ser visto como um processo de otimização. Nesse caso, os indivíduos representam soluções do problema e as variáveis do problema são codificados no equivalente aos genes. Estes algoritmos podem ser facilmente implementados e existem variantes especializadas para resolver várias classes de problemas.

Uma das maiores dificuldades apresentadas por estes algoritmos é a convergência prematura da população para soluções sub-ótimas antes do espaço de procura ser devidamente explorado. Várias estratégias foram desenvolvidas para reduzir este risco e, neste trabalho, estudamos a possibilidade de substituir a representação da população. Tradicionalmente as populações são representadas como coleções de indivíduos e nesta tese propomos a sua substituição por um multiconjunto (*multiset*). Esta nova forma de representação das populações, que denominamos multipopulações, permite manipular um conjunto de genomas e os seus clones, multi-indivíduos, de uma forma muito eficiente.

Adaptamos o processo evolutivo para otimizar multipopulações, estudamos o seu comportamento em vários tipos de algoritmos e problemas e desenvolvemos operadores genéticos especializados para trabalhar com a nova representação. Em resultado disso obtemos uma forma inovadora de manter uma elevada diversidade genética na população. As experiências realizadas permitiram-nos compreender melhor a dinâmica que a nova representação introduz no processo evolutivo e mostrar a sua eficácia.

Palavras chave:

Algoritmos evolutivos; populações; multiconjuntos; operadores genéticos.

Abstract

Evolutionary algorithms simulate the natural evolution of a population of individuals by iteratively applying genetic operators, recombination, mutation and selection of the fittest. The evolutionary process can be viewed as an optimization process. In this case, individuals represent problem solutions and the problem variables are encoded in that equivalent to the gene. These algorithms can be easily implemented and there are specialized variants to solve different classes of problems.

One of the biggest difficulties presented by these algorithms is the premature convergence of the population to suboptimal solutions before the search space is properly explored. Several strategies were developed to reduce this risk and, in this thesis, we studied the possibility of replacing the representation of the population.

Traditionally populations are represented as collections of individuals and in this thesis we propose its replacement by a multiset. This new form of population representation, which we call multipopulations, allows manipulating a set of genomes and their clones, multi-individuals, in a very efficient way.

We adapt the evolutionary process to optimize multipopulations, study their behavior on various types of algorithms and problems, and develop specialized genetic operators to work with the new representation. As a result, we get an innovative way to maintain a high genetic diversity in the population. The experiments allowed us to better understand the dynamics that the new representation introduces in the evolutionary process and show its effectiveness.

Keywords:

Evolutionary algorithms; populations; multisets; genetic operators.

Índice

Índice das figuras

Índice das Tabelas

Índice dos Algoritmos

Lista de Abreviaturas

AE	Algoritmo Evolutivo
MuCA	Multiset Crowding Algorithm
MDR	Multiset Decimation Replacement
MiTree	Multiset Indexed Tree
MuGA	Multiset Genetic Algorithm
MuGA-T	Multiset Genetic Algorithm Tool
MWM	Multiset Wave Mutation
SCA	Simple Crowding Algoritmo
SGA	Simple Genetic Algorithm
SMuGA	Symbiogenetic Multiset Algorithm

1 Introdução

A computação evolutiva é uma metáfora da evolução natural e o seu poder advém do facto de utilizar uma coleção de indivíduos que formam populações e evoluem em paralelo, utilizando o princípio da sobrevivência dos mais aptos. Muitas têm sido as áreas de aplicação desta técnica de otimização, cuja eficiência é incrementada através da especialização dos algoritmos. Estas especializações redefinem a forma como se codificam os genes e como estes são guardados dentro do cromossoma, os operadores genéticos aplicados ao cromossoma, bem como novas formas de interação dos indivíduos na população.

No entanto, a forma como se agrupam os indivíduos dentro da população, usualmente sob a forma de coleção, tem sido a regra ao longo desta especialização algorítmica.

1.1 Objetivo

Neste trabalho pretendemos explorar uma forma alternativa de representação das populações e dos indivíduos que a compõem, substituindo a coleção por um multiconjunto, *multisets*. Neste novo modelo de população, que designamos por multipopulação, os indivíduos são agrupados em clones, introduzindo uma nova dinâmica no processo evolutivo. A tendência dos algoritmos evolutivos em descobrir várias soluções de qualidade é aproveitada pela multipopulação através da preservação dos genomas, enquanto incrementam a quantidade de clones. Esta habilidade de preservar a quantidade e simultaneamente melhorar a qualidade difere dos algoritmos evolutivos convencionais.

O objetivo desta tese é desenvolver um conjunto de tecnologias que permitam fazer a substituição das tradicionais populações, baseadas em coleções, por multiconjuntos, aproveitando as possibilidades que o modelo fornece para desenvolver novas versões do processo evolutivo e dos operadores genéticos.

1.2 Metodologia

A ideia original de utilizar multisets para representar populações de algoritmos heurísticos foi proposta por (Correia, Moura-Pires, e Aparício 1999), que definiu formalmente a plataforma PLATO. Esta define de modo formal as populações, os indivíduos, os operadores genéticos e os algoritmos de otimização heurísticos, os quais

utilizam elementos definidos. Tomando como inspiração a PLATO, redefinimos a plataforma para a sua utilização em algoritmos evolutivos e testamos a sua eficácia e eficiência na otimização de vários tipos de problemas.

O artigo seminal desta tese, (Manso e Correia, 2009), fez a adaptação do algoritmo genético canónico, SGA, para a utilização de multiconjuntos na representação de populações, dando origem ao *Multiset Genetic Algorithm*, MuGA. Neste trabalho definimos o conceito de multi-indivíduo e de multipopulação, adaptamos o operador de seleção e o operador substituição de gerações, e introduzimos o operador de redimensionamento. Utilizando operadores genéticos tradicionais, constatamos experimentalmente a capacidade de otimização do MuGA e identificamos algumas fragilidades.

Redefinimos o operador de redimensionamento e utilizamos o MuGA para otimizar funções mais complexas codificadas por bits e verificamos, experimentalmente, a influência das multipopulações no processo evolutivo, nomeadamente a preservação da diversidade genética da população (Manso e Correia, 2011a) e a capacidade do algoritmo descobrir e reter vários ótimos na população (Manso e Correia, 2011). De forma a minimizar o esforço computacional, para a utilização de multipopulações desenvolvemos uma estrutura de dados baseada em árvores binárias balanceadas, MiTree – *Multiset indexed tree* (Manso e Correia, 2011b).

Para a otimização de problemas codificados por números reais com o MuGA, desenvolvemos os operadores de recombinação e de mutação baseados em multisets e o operador de substituição de multipopulações embutido com pesquisa local fornecida pelo algoritmo Nelder-Mead. Com esta configuração fizemos a otimização das funções propostas no *Congress on Evolutionary Computation* CEC2008 (Tang et al. 2007), onde obtivemos resultados que estão alinhados com os apresentados pelos algoritmos da competição.

Definimos o operador de mutação baseado em multisets, *Multiset Wave Mutation*, MWM, e utilizamos o MuGA para a otimização de funções enganosas (Manso e Correia, 2013a). Hibridizamos o MuGA com um algoritmo simbiótico e resolvemos problemas longos compostos por funções enganosas, (Manso e Correia, 2015) e (Correia e Manso, 2015b).

No decurso da nossa investigação foi desenvolvida a ferramenta MuGA-T, Anexo F, que permite fazer a exploração de multipopulações, utilizando vários tipos de algoritmos evolutivos e de operadores adaptados aos multisets. A ferramenta

implementa todos os algoritmos descritos nesta tese e permite executar todas as simulações. O código fonte está disponível para download no endereço https://github.com/Evolutionary-Algorithms.

1.3 Aspetos originais

Pelo facto de ser o primeiro trabalho explorando a ideia original de utilizar os multisets como suporte de populações de algoritmos evolutivos, o desenvolvimento de aspetos originais emergiu naturalmente. No decorrer da nossa investigação e experimentação, tivemos a necessidade de desenvolver e adaptar várias etapas do processo evolutivo, de forma a poderem ser utilizadas pela nova representação de população. Dentre os aspetos originais desta tese, salientamos a:

1) Implementação do conceito de multi-indivíduo e de multipopulação baseados em multisets e adaptação dos operadores de seleção de progenitores e substituição de gerações de forma a garantir as características invariantes;
2) Introdução de um novo operador, redimensionamento, para o controlo do número de cópias dos multi-indivíduos nas multipopulações; definição do *Multiset Genetic Algoritm*, MuGA, e verificação do seu desempenho na otimização de problemas codificados por bits e por permutações;
3) Aplicação de multipopulações em algoritmos de estado estacionário e definição dos procedimentos para a substituição de um multi-indivíduo dentro de uma multipopulação; definição do *Multiset Crowding Algorithm*, MuCA, e a sua aplicação a problemas rugosos e multimodais;
4) Definição do operador de substituição de gerações *ClearingReplace*, inspirado no proposto por (Petrowski 1996), que não necessita da parâmetros.
5) Utilização de multipopulações para suporte dos reprodutores e definição dos operadores de recombinação e de mutação para multi-indivíduos; otimização de problemas codificados por números reais utilizando o MuGA e os operadores adaptados;
6) Definição do operador de mutação *Mutiset Wave Mutation*, MWM, e o seu emprego para a otimização de funções enganosas; hibridização do MuGA com estratégias de simbiose, SMuGA, e a sua utilização para a otimização de problemas com funções enganosas complexas e longas.

7) Definição de uma métrica de diversidade genética normalizada e adaptada para várias representações de indivíduos.

As publicações geradas no âmbito deste trabalho estão enumeradas no Anexo A.

1.4 Estrutura da dissertação

No capítulo 2 apresentamos o estado da arte dos aspetos relacionados com o nosso trabalho.

No capítulo 3 é feita a definição formal de multipopulações e de multi-indivíduos que emergem da substituição dos conjuntos tradicionais pelos multisets como suporte da população de algoritmos evolutivos. Também são apresentadas as modificações dos operadores de seleção de progenitores e de substituição de gerações, de forma a manter as propriedades da multipopulação. Apresentamos um novo operador para controlo do número de cópias. Neste capítulo definimos também um conjunto de métricas que vão ser utilizadas nos capítulos seguintes para estudar o efeito das multipopulações nos algoritmos evolutivos.

No capítulo 4 definimos o algoritmo MuGA e verificamos experimentalmente a influência das multipopulações no processo evolutivo, utilizando um conjunto diversificado de problemas codificados por bits. Utilizamos o problema OneMax para estudar o efeito do número de cópias no processo evolutivo e a necessidade do operador de redimensionamento para o seu controlo. Escolhemos um problema representativo de uma classe de problemas e estudamos o comportamento do MuGA na sua otimização: O *Royal Road* para a otimização de problemas com planaltos; O *NK-Landscapes* para a otimização de funções rugosas; o *Knapsack* para a encontrar vários ótimos; e o *ChessQueen* para otimização de permutações.

No capítulo 5 utilizamos as multipopulações em conjunto com estratégias de nicho. Definimos o procedimento para a substituição e um indivíduo numa multipopulação e definimos o MuCA que utiliza estratégias de *crowding* com a competição dos progenitores com os respetivos descendentes. Definimos e o operador de substituição *ClearingReplace*, que utiliza uma estratégia de formação de formação de nichos através da eliminação dos indivíduos geneticamente mais próximos e utilizamos este operador no MuGA. Utilizamos o problema *Knapsack* e *NK-Landscapes* para estudar o comportamento das multipopulações nos algoritmos de nicho.

No capítulo 6 utilizamos as multipopulações para suporte dos progenitores e definimos os operadores de recombinação e de mutação para multi-indivíduos

codificados por números reais. Utilizamos o MuGA com estes novos operadores para estudar o comportamento das multipopulações na otimização do problema da esfera, e otimizamos as funções definidas no *Congress on Evolutionary Computation* CEC2008 para comparar a sua eficiência com os algoritmos que entraram na competição.

No capítulo 7 definimos o operador de mutação por onda, MWM, e o SMuGA, que utiliza uma estratégia simbiogenética em multipopulações. Usamos o SMuGA para a otimização de funções enganosas com genomas longos e complexos e comparamos os seus resultados com os publicados na literatura.

O capítulo 8 apresenta as conclusões gerais desta tese e o trabalho futuro.

2 Revisão bibliográfica

Os algoritmos evolutivos (AE) têm mostrado bons resultados na resolução de várias classes de problemas e, em especial, nos problemas de engenharia difíceis de resolver por outros métodos. Estes problemas são essencialmente caracterizados pela sua alta dimensionalidade, não linearidade e pouca informação sobre as suas propriedades.

A chave do sucesso dos AE baseia-se na utilização de uma coleção de soluções que evoluem simultaneamente, utilizando a metáfora da evolução natural proposta por Darwin.

2.1 Modelos de evolução

A evolução é uma teoria que explica a diversidade dos seres vivos existentes, defendendo que os seres vivos atuais tiveram origens noutros que os precederam. A grande variedade e diversidade de seres vivos que o mundo natural exibe teve como origem a acumulação de variações que o processo reprodutivo introduz nos descendentes. A evolução natural não é um processo aleatório, e nem todas estas variações são benéficas para os novos indivíduos, mas aquelas que o forem, tornam estes indivíduos mais aptos, permitindo que espalhem estas características nas gerações vindouras. A este processo de escolhas dos indivíduos mais aptos para se reproduzirem Darwin chamou de “seleção natural”, e a acumulação de diferenças deu origem às espécies (Charles Darwin 1859).

Os algoritmos evolutivos tentam mimetizar a evolução natural através da simplificação da representação dos indivíduos, do processo reprodutivo, da avaliação do grau de aptidão e do processo de seleção que promove a evolução das espécies. Esta simplificação é necessária para o tratamento computacional, no entanto, introduz problemas que não existem na evolução do mundo biológico.

Na natureza, devido à quantidade e diversidade de seres vivos que partilham o mesmo *habitat*, à interação entre elas, à complexidade interna e externa e à variedade dos mecanismos que afetam a sua reprodução, a variedade é a norma, e a evolução, um processo contínuo. Nos AE, devido à reduzida dimensão das populações artificiais, à rudimentar estrutura genética dos indivíduos, à simplificação do processo evolutivo e à eliminação de muitos outros fatores do meio natural, a perda da diversidade genética é uma realidade. Com a carência de diversidade genética da população, os AE perdem a

capacidade de exploração de novas áreas no espaço de procura e ficam presos em extremos locais.

2.2 Indivíduos e Populações

Na natureza nenhum ser vivo sobrevive de forma isolada. A resiliência da vida em recuperar de catástrofes e de proliferar em ambientes vantajosos é facultada pela capacidade emergente do conjunto de seres vivos que partilham o mesmo nicho ecológico. Na natureza os indivíduos são agrupados em populações, indivíduos da mesma espécie que partilham o mesmo habitat, e que interagem entre si de forma a produzir descendentes cada vez melhores, promovendo assim a evolução das espécies. No mundo natural, o ambiente força a seleção dos indivíduos que melhor aproveitam os recursos disponíveis e as espécies formam-se quando deixa de ser viável o acasalamento com indivíduos de outros nichos.

Os indivíduos têm uma representação interna, genótipo, que se traduzem em diversas características, que determinam a sua sobrevivência, o fenótipo. A transformação do genótipo no fenótipo é dada por uma série de processos com interferência do meio ambiente. O grau de aptidão é dado pela capacidade de sobreviver e de se reproduzir na população através da competição com os seus semelhantes.

Nos AE os indivíduos, que representam soluções do problema, são codificados internamente através de um conjunto dos seus genes. Os genes são a unidade fundamental de informação genética e representam as variáveis do problema que se pretende resolver. A computação evolutiva é muito utilizada para resolver problemas de otimização, sendo classificados em quatro grandes áreas, de acordo com o tipo de genes que os indivíduos possuem: a otimização binária, cujo objetivo é descobrir o valor lógico de cada gene; a otimização de valores reais, cujo intuito é descobrir o valor em vírgula flutuantes de cada gene; a otimização combinatória, que visa descobrir a ordem que os alelos devem ocupar no cromossoma; e, por fim, a otimização de estruturas, no qual o cromossoma é representado por uma estrutura não linear de genes, cujo objetivo é a determinação ótima dessa estrutura (Rothlauf 2006).

Nos seres vivos mais evoluídos, os genes estão agrupados em cromossomas, e o conjunto dos mesmos constitui o seu genoma. Os AE simplificam esta representação dos indivíduos e agrupam todos os genes num único cromossoma. O algoritmo canónico, *Simple Genetic Algorithm* (SGA) utiliza cromossomas haploides, em que cada gene armazena numa sequência de bits o valor da variável associada (Holland

1975). Outras representações têm sido propostas para a representação de vários valores alternativos para os genes dando origem a indivíduos multiploides (Collingwood, Corne, e Ross 1996). Para além do seu valor, os genes podem ter associado outro tipo de informação, tal como a probabilidade de mutação (Beyer e Schwefel 2002) ou a posição do alelo no cromossoma (D. E. Goldberg e Bridges 1990). Usualmente, os genes estão dispostos no cromossoma de forma linear, sob a forma de vetor. Os índices do vetor identificam o gene e estes permanecem nas mesmas posições ao longo do processo evolutivo. Os *Messy Genetic Algorithms* (D. E. Goldberg 1989) guardam a posição do gene no cromossoma e o respetivo alelo, e desta forma permitem aos genes o seu movimento no vetor, possibilitando o agrupamento de genes relacionados entre si. Uma forma diferente do cromossoma é uma estrutura, por exemplo, sob a forma de árvore (Koza 1992) ou de grafo (Doerr, Klein, e Storch 2007). Os indivíduos, para além do cromossoma, podem conter outro tipo de informação, como a espécie (Navalresearch e Spears 1995), a idade e o sexo (Ghosh, Tsutsui, e Tanaka 1998), ou a memória dos seus antepassados, como é o caso da *Particle Swarm Optimization* (Kennedy e Eberhart 1995).

As populações são usualmente constituídas por uma coleção de indivíduos que evoluem em paralelo através da utilização de operadores genéticos. Usualmente, são usados para produzir descendência, os operadores de seleção de progenitores para reprodução, os operadores de recombinação e de mutação, e os algoritmos de substituição de gerações. Os AE de estado estacionário substituem um indivíduo da população por um descendente; nos AE geracionais, a nova geração é formada através da competição entre os progenitores e os descendentes.

Uma forma diferente de representação de populações foi apresentada no *Compact Genetic Algorithm* proposto por Harik et al. (G. R. Harik, Lobo, e Goldberg 1999), na qual a população é um vetor de probabilidades para cada alelo do genoma. Esta representação deu origem a um novo conjunto de algoritmos conhecidos como *estimation of distribuition algorithms* (Hauschild e Pelikan 2011). Os modelos mais complexos permitem fazer a associações entre genes e estas dependências são representadas através de estruturas complexas como árvores (Dirk Thierens 2010) ou redes bayseanas (Pelikan 2005) .

A grande dimensão e complexidade dos problemas reais faz com que os AE canónicos não produzam resultados satisfatórios. Nos AE, quando aplicados a problemas difíceis, onde o espaço de procura é complexo, a dificuldade mais frequente

é a convergência prematura do algoritmo devido à falta de diversidade genética (Friedrich et al. 2008). Uma população pequena é incapaz de manter a diversidade genética durante várias gerações, e uma população grande necessita de elevados recursos computacionais para a sua evolução, e o aumento da sua dimensão não é garantia de diversidade. A pressão seletiva sobre os indivíduos com melhor aptidão possibilita que estes propaguem os seus genes em gerações sucessivas, levando a população à exploração das melhores soluções do nicho onde essa se encontra. Essa convergência da população, eventualmente prematura, impede que a recombinação das suas características lhes permita fazer a exploração de novas regiões do espaço. O balanceamento entre a capacidade de aproveitar as melhores soluções da população e a exploração do espaço de procura é fundamental para resolver problemas difíceis, e diversos algoritmos têm sido propostos para o conseguir (Glibovets e Gulayeva, 2013); (Črepinšek, Liu, e Mernik 2013).

2.3 Diversidade genética em algoritmos evolutivos

A diversidade genética da população proporciona a exploração global do espaço de procura através da recombinação das características existentes nos indivíduos; ao contrário, a sua falta provoca um aproveitamento das características existentes. Desta forma, a diversidade genética da população é de extrema importância na fase inicial do processo evolutivo no sentido de encontrar as regiões mais promissoras do espaço de procura, e a sua perda leva a uma adaptação dos indivíduos aos nichos que ocupam.

Uma população diversificada incrementa a robustez do processo evolutivo e é importante em diversos cenários de otimização A diversidade genética incrementa a eficiência da pesquisa, e diminui a possibilidade do algoritmo ficar retido em extremos locais, permitindo a exploração global do espaço de procura. Adicionalmente, uma população diversificada facilita o processo de decisão em ambientes ruidosos ou de incerteza, uma vez que fornece um conjunto de boas soluções (Sudholt 2020).

2.3.1 Diversidade populacional

A diversidade populacional pode ser definida como o grau de dissimilitude entre um conjunto de indivíduos e pode ser quantificada de diversas formas. Este cálculo pode ser feito ao nível do gene, em que são quantificadas as diferenças entre os alelos; ao nível individual, em que se considera o conjunto dos genes; ou através do fenótipo dos indivíduos, no qual são avaliadas as diferenças das características.

Ao nível do gene, podemos calcular o número de alelos diferentes, fixos, perdidos, ou relevantes para o problema (Wagner e Affenzeller 2005), calcular a distância de cada gene ao valor médio dos seus alelos, ou o seu desvio padrão. Podemos, ainda, calcular a probabilidade de um alelo ocorrer num gene da população e calcular a sua entropia (Diaz-Gomez e Hougen 2007).

Ao nível do indivíduo, podemos determinar a dissimilitude entre cada par utilizando a métrica de *Hamming* (Frederick, Sedlmeyer, e White 1993) para genes binários, ou a distância euclidiana, no caso de espaços numéricos. Utilizando as métricas anteriores a diversidade da população é calculada como a média da diversidade dos seus alelos.

Considerando a população como um todo, onde cada indivíduo ocupa um ponto no espaço, podemos calcular o seu volume, a média das distâncias ao centro de massa, ou a *minimum spanning tree* necessária para os unir (Lacevic e Amaldi 2010). A diversidade da população pode ser calculada eliminando os *outliers*, ou apenas utilizando alguns indivíduos representativos da população, como por exemplo, a frente de pareto (Fleischer 2003) nos algoritmos de otimização multiobjetivo.

As métricas apresentadas são específicas de uma representação e fornecem valores em escalas diferentes, o que dificulta a sua aplicação e comparação em diferentes tipos de problemas. De forma a contornar este problema, apresentamos nesta tese uma métrica normalizada no intervalo [0,1] e aplicada a diversas codificações de genes.

Existem várias estratégias que permitem promover e manter a diversidade genética na população. As estratégias podem ser simples, como a eliminação de duplicados ou a introdução de indivíduos aleatórios, ou mais complexas, como a definição de operadores genéticos especializados ou outros modelos de evolução, como se descreve em seguida.

2.3.2 Operadores genéticos e o seu impacto na diversidade

Os operadores genéticos são os responsáveis pela evolução dos indivíduos da população e, por consequência, pela diversidade que a mesma apresenta.

O operador de seleção desempenha um papel fundamental ao selecionar os indivíduos que se vão reproduzir. Se tiver uma pressão alta, apenas os melhores se vão reproduzir e a diversidade genética da população é afetada negativamente. Por outro lado, se tiver uma pressão baixa, a diversidade tende a aumentar, mas o progresso em direção ao ótimo torna-se mais lento.

O operador de recombinação é fundamental para a exploração do espaço entre os indivíduos e incrementa a eficiência dos algoritmos evolutivos (Jansen e Wegener 2005). Com a exceção da otimização de estruturas, este só introduz diversidade genética na população se os progenitores forem significativamente diferentes; caso contrário, produz descendentes semelhantes, e a diversidade genética introduzida é diminuta. Estratégias que previnam o incesto, (Craighurst e Worthy 1995), ou que restrinjam o acasalamento entre progenitores compatíveis em idade e sexo (Zhu, Yang, e Song 2006), ou do genótipo (Fernandes et al. 2001), incrementam a eficácia do operador.

A mutação é importante no processo evolutivo, pois é a responsável pela introdução de novos alelos na população ou na recuperação dos que foram perdidos. Uma taxa de mutação muito alta degenera em pesquisa aleatória, e uma taxa muito baixa, reduz a possibilidade de descoberta de bons alelos. A definição da taxa de mutação pode variar periodicamente, como no caso da hipermutação proposta por (Cobb 1990), ou pode adaptar-se às características da população, como o proposto por (D. Thierens 2002).

Para além da reprodução e da mutação, a natureza possui outros mecanismos, tais como a inversão, a transdução, a conjugação, a transposição e a translocação, que podem ser utilizados no processo evolutivo com vista à manutenção da diversidade genética das populações (Simões e Costa, 2001).

Os operadores genéticos anteriores são responsáveis pela criação de descendentes que vão competir por um lugar na população principal. A substituição de gerações pode ser: total, e, neste caso, é necessário garantir que a população dos descendentes seja suficientemente diversa para continuar o processo evolutivo; ou parcial, onde uma parte dos progenitores dá lugar aos descendentes.

A substituição de um progenitor por um descendente deve ser feita com base na aptidão, de forma a introduzir na população indivíduos mais bem-adaptados e promover a evolução da população.

A competição dos descendentes com os respetivos progenitores promove a formação de nichos. Nas estratégias de *crowding* (Mengshoel e Goldberg 2008), os progenitores competem com os descendentes, e a sua substituição pode ser determinística ou probabilística; na recombinação elitista (D. Thierens e Goldberg 1994), os dois melhores dos quatro indivíduos são preservados; na estratégia *keep-best*, o melhor pai e o melhor filho são preservados (Wiese e Goodwin 2001); por fim, na estratégia c*orrelative family-based*, o melhor indivíduo é escolhido e o outro é o mais diferente dele (Matsui 1999).

Uma forma diferente de competição é definida por (G. Harik 1995), que propõe que a competição seja com o indivíduo geneticamente mais próximo da população, *restrictive tournament selection*. (Lozano, Herrera, e Cano 2005) sugere uma estratégia que combina a contribuição do descendente para a diversidade da população e o seu grau de aptidão.

Uma forma distinta de promover a diversidade genética da população é diminuir a aptidão dos indivíduos que ocupam o mesmo nicho, ou seja, que sejam geneticamente semelhantes. Na estratégia de *fitness sharing*, proposta por (D. Goldberg e Richardson 1987), o valor da aptidão dos indivíduos é partilhada por aqueles que ocupam o mesmo nicho ecológico. Desta forma, as áreas densamente ocupadas tornam-se menos atrativas, promovendo a mobilidade dos indivíduos na população. Ao invés de partilhar os recursos pelos indivíduos da espécie, (Petrowski 1996) propõe a simplificação do método de partilha em que apenas o melhor individuo da espécie recebe o seu valor de aptidão, limpando, *clearing*, os restantes indivíduos do nicho.

Uma forma que a natureza encontrou para evoluir continuamente as espécies foi a morte. Todos os seres vivos crescem e envelhecem ao longo do tempo e o seu potencial de reprodução diminui até que a morte os elimine. Este conceito foi explorado por (Ghosh, Tsutsui, e Tanaka 1998), que utiliza o conceito de idade para diminuir a aptidão e, assim, limitar a sua capacidade de reprodução e de competição com os descendentes.

2.3.3 Modelos evolutivos orientados pela diversidade genética

O poder dos AE advém da utilização dos operadores genéticos apresentados, e a sua escolha e parametrização tem uma grande influência na evolução da população. O processo evolutivo pode sofrer alterações de forma a conservar e promover a diversidade genética da população. Em alguns algoritmos evolutivos multiobjectivo a diversidade pode ser um deles (Toffolo e Benini 2003). Os *Multinational algorithms* (R.K. Ursem 1999) atribuem a cada indivíduo uma nacionalidade, formando subpopulações que evoluem individualmente e que mantêm relações geridas por politicas globais. O *Diversity guided algorithms* (Rasmus K Ursem 2002) alterna entre os operadores de recombinação (fase de aproveitamento) e de mutação (fase de exploração), de acordo com a diversidade genética da população.

A maioria dos AE usa uma população de indivíduos e aplica sobre eles os operadores genéticos, de forma a promover a sua evolução. Devido à natureza aleatória do processo evolutivo, a população evolui de maneiras distintas e o resultado é diferente a cada

simulação. O modelo de evolução de populações independentes, *parallel genetic Algorithms* (Cantú-Paz 1998), explora esta característica através de um conjunto de populações que evoluem de forma independente e, por conseguinte, a diversidade de indivíduos do conjunto das populações é elevado.

No modelo de evolução distribuído por ilhas, a população é dividida em pequenas subpopulações, que evoluem de forma isolada, eventualmente com parâmetros e operadores genéticos diferentes (Alba et al. 2004), de forma a promover a especiação (Li et al. 2002). Os indivíduos ocasionalmente migram entre as ilhas, introduzindo, desta forma, material genético novo, tendo por objetivo evitar a estagnação em extremos locais. A migração constitui-se como novo operador genético com um conjunto novo de parâmetros: taxa de migração, política de integração dos migrantes e, finalmente, topologia das comunicações entre ilhas.

A topologia das ilhas exerce um papel importante no desempenho do sistema evolutivo, e os algoritmos celulares atribuem às ilhas uma posição no espaço, comunicando cada uma apenas com as vizinhas como se de um corpo celular se tratasse.

Nos modelos anteriores os indivíduos têm todos o mesmo objetivo, que é a otimização do mesmo problema, e embora haja uma especiação virtual, conseguem reproduzir-se entre eles.

Outro modelo de evolução é o que integra várias espécies diferentes, que partilham o mesmo ambiente e interagem entre si, mas que têm objetivos diferentes. A interação entre as espécies pode ser competitiva, onde cada uma tenta superar a outra (Seredynski 1994), ou cooperativa, na qual a interação entre as espécies permite a otimização de um objetivo comum (Potter e Jong 2000). A simbiogénese (Wallin, Ryan, e Azad 2005) é um exemplo de cooperação em que as espécie colaboram obtendo benefício mútuo (Manso e Correia 2015).

2.4 Multisets em algoritmos evolutivos

Os algoritmos evolutivos e os respetivos mecanismos de evolução genética permitem a liberdade de desenvolver algoritmos que não têm paridade no mundo natural (Correia 2010), como por exemplo, a utilização de multisets para representar populações

Os multisets são uma extensão dos conjuntos “set” de forma a suportar elementos repetidos (Blizard 1991), sendo também conhecidos em matemática como *bag* ou *mset*. O formalismo para a utilização de multisets como suporte de populações foi

inicialmente proposta por (Aparício, Correia, e Moura-Pires 1999). Este conceito matemático foi também utilizado para análise de populações (Schaefer, 2007 pp-39). O termo multiset aparece várias vezes na literatura e, na maioria dos casos, como sinónimo de coleção, o modelo convencional de representação de populações em AE. Do trabalho desta tese resultou o artigo que define o conceito de multi-indivíduo como um elemento do multiset, e de multipopulação como o multiset em si (Manso e Correia 2009). Na literatura o conceito de multipopulação aparece referenciado várias vezes, mas é sempre sinónimo de várias populações (Siarry, Pétrowski, e Bessaou 2002), e não de uma população organizada em multisets, como definido nesta tese.

A utilização dos multisets como suporte de algoritmos evolutivos tem dois grandes objetivos: incrementar a diversidade genética da população principal e incrementar a eficiência computacional com a avaliação automática de todos os clones. Esta estratégia de avaliação utiliza os indivíduos já avaliados da população para evitar novas avaliações à semelhança do que acontece com os algoritmos de avaliação por *caching* (Santos & Santos, 2001).

2.5 Comentários finais

Neste capítulo fizemos a revisão da literatura nos aspetos que estão conexos ao tema desta tese. Revimos a forma como os indivíduos são codificados e armazenados na população, bem como as estratégias para promover e preservar a diversidade genética nas populações dos algoritmos evolutivos.

A abordagem seguida neste trabalho é significativamente diferente das que são encontradas na literatura, substituindo o modelo tradicional de populações por populações baseadas em multiconjuntos (multisets). Este novo modelo necessita da adaptação dos operadores genéticos para a evolução dos indivíduos; no entanto, é insensível em relação ao algoritmo que os evolui.

Nos capítulos seguintes faremos a definição formal da multipopulação e adaptaremos o processo evolutivo para a utilizar. Implementaremos a substituição das populações simples por multipopulações em alguns algoritmos evolutivos, e estudaremos o seu comportamento.

3 Populações baseadas em multisets

Os *multisets*, ou em português multiconjuntos, são uma extensão dos conjuntos tradicionais que possuem características que os tornam eficientes para um grande número de aplicações. Em matemática podem ser encontrados em conjuntos de lógica difusa, em análise combinatória ou de permutações. Em computação podemos encontrar algoritmos para cálculo de potências, para pesquisar e ordenar dados, em autómatos ou ainda na prova do término de determinados programas, para citar alguns exemplos. Neste âmbito (Singh et al. 2007) apresenta um excelente resumo da sua aplicação na matemática e ciências computacionais. Neste trabalho vamos explorar a sua aplicação na representação de populações para algoritmos evolutivos.

3.1 Multisets

Um multiset é uma coleção de elementos onde são permitidas repetições e os elementos repetidos são considerados iguais entre si (Blizard 1991), isto é, não importa a ordem, mas apenas o seu valor e a sua quantidade. Existem estruturas de dados, como por exemplo as coleções, que suportam conjuntos de elementos iguais, no entanto, esses elementos repetidos são tratados como distintos entre si, pois ocupam posições distintas dentro da estrutura e não estão relacionados. Esta falta de estrutura para o tratamento de elementos repetidos leva ao desperdício de espaço e de tempo para o seu processamento.

A coleção de elementos (figura 3-1-a) pode ser representada pelo conjunto de elementos distintos (figura 3-1-b), chamado *conjunto de suporte*, e pela multiplicidade associada a cada elemento, *conjunto de multiplicidade* (figura 3-1-c). Podemos definir um multiset (figura 3-1-d) como um conjunto de pares <*elemento, cópias*>, onde *cópias* representa a multiplicidade associada ao objeto *elemento*.

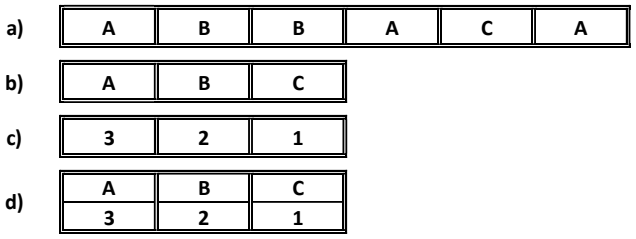


figura 3-1 - Definição de multiset: a) coleção com seis elementos; b) conjunto de suporte da coleção; c) conjunto de multiplicidade da coleção; d) multiset.

Computacionalmente é possível que dois elementos distintos, mas equivalentes, sejam considerados iguais e coexistam na mesma coleção. O armazenamento dos elementos iguais num conjunto pode ser feito através da repetição do elemento (figura 3-1-a) ou como dois conjuntos separados (figura 3-1-b e c). A segunda forma é computacionalmente mais eficiente e a sua implementação pode ser encontrada em várias bibliotecas de classes. Um levantamento das bibliotecas que fornecem a estrutura de dados multiset e as suas características pode ser consultado em (Manso e Correia, 2011a).

Outra forma de representar o multiset é através da introdução de um novo atributo nos elementos que permita guardar o número de cópias existentes (figura 3-1-d). Esta representação por pares de <*elemento,cópias*> foi explorada por (Manso e Correia 2011a), que propuseram a estrutura de dados MiTree (*Multiset indexed Tree*), baseada em árvores binárias balanceadas como suporte de dados do multiset e a sua utilização na representação de populações para algoritmos evolutivos.

De forma a poder utilizar os multisets para o processamento de indivíduos nas populações de algoritmos evolutivos é necessário definir um conjunto de operações que têm de ser executadas de forma eficiente:

- **Pesquisa:** Esta é a uma operação crítica da estrutura de dados pois as restantes operações dependem dela. A procura de elementos no multiset é sempre feita no conjunto de suporte e a sua representação compacta facilita a pesquisa sequencial. No entanto, por ser uma operação realizada frequentemente, este tipo de pesquisa torna a estrutura lenta. A MiTree utiliza pesquisa em árvores binárias balanceadas, com tempos de acesso logarítmicos; porém, as tabelas de dispersão são uma boa alternativa, uma vez que têm tempos de acesso constante.
- **Introdução:** Na introdução de um elemento num multiset é necessário pesquisar se o elemento já existe na estrutura de dados. Se existir, incrementa-se o seu número de cópias no conjunto do elemento; caso contrário, introduz-se um novo elemento no conjunto de suporte e o valor unitário no conjunto de multiplicidade.
- **Eliminação:** Na eliminação de um elemento do multiset é necessário pesquisar quantas cópias do elemento existem no conjunto. Se este número

for superior a um, a eliminação é feita através da diminuição de uma unidade do número de cópias. Se o número de cópias for igual a um, é feita a eliminação do elemento no conjunto de suporte e a respetiva entrada no conjunto da multiplicidade.

- **Indexação:** A indexação do multiset é uma operação que ajuda no desenho de operadores genéticos, uma vez que estes utilizam frequentemente números aleatórios gerados por uma distribuição de probabilidade. Nas populações tradicionais baseadas em conjuntos, o acesso aleatório a um elemento é feito através do sorteio de um índice, pois todos os elementos são indexados. O multiset (figura 3-2-a) deve fornecer a mesma funcionalidade e definir os índices de cada um dos elementos (figura 3-2-c), como se de uma coleção de elementos simples se tratasse. Nesta representação os elementos duplicados ocupam posições indexadas adjacentes (figura 3-2-b):

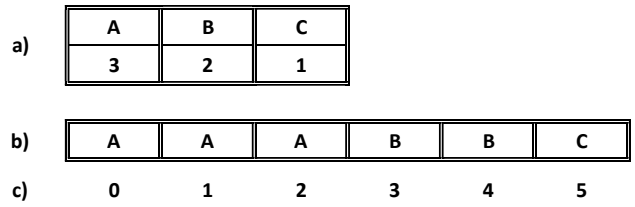


figura 3-2 - Indexação do Multiset: a) multiset; b) coleção equivalente ao multiset; c) índices dos elementos do multiset

3.1.1 Multisets como suporte de populações

Com a definição das operações anteriores, os multisets podem ser utilizados para a representação de populações para computação evolutiva. Esta representação contém características que a tornam uma boa alternativa aos conjuntos que tradicionalmente servem de suporte às populações:

- O conjunto de suporte contém os elementos distintos da população e pode ser usado para controlar a diversidade genética da população através da sua dimensão. Quanto maior a sua dimensão mais genótipos diferentes a população vai conter e, consequentemente, maior diversidade genética;
- A multiplicidade de cada elemento do conjunto de suporte pode ser usada para controlar os clones que cada indivíduo tem na população e, por conseguinte, a própria diversidade genética da população global;

- A introdução e eliminação de elementos repetidos não alteram a estrutura de suporte da população, mas apenas a sua multiplicidade, o que torna a estrutura de dados computacionalmente eficiente para introdução e eliminação de elementos repetidos;
- A representação compacta do multiset necessita de menor esforço computacional para a avaliação da população evitando avaliações de clones.

As características acima apresentadas podem ser exploradas para incrementar a eficiência dos algoritmos evolutivos e evitar um dos seus maiores problemas, a convergência prematura por falta de material genético na população.

A ideia original de utilização de multisets para a representação de populações foi originalmente proposta em 1999 (Aparício, Correia, e Moura-Pires 1999), e a sua concretização foi apresentada dez anos mais tarde pelo autor desta tese (Manso e Correia, 2009), recorrendo a coleções lineares de indivíduos aos quais foi adicionado o atributo *cópias*. Esta estrutura de dados cumpriu o objetivo de demonstrar o conceito e antever novas linhas de investigação para a utilização dos multisets; no entanto, a estrutura era lenta pelo facto de as pesquisas terem complexidade linear. A aceleração das pesquisas através da ordenação da estrutura transferiu a complexidade para a introdução e eliminação de elementos no conjunto de suporte, o que não resolveu o problema.

Com vista a colmatar as dificuldades identificadas, foi desenvolvida uma estrutura de dados baseada em árvores binárias balanceadas, MiTree, em que as operações são realizadas com uma ordem de complexidade logarítmica, $O(\log_n)$. A estrutura foi especialmente desenvolvida para o acesso indexado aos elementos (figura 3-3) de forma a utilizar os operadores genéticos tradicionais. A estrutura permitiu acelerar o processamento de indivíduos na população e a otimização de problemas com dimensões consideráveis em tempo aceitável. Apesar de a estrutura ter indexação com complexidade logarítmica, é competitiva com os conjuntos tradicionais devido ao facto de o conjunto de suporte ter menor dimensão e as introduções e eliminações de elementos serem mais eficientes.

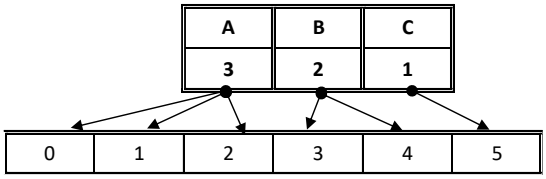


figura 3-3 - Indexação do Multiset

Na busca de uma estrutura mais eficiente, desenvolvemos uma estrutura híbrida em que o multiset foi separado da indexação. A solução escolhida utiliza uma representação de multisets baseados em tabelas de dispersão, que permitem introduzir, eliminar e pesquisar em tempo constante, O(1). Se conhecermos a dimensão máxima que a tabela pode conter, esta pode ser parametrizada para evitar que a estrutura cresça de forma dinâmica, aumentando a sua eficiência, e este atributo é tipicamente fornecido como parâmetro dos algoritmos evolutivos. A indexação dos elementos passou a ser feita através da transformação do multiset num conjunto tradicional indexado (figura 3-2-b) em que o acesso tem complexidade constante, O(1). Com a transformação do multiset em conjunto indexado, os operadores genéticos tradicionais podem ser utilizados de forma automática.

Através desta abordagem híbrida passamos a ter complexidade ótima nas operações de manipulação de indivíduos na população (pesquisa, inserção e eliminação) e, sacrificando um pouco o espaço, passamos a ter suporte indexado aos indivíduos. O espaço extra necessário para a indexação pode ser muito reduzido porque não é necessário ter cópias dos elementos, mas apenas as referências que estão na tabela de dispersão, como se pode ver na figura 3-3.

3.2 Multipopulações para computação evolutiva

Uma multipopulação (MP) é um multiset de indivíduos que evoluem através de um algoritmo evolutivo. A cada indivíduo da multipopulação está associado um número de cópias e ao par <*indivíduo*, *cópias*> chamaremos multi-indivíduo (MI). Assim sendo, podemos definir uma multipopulação como um conjunto multi-indivíduos.

indivíduo	aptidão
11111110	7
11111110	7
11111110	7
11110000	4
11110000	4
10001001	3
10001001	3
10000000	1

a)

multi-Indivíduo	aptidão
< 3, 11111110 >	7
< 2, 11110000 >	4
< 2, 10001000 >	3
< 1, 10000000 >	1

b)

figura 3-4 - Coleção de oito indivíduos do problema OneMax: a) População simples de oito indivíduos; b) Multipopulação com quatro multi-indivíduos que representam os oito indivíduos da população simples.

A figura 3-4 apresenta a equivalência entre uma população simples, baseada em conjuntos, e uma multipopulação baseada em multisets. Desta equivalência salientamos as seguintes características:

- Os oito indivíduos da população simples foram agrupados em três multi-indivíduos na multipopulação, o que permite guardar a mesma informação num espaço mais reduzido;
- Os multi-indivíduos possuem um atributo extra que lhes permite guardar o número de cópias. Este atributo tem um peso reduzido no espaço que o multi-indivíduo ocupa quando comparado com os restantes atributos que partilha com os indivíduos simples;
- O custo computacional para processar a multipopulação é menor do que o da população simples. Os algoritmos de pesquisa, ordenação, inserção e outros, demoram menos tempo porque a estrutura de dados possui menos elementos;
- A avaliação da multipopulação é computacionalmente mais eficiente, porque o número de genótipos a avaliar é menor. A avaliação de um multi-indivíduos na multipopulação corresponde à avaliação de todos os clones na população simples. A população simples necessitaria de oito chamadas à função de avaliação enquanto que a multipopulação necessitaria apenas de três;
- Os clones dos indivíduos são guardados nos números de cópias dos multi-indivíduos. Desta forma ficamos com a informação imediata de quantos clones tem cada genótipo. Esta informação é muito útil para controlar os clones na população e vai ser explorada para melhorar o processo evolutivo. A mesma informação poderia ser obtida a partir da população simples, mas o custo computacional seria mais elevado e o controlo de clones mais complexo.

Tradicionalmente a dimensão da população nos algoritmos evolutivos permanece inalterada ao longo do processo evolutivo, observando a regra de que os melhores indivíduos das novas gerações substituem os mais antigos através do princípio da sobrevivência dos mais aptos. Nas populações tradicionais cada indivíduo ocupa uma posição específica e a existência de clones não altera a dimensão da população.

Por definição, nesta tese, a dimensão da multipopulação, ou o número de multi-indivíduos, permanece constante ao longo do processo evolutivo, independentemente de quantos indivíduos simples represente. Esta premissa permite definir, à partida, a

quantidade de genótipos diferentes que a população vai conter e, por consequência, a aumentar a sua diversidade genética:

3.3 Algoritmos evolutivos com populações baseadas em multisets

As multipopulações são alheias em relação ao processo evolutivo que é aplicado sobre os indivíduos da população. Elas podem ser utilizadas por qualquer algoritmo evolutivo que use populações tradicionais, desde que se cumpram os seguintes requisitos:

1. O número de cópias incremente a probabilidade de seleção do genótipo do multi-indivíduo;
2. O operador de substituição insira os clones do genoma no número de cópias dos multi-indivíduos;
3. O número de cópias da população seja controlado.

O primeiro ponto garante a utilização do número de cópias do genoma pelo operador de seleção, pois o número de cópias de um multi-indivíduo numa multipopulação só tem um único propósito: incrementar a probabilidade de seleção do genoma. Se este valor for ignorado pelo operador de seleção, a população deixa de ser uma multipopulação para passar a ser um conjunto de indivíduos diferentes entre si.

O segundo ponto assegura que a introdução de clones na população pelo operador de substituição incrementa o número de cópias do multi-indivíduo e, consequentemente, a sua probabilidade de se reproduzir. Além disso, garante que o conjunto de suporte da população permanece constante e com indivíduos únicos.

E, finalmente, o terceiro ponto garante que o número de indivíduos de uma multipopulação é controlado de forma a limitar a pressão seletiva sobre os genomas. Se o número de cópias não for controlado, um conjunto limitado de genomas podem assumir o controlo da população através da quantidade de indivíduos que representam.

Se os três princípios acima enunciados forem respeitados, as populações tradicionais dos algoritmos evolutivos podem ser substituídas por multipopulações. A utilização das multipopulações traz o benefício de conter sempre um conjunto de genoma diferentes e, por conseguinte, o operador de recombinação consegue gerar indivíduos diferentes dos progenitores. A utilização do número de cópias para acomodar os clones tem o benefício de não desperdiçar bons indivíduos gerados pelos operadores de reprodução. Por fim, a limitação do número de clones de um genoma permite evitar que alguns multi-indivíduos bem-adaptados dominem a população.

3.3.1 Geração da multipopulação inicial

O algoritmo para a geração da população inicial deve garantir que são criados os multi-indivíduos necessários para que a população tenha a dimensão pretendida. Nas populações comuns isto não levanta nenhum obstáculo, uma vez que a introdução de indivíduos repetidos aumenta o tamanho da população, o que não acontece nas multipopulações. A dimensão do tamanho da população não deve exceder o número de elementos do espaço de procura e, se tal acontecer, o valor da dimensão deve ser reajustado.

3.3.2 Operador de seleção para reprodução

O operador de seleção para reprodução seleciona os indivíduos que se vão reproduzir para constituir uma nova geração. Este operador deve tendencialmente selecionar os melhores indivíduos de forma a que os seus genes sejam espalhados para as novas gerações. Por outro lado, também deve fornecer uma população com diversidade genética suficiente para que os progenitores possam trocar genes entre si e consequentemente gerar indivíduos diferentes de si próprios.

O número de cópias dos multi-indivíduos tem uma implicação direta sobre o operador de seleção para reprodução, dado que o operador será utilizado sobre a versão indexada da multipopulação (figura 3-3), equivalente à figura 3-4 a). O número de cópias de um multi-indivíduo na população principal aumenta a probabilidade de seleção do genótipo, ao introduzir tantos clones quanto o número de cópias na versão estendida da multipopulação. Qualquer algoritmo de seleção pode ser utilizado neste operador se o multiset for transformado em conjunto indexado.

O algoritmo *MTournamentSelection* (algoritmo 3-1) constitui-se como um procedimento que seleciona um conjunto de indivíduos para reprodução em populações baseadas em multisets através de torneios. Este tem três parâmetros: *parents,* que representa a população dos progenitores; *tourSize*, que representa a dimensão do torneio usado para a seleção; e *parentsProportion* que representa a proporção dos indivíduos que serão selecionados.

O algoritmo começa por calcular o número de indivíduos que serão selecionados através do parâmetro *parentsPorportion* e do tamanho da população. O tamanho das populações simples corresponde, naturalmente, ao tamanho da população, mas nas multipopulações, corresponde à dimensão do conjunto de suporte. Este operador

seleciona o mesmo número de indivíduos nos dois tipos de populações, apesar de a multipopulação representar, eventualmente, mais indivíduos.

```
MTournamentSelection(parents , tourSize, parentsProportion)
  popSize = parents.size * parentsProportion
  listIndividuals = parents.getIndividualsList()
  para k=0 até popSize
      best = selecionar um individuo aleatório de listIndividuals
      para t=1 até tourSize
         ind = selecionar um individuo aleatório de listIndividuals
         se best for pior que ind então
             best = ind
      próximo
      adicionar uma cópia de best a selectedPop
  próximo
  retornar selectedPop
fim
```

algoritmo 3-1 – MTournamentSelection – Operador de seleção de indivíduos para reprodução em populações baseadas em multisets

O algoritmo continua com a obtenção da lista de indivíduos que participaram na seleção. Assim, no caso das populações simples, corresponde aos elementos da população; no caso das multipopulações, corresponde ao conjunto dos seus multi-indivíduos depois de transformados em indivíduos simples, figura 3-4 a).

De seguida são realizados torneios com dimensão *tourSize*, em que os indivíduos são selecionados de forma aleatória na lista de indivíduos, e o indivíduo com melhor aptidão é copiado para a população dos selecionados. O parâmetro *tourSize* regula a pressão seletiva sobre os progenitores e que se manifesta na diversidade genética da população selecionada. Um valor unitário significa que os indivíduos são selecionados de forma aleatória, ou seja, sem competição. Se a população for simples, significa que não existe pressão seletiva, mas no caso da multipopulação existe a pressão seletiva introduzida pelo número de cópias. O número de cópias de um multi-indivíduo produz o mesmo número de clones na lista de indivíduos, o que aumenta de forma linear a probabilidade de ser selecionado para participar num torneio. De facto, quanto maior for o valor deste parâmetro, menor será a diversidade genética da população selecionada e a consequente aceleração do algoritmo para genótipos próximos dos selecionados. E se, por um lado, esta falta de diversidade é boa na fase final do processo evolutivo para

otimizar os picos descobertos, por outro lado, representa um grave problema em todas as restantes situações.

3.3.3 Operador de substituição de populações

Este operador tem como objetivo a introdução dos indivíduos gerados pelos operadores genéticos na população principal dos algoritmos evolutivos. Por norma, o número de indivíduos numa população permanece constante ao longo do processo evolutivo e, se mantivéssemos esta premissa com a utilização dos multisets, tal iria fazer com que o conjunto de suporte diminuísse com a introdução de clones. No entanto, esta situação mudaria apenas a forma como os indivíduos são armazenados na população, e as vantagens que a nova representação traria seriam residuais. Por outro lado, se mantivermos a dimensão do conjunto de suporte constante, deixando variar o número de indivíduos, a população terá sempre um conjunto de genótipos distintos, o que incrementará a diversidade genética da população. Com base nestas observações, decidimos que as multipopulações teriam um número de multi-indivíduos constante ao longo do processo evolutivo, e esta premissa é garantida por este operador.

Os algoritmos de substituição de multipopulações devem contemplar os procedimentos necessários para garantir que a população que continua o processo evolutivo contém exatamente o mesmo número de multi-indivíduos da população original. Assim o algoritmo *MTournamentReplacement*, algoritmo 3-2, apresenta um algoritmo de substituição de populações, baseado em torneios efetuados sobre a população dos pais (*parents*) e dos filhos (*offspring*). Este algoritmo seleciona os dois indivíduos, um pai e um filho, das listas que representam as populações que vão competir por um lugar na nova geração através de torneios. O pai é selecionado por um torneio de dimensão *tourPar* de indivíduos selecionados de forma aleatória na lista dos pais. Por seu turno, o filho é selecionado de forma idêntica, através de um torneio de dimensão *tourOff* na lista dos filhos. Deste modo, o melhor dos dois indivíduos selecionados sai da lista correspondente e ganha um lugar na nova geração.

O parâmetro *tourOff* representa a pressão seletiva sobre os descendentes e tem de ser estritamente positivo, ou seja, maior que um, para garantir que os elementos da nova geração tenham a oportunidade de sobrevivência. Já o parâmetro *tourPar* representa a pressão seletiva sobre a população principal e é um número positivo. O valor zero de *tourPar* significa que apenas os filhos entram nos torneios.

Estes torneios, sem reposição, são realizados até que a população tenha a dimensão da população original (*parents*), ou já não existam descendentes em número suficiente para realizar novos torneios. O algoritmo apresentado tem preferência pelos descendentes quando o valor da aptidão for igual ao do progenitor selecionado, premiando a introdução de novos indivíduos em detrimento dos mais antigos.

```
MTournamentReplacement (parents, tourPar, offspring, tourOff)
  inicializar newPop
  listParents = parents.getIndividualsList()
  listOffspring = offspring.getIndividualsList()
  enquanto newPop.#genotypes < parents.#genotypes e
           listOffspring.size >= tourSize
       parentT = selecionar tourPar indivíduos de listParents
       childrenT = selecionar tourOff indivíduos de listOffspring
       se bestOf(parentT) for melhor que bestOf(childrenT) então
            adicionar bestOf(parentT) a newPop
            remover bestOf(parentT) de listParents
       senão
            adicionar bestOf(childrenT) a newPop
            remover bestOf(childrenT) de listOffspring
       fim se
  fim enquanto

  enquanto newPop.genotypes < parents.genotypes
     randomP = remover um indivíduo de listParents
     adicionar randomP a newPop
  fim enquanto

  retornar newPop
fim
```

algoritmo 3-2 - MTournamentReplacement - Operador de substituição de populações baseadas em multisets

O algoritmo termina com a introdução de indivíduos, selecionados de forma aleatória na população principal, para garantir que a população que continua o processo evolutivo tem a dimensão da população original. Esta situação pode acontecer quando a diversidade genética da população dos descendentes for baixa, isto é, composta por muitos clones e bem-adaptados. Como o algoritmo tem preferência pelos descendentes, estes são armazenados na população selecionada em poucos multi-indivíduos. O

mesmo também pode acontecer quando o número de indivíduos na população dos descendentes for menor que o conjunto de suporte da população e estes forem todos escolhidos para a nova geração.

3.3.4 Operador de redimensionamento

A existência de clones na população acelera o processo de convergência e, tradicionalmente, as populações finais dos algoritmos evolutivos são compostas por vários clones que partilham um pequeno conjunto de genomas. Como nas multipopulações a dimensão do conjunto de suporte se mantém constante e os clones são guardados no número de cópias, este tem tendência a crescer, e se não for controlado, tem um efeito negativo no processo evolutivo da população. Se o número de cópias dos multi-indivíduos não for limitado, os indivíduos mais aptos podem ganhar um número tão elevado de cópias que o operador de seleção os escolhe quase sempre e, consequentemente, a diversidade genética da população selecionada é tão pobre que o algoritmo genético tem dificuldade em evoluir para novas soluções. Sem o controlo do número de cópias, as multipopulações sofrem do problema de convergência prematura, uma vez que a uma grande quantidade de cópias dos melhores indivíduos minimiza o efeito da diversidade genética introduzida pelo conjunto de suporte.

De forma a mitigar o problema acima referido, introduzimos um novo operador no processo evolutivo a que chamaremos redimensionamento. Este operador trabalha sobre o número de cópias dos indivíduos no sentido de as reduzir, evitando, desta forma, que se propaguem de forma excessiva nas gerações seguintes. O desenho deste operador orientou-se por três princípios:

1. Limitar o número total de indivíduos na multipopulação para impedir que um grupo de indivíduos domine a população;
2. Assegurar que os melhores indivíduos continuem com as múltiplas cópias geradas pelo processo evolutivo e, consequentemente, possam ter uma probabilidade de seleção para reprodução acrescida;
3. Não criar clones artificiais através do incremento do número de cópias, ou seja, o operador apenas pode reduzir as cópias, sem nunca as incrementar.

A primeira abordagem para a limitação do número de cópias dos multi-indivíduos foi feita pelo operador de redimensionamento fixo *FixedCeiling*, algoritmo 3-3, por nós apresentado em (Manso e Correia 2009). O operador tem como parâmetros a população

a redimensionar (*pop*) e o respetivo fator de redimensionamento (*factor*), dividindo o número de cópias de cada indivíduo por esse fator, e arredondando para cima o número de cópias com a função *ceil*. O parâmetro *factor* é um número real maior que um e a utilização da função *ceil* permite que quando os indivíduos ficam com mais do que uma cópia, estes a mantenham, devido ao arredondamento para o próximo inteiro, função *ceil*. Em cada geração o número de cópias dos indivíduos é atualizado, sendo que só conseguem manter as múltiplas cópias se o processo evolutivo conseguir gerar clones seus.

O operador *FixedCeiling* cumpre a missão de controlo do número de cópias; no entanto, apresenta algumas características que podem ser problemáticas. Assim, quando a função de avaliação apresenta vários extremos locais, é natural que os indivíduos que se encontrem nesses extremos ganhem várias cópias, por isso, as funções rugosas possuem tendencialmente mais indivíduos. Por consequência, o número de indivíduos na multipopulação é dependente da rugosidade da função de avaliação, o que dificulta a definição do parâmetro. Outra limitação do operador é a dificuldade de estimar o número de indivíduos em relação ao valor do parâmetro ao longo do processo evolutivo. Deste modo, o algoritmo começa a limitar as cópias logo no início da evolução, mesmo quando não existem indivíduos com um número exagerado de cópias:

```
FixedCeiling(pop,factor)
  para i = 0 até pop.genotypes
      pop[i].copies = ceil(pop[i].copies/factor)
  próximo
fim
```

algoritmo 3-3 - Algoritmo de redimensionamento Fixo – FixedCeiling

Resultados experimentais mostraram que o valor do parâmetro igual a 1.5 é um bom compromisso entre o número de indivíduos e a diversidade genética da população na otimização de função representadas por vetores de bits (Manso e Correia 2009):

Com o intuito de resolver as dificuldades identificadas, desenvolvemos uma versão adaptativa do operador de redimensionamento, *AdaptiveCeiling*, onde o valor do fator de redimensionamento foi substituído pela proporção máxima de indivíduos desejados na população relativamente ao número de multi-indivíduos (Manso e Correia 2011b). Neste operador o fator de redimensionamento deixou de ser fixo e passou a ser

calculado de acordo com o número de indivíduos que se espera que a população tenha (algoritmo 3-4).

```
AdaptiveCeiling (pop, maxProportion)
  factor = pop.genotypes*maxProportion / pop.individuals
  se factor > 1 então
    para i = 0 ate pop.genotypes
        pop[i].copies = ceil(pop[i].copies * factor)
    próximo
  fim se
fim
```

algoritmo 3-4 -Algoritmo do redimensionamento adaptativo – AdaptiveCeiling

O Algoritmo possui o parâmetro *maxProportion* que representa a proporção máxima desejada entre os genótipos da população (*pop.genotypes*) e número total de indivíduos (*pop.individuals*). O algoritmo calcula o fator a ser aplicado ao número de cópias (*factor*) e atualiza o número de cópias de cada indivíduo, utilizando a função *ceil*. O algoritmo evita a introdução de cópias artificiais através do teste do valor do fator. A função *ceil* garante ainda que cada individuo tenha pelo menos uma cópia e, como faz arredondamentos para cima, o parâmetro *maxProportion* garante que, sempre que um genoma ganha mais do que uma cópia, este a mantém até ser substituído.

3.4 Métricas para avaliação do processo evolutivo

De forma a poder avaliar a influência das multipopulações no processo evolutivo foram definidas um conjunto de métricas para nos ajudar nesta tarefa. Nas secções seguintes são definidas as métricas que foram utilizadas no processo de investigação e o nome da implementação computacional (em negrito dentro de parêntesis), que serve de referência nas tabelas de resultados dos capítulos seguintes e na parametrização das experiências com o MuGA-T (Anexo F), um software desenvolvido para explorar algoritmos evolutivos com vários tipos de populações e, especialmente, de multipopulações.

3.4.1 Critérios de paragem

A finitude é uma das propriedades que todos os bons algoritmos possuem, pois esta propriedade garante que os algoritmos terminam sempre e devolvem o valor calculado. Ora, a evolução é um processo contínuo e os algoritmos evolutivos necessitam de

definir explicitamente o término da evolução. Habitualmente o critério de paragem é dado ou pela estrutura da população ou pela capacidade computacional disponível.

O valor da aptidão do melhor indivíduo da população é uma das características que usualmente é utilizada; o valor pode ser o ótimo (*OptimumFound*), caso seja conhecido, ou então o valor de uma solução aceitável para o decisor (*ValueFound*). Outro critério de paragem pode ser a estagnação da população, quando o algoritmo não consegue fazer evoluir as melhores soluções durante um determinado número de gerações.

No esforço computacional como critério de paragem, definimos o tempo computacional disponível (*MaxTime*), o número máximo de gerações (*MaxGeneration***)** e ainda o número máximo de chamadas à função de avaliação (*MaxEvaluations***)**. Quando o critério de paragem é atingido, o algoritmo termina e devolve a população final.

3.4.2 Eficiência e eficácia

A eficiência de um algoritmo mede o esforço computacional que é necessário despender para o executar e depende do tamanho dos dados que processa (*n*), o que, no caso dos algoritmos evolutivos, corresponde ao tamanho da população. O tempo e o espaço necessário para a sua execução são as medidas mais usuais, e a otimização de uma normalmente condiciona a outra; por isso, os bons algoritmos são um compromisso entre o tempo e o espaço que necessitam para a sua execução. A complexidade dos algoritmos é por norma analisada através da métrica "*big-O*", O(n), que faz uma análise assintótica e atribui aos algoritmos uma classe: constante, logarítmica, polinomial, exponencial, entre outras. A notação O(n) é uma métrica para a análise do pior dos casos na execução dos algoritmos e é utilizado em algoritmos determinísticos, não sendo apropriada para algoritmos aleatórios. Medir o comportamento dos algoritmos no pior dos casos dá-nos a informação da sua eficácia, ou seja, a capacidade de o algoritmo resolver qualquer instância do problema.

Os algoritmos evolutivos utilizam a aleatoriedade de forma intensiva, pelo que a análise de complexidade teórica tem uma aplicabilidade limitada. Para a avaliação do desempenho destes algoritmos utilizam-se métricas retiradas da sua execução computacional.

A eficiência de um algoritmo evolutivo mede o esforço computacional que é necessário despender para obter o resultado pretendido. O tempo de processamento (*Time*) mede a eficiência do algoritmo; no entanto, esta métrica depende da

implementação do algoritmo, sequencial ou paralelo, do hardware computador onde é executado e da disponibilidade do sistema operativo para executar a evolução. O valor produzido por esta métrica está, portanto, dependente do sistema computacional onde a evolução é executada, e a sua comparabilidade é muito limitada.

O número de instruções executadas pelo algoritmo é uma métrica mais genérica, fiável e comparável, pelo que a contagem do número de instruções depende da implementação, da linguagem de programação e das bibliotecas utilizadas. Por forma a contornar estas limitações, habitualmente utiliza-se o número de chamadas à função de avaliação para atingir um determinado valor (*FuncsCallToValue*) e, no caso de ser conhecido, o valor ótimo (*FuncsCallToOptimum*). Estas métricas medem apenas o número de vezes que os descendentes são avaliados, não importando o esforço computacional que foi despendido para os gerar.

A eficácia mede a robustez do algoritmo, ou seja, a competência que o algoritmo tem para resolver as instâncias do problema que lhes são apresentadas. Esta capacidade de resolução de problemas é medida pela taxa de sucesso, calculada pelo rácio entre as simulações que tiveram sucesso e o número total de simulações. Finalmente, o sucesso de uma simulação pode ser medido pelo valor atingido pelo melhor indivíduo (*SuccessToValue*), que pode ser o ótimo (*SuccessRate*).

3.4.3 Estrutura da população

As métricas sobre a estrutura da população são muito importantes para compreender o sucesso ou o fracasso do processo evolutivo. A característica mais diferenciadora das multipopulações, em relação às populações tradicionais, é a possibilidade de acomodar os clones no número de cópias dos multi-indivíduos e, por isso, definimos métricas para compreender a sua influência no processo evolutivo.

O número de cópias numa multipopulação influencia a pressão seletiva sobre o respetivo genoma e é importante calcular o seu valor máximo na população (*MaxCopies*). Assim, um valor muito alto prejudica a diversidade genética da população, e um valor mais baixo prejudica a probabilidade de seleção para reprodução.

O número de indivíduos (*Individuals*) é uma métrica muito importante para as multipopulações, pois este número é variável e depende do número de cópias dos multi-indivíduos. Outra métrica importante é a capacidade encontrar os indivíduos ótimos (*NumOptimumsFound*) ao longo do processo evolutivo e a sua capacidade de os reter na população (*NumOptimumsPop*). De igual importância é a quantidade de máximos

diferentes (*NumMaximaPop***)** ou de ótimos diferentes (*NumOptimalGenomesPop*) que a população contém.

3.4.4 Estrutura dos indivíduos

Os operadores de reprodução incidem sobre os genes dos progenitores, e a diversidade genética é a pedra de torque para o sucesso dos algoritmos evolutivos. Sem diversidade genética, os algoritmos dificilmente conseguem gerar indivíduos diferentes dos seus pais e, por conseguinte, a população fica estagnada em extremos locais. Uma das características diferenciadoras das multipopulações em relação às populações tradicionais é a manutenção de um conjunto de genomas distintos, os multi-indivíduos, que aumentam a diversidade genética da população. Deste modo, a definição de métricas para a sua quantificação foi uma necessidade.

Diversidade genética em problemas codificados por genes binários

A equação 3.1 apresenta a métrica *genetic binary diversity,* que é aplicada a problemas codificados por bits e a populações representadas por multisets e calcula a diversidade ao nível dos genes (Manso e Correia 2009). Esta métrica consiste numa forma normalizada da distância de *Hamming* para genes binários e pode ser calculada de forma eficiente pela equação abaixo apresentada (3.1), onde *l* representa o número de bits que compõem o genótipo dos indivíduos, k_i representa o número de vezes que o alelo é 1, e, finalmente, *n* representa o tamanho da população:

$$genetic\ binary\ diversity\ = \frac{4}{n^2 l} \sum_{i=1}^{l} k_i\ (n - k_i) \qquad (3.1)$$

A população exibe uma diversidade máxima (1) quando o número de zeros e uns para cada alelo do gene é igual; apresenta uma diversidade mínima (zero) quando cada um dos alelos do gene possui o mesmo valor, o que equivale a dizer que os indivíduos são clones uns dos outros. As multipopulações nunca atingem a diversidade mínima, porque os clones são agrupados no número de cópias dos multi-indivíduos, e o conjunto de suporte da multipopulação é constituído por genótipos diferentes.

Diversidade genética genérica

A equação (3.1) utiliza a proporção entre zeros e uns dos alelos da população para calcular a diversidade genética; porém, os genes dos indivíduos podem ser representados por permutações de elementos ou por valores numéricos. Existem várias

formas de calcular a diversidade genética para as populações de indivíduos com cada uma das codificações; no entanto, não existe uma que sirva simultaneamente para as três: binários, permutações ou valores numéricos. No sentido de promover a comparação da diversidade genética entre populações codificadas por genes de tipos distintos, desenvolvemos uma medida de diversidade genética baseada na entropia de *Shannon*. A equação (3.2) apresenta a fórmula de cálculo da entropia para um conjunto de valores, *v*, cujas probabilidades de ocorrerem são p_i. Se a base do logaritmo, *k*, for a cardinalidade do conjunto de valores que *v* pode tomar, a fórmula fica normalizada no intervalo [0,1]. O seu valor máximo ocorre quando todas as probabilidades forem iguais:

$$H(v) = \sum_{i=1}^{n} -p_i \, log_k(p_i) \tag{3.2}$$

Para calcular a diversidade genética de populações com indivíduos codificados por permutações ou por valores reais, necessitamos de definir a forma de cálculo das probabilidades, p_i, da equação (3.2). Os alelos codificados por permutações têm um conjunto de valores possíveis igual ao tamanho do cromossoma e, deste modo, a equação (3.2) pode ser utilizada calculando a probabilidade de cada valor do alelo dentro da população. Este procedimento não se pode aplicar aos valores codificados por valores numéricos devido à infinidade de valores que os genes podem tomar. Se modificarmos o cálculo das probabilidades, p_i, de valores singulares para intervalos, a fórmula (3.2) pode ser utilizada nas três representações de genes. A figura 3-5 mostra os intervalos definidos por um conjunto de valores, v_i, e a equação (3.3) mostra a fórmula de cálculo da probabilidade, p_i, dos intervalos definidos pelos valores:

$$p_i = \frac{v_{i+1} - v_i}{max - min} \tag{3.3}$$

A probabilidade é definida pela proporção entre dois valores consecutivos e a dimensão do domínio do gene, valores *min* e *max* da figura 3-5.

Na figura 3-5 os valores v_i são os valores dos genes na população em caso de permutações ou valores numéricos. Os genes binários são um caso especial de valores numéricos e o valor *p* é dado pela proporção entre zeros e uns. A base do logaritmo, *k*, na equação (3.2) é 2 para binários, e para as permutações ou para valores numéricos vale o tamanho do gene. A correta definição da base do logaritmo normaliza a métrica no intervalo [0,1].

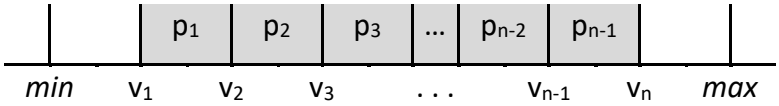


figura 3-5- Intervalos definidos por um conjunto de valores (v) e as respetivas probabilidades (p).

A diversidade máxima ocorre quando se verificarem três situações simultaneamente:

1. O número de indivíduos é maior ou igual ao número de genes do cromossoma, pois permite que cada um deles tenha um valor distinto nos alelos;
2. Os valores atingem o mínimo e o máximo, ou seja, ocupam todo o domínio;
3. Os valores estão igualmente espaçados no domínio, implicando igualdade de probabilidade de ocorrência.

Se as condições anteriores ocorrerem a probabilidade de cada alelo na população é igual e o valor da diversidade genética é 1. O valor mínimo é dado quando um alelo possui apenas um valor e a diversidade genética vale zero.

A diversidade genética de uma população é dada pela fórmula (3.4) e calcula a média da diversidade de cada alelo que o cromossoma contém:

$$Genetic\ Diversity(P) = \frac{\sum_{i=1}^{n} H(P_i)}{K} \qquad (3.4)$$

Nesta equação *H* representa a entropia (3.2) do vetor *Pi,* que contém os valores do alelo *i* de um cromossoma com *n* alelos, na população *P*. O valor *k* representa o número de indivíduos que a população *P* contém. No caso das populações simples, o valor de *k* é o número de indivíduos, já no caso das multipopulações é a soma de todas as cópias dos multi-indivíduos.

A diversidade genética (*GeneticDiversity***)** de uma população avalia a capacidade de evolução dos seus indivíduos através dos operadores genéticos. Uma diversidade próxima de 1 é, geralmente, uma população aleatória; uma população com diversidade próxima de 0 é uma população estagnada com todos os genomas semelhantes entre si. Uma forma indireta de medir a diversidade genética da população pode ser feita pela quantidade de genomas diferentes que estão presentes na população (*Genotypes*), ou pela quantidade de genomas diferentes que são selecionados para reprodução (*SelectedGenotypes*), no caso dos algoritmos geracionais.

3.4.5 Significância estatística

Quando aplicadas sobre algoritmos evolutivos que utilizam os dois tipos de populações, as métricas definidas anteriormente fornecem um conjunto de valores que podem ser comparados. Todavia, a aleatoriedade do processo evolutivo leva a que duas execuções diferentes gerem valores distintos para as métricas que avaliam as propriedades do algoritmo. Para contornar esta aleatoriedade foi gerado um conjunto de simulações, das quais foram computadas as respetivas médias e os desvios padrões. Para verificar se a utilização das multipopulações possuem um efeito significativo no processo evolutivo, utilizamos o teste de hipóteses *T-student* com 95% de significância, para verificar se as médias das métricas são efetivamente iguais. Se forem iguais, concluímos que não existe diferença estatística na utilização das multipopulações; se forem diferentes, podemos concluir que a utilização da multipopulação foi benéfica ou prejudicial, comparando o valor das duas médias. Numa métrica onde os valores maiores são os melhores, a exemplo do número de ótimos encontrados (*NumOptimumsFound*), o algoritmo com a média mais alta é melhor. Em sentido inverso, de que é exemplo o número de avaliações para chegar ao ótimo (*FuncsCallToOptimum*), o algoritmo com a média mais baixa é melhor.

3.5 Comentários finais

Os multisets são um tipo de conjunto que suporta elementos repetidos e cujas repetições são acomodadas na estrutura de dados sob a forma de um valor inteiro. Neste capítulo, explorámos as possibilidades da utilização dos multisets para representar as populações dos algoritmos evolutivos, definindo o conceito de multi-indivíduo como um par ordenado <*genoma, cópias*> e o conceito de multipopulação como um conjunto de multi-indivíduos. Relacionámos ainda os conceitos de multi-indivíduos como um conjunto de clones de indivíduos simples e definimos a equivalência entre uma multipopulação e uma população simples.

A utilização das multipopulações em algoritmos evolutivos implica que o processo evolutivo sofra algumas alterações. Através da equivalência entre as multipopulações e as populações simples, os operadores genéticos de seleção, reprodução e mutação tradicionais podem ser utilizados no processo evolutivo. Os operadores de substituição têm de ser adaptados para garantir a integridade da multipopulação.

Como a seleção de indivíduos para reprodução deve utilizar o número de cópias dos multi-indivíduos, apresentámos o operador *MTournamentSelection*, que utiliza torneios

para selecionar indivíduos numa multipopulação. A substituição de gerações precisa manter a dimensão do conjunto de suporte das multipopulações constante e, neste sentido, apresentámos o operador *MTournamentReplacement*, que introduz na população principal uma geração de descendentes através de torneios. Devido à estrutura dos multi-indivíduos, que conseguem acomodar os clones no número de cópias, o número total de indivíduos tende a crescer e é necessário controlá-lo de forma a não prejudicar o operador de seleção. Este controlo de cópias é feito por um novo operador, o redimensionamento, cuja finalidade é reduzir o número de cópias dos multi-indivíduos e, para tal, apresentamos o operador *AdaptiveCeiling* para executar essa tarefa.

Nos capítulos seguintes vamos analisar a influência das multipopulações no processo evolutivo de diferentes algoritmos.

4 Algoritmo genético com populações baseadas em multisets

Ao longo do processo evolutivo, as multipopulações conseguem manter um conjunto distinto de genomas na população e recebem os clones no número de cópias dos indivíduos. Estas duas propriedades influenciam o processo evolutivo da população, pois os clones influenciam o processo de seleção dos reprodutores, e a diversidade da população exerce influência na quantidade dos diferentes genótipos selecionados. Os progenitores selecionados para reprodução afetam de forma direta a nova geração de indivíduos e as multipopulações têm um grande influxo neste operador.

Para avaliar estes dois fenómenos introduzimos a multipopulação no algoritmo genético padrão, *Simple Genetic Algorithm* (SGA). Os algoritmos genéticos mimetizam o processo de evolução natural e representam um grupo importante de algoritmos evolutivos.

4.1 MuGA – Multiset Genetic Algorithm

A substituição das populações simples por multipopulações no algoritmo genético padrão deu origem ao *Multiset Genetic Algorithm* (MuGA), figura 4-1, que é um algoritmo genético geracional que utiliza multipopulações (MP) em todas as fases do processo evolutivo. Esta característica não impede a utilização dos operadores genéticos tradicionais, dado que as multipopulações possuem a capacidade de se transmutarem em populações simples, possibilitando, ainda, a definição de novos operadores genéticos que tiram partido dos multi-indivíduos:

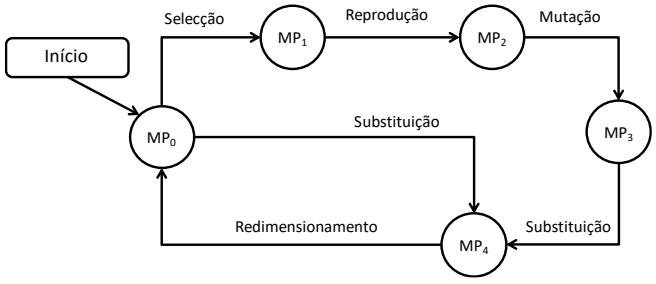


figura 4-1 MuGA - Multiset Genetic Algorithm: Operadores e multipopulações

O algoritmo MuGA, algoritmo 4-1, possui três parâmetros: o problema a resolver (*problem*), o número de multi-indivíduos da população (*popSize*) e o número de indivíduos selecionados para reprodução (*selectSize*). O MuGA começa com a geração da população inicial (MP_0) e com este procedimento, temos a garantia de que são gerados *popSize* multi-indivíduos do problema, que é passado por parâmetro, sendo o

seu número de cópias inicializado a um. De seguida, o algoritmo entra no ciclo geracional e este termina quando o critério de paragem for atingido.

A população MP_1 é produzida pelo operador de seleção que seleciona *selectSize* indivíduos de MP_0. Pode ser utilizado qualquer operador de seleção, desde que este utilize o número de cópias dos multi-indivíduos para aumentar a pressão seletiva sobre o genótipo. O operador genético de reprodução é aplicado sobre a população selecionada para reprodução (MP_1), gerando uma nova população (MP_2) e, sobre esta última, é aplicado o operador de mutação, dando origem a uma nova população (MP_3). Sempre que os operadores genéticos originem genótipos repetidos, estes são guardados no número de cópias dos multi-indivíduos.

Na substituição de gerações, a população inicial (MP_0) compete com os seus sucessores (MP_3) por um lugar na nova geração (MP_4). Este operador deve garantir, por definição, que a dimensão do conjunto de suporte da nova geração é igual ao da população dos progenitores (MP_0). A competição entre as duas gerações gera normalmente clones, o que faz com que o número de cópias dos indivíduos mais bem-adaptados cresça. O aumento do número de cópias de um genótipo é o resultado da competição dos melhores progenitores presentes na geração atual com os seus descendentes, que se espera serem igualmente bons ou melhores. Os bons indivíduos representam extremos locais da função de avaliação, e o seu número de cópias tende a crescer enquanto não forem substituídos. O processo iterativo termina com o redimensionamento das cópias da população (MP_4), e a evolução continua para uma nova geração até que o critério de paragem seja alcançado:

```
MuGA (problem, popSize , selectSize)
  MP0 = Criar e avaliar popSize multi-indivíduos de problem
  repete
    MP1 = selecionar selectSize Indivíduos de MP0
    MP2 = recombinar os indivíduos de MP1
    MP3 = mutar os indivíduos de MP2
    avaliar MP3
    MP4 = selecionar popSize multi-indivíduos de MP3 e MP0
    MP0 = redimensionar o número de cópias de MP4
  até atingir o critério de paragem
fim
```

algoritmo 4-1 - Algoritmo MuGA - Multiset Genetic Algorithm

Para verificar experimentalmente a influência das multipopulações nos algoritmos genéticos, desenhamos um conjunto de experiências em que o elemento variável é a representação da população. O MuGA otimiza multipopulações, baseadas em multisets, e o SGA otimiza populações simples baseadas em conjuntos. O MuGA necessita de operadores especializados na seleção de progenitores para a reprodução e a substituição de gerações. Estes operadores adaptados para multipopulações funcionam igualmente em populações simples, porque estes utilizam listas de indivíduos, e uma população simples já é uma lista de indivíduos.

Como operadores genéticos para a reprodução dos progenitores escolhemos os operadores usuais pelo seu bom funcionamento neste tipo de algoritmos. Estes operadores estão definidos para populações simples e podem ser utilizados em multipopulações, através da expansão dos multi-indivíduos num conjunto de clones (indivíduos simples), figura 3-4.

O operador de redimensionamento tem como finalidade a redução do número de cópias dos multi-indivíduos, pelo que só faz sentido se aplicado a multipopulações. Sobre as populações simples o mesmo não tem qualquer efeito, pois consideramos que um indivíduo simples possui um número de cópias unitário.

Por uma questão de simplicidade, redefinimos os operadores descritos no capítulo anterior de forma a omitir as populações nos parâmetros dos operadores, dado que elas fazem parte do processo evolutivo. A seguir são apresentados os operadores que foram usados nas experiências:

- *MTournamentSelection*(*tourSize, parentsProportion*) – Seleção de um conjunto de reprodutores de uma população utilizando torneios com reposição. A quantidade de indivíduos selecionados é calculada como uma proporção (*parentsProportion*) do tamanho da população principal. Se a população for simples, o tamanho é equivalente ao número de indivíduos; se for baseada em multisets, corresponde à dimensão do conjunto de suporte (número de genótipos). Desta forma, são selecionados o mesmo número de progenitores nos dois tipos de populações. A dimensão do torneio é dada pelo parâmetro (*tourSize*).
- *Uniform*(*probCrossover, probSwap*) - Reprodução por troca uniforme de genes. O operador é parametrizado pela probabilidade de cruzamento (*probCrossover*) e pela probabilidade de trocar um gene entre os progenitores (*probSwap*). A probabilidade de cruzamento controla o número de indivíduos que são

submetidos ao processo reprodutivo para a geração de descendentes. Os indivíduos que não se reproduzem ficam na população como clones dos seus progenitores.

- *Flipbit(probMutation)* - Mutação por inversão de bits. Cada gene de cada descendente é submetido a uma troca probabilística (*probMutation*) do seu valor binário.
- *MTournamentReplacement(tourPar, tourOff)* – Substituição da população principal através da competição de gerações utilizando torneiros. O operador utiliza torneios binários sem reposição para selecionar os indivíduos que vão continuar o processo evolutivo. O progenitor é selecionado através de um torneio de dimensão *tourPar* na população principal e o descendente é escolhido por um torneiro de dimensão *tourOff* na população dos descendentes.
- *AdaptiveCeiling(maxProportion)* – Redimensionamento do número de cópias dos multi-indivíduos através de um fator adaptativo. O parâmetro *maxProportion* define a proporção desejada entre o número de genótipos e o número de indivíduos da multipopulação.

4.2 Efeito do operador de redimensionamento no MuGA

As multipopulações são constituídas por multi-indivíduos que acomodam os seus clones no número de cópias. Esta característica permite que uma multipopulação tenha um número potencialmente infinito de indivíduos. Para aferir da necessidade e utilidade do operador de redimensionamento, utilizamos a função *OneMax,* equação 4.1, que conta o número de bits a 1 que o genoma possui. A função é unimodal, de fácil otimização pelos algoritmos evolutivos e amplamente estudada na literatura. O comportamento das multipopulações nesta função é similar ao que se pode encontrar em funções multimodais, quando o algoritmo descobre um máximo durante o processo evolutivo.

$$f(x) = \sum_{i=1}^{n} x_i \qquad (4.1)$$

O operador de redimensionamento tem como função reduzir o número de cópias que os multi-indivíduos vão ganhando ao longo do processo evolutivo. Em (Manso e Correia 2009), foi apresentado o primeiro estudo do efeito do operador de redimensionamento fixo, *FixedCeiling*, na otimização de uma multipopulação de 128 multi-indivíduos do problema *OneMax* pelo MuGA. A figura 4-2 apresenta a evolução

do número de indivíduos na multipopulação ao longo de 250 gerações, com o valor do fator de redimensionamento a variar entre 1.0 e 3.0. A experiência mostra que o MuGA sem o operador de redimensionamento (r=1.0) não controla o número de indivíduos (divisão por 1), e este cresce descontroladamente. Para valores do fator superiores, o operador permite controlar e estabilizar o número de indivíduos da população:

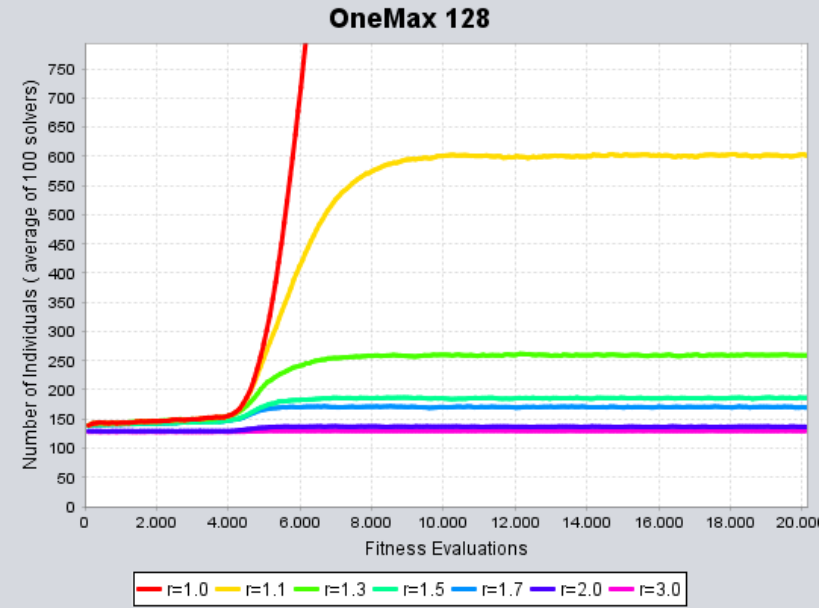


figura 4-2 Evolução do número de indivíduos no MuGA : Otimização do problema OneMax com o operador de redimensionamento FixedCeiling

As experiências realizadas com o operador *FixedCeiling* expuseram diversas fragilidades no seu funcionamento. O número de indivíduos na população depende muito das características da rugosidade da função de avaliação, o que dificultava a definição do valor do parâmetro. Por outro lado, não existe uma relação direta entre o valor do parâmetro e o número de indivíduos na população.

Item	**Descrição**
Populações	64 indivíduos e 64 multi-indivíduos do problema OneMax com 128 genes
Seleção	*MTournamentSelection(parentsProportion=2.0; toursize=2)*
Reprodução	*Uniform(probCrossover=0.75, probSwap=0.5)*
Mutação	*Flipbit(probMutation= 1/128)*
Substituição	*MTournamentReplacement(tourPar= 2, tourOff= 2)*

tabela 4-1-Configuração base das experiências para avaliar a influência das multipopulações nos algoritmos genéticos.

Estas dificuldades conduziram ao desenvolvimento do operador de controlo adaptativo, *AdaptiveCeiling*, por forma a facilitar o controlo do número de cópias dos multi-indivíduos. A tabela 4-1 apresenta as populações e os operadores genéticos da experiência, e a tabela 4-2 apresenta as variantes do operador de redimensionamento.

As multipopulações do MuGA são constituídas por 64 multi-indivíduos e o SGA utiliza uma população com 64 indivíduos simples. O operador de seleção escolhe 128 progenitores da população principal, *parentsProportion=2.0*, através de torneios binários, *toursize=2*. Os progenitores são recombinados com uma probabilidade de 75%, *probCrossover=0.75*, através de recombinação uniforme com probabilidade de 50% de fazer a troca de genes, *probSwap= 0.5*. A mutação da nova geração é feita com a inversão do valor do gene com uma probabilidade de 1/128, o que significa que em média é mudado um gene por indivíduo, *probMutation= 1/128*. A substituição de gerações é feita através de um torneio binário entre um progenitor e um descendente, que são selecionados em torneios de dimensão 2 (*tourPar= 2, tourOff= 2*) nas respetivas populações.

Experiência	**Operador de Redimensionamento**
NoRescaling	MuGA sem operador de redimensionamento
MuGA-0.5	MuGA com *AdptiveCeiling(maxProportion = 0.5)*
MuGA-1.0	MuGA com *AdptiveCeiling(maxProportion = 1.0)*
MuGA-1.5	MuGA com *AdptiveCeiling(maxProportion = 1.5)*
MuGA-2.0	MuGA com *AdptiveCeiling(maxProportion = 2.0)*
MuGA-4.0	MuGA com *AdptiveCeiling(maxProportion = 4.0)*
SGA	Algoritmo genético padrão com populações simples.

tabela 4-2- Configuração das experiências para avaliar a influência do operador de redimensionamento.

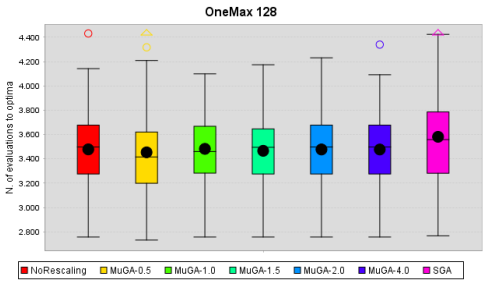
A tabela 4-2 apresenta as configurações das experiências para avaliar a influência do operador de redimensionamento. As experiências MuGA apresentam um conjunto de simulações onde o parâmetro *maxProportion* do operador varia entre 0.5 e 4.0. A experiência *NoRescaling* utiliza o MuGA sem nenhum operador de redimensionamento e, por seu turno, a experiência SGA, ao utilizar o algoritmo genético padrão, não necessita do operador porque utiliza populações simples.

A tabela 4-3 mostra o resultado das simulações após 7.000 chamadas à função de avaliação, e a figura 4-4 a evolução de diversas métricas ao longo do processo evolutivo. Os valores apresentados na tabela 4-3 são a média e o desvio padrão de 100 simulações após 7000 chamadas à função de avaliação:

	FuncsCallsToOptimum		GeneticDiversity		CeilingFactor		Individuals		MaxCopies	
Experiência	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
NoRescaling	3478,0	301,0	0,00543	0,00112			1460,5	214,1	1373,0	213,0
MuGA-0.5	3453,9	323,6	0,05430	0,00061	2,2	0,0	69,3	1,3	5,2	1,0
MuGA-1.0	3482,3	280,6	0,04816	0,00145	1,5	0,1	92,9	5,7	18,0	2,9
MuGA-1.5	3466,1	286,6	0,04344	0,00160	1,3	0,0	126,9	4,3	36,3	3,8
MuGA-2.0	3477,4	298,3	0,03710	0,00141	1,2	0,0	154,3	3,0	63,2	4,0
MuGA-4.0	3477,1	298,0	0,02283	0,00116	1,1	0,0	270,7	3,8	181,5	4,4
SGA	3581,4	404,9	0,00000	0,00000			64,0	0,0	1,0	0,0

tabela 4-3-Efeito do operador de redimensionamento após 7.000 chamadas à função de avaliação -100 simulações.

A função *OneMax* é de fácil otimização, e todas as experiências obtiveram uma taxa de sucesso de 100%. Todas as simulações com multipopulação alcançaram aquela taxa de sucesso com um menor número de chamadas à função de avaliação. Isto é tanto mais significativo se compararmos tais simulações com o algoritmo SGA, baseado em populações simples, sendo a diferença estatisticamente significativa, se aplicarmos o teste T com significância de 95% (figura 4-4).



a)

Comparison of means - confidence 95%

	NoRescaling	MuGA-0.5	MuGA-1.0	MuGA-1.5	MuGA-2.0	MuGA-4.0	SGA
NoRescaling	.	.	.	.	.	.	+
MuGA-0.5	.	.	.	.	.	.	+
MuGA-1.0	.	.	.	.	.	.	+
MuGA-1.5	.	.	.	.	.	.	+
MuGA-2.0	.	.	.	.	.	.	+
MuGA-4.0	.	.	.	.	.	.	+
SGA	-	-	-	-	-	-	.

b)

figura 4-3-Efeito do operador de redimensionamento no problema OneMax: a)gráfico de boxplot do número de chamadas à função de avaliação para obter o ótimo; b)resultado do teste de hipóteses.

Na figura 4-3 b) as comparações são feitas entre as linhas e as colunas. O ponto (.) significa que aceitamos a hipótese de que as médias são iguais; caso contrário, rejeitamos essa hipótese, com o sinal de positivo (+) a significar que a média é melhor, e o sinal negativo (-), que é pior. Com os resultados apresentados na tabela 4-3 podemos concluir que as simulações com multipopulações são estatisticamente equivalentes entre si e, todas elas, são melhores do que a simulação SGA.

A simulação MuGA-0.5, com o valor *maxProportion=0.5*, representa uma experiência extrema em que se desejam apenas metade dos indivíduos em relação ao número de genomas, o que é impossível, porque a quantidade de genomas se mantém constante na população, por definição. Esta simulação aplica um fator de redimensionamento que começa em 2.0, porque necessita de fazer uma redução do número de indivíduos para metade, enquanto que nas restantes simulações o fator começa em 1, pois não precisam de fazer qualquer redução.

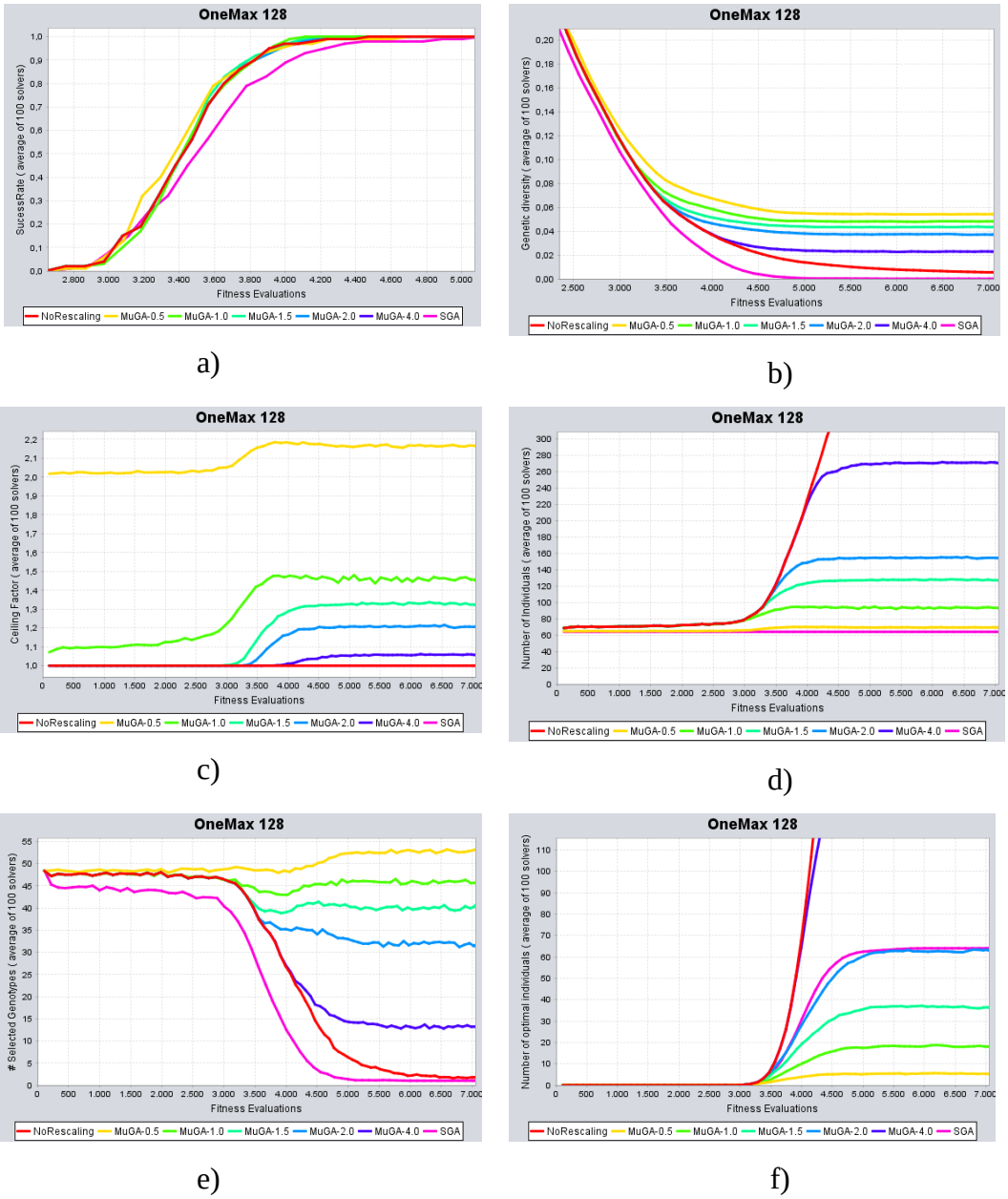


figura 4-4 Evolução dos parâmetros genéticos ao longo de 7.000 avaliações na otimização da função OneMax: a) Taxa de sucesso; b) Diversidade genética da população; c) Fator de redimensionamento; d) Número de indivíduos; e) Número de genótipos selecionados para reprodução; f) Número de ótimos na população

O fator de redimensionamento mantém-se estável até o processo evolutivo encontrar o indivíduo ótimo que, nas gerações seguintes, começará a ganhar cópias (figura 4-4 f) controladas pelo fator de redimensionamento, figura 4-4 c). Após algumas gerações, o número de cópias deste indivíduo estabiliza num valor que depende do valor de *maxProportion*. A experiência *NoRescaling* não aplica o operador, pelo que o número de cópias do ótimo (1373.0) fica descontrolado, e o tamanho da população cresce em cada geração. No final da experiência a população tem 1460.5 indivíduos em média e com tendência crescente. Esta experiência evidencia a necessidade de controlo do número de cópias na multipopulação.

A Experiência MuGA-0.5 define uma proporção de 50% de indivíduos em relação aos multi-indivíduos, um valor impossível, já que o valor mínimo de cópias é unitário. Esta experiência estabiliza num fator de redimensionamento elevado (2.2), que elimina a maior parte das cópias dos multi-indivíduos, quando o processo evolutivo estabiliza, ficando o ótimo com 5.2 cópias em média. Esta simulação é um caso extremo da utilização do operador *AdaptiveCeiling* que, no entanto, obtém a melhor marca em termos de eficiência, 3453.9 avaliações para o ótimo. A população desta simulação é composta por poucos clones, numa média de 69.3 indivíduos em 64 multi-indivíduos, e apenas o individuo ótimo tem múltiplas cópias. Este sucesso deve-se à elevada diversidade genética da população e à capacidade do algoritmo genético em aproveitar para gerar descendentes que superam os pais.

A diversidade genética da população decai ao longo do processo evolutivo, figura 4-4 b), em todas as simulações; nas multipopulações estabiliza de acordo com o valor do parâmetro *maxProportion*. Note-se, porém, que a diversidade genética da população está diretamente relacionada com o número de cópias dos multi-indivíduos, figura 4-4 d). Ora, como o número de indivíduos são controlados pelo operador de redimensionamento, a diversidade estabiliza com a evolução da população. Isto não acontece na experiência *NoRescaling* devido à falta de controlo do número de cópias, nem na experiência SGA, uma vez que a população não é baseada em multisets.

A diversidade genética na população principal influencia diretamente a diversidade genética da população que é selecionada para reprodução, figura 4-4 e). O número de genótipos selecionados nas multipopulações é sempre maior do que na simulação com populações simples, o que permite que o operador de recombinação gere uma maior diversidade de descendentes que, eventualmente, são melhores do que os progenitores, pois a função é unimodal. Esta diversidade de descendentes bem-adaptados permite

chegar ao máximo mais rapidamente, o que explica a maior eficiência do MuGA em relação ao SGA (figura 4-4 a). Quando o ótimo é atingido, o SGA propaga os clones na população, e a sua diversidade genética cai para valores próximos de zero, pois a população é constituída por clones do ótimo (figura 4-4 f). Se no MuGA houver controlo do número de cópias, a diversidade genética da população estabiliza e consequentemente, normaliza também a diversidade da população dos reprodutores, os quais trocam material genético entre si, incorporado na população pelo operador de substituição, que privilegia os descendentes em caso de empate. Este controlo do número de cópias tem consequências na diversidade dos reprodutores e, deste modo, a reciclagem de material genético da população principal explica a eficiência do MuGA em problemas difíceis. Sem o controlo de cópias, experiência NoRescaling, nada disto é possível.

A otimização da função *OneMax* é muito fácil e a parametrização do operador de redimensionamento deve ter em conta que o MuGA será utilizado para otimizar problemas mais difíceis. Por defeito escolhemos o valor de *maxProportion=1.5* do operador de *AdaptiveCeiling* pelas seguintes razões:

a) Este valor permite que a população principal tenha multi-indivíduos em número que se aproxima do dobro do conjunto de suporte. Note-se que um grande número de cópias atrasa a eficiência do processo evolutivo devido à diminuição da diversidade genética, ao passo que um número pequeno de cópias prejudica a pressão dos bons indivíduos sobre os restantes. Através do resultado da nossa experiência com o MuGA, consideramos que um valor de cópias igual ao tamanho do conjunto de suporte é um valor equilibrado;
b) O número de cópias da multipopulação está dividido por vários indivíduos. Na experiência MuGA-1.5 o ótimo tem em média 36.3 cópias num total de 126.9 indivíduos, o que implica aproximadamente 26.6 cópias espalhadas pelos restantes indivíduos. Nas experiências MuGA-0.5 e MuGA-1.0, o ótimo concentra a quase totalidade das cópias.
c) As várias cópias dos multi-indivíduos na população principal incrementam a probabilidade de haver multi-indivíduos com várias cópias na população selecionada para reprodução. O número de cópias dos reprodutores será utilizado nos capítulos seguintes para o desenho de operadores genéticos baseados em multisets.

4.3 Influência das multipopulações no esforço computacional

Para avaliarmos o esforço computacional das multipopulações no processo evolutivo, executamos o algoritmo MuGA e SGA durante 100 gerações com os parâmetros definidos na secção anterior. A simulação MuGA-64 e SGA-64 evoluem populações com 64 elementos e a SGA-128 tem 128.

Em cada geração são selecionados 128 indivíduos em cada um das simulações, e por isso a simulação a SGA-128 utiliza o operador *MTournament(1.0, 2*), e as restantes *MTournament(2.0, 2)*.

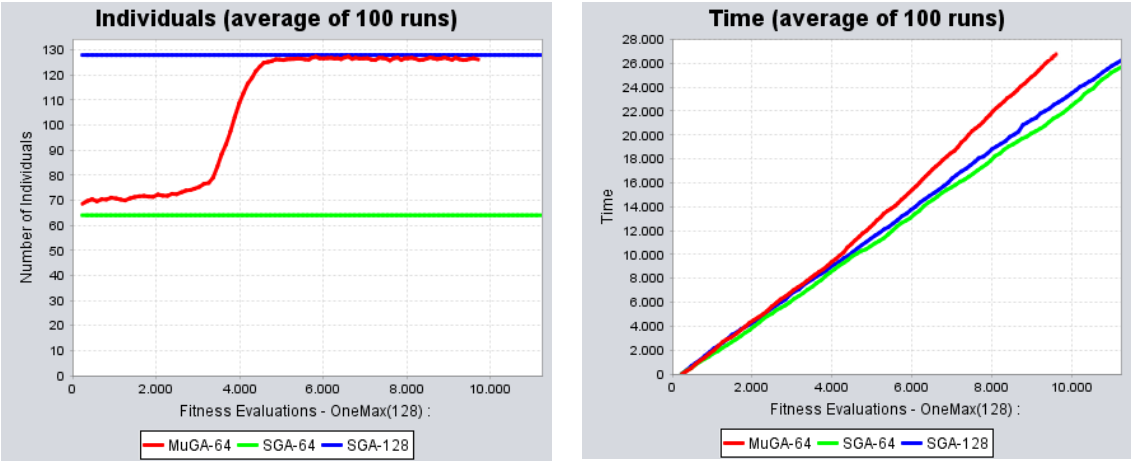
A tabela 4-4 apresenta o resultado da execução computacional de 100 simulações, e a figura 4-5 a evolução do número de indivíduos e do tempo de simulação em milissegundos. Os tempos de execução são provenientes de numa máquina virtual com processador Intel Xeon E5-2620, que possui 8 unidades de processamento e 16 GB de memória RAM. As multipopulações demoraram mais tempo na execução, contudo, representam um acréscimo de cerca de 3% do tempo total.

As multipopulações necessitaram de menos chamadas à função de avaliação para a executar 100 gerações, porque a avaliação de um multi-indivíduo equivale à de todos os seus clones. Os operadores genéticos produzem indivíduos iguais, que são avaliados de forma eficiente pela multipopulação como se pode ver na figura 4-5 a).

O MuGA evolui uma população com 64 multi-indivíduos, que através da evolução vão ganhando clones até ao máximo expectável de 128 figura 4-5 a). A figura 4-5 b) apresenta o tempo de simulação em relação ao número de chamadas à função de avaliação e, nela podemos ver que os tempos são equiparáveis até cerca das 4.000 avaliações. A partir deste momento o MuGA começa a ganhar multi-indivíduos, figura 4-5 a), e as gerações do MuGA necessitam de avaliar menos descendentes, porque os clones são agrupados nos multi-indivíduos.

	FunctionCalls		Time (ms)		Individuals	
Experiência	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-64	9529,23	62,34	26877,52	59,46	126,50	3,61
SGA-64	11113,73	46,09	25940,89	89,10	64,00	0,00
SGA-128	11108,44	41,00	26466,47	62,84	128,00	0,00

tabela 4-4- Esforço computacional na otimização da função OneMax após 100 gerações -100 simulações.



a) b)

figura 4-5 Evolução da função OneMax durante 100 gerações: a) Número de indivíduos na população b) Tempo de otimização em milissegundos.

4.4 Influência das multipopulações no processo evolutivo

Nas secções seguintes vamos analisar a influência das multipopulações no processo evolutivo e comparar os resultados com as populações simples. A tabela 4-5 apresenta a configuração base das experiências para avaliação da influência das multipopulações no processo evolutivo. Na verdade, as experiências que a seguir se descrevem otimizam diversos tipos de problemas, com propriedades distintas, de forma a inferir a influência das multipopulações nessas classes de problemas. Todos os problemas são constituídos por 128 variáveis que são codificadas em 128 genes.

Para a otimização dos problemas utilizamos algoritmos genéticos que usam dois tipos distintos de populações: o SGA usa populações tradicionais, enquanto o MuGA utiliza populações baseadas em multisets. Devido ao facto de o MuGA utilizar multipopulações, onde o número de multi-indivíduos é constante, variando apenas o seu número de cópias, definimos o número de indivíduos nas populações do SGA correspondente ao limite mínimo, 64 indivíduos, e o limite máximo expectável de indivíduos que a multipopulação poderá conter, 128 indivíduos. A experiência MuGA-64 contempla uma multipopulação de 64 multi-indivíduos que, ao utilizar o operador de redimensionamento parametrizado em 1.5, faz com que a população suporte sensivelmente o dobro dos indivíduos. Por esta razão, a experiência SGA-64 utiliza uma população com 64 indivíduos, e a SGA-128 utiliza uma população com 128.

O operador de seleção *MTournamentSelection* utiliza torneios de dimensão 2, *toursize=2*, para selecionar 128 indivíduos para reprodução. No caso das populações com 64 elementos, o operador é configurado com o parâmetro *parentsProportion=2.0*;

na população com 128 indivíduos, com o parâmetro *parentsProportion=1.0*. Assim, em todas as experiências são selecionados o mesmo número de reprodutores que vão dar origem à nova geração.

Item	Descrição
Experiência	MuGA-64 – MuGA com uma multipopulação com 64 multi-indivíduos SGA-64 – SGA com uma população de 64 indivíduos simples SGA-128 – SGA com uma população de 128 indivíduos simples
Indivíduos	128 genes
Seleção	*MTournamentSelection*(*parentsProportion={2.0,1.0}, toursize= 2*)
Recombinação	*Uniform*(*probCrossover=0.75*, *probSwap = 0.5*)
Mutação	*Flipbit*(*probMutation= 1/128*)
Substituição	*MTournamentReplacement*(*tourPar= 2, tourOff= 2*)
Redimensionamento	*AdaptiveCeiling*(maxProportion=1.5)

tabela 4-5-Configuração base das experiências para avaliar a influência das multipopulações no processo evolutivo com algoritmos genéticos.

Os operadores genéticos de reprodução e de mutação operam sobre indivíduos simples, pelo que as multipopulações da experiência MuGA-64 são transformadas em populações simples. O operador de reprodução *Uniform* é aplicado com uma probabilidade de 75% (*probCrossover=0.75*) aos reprodutores, e os seus genes são trocados com uma probabilidade de 50% (*probSwap=0.5*). O operador de mutação *Flipbit* é usado nos genes dos indivíduos com uma probabilidade de 1/128, o que permite a existência, em média, de uma mutação por indivíduo.

A substituição de gerações é feita com o operador *MTournamentReplacement***,** que utiliza torneios binários para selecionar um progenitor (*tourPar=2*) e um descendente (*tourOff=2*) para competirem por um lugar na geração seguinte.

Este conjunto de operadores e os seus respetivos parâmetros constituem a base das experiências efetuadas, através das quais vamos estudar a influência das multipopulações no processo evolutivo.

4.4.1 Funções com planaltos – Royal Road R1

A utilização dos algoritmos genéticos permite a obtenção de boas soluções na resposta a problemas do mundo real; no entanto, não existe uma teoria universalmente aceite que explique a razão do seu bom funcionamento. A fundamentação teórica dos algoritmos genéticos foi tratada, desde a sua fundação, por Jonh Holland através da "teoria dos esquemas" (Holland 1975). Apesar de não fornecer uma explicação cabal para o funcionamento dos algoritmos genéticos, a teoria fornece pelo menos uma razão lógica e válida. Com efeito, Holland defende que os blocos construtores, subsequências formadas por elementos com elevado nível de adaptação, têm uma maior probabilidade de serem selecionados para reprodução, porque esses indivíduos têm um grau de aptidão superior à média. Por sua vez, o operador de cruzamento tende a combinar os blocos construtores nos descendentes, formando outros cada vez maiores, que são introduzidos na população. A propagação e o crescimento dos blocos construtores ao longo das gerações faz com que o algoritmo convirja para boas soluções do problema.

Para o estudo da interação dos blocos construtores no desempenho dos algoritmos genéticos,(Mitchell, Holland, e Forrest 1991) desenvolveram um conjunto de funções designadas por *Royal Road*. A primeira destas, a função *R1*, está apresentada na equação 4.2, que conta o número de esquemas de dimensão *C* contidos no cromossoma *x*, começando nas posições múltiplas de *C*:

$$\mathrm{R1}(x) = \sum_{i=1}^{q} C_s \delta_s(x), \qquad onde\ \delta_s(x) = \begin{cases} 1 & x \in S_i \\ 0 & caso\ contrário \end{cases} \tag{4.2}$$

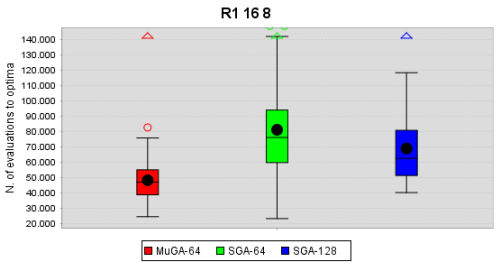
Nesta equação, q é o número de esquemas, $S = (S_1, \ldots, S_q)$ e C_s é o tamanho do esquema. Para avaliar o efeito das multipopulações na propagação e crescimento de esquemas, efetuamos uma experiência para otimizar a função *R1* com dimensão de 128 bits, divididos em 16 esquemas de dimensão 8.

Como critério de paragem, definimos o valor de 200.000 chamadas à função de avaliação, e para obter uma significância estatística forte, executamos 100 simulações. A tabela 4-6 apresenta o resultado das métricas evolucionárias no final das simulações, e a figura 4-7 apresenta a progresso das métricas ao longo do processo evolutivo.

A função *R1* é de fácil otimização pelos algoritmos genéticos, e todas as simulações obtiveram uma taxa de sucesso de 100%. Quanto ao esforço que é necessário para otimizar a função (*FuncsCallsToOptimum*), o MuGA é mais eficiente do que o SGA, e esta eficiência é estatisticamente significativa, como se pode verificar na figura 4-6.

	FuncsCallsToOptimum		SucessRate		Individuals		SelectedGenotypes		GeneticDiversity	
Experiência	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-64	48095,1	13332,5	1,0	0,0	126,2	3,8	39,1	3,6	0,0449	0,0018
SGA-64	81038,7	28430,2	1,0	0,0	64,0	0,0	1,0	0,0	0,0000	0,0000
SGA-128	68782,4	24372,3	1,0	0,0	128,0	0,0	1,0	0,0	0,0000	0,0000

tabela 4-6- Resultado da otimização da função R1 após 200.000 chamadas à função de avaliação - 100 simulações.



a)

Comparison of means - confidence 95%

	MuGA-64	SGA-64	SGA-128
MuGA-64	.	+	+
SGA-64	-	.	-
SGA-128	-	+	.

b)

figura 4-6 Número de chamadas à função de avaliação para obter o ótimo na função R1-16-8: a) gráfico de boxplot*; b) Resultado do teste de hipóteses t-test.*

Na figura 4-7 a) podemos ver a evolução da taxa de sucesso das simulações e o resultado na tabela 4-6. O MuGA precisa de menos esforço computacional, a saber 48095.1 chamadas à função de avaliação para atingir os 100%, enquanto o SGA-64 necessita de 81038.7 e o SGA-128 de 68782.4, em média. Isto demonstra que o incremento do número de indivíduos no SGA ampliou a sua eficiência, e este aumento estatisticamente significativo, como se pode ver na figura 4-6. O número de indivíduos no MuGA começa em 64 e vai crescendo ao longo do processo evolutivo, estabilizando em 126.2 no final da simulação.

O sucesso do MuGA está diretamente relacionado com a diversidade genética apresentada pela população ao longo do processo evolutivo, figura 4-7 b). No início da evolução, a diversidade genética cai abruptamente em todas as simulações, pois a população é conduzida para as regiões do espaço mais promissoras. Todavia, enquanto nas simulações com o SGA a diversidade genética continua a decair até atingir valores próximos de 0, o MuGA mantém essa diversidade estável (0.0449). Esta diversidade genética tem influência direta na quantidade de indivíduos distintos que são selecionados para reprodução, figura 4-7 c), um número que no MuGA se mantém estável (próximo dos 40 genomas diferentes, em todas as fases do processo evolutivo), ao passo que no SGA decai continuamente até estabilizar no valor 1. Também a

quantidade de clones do ótimo contidos na população, figura 4-7 e), confirma as observações anteriores, uma vez que no SGA o número de ótimos na população corresponde à sua dimensão, 64 ou 128, respetivamente, quando no MuGA estabiliza no valor 38,4.

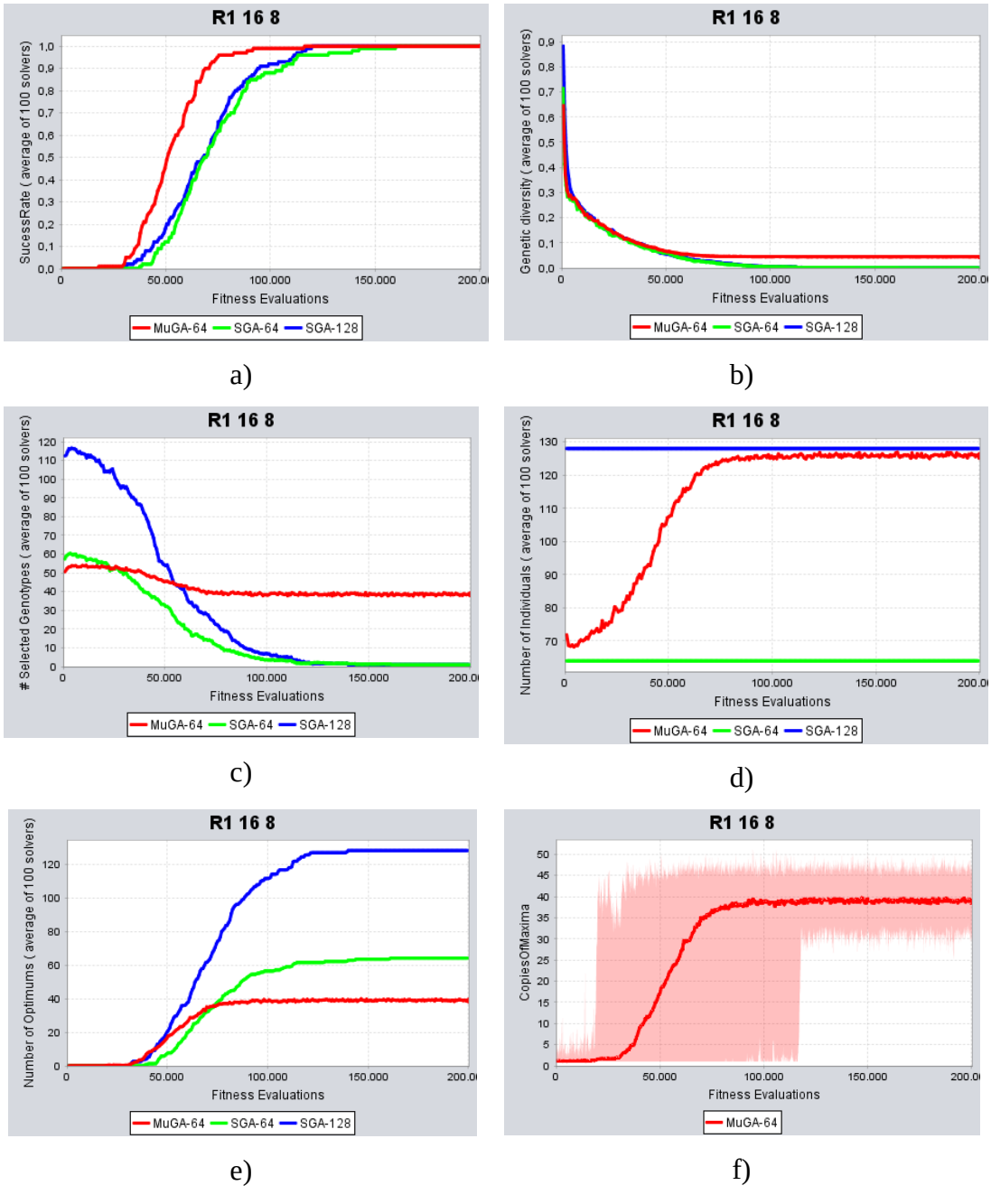


figura 4-7 Otimização da função R1-16-8: Evolução dos parâmetros genéticos ao longo de 200.000 avaliações: a) Taxa de sucesso; b) Diversidade genética da população; c) Número de genótipos selecionados para reprodução d) Número de indivíduos na população; e) Número de clones ótimos na população; f) Número cópias do melhor individuo na multipopulação do MuGA (mínimo, média e máximo)

No MuGA o número de indivíduos varia ao longo do processo evolutivo, aumentando progressivamente no início e estabilizando quando as simulações atingem o ótimo figura 4-7 d). A evolução do número de cópias do melhor individuo da população no MuGA ao longo do processo evolutivo está representado na figura 4-7 f): a linha contínua apresenta a média das 100 simulações e o sombreado delimita o valor máximo e o mínimo. No início da evolução, os melhores indivíduos ganham poucas cópias, porque estes são rapidamente substituídos por indivíduos melhores, que voltam a ganhar cópias, e assim sucessivamente, até ser alcançado o ótimo. Quando o ótimo é atingido, este continua a ganhar cópias, uma vez que não é substituído. O número de cópias é controlado pelo operador de redimensionamento e estabiliza em valores a rondar os 40.

A experiência realizada mostra que a utilização de multipopulações na otimização da função *R1* aumenta a eficiência do processo evolutivo, o que se explica pelo número de cópias dos indivíduos portadores dos melhores esquemas da população. Este ganho de eficiência só se pode dever à estrutura da população, pois os operadores genéticos de reprodução e mutação são os mesmos para todas as simulações.

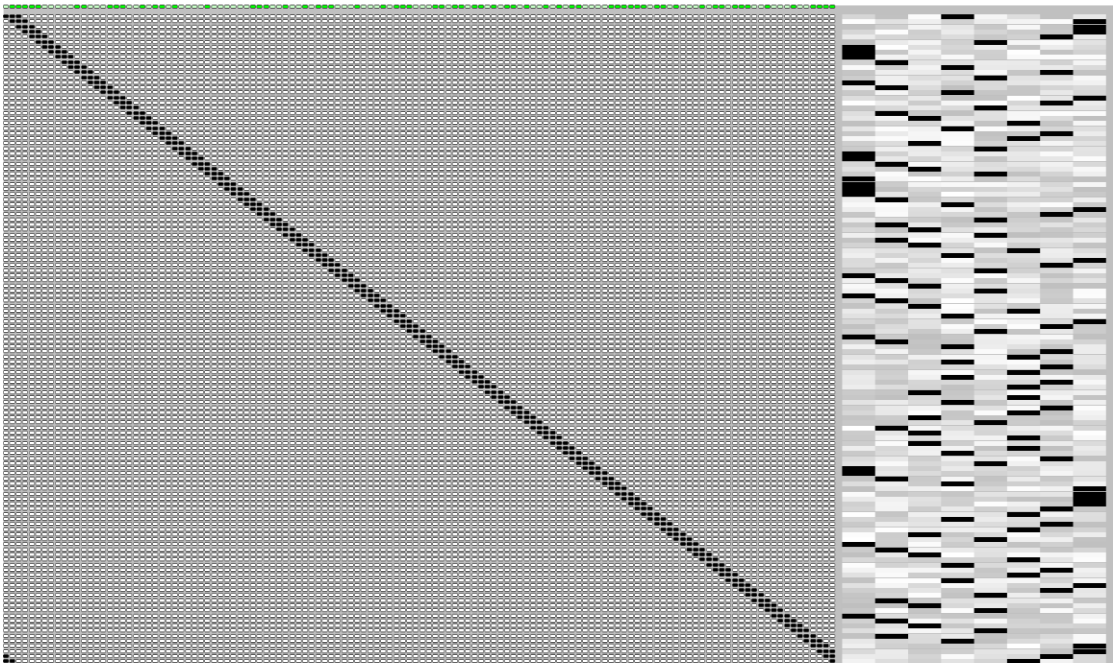
4.4.2 Funções rugosas – NK-Landscapes

A computação evolutiva é comummente utilizada para otimizar processos complexos, onde os métodos matemáticos ou não estão disponíveis, ou não podem ser utilizados pela complexidade da função que o descreve. Tal complexidade está diretamente relacionada com a rugosidade da paisagem delineada pela sua representação gráfica, e as funções complexas apresentam por regra vários extremos locais, o que dificulta a sua otimização.

Para estudar o efeito das multipopulações na otimização de funções rugosas, utilizamos o modelo NK, definido por Stuart Kaufman, que desenvolveu o modelo para estudar a interação entre os genes e a sua expressão no fenótipo em modelos biológicos. A versatilidade do modelo permite estudar muitos outros domínios, desde as ciências sociais, as ciências económicas ou as ciências da computação (Hwang et al. 2018). O modelo permite simular a epistasia, ou seja, a interação de vários genes para influenciar uma característica, e a partilha de genes por várias características. O modelo é parametrizado pelo número características (N), também chamadas regras, e pelo número de genes que influenciam a característica (K). Na equação 4.3, a função NK(x)é representada como a soma de todas as características expressas no cromossoma:

$$NK(x) = \sum_{i=1}^{N} F_i(x_{i1;}x_{i2;\ldots;}x_{ik}) \quad (4.3)$$

Nesta equação F_i representa a função que avalia a característica i e $x_{i1;}x_{i2;\ldots;}x_{ik}, \ldots, x_{ik;}$, os genes do cromossoma x que contribuem para a expressão dessa característica. O modelo mais simples tem o valor de *K=1*, onde cada gene influencia apenas uma característica, e a função 4.3 é equivalente à função *OneMax* descrita na função 4.1, com uma paisagem suave. O valor *K* define o nível de epistasia entre os genes, e o valor de *N* influencia o número de características influenciadas por um gene. Quanto maior o *N* e o *K*, mais rugosa se torna a paisagem. No modelo binário, onde cada gene só pode assumir o valor 0 ou 1, a contribuição de cada característica para o cálculo da adaptação pode assumir 2^k valores distintos.



a) b)

figura 4-8 Representação gráfica do problema NK-128-3-128: a) genes que influenciam a característica b) valor do fenótipo para cada combinação de valores do gene.

A figura 4-8 apresenta a representação gráfica da função *NK-128-3-128*, cujos valores da aptidão do fenótipo, correspondente ao valor dos genes, estão no Anexo B. A função *NK-128-3-128* é um problema com 128 variáveis, que indexam 128 características (*N=128*) e uma epistasia de 3 genes (*K=3*), ou seja, cada gene influencia 3 regras. O problema considera o cromossoma circular, e os genes que influenciam as características estão dispostos sequencialmente no cromossoma.

A figura 4-8 mostra os genes que indexam as características, do lado esquerdo, e cada um dos 8 valores do fenótipo em tons de cinzento no lado direito. O valor a negro representa o valor máximo da característica indexada pelos 3 bits. No topo está apresentado o genoma do indivíduo ótimo: a cor verde clara representa o valor 0 e o escuro o valor 1.

No problema *NK-128-3-128*, os valores do fenótipo expresso pelos genes foram gerados com valores aleatórios no intervalo [0,1]. Também foi gerado de forma aleatória um genoma, que representa o indivíduo ótimo, e foi colocado o valor 1 nas respetivas posições. Desta forma, sabemos à partida o valor da aptidão do melhor indivíduo e, assim, podemos inferir o sucesso do processo evolutivo.

	FuncsCallsToOptimum		SucessRate		Individuals		SelectedGenotypes		GeneticDiversity	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-64	17694,6	14155,8	1,00	0,00	138,6	5,4	40,2	2,9	0,0557	0,0016
SGA-64	11357,5	22100,3	0,33	0,47	64,0	0,0	1,0	0,1	0,0001	0,0005
SGA-128	15271,8	25146,4	0,48	0,50	128,0	0,0	1,0	0,0	0,0000	0,0000

tabela 4-7- Resultado da otimização da função NK-128-3-128 após 100.000 chamadas à função de avaliação.

A tabela 4-7 mostra o resultado da evolução depois de 100.000 chamadas à função de avaliação e a mesma tabela apresenta ainda a evolução dos parâmetros ao longo do processo evolutivo. Nesta experiência, apenas o MuGA alcançou uma taxa de sucesso de 100%, enquanto o SGA-64 obteve 33%, e o SGA-128, 48%. A evolução da taxa de sucesso de cada uma das simulações pode ser vista na figura 4-9 a). No caso do MuGA, o número de chamadas à função para obter o ótimo (*FuncsCallToOptimum*) diz respeito a todas as simulações, enquanto no SGA-64 e no SGA-128 se refere apenas às simulações que terminaram, razão que explica o valor mais baixo.

A figura 4-9 d) apresenta a evolução da diversidade genética nas simulações, e a figura 4-9 c) a evolução do número de indivíduos selecionados para reprodução. No algoritmo SGA, a diversidade genética cai para valores próximos de zero, revelando que a população é constituída por um grande número de clones, e o número de genomas distintos é aproximadamente 1. Por outro lado, o MuGA estabiliza a população com uma diversidade genética de 0.0558, permitindo que sejam selecionados cerca de 40 genótipos diferentes em cada geração. O valor da função *NK* é a soma do valor de todas as características, o fenótipo, que é avaliado por 3 genes distintos, e a alteração de um gene altera o valor de 3 características simultaneamente. Tal constatação obriga que a

função tenha vários extremos locais e, para que o processo evolutivo tenha sucesso, é necessário, por norma, mais do que um bit para escapar aos extremos locais. A otimização da função NK implica que os indivíduos sofram mutações benéficas, influenciando positivamente a sua aptidão, ainda que essas mutações sejam espalhadas pela população através do operador de recombinação.

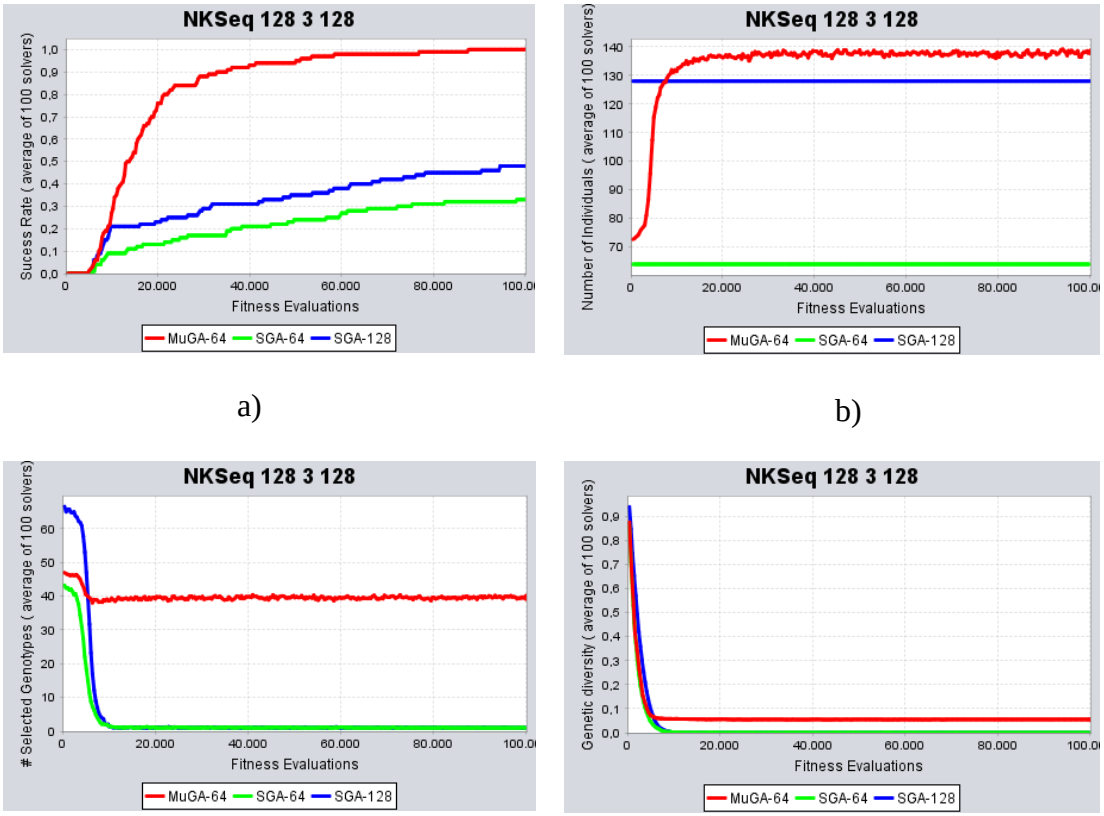


figura 4-9 - Otimização da função NKSeq128-3-128: Evolução dos parâmetros genéticos ao longo de 100.000 avaliações: a) Taxa de sucesso; b) Número de indivíduos; c) Número de genótipos selecionados para reprodução; f) Diversidade genética da população

O operador de mutação altera em média 1 bit por indivíduo em cada iteração, mas se várias destas mutações permanecerem na população, o operador de recombinação agrupa-as nos descendentes e o algoritmo escapa aos extremos locais. Este fenómeno acontece apenas no MuGA, possuidor de uma diversidade genética que lhe permite selecionar indivíduos com pelo menos 40 genes diferentes, agrupados pelo operador de recombinação nos descendentes. É isto que explica a taxa de sucesso do algoritmo, pois no SGA a diversidade genética é muito reduzida. As mutações genéticas benéficas são assimiladas pela população de forma muito rápida, não dando tempo para o aparecimento de novas mutações que possam ser assimiladas em conjunto.

O sucesso do MuGA na otimização desta função deve-se essencialmente à diversidade genética da multipopulação, que consegue suportar várias mutações genéticas nos seus multi-indivíduos, transmitidas aos descendentes pelo operador de recombinação. O número de cópias tem um contributo positivo na seleção dos melhores indivíduos, o que, aliado à diversidade genética selecionada, permite ao MuGA escapar aos extremos locais e obter uma taxa de sucesso plena.

4.4.3 Funções com vários ótimos - Knapsack

Os algoritmos evolutivos possuem a capacidade de fornecer várias boas soluções, eventualmente ótimas, para o problema a resolver, uma faculdade muito útil para o decisor que opera em ambientes dinâmicos, pois permite-lhe escolher entre várias opções. Estas alternativas permitem a que se faça uma análise da sensibilidade dos parâmetros da função de avaliação, que muitas vezes, estão codificados de forma difusa. Estes parâmetros podem representar valores, por exemplo, o valor de mercado de um item ou o nível de stock das matérias primas, difíceis de quantificar de forma precisa, e, por esta razão a faculdade de os algoritmos fornecerem várias opções é muito importante.

Para analisar a capacidade de o MuGA descobrir e manter soluções ótimas na população utilizamos o problema da mochila binária, *knapsack*. Ora, a mochila binária acarreta o problema de otimização combinatória NP completo (Pisinger 2005), com várias aplicações a várias circunstâncias do mundo real. Existem diversas aplicações práticas do problema na área de gestão de recursos, da criptografia ou da engenharia e, por conseguinte, tem sido objeto de resolução através de vários métodos: *backtracking*, métodos gulosos, programação dinâmica e computação evolucionária.

A definição do problema envolve um conjunto de itens (*n*) que possuem um peso (*w*) e um valor (*p*) dos quais serão escolhidos alguns itens para meter numa mochila. O objetivo do algoritmo é maximizar o valor dos itens na mochila, equação 4.4, sem violar o seu peso máximo (*C*), equação 4.5.

$$p(x) = \sum_{i=1}^{n} p_i x_i \qquad (4.4)$$

Nesta equação, $x = (x_1, \dots x_n)$ é um vetor de variáveis lógicas que definem se o item está na mochila ($x_i = 1$) ou não ($x_i = 0$), e p é um vetor com o valor dos itens:

$$\sum_{i=1}^{n} w_i x_i \leq C \qquad (4.5)$$

Na equação 4.5, w é um vetor com o peso de cada item, e C representa a capacidade máxima da mochila. No caso de o peso dos itens selecionados excederem a capacidade

da mochila, *C*, estamos perante uma solução inviável. Para lidar com estas soluções, podemos tomar um de três caminhos:

1. Descartar a solução – neste caso perdemos toda a informação que a solução contém e a aproximação ao ótimo é feita apenas por soluções válidas;
2. Reparar a solução – neste caso aplica-se um algoritmo para tornar a solução viável através da eliminação ou substituição de itens;
3. Penalizar a solução – neste caso a solução é introduzida na população, mas a sua aptidão é diminuída de forma a refletir a sua inviabilidade;

Na resolução deste problema, optamos por penalizar as soluções inviáveis através da fórmula da equação 4.6, que penaliza o excesso de peso da mochila de forma quadrática. Desta forma a aproximação ao ótimo também pode ser feita através de soluções inválidas:

$$penalização(x) = \left(\sum_{i=1}^{n} w_i x_i - C\right)^2 \quad (4.6)$$

O problema Knapsack 128, presente no Anexo C, foi gerado de forma aleatória com 128 itens, cujo valor e peso estão fortemente correlacionados. A capacidade da mochila foi parametrizada para 50% do peso total dos itens, equação 4.7, os pesos foram gerados com valores inteiros de forma aleatória no intervalo [1,128], equação 4.8, e o valor de cada item foi gerado adicionando ao seu peso um valor inteiro gerado aleatoriamente entre zero e um décimo do seu peso, equação 4.9:

$$C = 0.5 \sum_{i=1}^{n} w_i \quad (4.7)$$

$$w_i = random\ integer\ [1,128] \quad (4.8)$$

$$p_i = w_i + random\ integer\ [0, wi/10] \quad (4.9)$$

O problema Knapsack 128-0.5, com uma capacidade da mochila de 50% do peso total dos itens (4283), possui um valor ótimo de 4615, que pode ser obtido através de programação dinâmica, e existem milhares de combinações que permitem obtê-lo. A figura 4-10 apresenta o esquema dos ótimos conhecidos: a verde estão os itens que fazem parte de todas as soluções; a vermelho, os que não fazem parte de nenhuma solução; e a branco com asterisco, os itens que variam de solução para solução.

figura 4-10 Knapsack 128-0.5: Esquema dos ótimos do problema: a verde os itens que fazem parte de todos os ótimos; a vermelho os que não fazem parte de nenhum ótimo; a branco com asteriscos os itens que variam nas soluções ótimas

A tabela 4-8 apresenta o resultado final da otimização depois de 1.500.000 chamadas à função de avaliação, e a figura 4-11 apresenta a evolução das estatísticas ao longo do processo evolutivo. O *knapsack* é um problema difícil de otimizar pelo algoritmo genético simples quando a sua dimensão é elevada. Esta dificuldade na otimização foi apresentada em (Manso e Correia 2011 a) e (Manso e Correia 2011b), onde os autores apresentam os resultados da otimização de problemas Knapsack com 50, 100 e 250 itens, pelo MuGA e por outros algoritmos evolutivos.

	SucessRate		Individuals		NumOptimaFound		NOptGenomesPop		GeneticDiversity	
Experiência	Média	Desv.Pad.	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-64	1,0	0,0	126,8	5,2	225,58	108,38	61,8	9,5	0,1087	0,0060
SGA-64	0,0	0,0	64,0	0,0	0,00	0,00	0,0	0,0	0,0001	0,0004
SGA-128	0,0	0,0	128,0	0,0	0,00	0,00	0,0	0,0	0,0001	0,0004

tabela 4-8- Resultado da otimização da função Knapsack 128-0.5 após 1.500.000 chamadas à função de avaliação.

Na otimização do problema Knapsack 128-0.5, o MuGA alcançou uma taxa de sucesso de 100%, enquanto o SGA não conseguiu fazer nenhuma otimização, figura 4-11 a). A razão do sucesso do MuGA está ligada à diversidade genética da população introduzida pelo multiset, que lhe permite a escolha para reprodução de uma grande quantidade de genótipos diferentes figura 4-11 d).

A recombinação dos indivíduos selecionados permite a geração de descendentes bem-adaptados, os quais superam os extremos locais da função. Este processo começa logo nas primeiras fases da evolução, como se pode ver na figura 4-11 b), onde as soluções do MuGA são sempre melhores que o algoritmo SGA, pois este fica preso em extremos locais logo nas primeiras fases do processo evolutivo, quando a população fica estagnada pelo facto de o operador de seleção escolher muito poucos genomas para reprodução, figura 4-11 f), e, deste modo, o algoritmo só consegue progredir por mutações.

O MuGA, por outro lado, aproveita as mutações benéficas nos descendentes para as espalhar e as combinar com outras na população através do operador de recombinação. Os indivíduos que representam extremos locais começam a ganhar cópias logo no início da evolução, pois o número de indivíduos sobe rapidamente para valores próximos de 128, figura 4-11 e), aumentando assim a probabilidade de serem selecionados para reprodução.

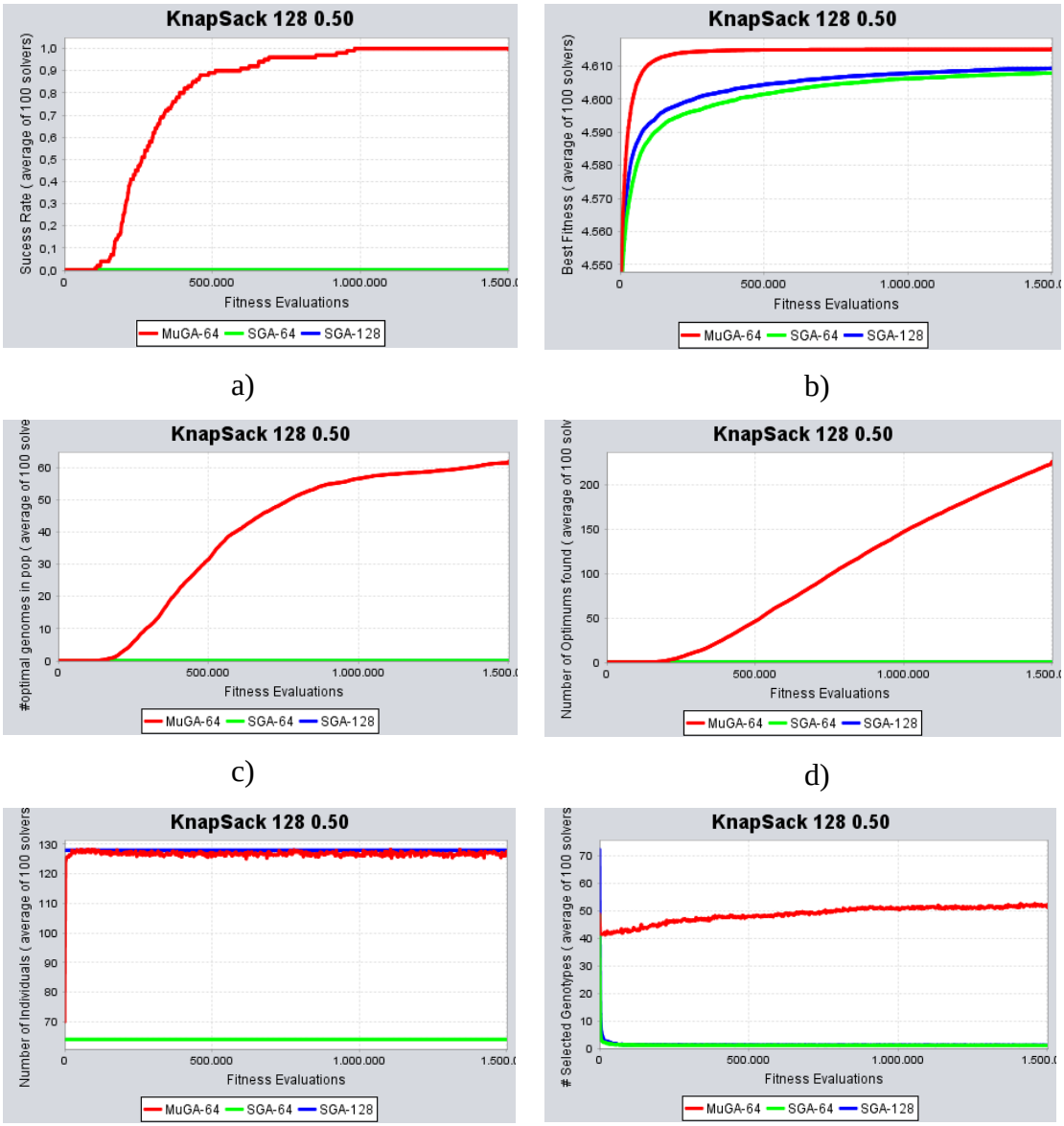


figura 4-11 Knapsack 128-0.5: Evolução dos parâmetros genéticos ao longo de 1.500.000 avaliações: a) taxa de sucesso; b) aptidão do melhor indivíduo; c) número de genomas ótimos na população; d) número de ótimos encontrados; e) número de indivíduos da população; f) número de genomas selecionados para reprodução

As multipopulações do MuGA não só ajudam os operadores de recombinação e de mutação a evoluir a população para os ótimos, mas também possuem a capacidade de os reter na população. Esta característica pode ser verificada na figura 4-11 c): no final da evolução a população de 64 multi-indivíduos possui 61.8 ótimos em média. A grande variedade de indivíduos a que nos referimos tem reflexo na diversidade genética dos genomas selecionados para reprodução, em média 51.4 de 64 possíveis, e explica a capacidade de explorar o espaço de procura e encontrar os vários ótimos. Como o operador de substituição de gerações *MTournamentReplacement* tem preferência pelos

descendentes, em caso de igualdade no valor da aptidão, o MuGA vai descobrindo ótimos sucessivos: 225.58 em média no final da simulação. Na figura 4-11 d) podemos ver que a tendência para encontrar novos ótimos é real.

4.4.4 Otimização de permutações – ChessQueen

Na computação evolutiva, os genes representam uma variável do problema e o cromossoma dos indivíduos uma solução. Na resolução dos problemas anteriores, os genes eram representados por variáveis lógicas e o objetivo do algoritmo era descobrir o valor, 0 ou 1, de cada uma delas. Nos cromossomas dos indivíduos, o *locus* de cada gene é estático e é decidido na definição do problema.

Outro tipo diferente de problema é aquele em que sabemos o valor dos genes e precisamos de saber qual a sua posição, *locus*, no cromossoma. Estes problemas são designados pela expressão "otimização combinatória", e têm sido objeto de extensa pesquisa pela complexidade da sua resolução pelos métodos matemáticos tradicionais e pela sua aplicação prática a questões do mundo real.

Para averiguar a influência das multipopulações na otimização de problemas baseados em permutações, utilizamos o problema das rainhas no tabuleiro de xadrez. O objetivo do problema é colocar *n* rainhas num tabuleiro de xadrez de dimensão *n* x *n* de forma a que não haja ataques entre elas. A figura 4-12 apresenta uma solução do problema de dimensão 4 e a respetiva representação do cromossoma. Este é um problema de minimização e a avaliação do cromossoma é feita pelo número de ataques que as rainhas fazem no tabuleiro.

Os operadores genéticos de recombinação e de mutação definidos na tabela 4-5 têm de ser adaptados por forma a que possam lidar com a representação de permutações no cromossoma. A reprodução por troca uniforme de genes entre os progenitores precisa de ser redefinida de forma a que, ao fazer a troca de genes entre os progenitores, gere descendentes viáveis.

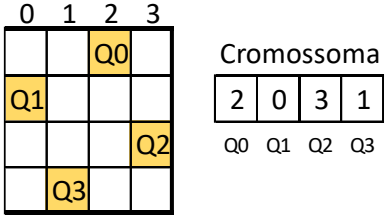


figura 4-12 Uma solução do problema das rainhas no tabuleiro de xadrez 4x4: Fenótipo e Genótipo.

A figura 4-13 apresenta o operador *Uniform Permutation Crossover* (UPX), uma variante do operador utilizado nas otimizações anteriores para cromossomas baseados em permutações, baseado no que foi apresentado em (Hartmann 1998). Tal como acontece na versão para genes binários, é aplicado a dois progenitores que vão gerar dois descendentes através de uma máscara, determinando quais serão os genes doados por um progenitor. O operador UPX opera em três fases:

1. Copia os genes de um progenitor utilizando máscara.
2. Copia os genes do outro progenitor que ainda não estejam no cromossoma, utilizando a máscara.
3. Completa o cromossoma com os genes que faltam, utilizando a ordem definida no outro progenitor.

O operador UPX, figura 4-13, faz a recombinação dos genes dos progenitores através de uma máscara, gerada de forma aleatória, preservando ao máximo a ordem de ambos os progenitores nas duas primeiras fases, e utiliza uma terceira fase para introduzir os genes em falta, baseados também na ordem de um progenitor. Este operador é menos disruptivo do que o apresentado em (Hartmann 1998), usado na resolução do problema de alocação de tarefas, que utiliza apenas as fases 1 e 3.

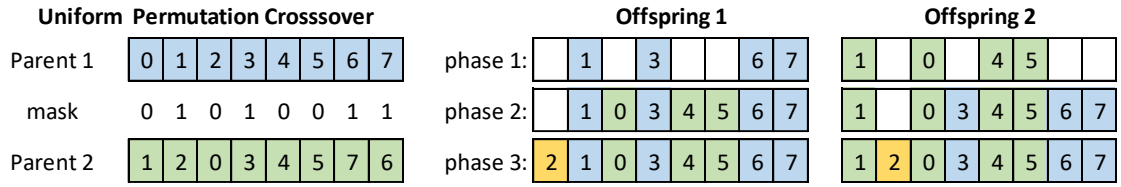


figura 4-13 UPX - Uniform Permutation Crossover

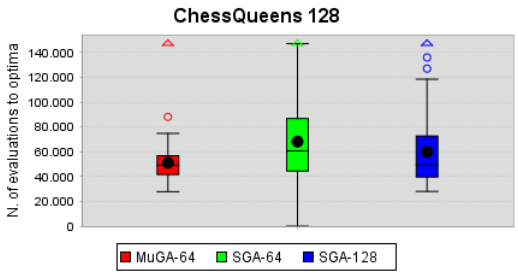
Como operador de mutação, utilizamos o operador *SwapGenes***,** que faz a troca de genes entre duas posições aleatórias do cromossoma.

A tabela 4-9 apresenta o resultado da otimização da função *ChessQueen* 128, que é um problema para a colocação de 128 rainhas num tabuleiro de 128x128, utilizando os parâmetros da tabela 4-5. Neste caso, o operador de recombinação foi substituído pelo operador *UPX***,** e o operador de mutação pelo operador *SwapGenes*.

	SucessRate		FuncsCallsToOptimum		Individuals		NumOptimaFound		NOptGenomesPop		GeneticDiversity	
Experiência	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	DesvPad
MuGA-64	1,00	0,00	51182,0	12754,7	130,5	4,7	689,6	93,2	64,0	0,0	0,109	0,009
SGA-64	0,98	0,14	68261,1	32408,0	64,0	0,0	222,6	74,9	4,0	1,7	0,004	0,002
SGA-128	1,00	0,00	59900,2	28181,5	128,0	0,0	334,2	103,8	6,9	2,2	0,006	0,003

tabela 4-9- Resultado da otimização da função ChessQueen 128 após 200.000 chamadas à função de avaliação.

O MuGA conseguiu fazer a otimização de todas as simulações, e o esforço de otimização consumiu menos chamadas à função de avaliação do que os restantes algoritmos. O SGA apenas obteve uma taxa de sucesso de 100% com uma população de 128 indivíduos, enquanto o SGA-64 conseguiu otimizar 98% das simulações. A eficiência do MuGA é melhor quando comparada com o SGA, e esta diferença é estatisticamente significativa, figura 4-14.



a)

Comparison of means - confidence 95%

	MuGA-64	SGA-64	SGA-128
MuGA-64	.	+	+
SGA-64	-	.	.
SGA-128	-	.	.

b)

figura 4-14 ChessQueens-128 - Número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot*; b) Resultado do teste de hipóteses t-test*

O problema tem muitos ótimos e vários deles foram encontrados pelos algoritmos: o algoritmo SGA-64 obteve 222.6, o SGA-128 atingiu 334.2, e o MuGA obteve 689.6 ótimos na média de 100 simulações.

O algoritmo genético é muito eficaz para encontrar novos ótimos e, enquanto algoritmo de substituição de gerações, manifesta a preferência por soluções vindas dos descendentes, os algoritmos têm uma tendência crescente para encontrar novos ótimos. A figura 4-15 c) mostra a evolução do número de ótimos encontrados durante o processo evolutivo, e podemos verificar que a tendência de descoberta é mais acentuada no MuGA do que nos SGA. Esta observação é explicada pela diversidade genética da população, tabela 4-9, e pela quantidade de genomas diferentes selecionados para reprodução, figura 4-15 b), onde o MuGA seleciona mais genomas e, por isso, consegue uma maior variedade na população dos descendentes.

A diversidade genética da multipopulações no MuGA permite que o processo evolutivo seja mais eficiente e eficaz, mas também admite que os ótimos encontrados se mantenham na população como podemos ver na figura 4-15 d). No final da simulação, verificamos que os algoritmos SGA, com populações simples, possuem 4.0 e 6.9 genomas ótimos nas respetivas populações, enquanto o MUGA apresenta 64

ótimos, o que significa que a população é constituída apenas por soluções ótimas. O SGA encontra os ótimos e, durante o processo evolutivo, vai-os substituindo por novas soluções ótimas, eventualmente clones. Por seu turno, o MuGA, quando encontra a soluções ótimas, incrementa o número cópias, caso o novo ótimo seja um clone de um progenitor, ou substitui um indivíduo, caso seja um genoma novo.

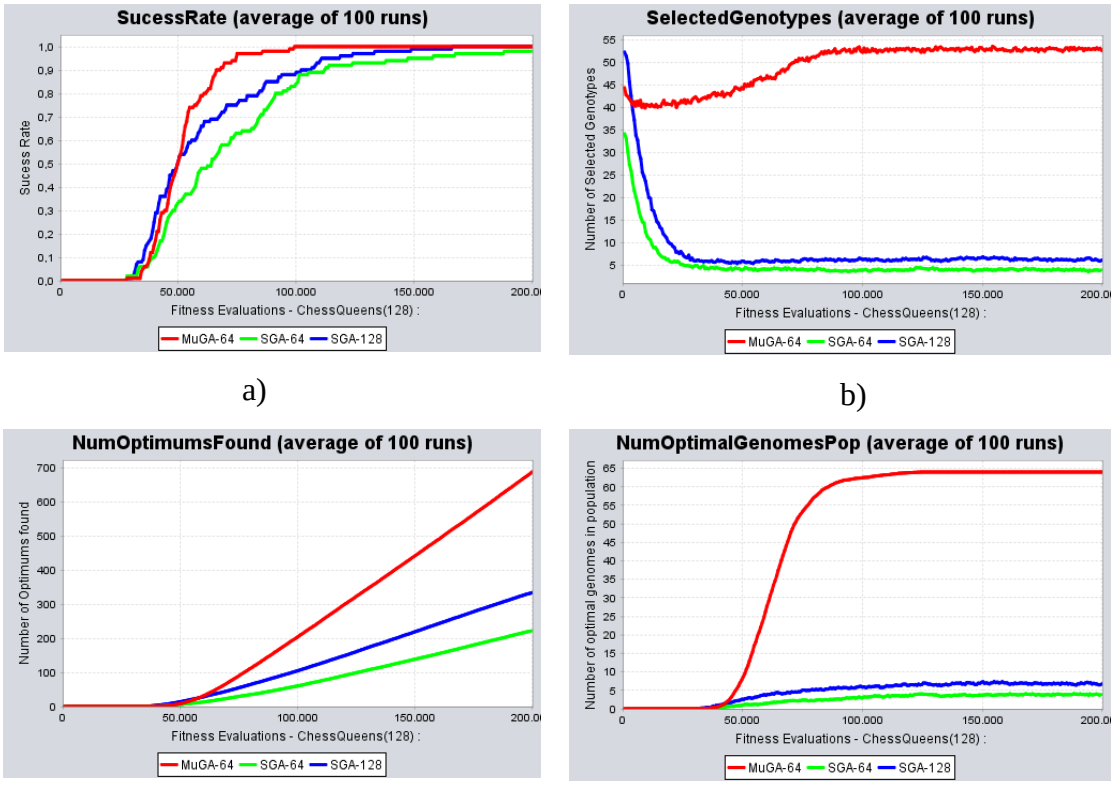


figura 4-15 ChessQueen 128: Evolução dos parâmetros genéticos ao longo de 200.000 avaliações: a) Taxa de sucesso; b) diversidade genética; c) Número de genomas ótimos encontrados; d) Número de genomas ótimos na população

4.5 Comentários finais

Neste capítulo verificamos experimentalmente a influência das multipopulações no algoritmo genético. Definimos o *Multiset Genetic Algorithm*, MuGA, e verificamos experimentalmente a influência das multipopulações no processo evolutivo, utilizando um conjunto diversificado de problemas. Todos eles foram parametrizados para conterem 128 genes, e a influência foi verificada através da comparação dos resultados com o algoritmo genético padrão, SGA. Desenhamos um conjunto de experiências onde cada algoritmo evolui 128 indivíduos em cada geração. A experiência MuGA-64 utiliza uma multipopulação com 64 multi-indivíduos; o SGA-64, uma população simples com 64 indivíduos; e o SGA-128, uma população de 128 indivíduos.

A função *OneMax* foi utilizada para comprovar a necessidade do operador de redimensionamento para o controlo do número de cópias durante o processo evolutivo. Sem este operador, as multipopulações do MuGA possibilitariam aos melhores indivíduos tomarem conta da população, promovendo a sua deriva genética. Com o operador de redimensionamento *AdaptiveCeiling*, o MuGA controla o número de cópias, e o algoritmo torna-se mais robusto.

As multipopulações necessitam de menor esforço para avaliar a população, quando comparadas com as populações simples, porque a avaliação de um multi-indivíduo equivale à avaliação de todos os seus clones. Relativamente ao tempo de execução, as multipopulações são ligeiramente piores (3%), no entanto este esforço é largamente compensado pela sucesso no processo evolutivo.

A função *Royal Road R1* foi utilizada para testar o comportamento do MuGA em problemas com planaltos. Embora o algoritmo SGA-128 seja mais eficiente do que o SGA-64, o que significa que o aumento do número de indivíduos é benéfico neste tipo de problemas, o MuGA é mais eficiente do que ambos os SGA.

Na otimização de funções rugosas, utilizamos o problema *NK-landscapes* e o MuGA foi muito eficaz, encontrando o ótimo em todas as simulações, enquanto que ambas as versões do SGA tiveram muitas dificuldades na otimização da função, e a sua eficácia foi muito reduzida.

Para a otimização de problemas com vários ótimos utilizamos o problema *Knapsack* e, uma vez mais, o MuGA obteve excelentes resultados, enquanto o SGA não atingiu o ótimo em qualquer simulação.

Para a otimização de problemas representados por permutações, definimos o operador *Uniform Permutation Crossover*, UPX, para fazer o cruzamento de dois indivíduos de forma a preservar ao máximo as características dos progenitores na geração dos descendentes. Utilizamos o problema *ChessQueen*, que também apresenta vários ótimos e todos algoritmos foram eficazes na sua resolução, no entanto, o MuGA é o mais eficiente deles todos. Neste problema pudemos verificar que o MuGA não só consegue encontrar os ótimos de forma mais eficiente, como também os consegue manter na população.

A superioridade do MuGA em relação ao SGA, nos problemas analisados, só pode ser atribuída às multipopulações, pois o algoritmo que as evolui é o mesmo, e os operadores genéticos também. As multipopulações garantem uma heterogeneidade genética, muito útil para a construção das novas gerações. Esta diversidade traduz-se

na quantidade de genomas distintos selecionados para reprodução e na qualidade dos descendentes, mas durante o processo evolutivo, as populações simples perdem diversidade, através da incorporação de clones, fazendo com que o operador de reprodução perca a sua utilidade ao recombinar indivíduos iguais. Este fenómeno não se verifica nas multipopulações: o operador de seleção seleciona sempre vários indivíduos distintos, e o operador de recombinação não perde a sua eficácia. Nas experiências anteriores verificamos que os melhores indivíduos ganham cópias ao longo do processo evolutivo, o que incrementa a sua probabilidade de seleção para reprodução.

Como conclusão podemos afirmar que a diversidade genética introduzida pelo conjunto de suporte da multipopulação, juntamente com a pressão seletiva introduzida pelo número de cópias dos multi-indivíduos, obriga a que o algoritmo genético tenha um acréscimo significativo em termos de eficiência e de eficácia.

5 Algoritmos de nicho com populações baseadas em multisets

Nos algoritmos genéticos, a perda da diversidade genética ao longo do processo evolutivo faz com que estes não sejam adequados para a otimização de funções complexas e multimodais. Estas dificuldades são relatadas por vários autores (Mahfoud 1995) e foram expostas no capítulo anterior, onde o SGA teve muitas dificuldades em otimizar a função *NK-128-3* e não conseguiu otimizar a função *Knapsack 128-0.5*. Mas, por outro lado, o MuGA não teve qualquer dificuldade a otimizar as duas funções, porque as multipopulações oferecem uma diversidade genética que as populações simples são incapazes de fornecer.

Para resolver estes problemas foram propostos uma classe de algoritmos que exploram o conceito natural de nicho. Os algoritmos de nicho são baseados no fenómeno da formação de espécies na evolução natural, onde um conjunto de indivíduos partilham o mesmo espaço e os seus recursos. Nestes algoritmos a população é dividida em várias espécies, subpopulações, onde os indivíduos têm interações com os seus semelhantes. A manutenção de várias espécies, ou seja, de indivíduos semelhantes entre si, mas distintos da restante população, promove a diversidade genética da população global.

Os algoritmos de nicho têm como principal objetivo a manutenção da diversidade genética na população, de forma a evitar a convergência prematura da população e a facilitar o encontro de vários máximos, eventualmente ótimos. Encontrar muitos ótimos, e de preferência significativamente diferentes, é importante para um grande número de problemas, tais como a otimização de múltiplos objetivos em simultâneo. Existem várias modelos de evolução que conseguem este objetivo, mas os mais representativos são os modelos de partilha do valor de aptidão, *fitness sharing*, e os algoritmos de aglomeração, *crowding*. Em (Mengshoel e Goldberg 2008) é apresentado um estudo destes algoritmos e das suas variantes.

Nas secções seguintes vamos fazer a adaptação do processo evolutivo por *crowding* e por *fitness sharing*, para o usarmos em multipopulações e verificarmos a sua influência na otimização das funções NK-128-3 e Knapsack 128 0.5.

5.1 Influência das multipopulações no algoritmo de crowding

O MuGA é um algoritmo evolucionário geracional em que a população evolui através de gerações e, em cada geração, parte da população principal é substituída pelos

seus descendentes. Ora, na natureza, a evolução não se processa desta forma, pois a morte e o nascimento de indivíduos ocorrem de forma continuada. Todavia, estes atos contínuos de nascimento e morte podem ser simulados por algoritmos evolutivos de estado estacionário, onde um pequeno conjunto de indivíduos, tradicionalmente 2, é selecionado para reprodução e produz descendência, que, por sua vez, é incorporada novamente na população. O exemplo mais simples deste tipo de algoritmo é o de aglomeração, *crowding*, onde os descendentes competem com os pais por um lugar na população. Os operadores genéticos produzem indivíduos semelhantes aos progenitores, uma vez que partilham os seus genes, e a competição entre indivíduos semelhantes por um lugar na população promove a formação de aglomerados. A substituição de indivíduos semelhantes por outros com maior aptidão promove a evolução da população e preserva a sua diversidade genética.

```
MuCA (problem, popSize)
  MP = criar e avaliar popSize multi-indivíduos de problem
  repete
    par1, par2 = Selecionar dois indivíduos de MP
    child1, child2 = Recombinar os indivíduos par1 e par2
    child1, child2 = Mutar os indivíduos child1 e child2
    avaliar child1 e child2
    substituir (MP, par1, child1)
    substituir (MP, par2, child2)
   MP = redimensionar o número de cópias de MP
  até atingir o critério de paragem
fim
```

algoritmo 5-1 - Multiset Crowding Algorithm – MuCA

O *Multiset Crowding Algoritm*, MuCA, apresentado no algoritmo 5-1 é um algoritmo de *crowding* simples em que as populações tradicionais foram substituídas por multipopulações. Ele começa por gerar a multipopulação que irá conter *popsize* multi-indivíduos, onde todos têm genomas diferentes, por definição, e com o valor de cópias inicializado a 1. De seguida, o algoritmo entra no ciclo iterativo até que o critério de paragem seja atingido.

No processo iterativo, o algoritmo seleciona dois indivíduos da população para reprodução. A seleção dos indivíduos, *par1* e *par2*, da população tem de ter em conta o número de cópias do multi-indivíduos de forma a que este incremente a probabilidade de seleção. Esta operação tem de ser realizada sobre a lista de indivíduos que a multipopulação representa.

```
Substituir(MP , parent, child)
  se child for melhor ou igual que parent
      se MP não contiver child
        remover todos os parent de MP
      fim se
      adicionar child a MP
   fim se
fim
```

algoritmo 5-2 – Algoritmo de substituição de indivíduos por crowding em multipopulações

Os dois indivíduos selecionados geram dois descendentes, através dos operadores de recombinação e de mutação, que depois competem com os pais por um lugar na população. A substituição de um indivíduo numa multipopulação deve preservar as suas características invariáveis: o número de multi-indivíduos permanece constante, e os clones são introduzidos nos números de cópias. O algoritmo 5-2 apresenta a forma como dois indivíduos, *parent* e *child*, competem por um lugar na multipopulação, *MP*. De forma a promover a evolução da população, a substituição só ocorre se o descendente (*child*) for melhor ou igual ao progenitor (*parent*), quando comparados no valor da sua aptidão. A substituição dos progenitores pelos descendentes, quando possuem igual aptidão, premeia a novidade e promove a busca de novas soluções em problemas multimodais.

Se a substituição ocorrer, o descendente é sempre introduzido na população, mas pode não haver necessidade de eliminar o progenitor. Se o genoma do descendente já existir na população, a introdução do descendente é feita através do incremento do número de cópias do multi-indivíduo correspondente, e o progenitor continua na população. Mas, se o genoma do descendente não existir na população, o multi-indivíduo correspondente ao genoma do progenitor é eliminado e substituído por um

novo multi-indivíduo equivalente ao descendente. Neste caso, o descendente elimina o progenitor e todos os seus clones.

O algoritmo 5-2 incrementa o número de cópias dos multi-indivíduos, sempre que seja gerado um descendente melhor do que o progenitor e o genoma já esteja na multipopulação. Este número de cópias precisa ser controlado e, por essa razão, o MuCA termina o ciclo iterativo com o redimensionamento do número de cópias da multipopulação.

O algoritmo *crowding* tradicional (De Jong 1975)não fornece um desempenho satisfatório em problemas mais complexos devido a vários erros de substituição, pois os pais podem provir de nichos distintos, e os seus descendentes podem substituí-los mesmo sendo muito diferentes. De forma a tornar o algoritmo mais robusto, (Mahfoud 1995) propõe o algoritmo *deterministic crowding* , que faz emparelhamento dos progenitores com os descendentes de forma a que a competição seja feita entre os pais e filhos mais semelhantes, algoritmo 5-3:

```
d1 =distance(child1,par1) + distance(child2,par2)
d2 =distance(child1,par2) + distance(child2,par1)
se d1 < d2
  substituir (MP, par1, child1)
  substituir (MP, par2, child2)
senão
  substituir (MP, par1, child2)
  substituir (MP, par2, child1)
fim se
```

algoritmo 5-3 –Algoritmo de emparelhamento entre progenitores e descendentes – Deterministic Crowding

Para determinar a influência das multipopulações no algoritmo de *crowding*, fizemos as experiências que estão descritas na tabela 5-1. As experiências MuCA utilizam o algoritmo de *crowding* com multipopulações, e as experiências SymbCoev usam-no com populações simples. As experiências que têm o sufixo “d” no nome do algoritmo utilizam o algoritmo algoritmo 5-3 para fazer o emparelhamento entre os progenitores e os descendentes. As experiências com multipopulações utilizam 64 multi-indivíduos, e as populações simples utilizam populações de 64 e de 128 indivíduos:

Experiência	Descrição
MuCA-64	MuCA com uma multipopulação com 64 multi-indivíduos
MuCAd-64	MuCA com uma multipopulação com 64 multi-indivíduos e emparelhamento
SCA-64	SCA com uma população simples de 64 indivíduos
SCAd-64	SCA com uma população simples de 64 indivíduos e emparelhamento
SCA-128	SCA com uma população simples de 128 indivíduos
SCAd-128	SCA com uma população simples de 128 indivíduos e emparelhamento

tabela 5-1-Configuração base das experiências para avaliar a influência das multipopulações no processo evolutivo com algoritmos de crowding.

Os operadores genéticos utilizados para gerar a descendência são os mesmos que foram utilizados nas experiências anteriores. Para fazer a reprodução dos progenitores servimo-nos do operador de recombinação uniforme, *Uniform*(*probCrossover=1.0*, *probSwap = 0.5*), que é sempre aplicado aos progenitores, e a probabilidade de trocar um gene é de 50%. Para fazer a mutação, utilizamos a troca de bits, *Flipbit*(*probMutation= 1/128)*, que altera em média um gene por indivíduo.

As multipopulações utilizam o operador *AdaptiveCeiling*(*maxProportion*=1.5) para fazer o redimensionamento das cópias na multipopulação.

5.1.1 Funções Rugosas – NK 128-3-128

Os resultados da otimização da função NK 128-3-128 estão apresentados na tabela 5-2, e a figura 5-1 mostra a evolução dos parâmetros ao longo da evolução. Após 100.000 chamadas à função de avaliação, podemos verificar que a otimização da função NK 128-3, graças ao algoritmo de *crowding* e à utilização de multipopulações, é mais eficiente e eficaz do que o mesmo algoritmo aplicado às populações simples.

A experiência MuCA-64 conseguiu uma taxa de sucesso de 100%, e a MuCAd-64 apenas de 99%. A introdução do emparelhamento entre progenitores e descendentes na experiência MuCAd-64 tornou-a menos eficiente e menos eficaz. Menos eficiente, porque o emparelhamento aumentou a diversidade genética da multipopulação, que se reflete na velocidade de convergência, como podemos ver na figura 5-1 a), acontecendo ainda que uma das simulações não obteve sucesso durante o período evolutivo definido. A figura 5-1 d) mostra a evolução do número máximo de cópias que os multi-indivíduos possuem. Podemos verificar que, sem emparelhamento (MuCA-64) o número máximo de cópias é muito maior do que a simulação com emparelhamento (MuCA-64d) e, como

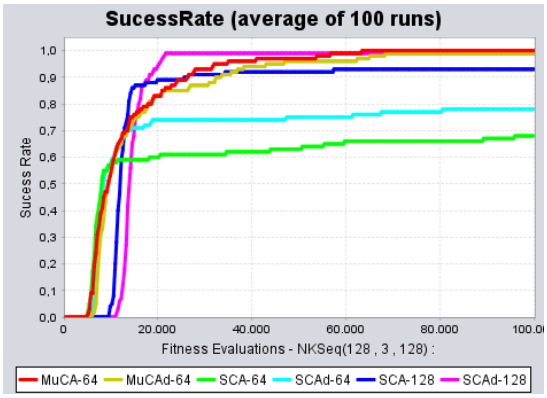
ambas as experiências têm o mesmo número de indivíduos (figura 5-1 c), as cópias encontram-se nos melhores, aumentando a sua probabilidade de serem selecionados. Na experiência MuCA-64d, a pressão seletiva está diluída por mais indivíduos.

	FuncsCallsToOptimum		SucessRate		Individuals		GeneticDiversity		Genotypes	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuCA-64	13164,6	10756,8	1,00	0,00	135,2	2,1	0,062	0,002	64,0	0,0
MuCAd-64	14333,0	12574,6	0,99	0,10	135,6	4,5	0,069	0,005	64,0	0,0
SCA-64	8683,2	16018,6	0,68	0,47	64,0	0,0	0,000	0,002	1,0	0,4
SCAd-64	8980,7	12344,0	0,78	0,42	64,0	0,0	0,015	0,008	3,5	1,5
SCA-128	12024,1	6500,5	0,93	0,26	128,0	0,0	0,000	0,000	1,0	0,0
SCAd-128	14244,1	2637,1	0,99	0,10	128,0	0,0	0,016	0,006	5,9	3,4

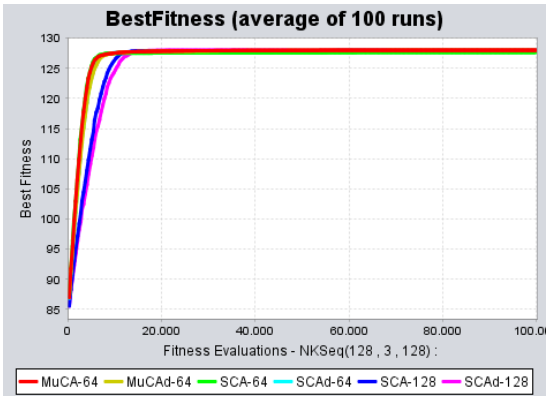
tabela 5-2- Resultado da otimização da função NK 128-3-128 com crowding após 100.000 chamadas à função de avaliação

Nas experiências com populações simples, verificamos que o emparelhamento entre pais e filhos foi benéfico e isso reflete-se na taxa de sucesso obtido. Esta melhoria da eficácia pode ser explicada pela diversidade genética apresentada pelas populações, figura 5-1 e), onde podemos verificar que a diversidade genética das populações sem emparelhamento cai para o valor mínimo quando existe apenas um genótipo na população, figura 5-1 c). O emparelhamento introduz e mantém uma maior diversidade genética e, no final das simulações, a experiência SCAd-64 tem uma população com 3.5 genomas diferentes, ao passo que a experiência SCAd-128 tem 5.9. Estes valores estão muito distantes dos 64 genomas diferentes que as multipopulações apresentam e que são responsáveis pelo seu bom desempenho.

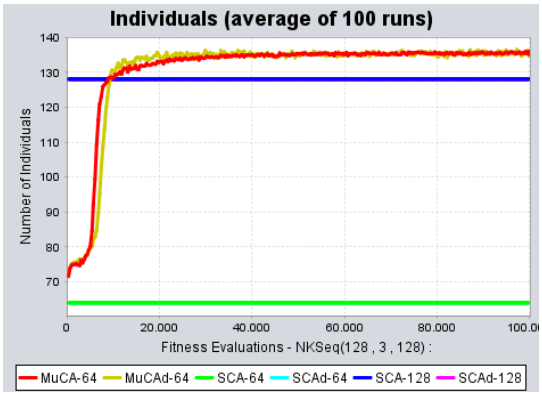
A figura 5-1 f) mostra o número de genótipos diferentes da população ao longo do processo evolutivo, e a figura 5-1 e) a diversidade genética da população. O algoritmo de *crowding* apenas consegue manter a diversidade genética na população durante algumas gerações, após as quais os erros de substituição levam à deriva genética da população. Quando um par de indivíduos gera um descendente mais apto que eles existindo já um genoma na população, um dos pais é substituído por outro genoma da população, e a quantidade de genomas decresce. Assim, o tamanho da população maior atrasa a deriva genética, e o emparelhamento abranda-a.



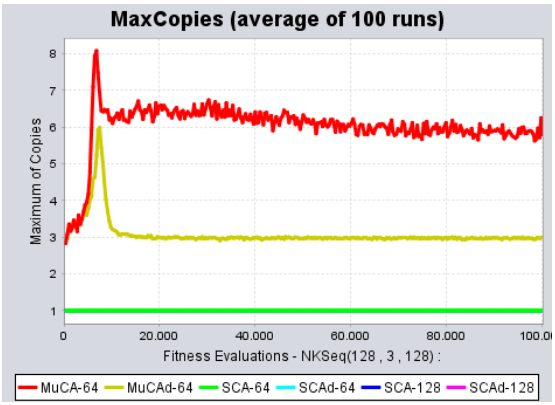
a)



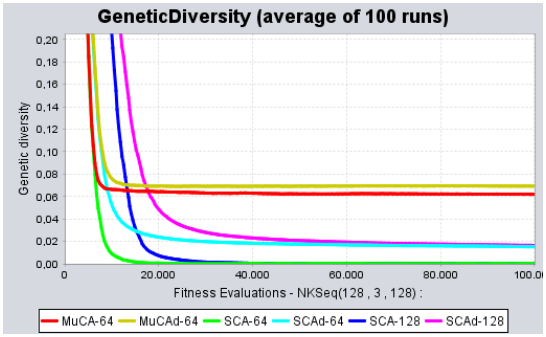
b)



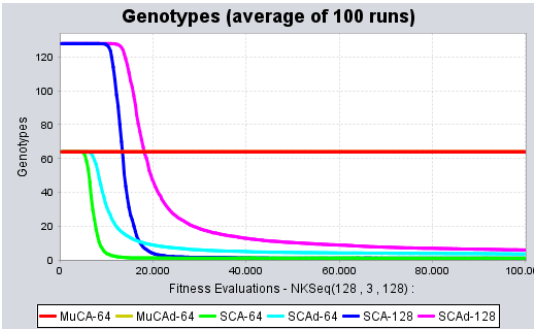
c)



d)



e)



f)

figura 5-1 Crowding na função NK 128-3: Evolução dos parâmetros genéticos ao longo de 100.000 avaliações: a) taxa de sucesso; b)aptidão do melhor indivíduo; c)número de indivíduos; d)número máximo de cópias dos multi-indivíduos; e)diversidade genética da população; d)número de genomas distintos na população;

Nas multipopulações, estes erros de substituição não existem porque, quando o genoma do descendente já existe na população, o multi-indivíduo correspondente ganha mais uma cópia e o progenitor não é eliminado, pois o conjunto de suporte da multipopulação mantém-se constante. Isto traz duas vantagens: o aumento da probabilidade de seleção dos bons indivíduos, através do incremento do número de cópias, e o aumento da diversidade genética pela não eliminação do progenitor.

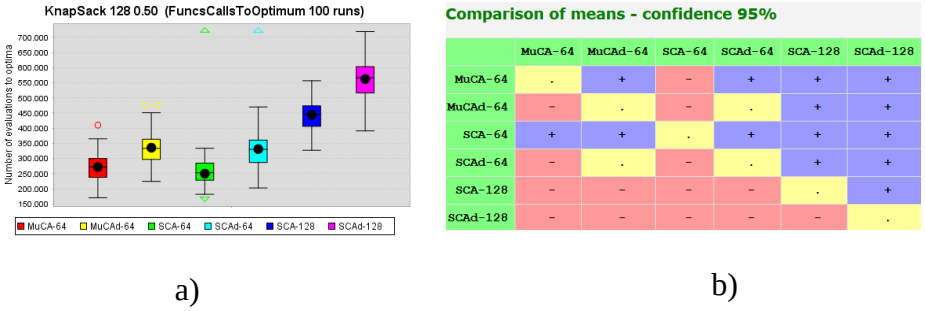
5.1.2 Funções com vários ótimos – Knapsack 128 0.5

A tabela 5-3 mostra o resultado da otimização da função Knapsack 128 0.5, após 1 milhão de chamadas à função de avaliação, e a figura 5-3 apresenta a evolução das estatísticas ao longo do processo evolutivo. Todas as experiências foram bem-sucedidas, exceto a SCA-64, que obteve sucesso em 97% das simulações.

	FuncsCallsToOptimum		SucessRate		Individuals		Genotypes		NumOptimaFound		NumOptimalGenomesPop	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuCA-64	271901,6	46401,5	1,00	0,00	134,2	4,9	64,0	0,0	390,5	84,2	63,8	0,4
MuCAd-64	335783,8	52772,1	1,00	0,00	134,2	5,1	64,0	0,0	265,0	73,8	59,2	5,7
SCA-64	249717,3	58893,0	0,97	0,17	64,0	0,0	2,1	2,0	46,7	40,8	2,1	2,0
SCAd-64	331200,6	53941,0	1,00	0,00	64,0	0,0	53,3	4,2	222,4	74,0	49,6	6,6
SCA-128	444716,6	44845,2	1,00	0,00	128,0	0,0	29,2	13,3	154,2	56,6	29,1	13,2
SCAd-128	562481,9	65506,6	1,00	0,00	128,0	0,0	120,8	4,1	115,1	37,5	56,0	13,3

tabela 5-3- Resultado da otimização da função Knapsack 128-05 com crowding após 1.000.000 chamadas à função de avaliação.

Dentro das experiências que tiveram 100% de sucesso, a MuCA-64 foi a mais eficiente, e esta diferença é estatisticamente significativa, como se pode ver na figura 5-2. A experiência SCA-64 necessita de menos chamadas à função de avaliação, mas a deriva genética da população não permitiu que tivesse sucesso pleno. O emparelhamento de progenitores com os descendentes tornou a experiência SCAd-64 imune à deriva genética e fez com que esta simulação fosse completamente bem-sucedida. Nas multipopulações, o emparelhamento teve um efeito negativo na eficiência do algoritmo. A simulação MuCAd-64 é menos eficiente do que a simulação SCAd-64, embora esta diferença não seja estatisticamente significativa, figura 5-2.



a) b)

figura 5-2 Knapsack-128 0.5 - Número de chamadas à função de avaliação para obter o ótimo com crowding– 100 simulações: a) gráfico de boxplot*; b) Resultado do teste de hipóteses t-test*

A figura 5-3 c) apresenta a evolução da diversidade genética da população, e a figura 5-3 d) a quantidade de genótipos diferentes que a população contém. Em todas as experiências, a diversidade genética estabiliza em valores que permitem gerar

descendentes diferentes dos progenitores, o que explica a eficácia das simulações. A experiência SCA-64 não teve sucesso em 3% das experiências devido à falta de diversidade genética da população, com apenas 2.1 genótipos diferentes, em média no final das simulações.

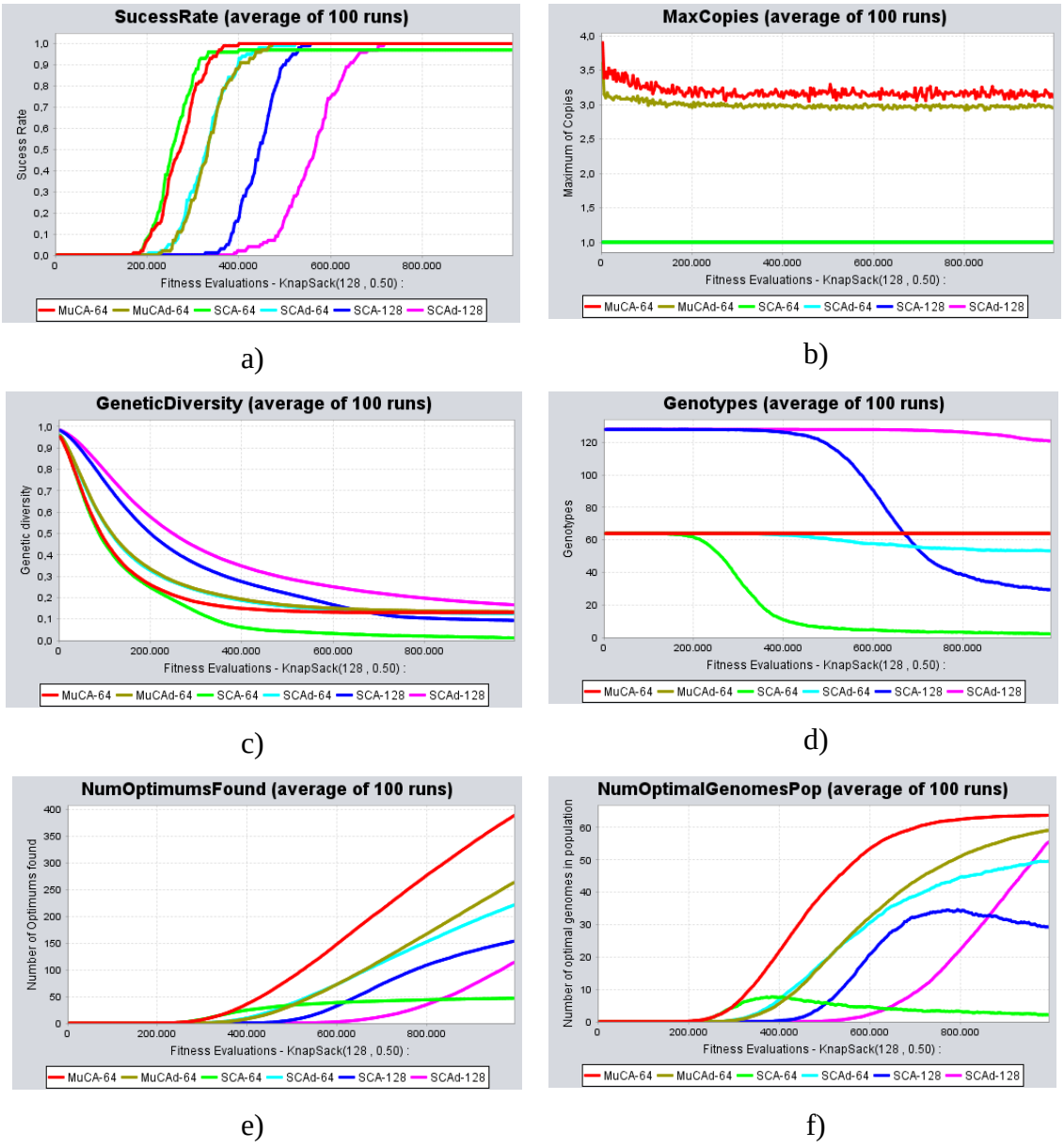


figura 5-3 Crowding na função Knapsack 128-0.5: Evolução dos parâmetros genéticos ao longo de 1.000.000 avaliações: a) taxa de sucesso; b) número de indivíduos na população; c) diversidade genética da população; d) número de genomas distintos na população; e) número de ótimos encontrados; f) número de ótimos distintos na população

A figura 5-3 e) mostra a quantidade de ótimos que foram encontrados ao longo do processo evolutivo, sendo que todas as simulações bem-sucedidas encontraram vários ótimos. As simulações com multipopulações obtiveram mais ótimos do que as

populações simples, e a sua tendência para obter novos ótimos é mais acentuada do que as experiências baseadas em populações simples.

Outra característica apresentada pelas multipopulações foi a capacidade de retenção de ótimos na população. Como a função tem milhares de ótimos, as multipopulações, depois de os encontrarem, conseguem retê-los até serem substituídos por novos ótimos, por força do operador de substituição de indivíduos, definido no algoritmo 5-2, que dá vantagem aos descendentes em caso de empate como os progenitores.

A experiência SCA-64, apesar de ter obtido pelo menos um ótimo em 97% das vezes, conseguiu obter 40,2 ótimos, mas manteve na população apenas uma média de 2,1. O emparelhamento da população na simulação SCAd-64 prejudicou a eficiência na obtenção dos ótimos, por força da diversidade genética introduzida, mas tornou a população mais robusta, tornando-a mais eficaz na descoberta de ótimos e na capacidade de os reter na população.

A simulação MuCAd-64, que utiliza emparelhamento, encontrou menos ótimos do que a simulação MuCA-64, e afetou a sua tendência de crescimento. A figura 5-7 b) mostra que o número máximo de cópias dos multi-indivíduos não é muito maior na simulação MuCA-64, mas é o suficiente para a tornar mais eficiente.

5.2 Influência das multipopulações no algoritmo de clearing

A partilha de recursos baseia-se na ideia de que os indivíduos que ocupam uma região dividem entre si os recursos disponíveis, sendo o valor da aptidão dos indivíduos proporcional à quantidade de recursos consumidos. Desta forma, o agrupamento de indivíduos num nicho é penalizado, e as regiões do espaço densamente ocupadas tornam-se pouco atrativas. A partilha do valor de aptidão por vários indivíduos necessita de uma métrica de similitude, para fazer o agrupamento, e de uma função para fazer a distribuição do valor da aptidão. A determinação do nicho implica a definição de um raio, mas a definição da função de avaliação levanta alguns problemas, nomeadamente de escala.

O método *clearing* permite contornar algumas das dificuldades anteriores, através da atribuição dos recursos aos melhores indivíduos do nicho, em vez de o partilhar com todos. Os indivíduos dominantes eliminam os indivíduos dominados através da supressão do seu valor de aptidão.

O algoritmo 5-4 apresenta a adaptação do algoritmo de *clearing* proposto por (Petrowski 1996) para multipopulações. O algoritmo original possui como parâmetros

a capacidade de cada nicho e o seu raio de limpeza (*clearing radius*). O algoritmo forma um conjunto de nichos através da eliminação dos indivíduos mais fracos e cada nicho pode ter mais do que um vencedor. O algoritmo produz um número variável de indivíduos vencedores e, portanto, não pode ser utilizado diretamente como operador de substituição em multipopulações.

```
ClearingReplacement(parents, offspring)
  allPop = parents + offspring
  ordenar allPop por ordem descendente
  newPop = população vazia
  enquanto newPop.genotypes < parents.genotypes
    winner = allPop.removeFirst()
    newPop.add(winner)
    se newPop.size + allPop.size > parents.size
      looser = individuo mais semelhante a winner
      allPop.remove(looser)
    fim se
  fim enquanto
  retornar newPop
fim
```

algoritmo 5-4 - Algoritmo ClearingReplace

Como as multipopulações necessitam de manter constante a dimensão do conjunto de suporte, o algoritmo 5-4 aplica uma variante do algoritmo original, que seleciona um conjunto constante de indivíduos, mantendo a noção de nicho. O algoritmo é mais simples do que original e não necessita de parâmetros.

O operador *ClearingReplace* recebe como parâmetros a população principal (*parents*) e a dos descendentes (*offspring*), que vai ser incorporada na principal. O algoritmo começa por juntar as duas populações (*allPop*). Se a população for simples os clones devem ser eliminados, se for uma multipopulação os clones são incorporados no número de cópias. Esta forma de juntar as duas populações permitem que as populações simples tenham sempre uma quantidade fixa de genomas diferentes, tal como acontece nas multipopulações.

O algoritmo continua com a ordenação da população *allPop* por ordem decrescente da sua aptidão. Se o algoritmo de ordenação for estável, isto é preservar a ordem de elementos iguais, devem-se baralhar os indivíduos em *allPop* de forma a que os indivíduos com igual aptidão ocupem posições aleatórias quando forem ordenados.

O algoritmo entra num ciclo iterativo que permite escolher um conjunto fixo de elementos (*parents.genotypes)* que irão passar para a geração seguinte. Estes elementos correspondem a indivíduos, no caso das populações simples, e a multi-indivíduos, no caso das multipopulações. Em cada iteração o melhor indivíduo, *winner*, está no topo da lista, porque está ordenada, e esse elemento passa direto para a próxima geração. Este passo é repetido *parents.genotypes* de forma a completar a nova geração. Se for necessário, é removido da população *allPop* o elemento geneticamente mais parecido com o *winner*.

O algoritmo de *ClearingReplace* foi desenvolvido para aumentar o desempenho dos algoritmos genéticos em funções multimodais, e nas secções seguintes, apresentamos os resultados da sua aplicação a multipopulações. O algoritmo 5-4 garante que um conjunto fixo de genótipos bem-adaptados sejam escolhidos para continuar o processo evolutivo. Este operador tem a virtude de transportar para as populações simples uma das características mais marcante das multipopulações: a manutenção de um conjunto constante de genomas distintos. Com a utilização deste operador nos algoritmos genéticos, a diferença entre os dois tipos de populações resume-se ao facto de os multi-indivíduos acomodarem os clones no número de cópias.

As experiências seguintes utilizam multipopulações com 64 multi-indivíduos, que representam aproximadamente 128 indivíduos simples através do seu número de cópias, e populações simples, que representam 64 e 128 indivíduos com genomas diferentes, ou seja, sem clones. Essas experiências evidenciam a influência da utilização de clones no processo evolutivo.

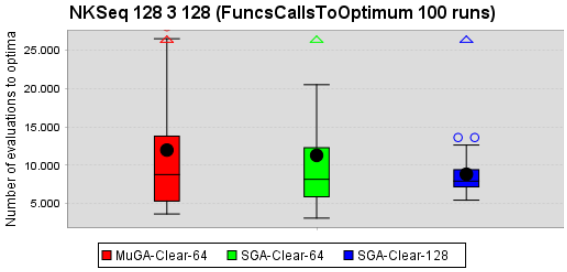
5.2.1 Funções Rugosas – NK 128-3-128

A tabela 5-4 apresenta o resultado da otimização da função NK 128-3, após 100.000 chamadas à função de avaliação; a figura 5-4 mostra o resultado estatístico da comparação entre as médias do número de chamadas à função de avaliação para obter um ótimo; e, finalmente, a figura 5-5, a evolução dos parâmetros genéticos ao longo do processo evolutivo.

	FuncsCallsToOptimum		SucessRate		Individuals		GeneticDiversity		Genotypes	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-Clear-64	11977,0	10058,0	1,00	0,00	135,4	5,6	0,061	0,002	64,0	0,0
SGA-Clear-64	11291,8	10530,5	1,00	0,00	64,0	0,0	0,075	0,003	64,0	0,0
SGA-Clear-128	8827,0	2836,8	1,00	0,00	128,0	0,0	0,459	0,033	128,0	0,0

tabela 5-4- Resultado da otimização da função NK 128-3-128 com clearing após 100.000 chamadas à função de avaliação.

Todas as experiências tiveram sucesso pleno, sendo que as mais eficientes foram as que utilizam populações simples e a experiência SGA-Clear-128 é estatisticamente melhor do que as restantes, como se pode ver na figura 5-4.



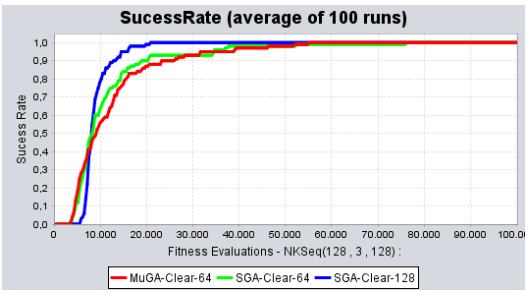
a)

Comparison of means - confidence 95%

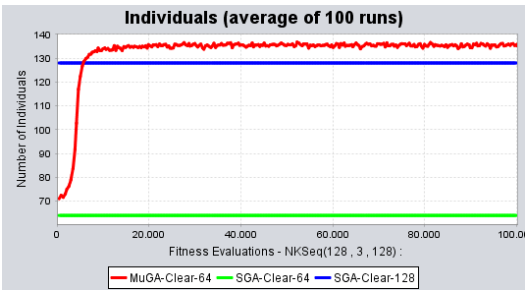
	MuGA-Clear-64	SGA-Clear-64	SGA-Clear-128
MuGA-Clear-64	.	.	-
SGA-Clear-64	.	.	-
SGA-Clear-128	+	+	.

b)

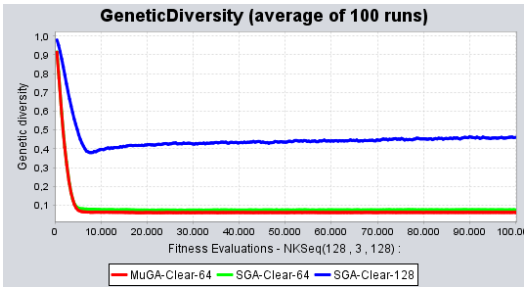
figura 5-4 Clearing na função NK-128-3-128: número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot; b)resultado do teste de hipóteses



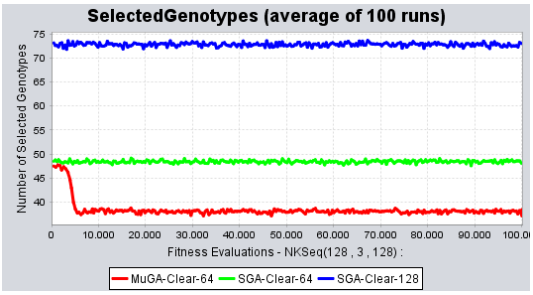
a)



b)



c)



d)

figura 5-5 Clearing na função NK 128-3; Evolução dos parâmetros genéticos ao longo de 1.000.000 avaliações: a) Taxa de sucesso; b) Número de indivíduos na população; c) Diversidade genética da população; d) Número de genomas selecionados para reprodução.

Na otimização desta função, os clones presentes na experiência MuGA-Clear-64 prejudicaram o seu desempenho na obtenção do único ótimo. A perda da diversidade genética introduzida pelo número de cópias (figura 5-5 c), traduz-se numa menor quantidade de genomas distintos selecionados para reprodução (figura 5-5 d), o que explica o inferior desempenho da simulação com multipopulações. A função NK 128-3-128 é uma função rugosa, com muitos extremos locais, e os multi-indivíduos acumulam cópias nesses máximos (figura 5-5 b) logo nas primeiras fases do processo evolutivo. A pressão seletiva feita por estes clones faz com que o algoritmo MuGA-Clear-64 tenha um desempenho ligeiramente inferior ao SGA-Clear-64. O bom desempenho da experiência SGA-Clear-128 deve-se à elevada diversidade genética da população (figura 5-5 c), o que lhe permite selecionar para reprodução um conjunto maior de genomas (figura 5-5 d), e ultrapassar os extremos locais na obtenção do ótimo.

5.2.2 Funções com vários ótimos – Knapsack 128 0.5

A tabela 5-5 apresenta o resultado da otimização da função Knapsack 128-05, após 1 milhão de chamadas à função de avaliação; a figura 5-6 mostra o resultado estatístico da comparação entre as médias do número de chamadas à função de avaliação para obter um ótimo; e, finalmente, a figura 5-7, a evolução dos parâmetros genéticos ao longo do processo evolutivo.

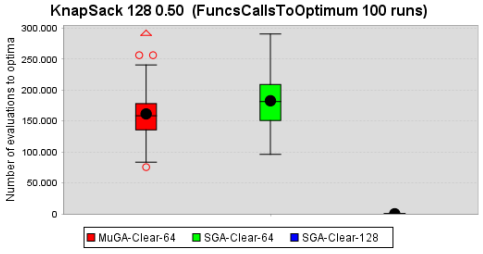
	FuncsCallsToOptimum		SucessRate		Individuals		Genotypes		NumOptimaFound		NumOptimalGenomesPop	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-Clear-64	161491,9	39743,1	1,00	0,00	129,7	4,0	64,0	0,0	397,0	53,8	16,7	2,4
SGA-Clear-64	182704,8	39808,0	1,00	0,00	64,0	0,0	64,0	0,0	271,7	35,5	11,6	2,8
SGA-Clear-128	0,0	0,0	0,00	0,00	128,0	0,0	128,0	0,0	0,0	0,0	0,0	0,0

tabela 5-5- Resultado da otimização da função Knapsack 128-0.5 com Clearing após 1000000 chamadas à função de avaliação.

As experiências com populações de dimensão 64 tiveram sucesso pleno, enquanto a experiência SGA-Clear-128 não conseguiu obter qualquer ótimo. A experiência com multipopulações foi mais eficiente do que a SGA-Clear-64, e esta diferença é estatisticamente significativa, como se pode verificar na figura 5-6 b). Foi omitida a experiência SGA-Clear-128 na (figura 5-6), porque não convergiu.

Na otimização desta função, os clones tiveram um efeito positivo, e as experiências com multipopulações não só foram mais eficientes (figura 5-7 a), como conseguiram obter mais ótimos (figura 5-7 e), e permitiram que estes permanecessem na população (figura 5-7 f).

A menor diversidade genética da população (figura 5-7 c) que se traduz numa menor quantidade de genomas selecionados (figura 5-7d), deve-se ao facto de a população ter clones. Como a função possui vários ótimos, estes clones fornecem uma pressão seletiva positiva para acelerar a convergência. Depois de encontrar o primeiro ótimo, os clones favorecem a exploração do espaço na busca dos restantes.



a)

Comparison of means - confidence 95%

	MuGA-Clear-64	SGA-Clear-64
MuGA-Clear-64	.	+
SGA-Clear-64	-	.

b)

figura 5-6 Clearing na função Knapsack-128; Número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot*; b) Resultado do teste de hipóteses t-test*

A experiência SGA-Clear-128 não convergiu devido ao facto de possuir uma diversidade genética demasiado elevada, pois os indivíduos estão espalhados pelos inúmeros máximos, cuja recombinação não permite evoluir. Todos os algoritmos selecionam 128 indivíduos para reprodução, cujos descendentes vão ser incorporados na população inicial. Nas populações com 64 elementos, a seleção dos 128 descendentes incrementa a pressão sobre os bons indivíduos da população principal, devido à sua menor dimensão. Os resultados experimentais demonstram que a pressão dos descendentes sobre a população principal é necessária para o bom funcionamento do operador em populações simples. As multipopulações são mais robustas a este parâmetro por causa da pressão seletiva introduzida pelo número de cópias dos multi-indivíduos.

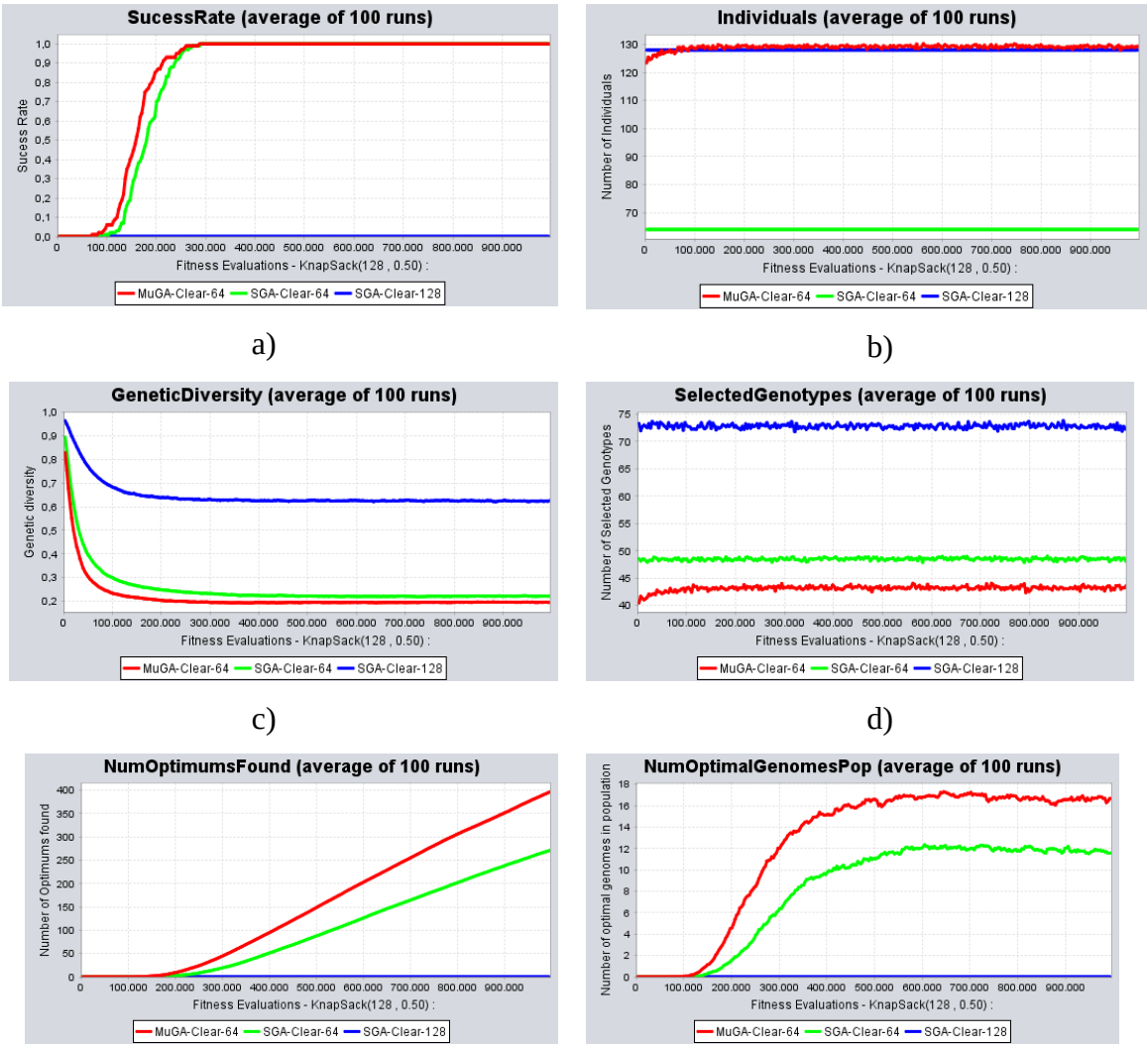


figura 5-7 Knapsack 128-0.5: Evolução dos parâmetros genéticos ao longo de 1.000.000 avaliações: a) Taxa de sucesso; b) Número de indivíduos na população; c) Diversidade genética da população; d) Número de genomas na população; e) Número de ótimos encontrados; f) Número de ótimos na população.

5.3 Comparação dos algoritmos evolutivos com multipopulações

O operador de substituição de geração por RTS (G. Harik 1995), *Restrictive Tournament Selection*, pode ser implementado no MuGA fazendo uma extensão ao operador *MTournamentReplacement*. O Operador *RestrictiveReplacement* faz a competição de cada descendente com um grupo de progenitores selecionados aleatoriamente na população principal. No grupo selecionado é escolhido o indivíduo geneticamente mais próximo, e este é substituído, caso seja o pior. No caso de ser uma multipopulação, e se o genoma do descendente já estiver na multipopulação, não é necessário fazer a eliminação, e o multi-indivíduo correspondente ganha mais uma cópia.

Os resultados do operador *RestrictiveReplacement* na otimização dos problemas *NK-Landscapes* e *Knapsack*, descritos anteriormente, estão representados no Anexo D, e o seu desempenho é semelhante aos apresentados nesta seção.

As simulações anteriores demonstraram que a utilização das multipopulações torna os algoritmos evolutivos mais eficazes, e incrementa o seu sucesso evolutivo em diversos aspetos. Um dos aspetos que julgamos mais importantes é a preservação da diversidade genética da multipopulação, que evita a convergência prematura, tornando o uso do operador de recombinação eficaz. Esta característica faz com que a multipopulação encontre as melhores soluções e, depois de o fazerem, as consigam manter na população.

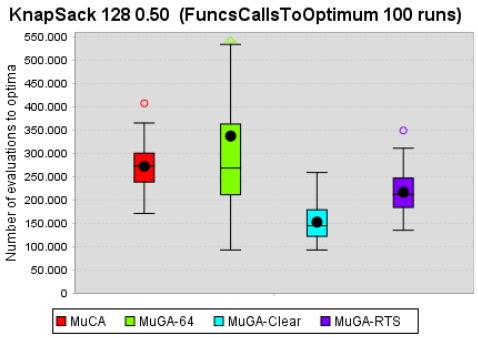
A tabela 5-6 apresenta o resultado da otimização da função *KnapSack 0.5* com os algoritmos evolutivos acima descritos no final de 2.500.000 chamadas à função de avaliação.

	FuncsCallsToOptimum		GeneticDiversity		NumOptimalGenomesPop		NumOptimumsFound	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	DesvPad	Média	Desv.Pad.
MuCA	271901,6	46401,5	0,13	0,00	64,000	0,000	885,7	69,7
MuGA	336795,4	258928,1	0,11	0,00	62,780	8,583	370,7	126,2
MuGA-Clear	152042,0	36909,5	0,17	0,01	13,830	3,282	701,1	53,8
MuGA-RTS	216255,8	41455,1	0,14	0,01	62,800	0,791	1008,8	54,0

tabela 5-6- Resultado da otimização da função Knapsack 128-0.5 com multipopulações após 2.500.000 chamadas à função de avaliação.

Todas as simulações têm uma taxa de sucesso de 100%, e a figura 5-8 apresenta os resultados estatísticos do número de chamadas à função de avaliação para obter o primeiro ótimo.

A simulação *MuGA-Clear*, que utiliza o operador *ClearingReplace*, é a simulação mais eficiente, e o seu resultado é estatisticamente melhor que as restantes, como se pode ver na figura 5-8 b). Em sentido contrário, a simulação MuGA com o operador *MTournamentReplacement* é a que tem pior desempenho, pois não tem implementa nenhuma estratégia de preservação da diversidade genética da população para além daquela que advém de usar a multipopulação. Este facto pode ser visualizado na figura 5-9 b), que mostra a evolução da diversidade genética durante o processo evolutivo. Os algoritmos que implementam estratégias de nicho têm uma diversidade genética superior, o que contribui para a descoberta do ótimo mais cedo figura 5-9 b).



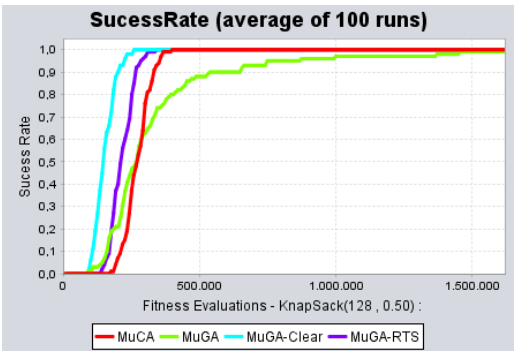
a)

Comparison of means - confidence 95%

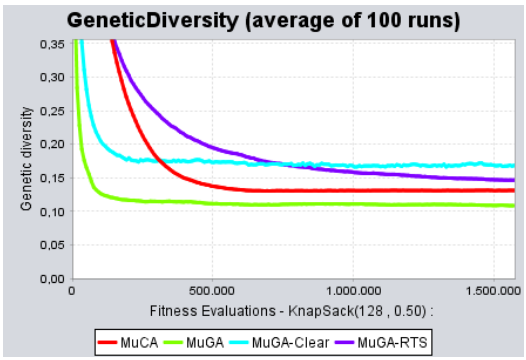
	MuCA	MuGA	MuGA-Clear	MuGA-RTS
MuCA	.	+	-	-
MuGA	-	.	-	-
MuGA-Clear	+	+	.	+
MuGA-RTS	+	+	-	.

b)

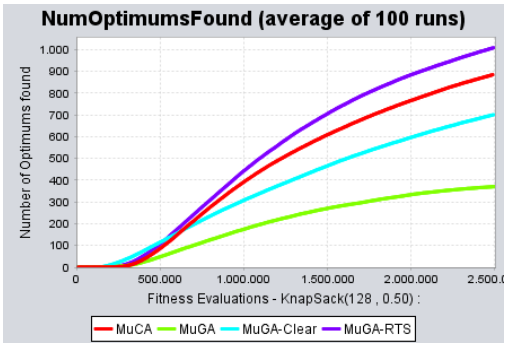
figura 5-8 Multipopulações na função Knapsack-128 0.5; Número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot*; b) Resultado do teste de hipóteses t-test*



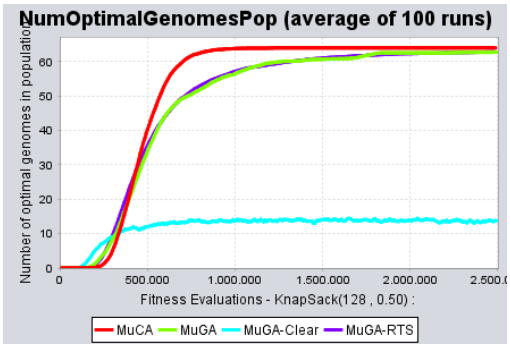
a)



b)



c)



d)

figura 5-9 Knapsack 128-0.5: Evolução dos parâmetros genéticos ao longo de 2.500.000 avaliações: a) Taxa de sucesso; b) Diversidade genética população; c) Número de ótimos encontrados; d) Número de ótimos na população.

Todos os algoritmos conseguiram obter vários ótimos e conseguiram-nos manter na população. De salientar a capacidade do operador *RestrictiveReplacement* no MuGA, MuGA-RTS, na descoberta de cerca de mil ótimos, em média, com uma população de 64 multi-indivíduos, e a sua capacidade de os conservar na população. Este resultado, que também pode ser visto com menor intensidade nas restantes simulações, mostra uma boa capacidade exploratória das multipopulações.

5.4 Comentários finais

Neste capítulo, adaptámos o uso das multipopulações para algoritmos de nicho, que permitem a manutenção da diversidade genética em populações simples. Definimos o procedimento para a substituição de um indivíduo dentro de uma multipopulação, para ser utilizado em algoritmos evolutivos de estado estacionário. Utilizando o procedimento anterior, definimos o *Multiset Crowding Algorithm*, MuCA, um algoritmo de estado estacionário que utiliza multipopulações. Fizemos ainda a definição do MuCAd, que utiliza um procedimento determinístico, *deterministic crowding*, para o emparelhamento da competição dos pais com os descendentes, por forma a evitar erros de substituição. Este emparelhamento em multipopulações não trouxe quaisquer benefícios, uma vez que o conjunto de suporte da multipopulação já fornece um mecanismo que previne a convergência prematura; no entanto, é necessário para os algoritmos que utilizam populações simples. Na otimização das funções o algoritmo MuCA tem melhor desempenho do que os algoritmos que utilizam populações simples.

Definimos um operador de substituição, *ClearingReplacement*, inspirado na formação de espécies dentro de multipopulações através da partilha de recurso. O operador não precisa de parâmetros para ser utilizado nas multipopulações e permite que as populações simples conservem, efetivamente, a diversidade genética da população. Tal diversidade, introduzida pelo operador, possibilita que as populações simples sejam simultaneamente eficientes e eficazes, desde que o algoritmo genético seja corretamente configurado. Mesmo assim, as multipopulações são mais robustas e menos sensíveis aos parâmetros do algoritmo genético. Nas simulações com o *ClearingReplacement*, o desempenho superior das multipopulações só pode ser atribuído ao número de cópias dos multi-indivíduos, uma vez que os dois tipos de população armazenam um conjunto de genomas únicos.

A utilização de multipopulações na otimização de funções multimodais permitem que os algoritmos descubram mais ótimos e que os mantenham na população.

As multipopulações mostraram um excelente desempenho na otimização dos problemas NK-Landscapes e Knapsack e é nossa convicção as propriedades exploratórias, introduzidas pelos genomas todos diferentes, e o aproveitamento introduzido pela pressão seletiva do número de cópias, se mantêm na otimização de outros tipos de problemas.

6 Otimização de problemas codificados por números reais

Os problemas de otimização codificados por números reais têm sido foco de investigação nos anos mais recentes, devido à necessidade em obter melhores soluções para um conjunto alargado de problemas. O problema geral de otimização pode ser matematicamente descrito por pela seguinte equação:

$$minimizar\ f(x), \forall x \in S \tag{6.1}$$

A função f, equação 6.1, é uma função real que mapeia o vetor $x = [x_1,x_2,\ldots,x_n]$, composto por n variáveis reais definidas no espaço S em $\mathcal{R}$. Cada componente x_i pode ter valores do intervalo S_i definido em $[l_i , u_i]$, onde l_i representa o limite inferior do intervalo e u_i o limite superior. O objetivo do processo de otimização é encontrar o vetor x^* que minimize o valor de f. Os problemas de maximização seguem uma formulação idêntica à definida na equação 6.1.

A resolução destes problemas envolve a otimização de vetores multidimensionais, cuja complexidade do espaço de procura cresce exponencialmente com o número de variáveis a otimizar. Os algoritmos evolutivos são uma das estratégias utilizadas, pela sua capacidade em lidar com problemas de grandes dimensões e de diversos tipos. Os mais bem-sucedidos representam os genes através de números reais e utilizam operadores genéticos adaptados a esta representação.

A utilização de multipopulações com os operadores genéticos tradicionais não traz qualquer vantagem significativa ao processo evolutivo, uma vez que os operadores genéticos utilizam distribuições de probabilidade, e a geração de indivíduos iguais aos seus progenitores é praticamente nula. A formação de multi-indivíduos é composta apenas pelos indivíduos que não sofreram modificações pelos operadores genéticos, ou seja, são clones dos progenitores, e são reintroduzidos na população principal incrementando o número de cópias. Os indivíduos mais aptos são tendencialmente selecionados para reprodução e, ao serem reintroduzidos na população, incrementam a probabilidade de nova seleção do genótipo. Todavia, quando aplicado a problemas difíceis e de dimensionalidade elevada, esta vantagem não é suficiente para promover de forma significativa o sucesso do algoritmo.

Nos capítulos anteriores, os operadores genéticos de recombinação e mutação utilizavam os multi-indivíduos como uma lista de clones e aplicavam os operadores tradicionais sobre eles. No entanto, a informação sobre o número de cópias de um genoma selecionado para reprodução é uma indicação indireta do valor da sua aptidão:

se o genoma foi selecionado muitas vezes é porque está bem-adaptado. Esta informação sobre o número de cópias de um genoma pode ser utilizada para desenvolver operadores genéticos adaptados a multi-indivíduos, no intuito de incrementar o sucesso da utilização das multipopulações em algoritmos evolutivos.

6.1 Adaptação dos operadores genéticos para multipopulações

Um dos problemas dos algoritmos evolutivos é a perda de diversidade genética pela aplicação repetida dos operadores genéticos a indivíduos, por norma bem-adaptados, principalmente nas fases mais avançadas do processo evolutivo. Nos espaços contínuos, a diversidade genética introduzida pelas multipopulações, através de um conjunto de genótipos diferentes, é reduzida devido ao número potencialmente infinito de soluções que os operadores genéticos conseguem produzir e à proximidade das soluções. Por forma a contornar este problema, redefinimos os operadores genéticos para operar sobre multi-indivíduos, tirando partido do facto de terem genomas diferentes e de possuírem um número de cópias que reflete indiretamente a sua aptidão.

6.1.1 Seleção para reprodução

Na população principal, o número de cópias de um multi-indivíduo tem o propósito de incrementar a probabilidade de seleção do genoma, e qualquer operador de seleção sobre multipopulações deve ter em conta esse facto. Os indivíduos mais aptos são selecionados várias vezes, e estes são armazenados no número de cópias dos multi-indivíduos correspondentes. Tendencialmente, os genomas mais bem-adaptados da população irão gerar multi-indivíduos com várias cópias na população selecionada para reprodução, e esta característica vai ser explorada pelos operadores genéticos de recombinação e de mutação.

6.1.2 Recombinação

Os operadores convencionais de recombinação sobre números reais operam sobre dois ou mais progenitores para gerarem um conjunto de descendentes, por regra igual ao número de progenitores. Esses operadores de recombinação sobre multi-indivíduos devem ter em consideração que cada multi-indivíduo representa um conjunto de indivíduos, clones, e, para manter o paralelismo com os seus homólogos, devem gerar um número de indivíduos igual dos progenitores.

O algoritmo 6-1 representa o algoritmo canónico de recombinação de dois multi-indivíduos, *mI1* e *mI2*, e devolve uma lista com os descendentes, *descendants*, cujo

número é igual ao dos clones que os multi-indivíduos representam. O algoritmo produz um número de iterações igual ao máximo de cópias dos progenitores, *maxItera*, aproveitando os descendentes, se a iteração for menor ou igual ao número de cópias do respetivo multi-indivíduo.

```
MultisetRecombination (mI1, mI2)
  descendants = lista vazia
  copies1 = mI1.copies
  copies2 = mI2.copies
  maxItera = máximo entre copies1 e copies2
  para itera de 1 até maxItera
     mI1.copies = mínimo entre itera e copies1
     mI2.copies = mínimo entre itera e copies2
     child1, child2 = Recombinar( mI1 , mI2 )
     se itera <= copies1
        adicionar child1 a descendants
     se itera <= copies2
        adicionar child2 a descendants
  próximo
  retornar descendants
fim
```

algoritmo 6-1 – Algoritmo canónico de recombinação de dois multi-indivíduos

A primeira iteração do operador de recombinação de multi-indivíduos é a aplicação do operador de recombinação convencional, pois altera o número de cópias para o valor unitário. Nas iterações subsequentes, o número de cópias é incrementado, e este valor vai influenciar a geração dos descendentes.

O operador de recombinação dos progenitores, *Recombinar*, deve ser um algoritmo que tire partido da informação do número de cópia associado a cada multi-indivíduo para a geração dos dois descendentes, *child1*, *child2*. Ora, a extensão do algoritmo 6-1 para operadores que necessitem de um número superior a dois indivíduos para ser aplicado é um processo trivial.

Os operadores de recombinação para indivíduos codificados por números reais permitem gerar descendentes que podem fazer simultaneamente a exploração e o

aproveitamento dos genes dos progenitores, figura 6-1, através da expansão do espaço por um fator α.

figura 6-1 Aproveitamento e exploração no operador de recombinação de indivíduos codificados por números reais.

Habitualmente os operadores de recombinação utilizam os valores expandidos, p1' e p2', para gerar os descendentes. Se os descendentes gerados estiverem entre os valores originais, p1 e p2, promovem o aproveitamento e a convergência do algoritmo; se forem gerados fora do intervalo delimitado pelos genes dos progenitores, incrementam a exploração. A performance de um operador de recombinação num problema é determinada pela combinação da capacidade de explorar novas regiões do espaço de procura, mas também pelo aproveitamento dos bons genes gerados pelo processo evolutivo (Herrera, Lozano, e Sanchez 2003).

Nas populações tradicionais, não existe qualquer informação acerca do número de cópias dos indivíduos na população selecionada, pelo que a recombinação de indivíduos iguais é um desperdício de capacidade computacional. Já a utilização de multi-indivíduos na recombinação garante que os progenitores têm genomas diferentes, e o seu número de cópias pode influenciar a região onde os descendentes vão ser gerados. Nas secções seguintes são apresentados dois operadores de recombinação entre multi-indivíduos.

Centroid Crossover

Em (Manso e Correia, 2011) foi apresentado o operador *Centroid Crossover* (*CX*), figura 6-2, que permite fazer a recombinação de dois multi-indivíduos com a geração de vários descendentes. O operador utiliza o número de cópias dos progenitores, *p1* e *p2*, para definir o local, *c1* e *c2*, onde os descendentes vão ser gerados. O valor *cm* representa o centro de massa da variável *x*, equação 6.2, e é calculado utilizando o número de cópias dos progenitores, *pi.copies*, e o valor do seu gene, *pi.value*. Desta forma, o centro de massa aproxima-se do progenitor que tem mais cópias.

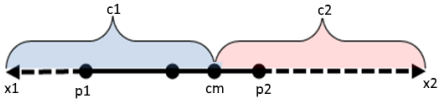


figura 6-2 Operador de cruzamento CX - Centroid Crossover

Os pontos x_i são calculados a partir do centro, o ponto *cm*, tendo como comprimento a distância entre o valor do gene dos progenitores multiplicado pelo fator α, o que permite expandir os limites onde vão ser gerados os descendentes *c1* e *c2*. O operador gera um número de descendentes c1 igual ao número de cópias de p1, e um número de descendentes c2 igual ao número de cópias de p2, utilizando a equação 6.4.

$$cm = \frac{p1.copies * p1.value + p2.copies * p2.value}{p1.copies + p2.copies} \tag{6.2}$$

$$\begin{cases} x1 = p1.value - (p2.value - cm)\alpha \\ x2 = p2.value + (cm - p1.value)\alpha \end{cases} \tag{6.3}$$

$$CX(p1, p2) = \begin{cases} c1 = uniform(x1, cm) \\ c2 = uniform(cm, x2) \end{cases} \tag{6.4}$$

Assumindo que o indivíduo com mais cópias, *pm*, está numa melhor posição do espaço, porque foi selecionado mais vezes, podemos presumir que a direção *cm→pm* aponta para melhores indivíduos e ainda que uma expansão nessa direção é benéfica (6.5).

$$\begin{cases} x1 = p1.value - (p2.value - cm)\alpha * p1.copies \\ x2 = p2.value + (cm - p1.value)\alpha * p2.copies \end{cases} \tag{6.5}$$

O operador *Multicopy Centroid Crossover*, MuCX, utiliza o algoritmo algoritmo 6-1 e a equação 6.5 para a geração dos descendentes.

Na figura 6-3 estão representados alguns descendentes (os pontos a negrito), que são gerados pela recombinação de dois multi-indivíduos, *p0* e *p1*, com dois genes cada um e cujo números de cópias são 2 e 4, respetivamente. A figura 6-3 a) apresenta alguns descendentes gerados pelo operador *CX*, onde são notórios quer o deslocamento do centro de massa em relação ao centro geométrico, quer uma maior densidade junto do indivíduo com mais cópias (*p1*). A figura 6-3 b) apresenta os descendentes gerados pelo operador *MuCX* onde é visível a expansão de geração em ambos os indivíduos, sendo maior no indivíduo com mais cópias (*p1*).

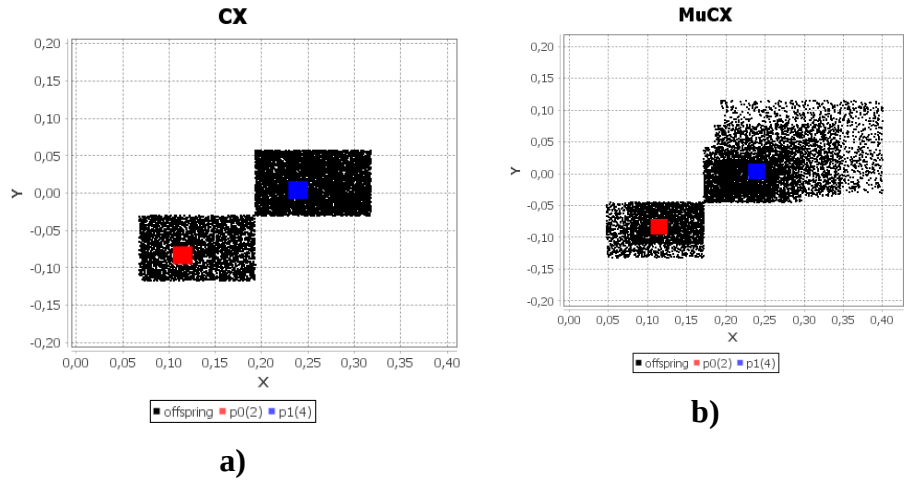


figura 6-3 Indivíduos gerados por recombinação de dois multi-indivíduos: a) Centroid Crossover-CX; b) Multicopy Centroid Crossover – MuCX

Arithmetical Crossover

O operador de recombinação aritmético, equação 6.7, utiliza uma distribuição de probabilidade uniforme, equação 6.6, para calcular uma variável aleatória, que vai servir para definir a proporção com que cada progenitor vai contribuir para a geração no valor do gene dos descendentes. O operador utiliza o parâmetro $\propto$, um valor positivo que permite fazer a exploração do espaço de procura, figura 6-1. Com o valor $\propto = 0$ o operador faz apenas o aproveitamento dos genes, mas com valores superiores, pode fazer a exploração do espaço que rodeiam os genes.

$$\beta = uniform(-\propto, 1+\propto) \tag{6.6}$$

$$AX(p1,p2) = \begin{cases} c1.value = \beta * p1.value + (1-\beta) * p2.value \\ c2.value = \beta * p2.value + (1-\beta) * p1.value \end{cases} \tag{6.7}$$

A equação 6.6 pode ser adaptada para a utilização do número de cópias dos multi-indivíduos de forma a incrementar a área de exploração do operador. O operador *Multiset Arithmetic Crossover*, *MuAX*, utiliza a equação 6.8 para gerar uma variável aleatória que vai ser utilizada na geração dos descendentes, equação 6.7.

$$\beta = uniform(-\propto * p1.copies, 1+\propto * p2.copies) \tag{6.8}$$

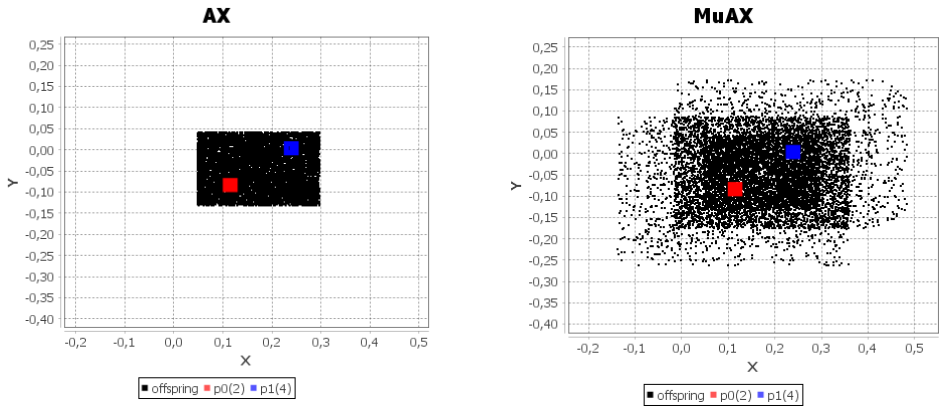


figura 6-4 Indivíduos gerados por recombinação de dois multi-indivíduos: a) Arithmetic Crossover – AX; b) Multiset Arithmetic Crossover – MuAX

A figura 6-4 apresenta o resultado do operador de recombinação aritmético tradicional, *AX*, e o operador de recombinação aritmético em multi-indivíduos, *MuAX*, com os mesmos progenitores apresentados na figura 6-3. Na figura 6-4 nota-se a dispersão da região de geração dos descendentes do operador *MuAX*, que utiliza o número de cópias dos multi-indivíduos, em relação ao operador *AX*, que utiliza indivíduos simples. Devido à simetria na geração dos descendentes, equação 6.7, a diferença do número de cópias entre os progenitores manifesta-se no alongamento da região que rodeia o eixo que une os progenitores.

6.1.3 Mutação

O operador de mutação em problemas com genes codificados por números reais é considerado um operador secundário, por causa da capacidade de exploração do operador de recombinação, que está condicionada pela distância entre o valor dos genes dos progenitores. Ora, com a diminuição da distância inerente à convergência do processo evolutivo, o operador perde a capacidade de sair dos extremos locais. E é nestas situações que o operador de mutação é imprescindível. A sua função consiste em introduzir diversidade genética nos alelos do cromossoma, através de valores aleatórios gerados por uma distribuição de probabilidade. Um multi-indivíduo pode ser considerado como um conjunto de clones, e cada um dos clones, identificado com o número da cópia pode ser utilizado para alterar a dimensão e a frequência da mutação.

O algoritmo 6-2 apresenta o algoritmo canónico de mutação de um multi-indivíduo: considera o multi-indivíduo (*mi*) um conjunto de clones, aos quais vai ser atribuído um número de cópia, alterado por um operador de mutação. O número de cópia do multi-

indivíduo deve ser utilizado para redimensionar a proporção da mutação e a probabilidade de a mesma acontecer.

```
MultisetMutation(mi)
  mutants = lista vazia
  copies = mi.copies
  para itera de 1 até copies
     mi.copies = itera
     mu = Mutar( mi )
     adicionar mu a mutants
  próximo
  retornar mutants
fim
```

algoritmo 6-2 – Algoritmo canónico de mutação de um multi-indivíduo.

A forma de mutação mais simples consiste em atribuir ao gene o valor dado por uma distribuição de probabilidade gaussiana centrada no valor do gene e redimensionada para as dimensões do intervalo. A equação 6.9 executa a mutação de um gene (*p.value*), num indivíduo onde o número de cópia (*p.copy*) é utilizado para alterar a variação da distribuição normal de forma exponencial (β). O valor fornecido pela função gaussiana é multiplicado por um fator $\propto$, que representa uma proporção da dimensão do intervalo onde o gene está definido.

$$mut(p) = p.value + \propto * gauss(0\ , \frac{1}{\beta^{p.copy-1}}) \qquad (6.9)$$

A figura 6-5 apresenta o efeito do número de cópias na função de densidade de gaussiana com média 0 e $\beta = 2$. Com o incremento do número de cópia (*p.copy*) a dimensão da mutação vai ser cada vez menor, e ela fica cada vez mais perto do valor original do gene.

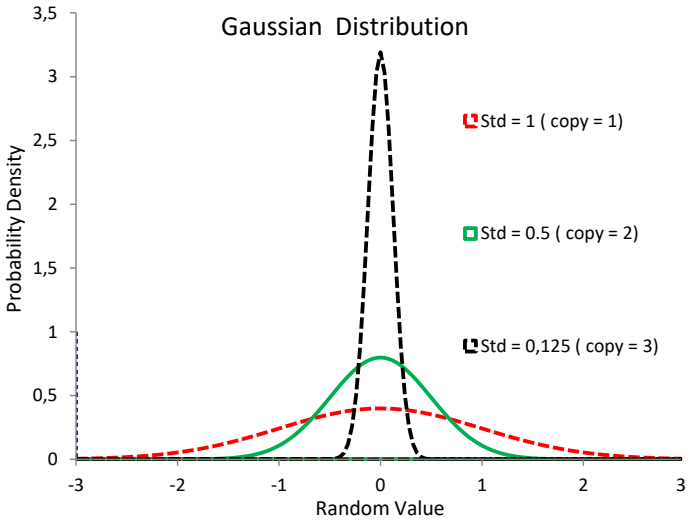


figura 6-5 Função de densidade de probabilidade da função gaussiana aplicada a um multi-indivíduo com 3 cópias.

No MuGA o operador de mutação é precedido do operador de recombinação, que utiliza processos estocásticos. Isto significa que os multi-indivíduos com mais de uma cópia não sofreram recombinação e são clones de indivíduos que já existem na população principal. Por via desta constatação, a probabilidade de mutação dos genes dos clones deve aumentar com o número de cópias, porque a probabilidade de destruir indivíduos gerados pela recombinação é muito baixa. A equação 6.10 mostra o cálculo da probabilidade da mutação de um gene de um indivíduo *p*, onde ∂ representa o número médio de genes a mudar num cromossoma, *p.copy*, o número de cópia, e *p.lenght* é o número de genes do cromossoma.

$$probability(p) = \frac{\partial * p.copy}{p.lengt} \qquad (6.10)$$

Porque a magnitude do valor da mutação de um gene do indivíduo é independente dos restantes elementos da população, os valores das mutações são elevados, e a sua utilização deve ser feita com parcimónia para não destruir os indivíduos gerados pelo operador de recombinação. Resultados experimentais indicam-nos que ∂=0.1, ou seja, em cada 10 indivíduos simples, um gene é um valor adequado para a generalidade dos problemas.

A figura 6-6 apresenta o resultado da mutação de um multi-indivíduo com três cópias, utilizando o operador de mutação gaussiano com $\propto = 0.1$. Na figura 6-6 a), utilizando o operador de mutação gaussiana, Gauss, os clones do indivíduo são tratados de forma independente, e as mutações assumem que cada um dos clones tem um número de cópia igual a 1. Enquanto na figura 6-6 b), utilizamos o operador

adaptado a multi-indivíduos, *MuGauss*, cada clone possui um número de cópias distinto. Este operador não perde a capacidade de exploração do operador original, pois a primeira cópia é tratada como se de um indivíduo simples se tratasse, *child(1)*. As cópias seguintes ganham uma capacidade de exploração acrescida, porque a dimensão da mutação vai ser cada vez menor e a probabilidade de ocorrer é cada vez maior. Devido à baixa probabilidade de aplicação do operador, eq. 6.10, os indivíduos mutantes concentram-se nos eixos; porém, na figura 6-6 b), é visível a concentração de mutantes em redor do indivíduo original, fruto do aumento da probabilidade de mutação e da redução do domínio introduzido pelo número de cópias. É notório o aumento da densidade de pontos verdes, os mutantes da 2ª cópia, e dos negros, os mutantes da 3ª cópia, em relação ao indivíduo original, representado pelo ponto rosa.

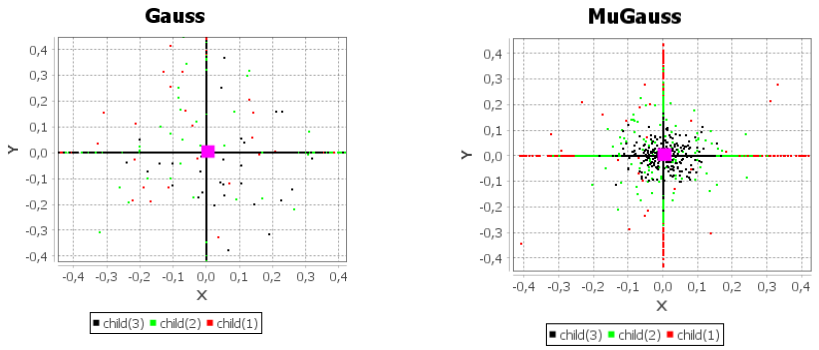


figura 6-6 Operador de mutação aplicado a um multi-indivíduo com 3 cópias: a) Mutação Gaussiana - Gauss; b) Multi-mutação Gaussiana - MuGauss.

6.1.4 Substituição

Neste tipo de problemas, a probabilidade de os operadores genéticos produzirem genomas iguais aos da população é praticamente nula devida à natureza aleatória dos operadores genéticos de recombinação e de mutação. No entanto, estes operadores são aplicados de forma probabilística, e a população dos descendentes pode conter indivíduos a quem não foram aplicados operadores genéticos, sendo, portanto, clones dos progenitores. Estes clones são indivíduos bem-adaptados, porque foram selecionados para reprodução e, quando voltam a ser introduzidos na população, geram multi-indivíduos, cujo número de cópias vai ser utilizado para incrementar a pressão seletiva sobre o genoma.

Qualquer operador de substituição pode ser utilizado na otimização desta classe de problemas, desde que o operador mantenha constante o número de multi-indivíduos e que os genomas repetidos sejam acomodados no número de cópias.

6.2 Efeito dos operadores genéticos baseados em multiconjuntos

Por forma a verificar o efeito dos operadores genéticos adaptados na otimização de problemas codificados por números reais, escolhemos a minimização do problema da esfera, equação 6.11. A função é separável, pois as variáveis podem ser otimizadas de forma independente e unimodal, por possui apenas um mínimo que coincide com o valor ótimo, o valor zero. Esta função é de fácil otimização por várias classes de optimizadores. A utilização desta função para o teste permite aferir a capacidade e a velocidade de convergência para o ótimo de um algoritmo, se o processo evolutivo descobrir a região onde se encontra.

$$Sphere(x) = \sum_{i=1}^{n} x_i^2 \qquad (6.11)$$

A tabela 6-1 apresenta os parâmetros das experiências para minimizar o problema 6.11 com 128 variáveis. Nas multipopulações, foram utilizados 128 multi-indivíduos, enquanto as experiências com populações simples utilizam 128 e 200 indivíduos, respetivamente.

Item	Descrição
Seleção	*MTournamentSelection(parentsProportion=0.5; toursize=3)*
Recombinação	*AX (probCrossover=0.75); MuAX (probCrossover=0.75)*
Mutação	*Gauss(* $\propto = 0.1, \partial = 0.1$*), MuGauss(* $\propto = 0.1, \partial = 0.1$*)*
Substituição	*Geracional*
Redimensionamento	*AdaptiveCeiling(maxProportion=1.5)*

tabela 6-1-Configuração da experiência para avaliar a influência dos operadores genéticos baseados em multi-indivíduos

Em cada geração foram selecionados para reprodução uma quantidade de progenitores que equivale a 50% do número de elementos da população (*parentsProportion=0.5*) através de um torneio com dimensão 3 (*toursize=3*). A recombinação é feita com uma probabilidade de 75% (*probCrossover=0.75*) através do operador de recombinação aritmético para indivíduos simples, AX; para multi-indivíduos é utilizado o operador aritmético adaptado, *MuAX*. A mutação foi feita pelo operador de mutação gaussiano para indivíduos simples, Gauss; e para multi-indivíduos usou-se o operador adaptado, *MuGauss*. Este operador é utilizado com $\propto = 0.1$, que

representa 10% da dimensão do domínio onde o gene está definido ($\partial=0.1$) e significando que é alterado, em média, um gene por cada 10 indivíduos simples.

A substituição da população é feita de forma geracional, com a população dos descendentes a ser completada pelos melhores indivíduos da população principal. O redimensionamento do número de cópias da multipopulação é feita pelo operador *AdaptiveCeiling* com uma proporção máxima de clones de 50%, *maxProportion=1.5*.

Foram efetuadas seis experiências, tabela 6-2, para testar os operadores de recombinação e mutação adaptados e fazer a sua comparação como a versão original. Para cada uma das experiências, foram executadas 100 simulações e, na tabela 6-3, apresentado o resultado das experiências após 250.000 chamadas à função de avaliação.

Item	**Descrição**
MuGA -AX-Gauss	Utilização dos operadores simples AX e Gauss em multi-indivíduos
MuGA -AX-MuGauss	Utilização do operador simples AX e o operador adaptado Gauss em multi-indivíduos
MuGA -MuAX-Gauss	Utilização do operador adaptado MuAX e o operador simples Gauss em multi-indivíduos
MuGA -MuAX-MuGauss	Utilização dos operadores adaptados MuAX e MuGauss em multi-indivíduos
SGA-AX-Gauss-128	Utilização dos operadores simples AX e Gauss numa população com 128 indivíduos simples
SGA-AX-Gauss-200	Utilização dos operadores simples AX e Gauss numa população com 200 indivíduos simples

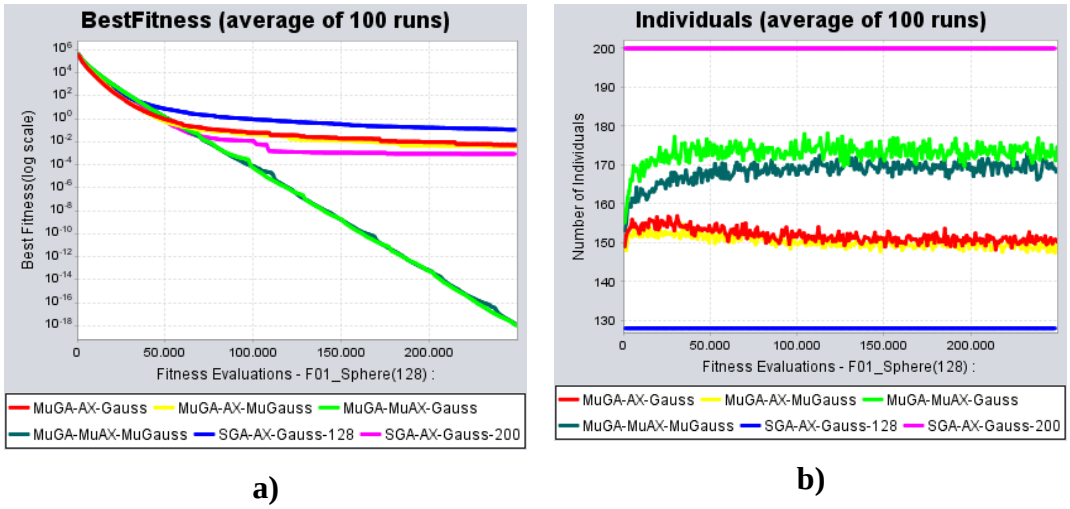
tabela 6-2-Experiências para verificar o efeito dos operadores genéticos adaptados para multi-indivíduos

Com a configuração escolhida, tabela 6-1, apenas as simulações com o operador de recombinação aritmético adaptado, *MuAX*, conseguiram obter valores próximos do ótimo. Em todas as outras simulações, houve convergência prematura da população. Como a função não tem extremos locais, a convergência deu-se pela perda da diversidade genética ao nível dos genes, o que desacelerou o processo de otimização.

	BestFitness		Individuals		SelectedGenotypes		MeanCopiesSelection		MaxCopiesSelection	
Solver	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-AX-Gauss	5,2E-03	1,7E-02	149,9	8,8	41,5	3,6	1,6	0,2	5,1	2,0
MuGA-AX-MuGauss	3,9E-03	1,3E-02	148,2	7,7	41,5	3,5	1,6	0,1	5,4	1,9
MuGA-MuAX-Gauss	1,0E-18	2,0E-18	174,1	15,6	37,6	3,6	1,7	0,2	7,4	3,0
MuGA-MuAX-MuGauss	1,2E-18	3,0E-18	165,8	11,9	38,7	4,0	1,7	0,2	6,2	2,4
SGA-AX-Gauss-128	1,1E-01	9,6E-02	128,0	0,0	39,9	3,4	1,0	0,0	1,0	0,0
SGA-AX-Gauss-200	8,9E-04	6,2E-03	200,0	0,0	60,5	4,2	1,0	0,0	1,0	0,0

tabela 6-3- Resultado da otimização da função Sphere-128

A figura 6-7 a) apresenta a evolução do valor de aptidão do melhor indivíduo ao longo do processo evolutivo, verificando-se a estagnação da otimização nas experiências que utilizam o operador tradicional de recombinação aritmético, AX. A evolução continua nas experiências com o operador *MuAX*. A figura 6-7 b) mostra o número de indivíduos na população, constatando-se que as experiências com o operador *MuAX* envolvem mais indivíduos, devido à expansão do espaço onde os descendentes são gerados, e que, no algoritmo simples, SGA, um maior número de indivíduos conduz a um melhor resultado. Resultados experimentais permitem-nos concluir que o SGA com populações maiores pode otimizar a função 6.1; no entanto, o MuGA consegue o mesmo com populações bastante menores.



a) **b)**

figura 6-7 Evolução dos parâmetros genéticos na otimização do problema Sphere-128: a) Valor do melhor indivíduo b) Número de indivíduos na população.

A figura 6-8 mostra em detalhe a evolução do melhor indivíduo em cada uma das 100 simulações: a região em tom rosa está delimitada pelo valor da melhor e pela pior das experiências; a linha vermelha representa a média. Nas experiências com os operadores adaptados, figura 6-8 b), a evolução é mais lenta no início, mas a evolução é contínua em todas as experiências. A exploração da região expandida ao longo do eixo que une os progenitores, feita pelo operador *MuAX*, permitiu que a população mantivesse diversidade genética suficiente em todos os alelos ao longo de todo o processo evolutivo e, deste modo, obteve sucesso em todas as simulações. Com a utilização do operador *AX*, figura 6-8 a), em algumas simulações, verificamos a convergência prematura de alguns genes para valores muito próximos, mas, como as capacidades de exploração do operador são limitadas pela distância entre o valor dos

alelos, a otimização estagna. No caso do *MuAX*, a sua capacidade de exploração é muito maior, e os genes não convergem de forma isolada.

a) **b)**

figura 6-8 Evolução do melhor indivíduo na otimização do problema Sphere-128: a) SGA com operadores simples b)MuGA com operadores adaptados para multi-índividuos

Esta capacidade de expansão pode ser vista no número médio de cópias, *MeanCopySelection*, e no máximo de cópias, *MaxCopySelection*, na tabela 6-3. O elevado número de cópias produzido pelo operador de seleção é utilizado pelo operador *MuAX* para expandir significativamente a região onde os descendentes são gerados.

A utilização do operador de mutação *MuGauss* não trouxe vantagens, e houve apenas uma ligeira desvantagem: como a função é unimodal e suave, as mutações, eventualmente, podem danificar os indivíduos gerados pelo operador de recombinação através da introdução de valores aleatórios.

6.3 Resultados experimentais num conjunto de funções padrão – CEC2008

Em (Manso e Correia, 2011) foram apresentados os primeiros resultados do MuGA na otimização de funções codificadas por números reais, utilizando as funções da competição CEC2008, cujas características estão apresentadas no Anexo E.

Este conjunto é composto por diferentes tipos de funções com grau de dificuldade diferenciados. Todas as funções têm como mínimo o valor zero, à exceção da função F7, uma função fractal cujo mínimo é desconhecido. O MuGA foi parametrizado com o operador de seleção, de mutação e de redimensionamento da tabela 6-1. Utilizamos o operador de recombinação *MuCX* e, para a substituição de populações, utilizamos um método com pesquisa local baseado no simplex de Nelder-Mead, o NMS (Nelder e Mead, 1965). Nesta estratégia de substituição, os melhores descendentes e os melhores

indivíduos da população principal participam no simplex, figura 6-9, para a busca local do melhor indivíduo. O número de cópias dos indivíduos que participam no simplex é utilizado para calcular o centro de massa, M, dos indivíduos (x1,x2, x3 ,W, B e G) e alongar a linha onde serão gerados os novos indivíduos (Co, C1, R e E). O método NMS é utilizado para evoluir a população do simplex, onde alguns indivíduos do simplex originais são substituídos por outros gerados ao longo da reta que une o pior deles, W, com o centro de massa, M. O simplex evolui sempre enquanto o processo gerar um indivíduo que seja melhor do que o melhor do simplex original, B.

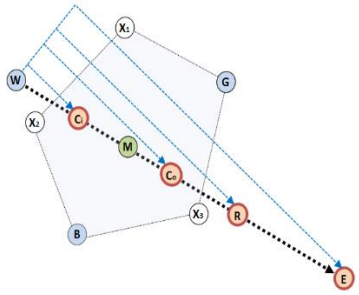


figura 6-9 Simplex Nelder-Mead.

Parametrizamos as funções com 100 variáveis, definimos o critério de paragem em 500.000 chamadas à função de avaliação, e calculamos a média da execução de 25 simulações para cada função, tal como foi definido na competição do evento. Desta forma, os nossos resultados podem ser comparados com os dos algoritmos apresentados no evento *CEC'2008 Special Session and Competition on Large Scale Global Optimization*: MLCC- Multilevel Cooperative for Large Optimization (Zhenyu Yang, Ke Tang, e Xin Yao 2008), EpusPSO-Efficiente Population Utilization for Particle Swarm Optimization (Sheng-Ta Hsieh et al. 2008), jDEdynNP-F - Self-Adaptive Differential Evolution Algorithm (Brest et al. 2008), MTS- Multiple Trajectory Search (Lin-Yu Tseng e Chun Chen 2008), DEwSAcc - Differential Evolution With Self-adaptation and Cooperative Co-evolution (Zamuda et al. 2008), DMS-PSO –Dynamic Multi-swarm Particle Swarm Optimizer (Zhao et al. 2008), LSEDA-gl - Restart Univariate Estimation of Distribution Algorithm Sampling under Mixed Gaussian and L´evy probability Distribution (Auger e Hansen 2005).

A tabela 6-4 apresenta o valor médio obtido pelos diferentes algoritmos na otimização das funções, e a *figura 6-10* mostra o seu ranking. O MuGA tem resultados competitivos, quando comparado com algoritmos bem afinados, tendo utilizado um conjunto alargado de técnicas de otimização, algumas delas talhadas para a otimização de funções com números reais. O MuGA não teve qualquer dificuldade em otimizar a

função da esfera, F1, onde obteve valores comparáveis com os melhores algoritmos. Nas restantes funções, obteve resultados medianos quando comparado com os restantes algoritmos.

	F1	F2	F3	F4	F5	F6	F7
MLCC	6,82E-14	2,53E+01	1,49E+02	4,39E-13	3,41E-14	1,11E-13	-1,54E+03
EPUS-PSO	7,47E-01	1,86E+01	4,99E+03	4,71E+02	3,72E-01	2,06E+00	-8,55E+02
JDEdynNP-F	5,68E-14	4,29E-01	1,12E+02	5,46E-14	2,84E-14	5,68E-14	-1,48E+03
MTS	0,00E+00	1,44E-11	5,17E-08	0,00E+00	0,00E+00	0,00E+00	-1,49E+03
DewSAcc	5,68E-14	8,25E+00	1,45E+02	4,38E+00	3,07E-14	1,13E-13	-1,37E+03
DMS-PSO	0,00E+00	3,64E+00	2,83E+01	1,83E+002	0,00E+00	0,00E+00	-1,14E+03
LSEDA-gl	5,68E-14	2,21E-13	2,81E+02	1,31E+02	2,84E-14	9,78E-14	-1,46E+03
MUGA	9,55E-26	6,54E-01	1,06E+02	1,39E-12	1,11E-16	5,73E-14	-1,46E+03

tabela 6-4- – Comparação dos resultados do MuGA com os algoritmos que competiram no CEC2008

O MuGA é o único representante dos algoritmos genéticos, o que vem demonstrar que este tipo de algoritmos, quando devidamente parametrizado, pode ser utilizado para a otimização de funções reais.

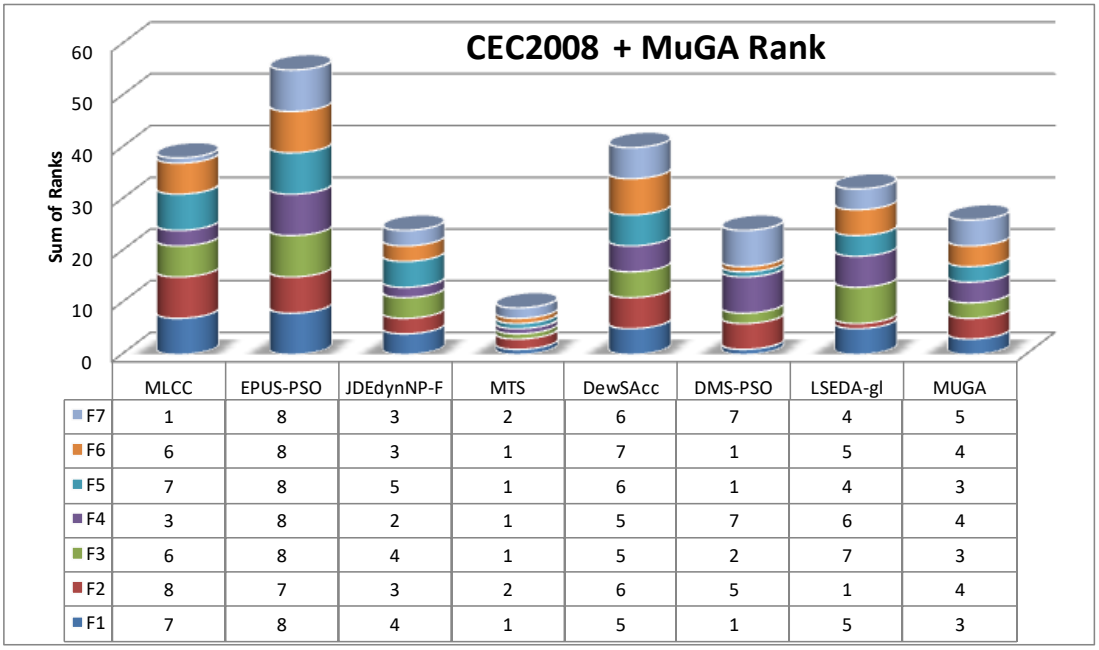


figura 6-10 Ranking dos algoritmos na otimização do Benchmark CEC2008.

6.4 Comentários finais

Neste capítulo adaptámos os operadores genéticos de recombinação e de mutação para tirarem partido do número de cópias dos multi-indivíduos. Apresentámos ainda o algoritmo canónico de recombinação e de mutação de multi-indivíduos e fizemos a

adaptação de alguns operadores tradicionais de forma a serem utilizados na otimização de problemas codificados por números reais.

Definimos o operador *Multiset Centroid Crossover, MuCX*, que desloca o centro de massa dos indivíduos em direção ao indivíduo com mais cópias, e o operador *MuAX*, que utiliza o número de cópias para expandir a área onde os descendentes são gerados. Definimos o operador *MuGauss* para fazer a mutação de multi-indivíduos, utilizando o número de cópias para modificar a amplitude e a frequência da mutação nos genes dos clones representados pelo multi-indivíduo.

Na otimização da função da esfera com 128 variáveis e uma população de 128 multi-indivíduos, o MuGA com operador *MuAX* demonstrou um desempenho muito superior em relação às restantes simulações. Na configuração escolhida para as experiências, pudemos constatar que os operadores genéticos tradicionais não conseguem otimizar a função por causa da perda de diversidade genética na população, um problema que pode ser contornado pela utilização dos operadores genéticos adaptados para multi-indivíduos. Por verificação experimental, constatámos que os algoritmos genéticos com populações simples necessitam de uma população muito maior para manter a diversidade genética necessária à convergência do algoritmo para o ótimo, quando comparado com o MuGA. Neste tipo de problemas, as multipopulações, enquanto suporte da população principal, não apresentam vantagens significativas, uma vez que a probabilidade de geração de dois indivíduos iguais é praticamente nula. No entanto, quando utilizadas para suporte dos progenitores selecionados para reprodução, permitem a utilização de operadores que exploram o número de cópias dos multi-indivíduos para alterar o comportamento dos operadores genéticos tradicionais.

Utilizámos o MuGA para a otimização de um conjunto de funções muito variado e comparámos o seu resultado com os algoritmos que representam o estado da arte. Nesta comparação, o MuGA, com o operador *MuCX* e utilizando o método simplex Nelder-Mead para busca local, obteve um resultado que ficou na média dos algoritmos apresentados na literatura. No Anexo E são apresentados os resultados do MuGA na otimização das funções com as 100, 500 e 1000 variáveis do Benchmark CEC2008, utilizando o operador *MuAX*. Os resultados obtidos são consistentes com os apresentados anteriormente, e o MuGA consegue um resultado comparável aos bons algoritmos apresentados na competição.

Os resultados experimentais produzidos pelas simulações e a experiência prática, acumulada durante o processo de investigação neste tipo de funções, deixam-nos

otimistas no desenvolvimento de operadores baseados em multisets. O MuGA mostrou um desempenho consistente na otimização de funções de natureza muito diversa, o que faz prever o espectro alargado da sua utilização para otimização de funções genéricas.

7 Hibridização do MuGA

Os modelos de coevolução, que exploram a capacidade de várias espécies evoluírem em conjunto, podem ser usados para estudar tanto as configurações naturais como para resolver problemas de otimização. Estes modelos são uma ferramenta eficaz e configurável para modelar diferentes tipos de evolução em multiespécies: parasitismo, comensalismo, mutualismo e interações cooperativas. A evolução competitiva de várias espécies mostrou ser uma técnica de otimização útil, fornecendo melhores indicadores do que os apresentados pela evolução de uma única população. A coevolução de uma população de soluções com uma população geradora de problemas empurra as duas populações para soluções cada vez melhores, um evento chamado "corrida às armas" (Rosin e Belew 1997).

Mas também, a simbiose é uma forma de coevolução cooperativa, que vem ganhando relevância na biologia (Daida et al. 1996). Em sistemas artificiais, a coevolução simbiogenética demonstrou melhorar os algoritmos de otimização evolutiva através da especialização dos diferentes componentes da colaboração simbiótica (Wallin, Ryan, e Azad 2005). Neste caso de coevolução cooperativa, existe uma espécie de divisão de trabalho entre os diferentes tipos de simbiontes: cada hospedeiro é combinado com um conjunto de parasitas que interagem em colaboração, avaliada como uma solução para o problema de otimização. Este fenómeno repete-se em diferentes hospedeiros e parasitas. A evolução simbiogenética artificial mostrou ser útil na resolução de problemas enganosos, uma classe de funções que é especialmente difícil de otimizar devido ao facto de o ótimo ser cercado por regiões de soluções de baixa qualidade.

Neste capítulo apresentamos o algoritmo *Symbiogenetic* MuGA (SMuGA), que usa a inspiração natural da simbiogénese para resolver problemas enganosos de grande dimensão, não resolvidos pela versão simples do MuGA.

7.1 Symbiogenetic MultiSet Algorithm - SMuGA

A simbiose é um conjunto de teorias naturais que tentam explicar a relação natural entre indivíduos que vivem juntos e como essa relação é vital para a sobrevivência do grupo. Na natureza, a simbiose ocorre e envolve uma relação constante e íntima entre espécies diferentes (Daida et al. 1996). Essa relação é mais do que a interação ecológica pela inclusão de diversas vertentes: o mutualismo, onde ambos os indivíduos ganham

vantagens da aliança; o comensalismo, no qual um individuo ganha vantagem sem que o outro tenha algum inconveniente; e, finalmente, o parasitismo, onde um indivíduo ganha vantagens e o outro é prejudicado pela relação.

A teoria da simbiose fornece um operador genético adicional ao processo evolutivo artificial e é aplicada com sucesso para resolver uma ampla gama de problemas difíceis. Para uma revisão da simbiogénese enquanto mecanismo para construir sistemas adaptativos complexos, remetemos para (Heywood e Lichodzijewski 2010).

O algoritmo SMuGA é inspirado no algoritmo de coevolução simbiogenética (SymbCoev), proposto por Wallin et al., em 2005, que explora uma relação hospedeiro-parasita para otimização de funções enganosas concatenadas. Embora os nomes "hospedeiro" e "parasita" sugiram uma relação parasitária, a interação entre duas espécies é benigna e os ganhos de parasitas não são prejudiciais para os anfitriões. O SymbCoev permite otimizar as funções enganosas concatenadas, e o MuGA por si só provou ser um algoritmo eficiente na otimização de tais funções com um tamanho moderado (Manso e Correia, 2013a). No entanto, quando o tamanho dos problemas aumenta, o MuGA apresenta dificuldades para a sua otimização. O SMuGA é um algoritmo que usa duas espécies cooperativas, hospedeiros e parasitas, que evoluem juntos num relacionamento mutualista. Os parasitas são compostos por um tuplo <*posição, genoma*>, onde a posição representa o local do genoma do hospedeiro em que o parasita atua, e o o genoma representa o material genético do parasita.

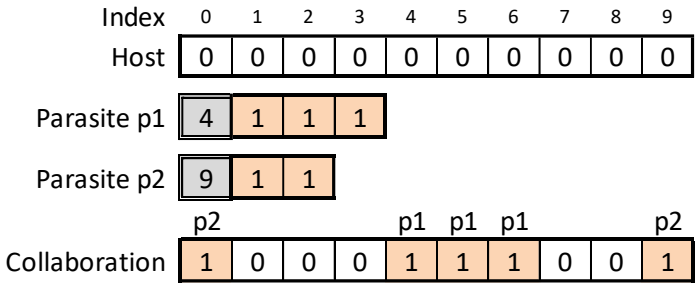


figura 7-1 - Colaboração formada pela simbiose de um hospedeiro e um parasita

No SMuGA, o genoma do hospedeiro é substituído pelo genoma do parasita no local definido pelo atributo *posição*, figura 7-1. O parasita considera o genoma hospedeiro como um círculo, ou seja, quando a cópia do genoma do parasita para o hospedeiro atinge o limite do genoma, a cópia continua no início. Na figura 7-1 o parasita *p1* possui três genes e pode ser aplicado no genoma hospedeiro nos alelos 4, 5 e 6; o parasita *p2* possui dois genes e pode ser aplicado nos alelos 9 e 0 do genoma do hospedeiro. A

colaboração é a combinação de genes hospedeiros com os genes introduzidos pelos parasitas *p1* e *p2*.

O SymbCoev tem algumas deficiências identificadas pelos autores. Na verdade, o tamanho dos parasitas é estático e definido como um parâmetro, e a colaboração faz-se de um parasita para um hospedeiro, que só pode ser infetado por um parasita de cada vez. Os melhores resultados obtidos pelo algoritmo ocorrem quando a proporção do genoma do parasita é semelhante ao tamanho das funções a serem otimizadas, os blocos de construção (BB), mas o desempenho degrada-se rapidamente à medida que o tamanho dos parasitas se afasta do tamanho dos BB. Outra fraqueza do algoritmo SymbCoev resulta do facto de a colaboração ser de um hospedeiro com um parasita, o que limita a sua aplicabilidade a problemas separáveis.

O SMuGA foi projetado para suprimir aquelas duas deficiências, combinando o conceito de simbiose com o potencial das populações baseadas em multisets presentes na otimização deste tipo de funções. Na próxima secção, mostramos a representação e a evolução, quer da população parasitária, quer população hospedeira, e a interação entre elas utilizando multipopulações.

7.2 Modelo de Evolução dos parasitas

De forma a automatizar a escolha do tamanho dos parasitas, deixamos que o seu tamanho seja o resultado da evolução; eliminamos, assim, a necessidade de especificar mais um parâmetro no processo evolutivo. Como mencionado anteriormente, o trabalho de (Wallin, Ryan, e Azad 2005) mostrou que havia uma dependência muito forte do desempenho em relação ao tamanho do parasita: quando o tamanho dos parasitas se aproxima do tamanho do BB, o desempenho é bom; no entanto, ele decai muito rapidamente com os desvios da dimensão ideal.

No algoritmo SMuGA, o utilizador não tem de saber o tamanho do BB, porque o comprimento do genoma do parasita é variável e ajusta-se ao longo do processo evolutivo. Do conhecimento adquirido durante a investigação realizada, este é o primeiro modelo de parasitas que pode variar o seu comprimento ao longo do processo evolutivo. Este sistema é importante para a resolução de problemas em que o tamanho do BB não é conhecido ou tem um tamanho variável. O tamanho dos parasitas é alterado por operadores genéticos de recombinação e mutação. O operador de seleção dá oportunidade aos parasitas, que têm um bom desempenho na população hospedeira, para se reproduzir e transmitir o seu conteúdo, material genético e posição aos seus

descendentes. De acordo com a teoria da sobrevivência do mais apto, os melhores parasitas, o que inclui a posição de aplicação e o material genético, espalharão os seus genes para as gerações subsequentes, descobrindo e otimizando simultaneamente a posição, o tamanho e o valor dos alelos.

7.2.1 Recombinação do parasita

Quando dois parasitas se recombinam, podem ocorrer quatro situações:

1. Os parasitas não compartilham posições no genoma do hospedeiro;
2. Os parasitas ocupam posições consecutivas no genoma do hospedeiro;
3. Os parasitas compartilham algumas posições no hospedeiro;
4. Todas as posições de um dos parasitas ocupam posições do outro.

No primeiro caso, como os parasitas infetam diferentes regiões do genoma hospedeiro, a recombinação entre os dois parasitas não pode ser realizada. Em todos os outros casos, a ideia subjacente a este operador não é apenas recombinar o material genético, mas também para introduzir diferentes comprimentos do genoma. Nós elegemos a recombinação de genomas do parasita como o operador principal para aumentar e diminuir o comprimento dos parasitas.

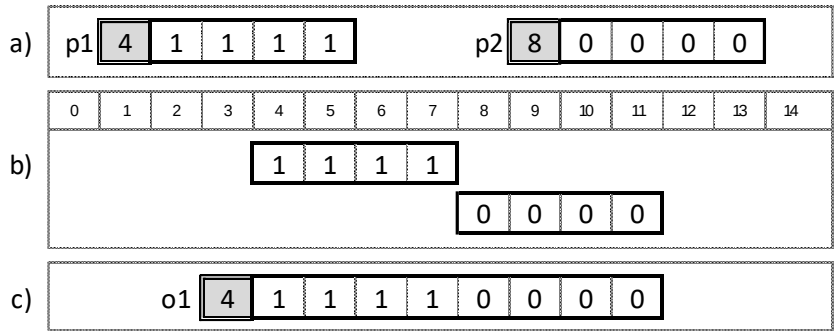


figura 7-2 -Recombinação pela união de parasitas consecutivos: a) parasitas selecionados; b) posições ocupadas por parasitas no genoma; c) resultado da recombinação p1 e p2.

No segundo caso, figura 7-2, em que os parasitas ocupam locais consecutivos no genoma hospedeiro, o resultado da recombinação é a união dos genomas, gerando um único parasita. O descendente *o1*, figura 7-2 c, tem um genoma cujo tamanho é a soma do tamanho dos genomas dos progenitores. Este tipo de reprodução concatena o genoma dos parasitas, fazendo-o crescer.

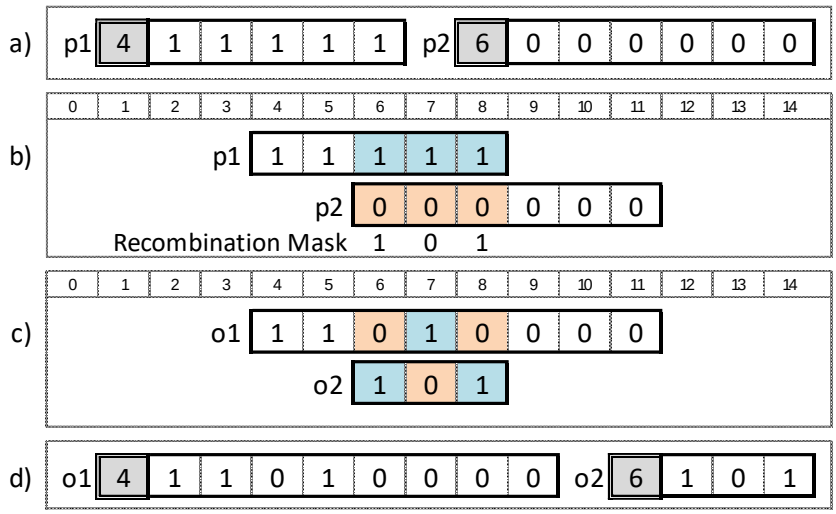


figura 7-3 -Recombinação pela parcela de algumas posições no hospedeiro: a) parasitas selecionados; b) posições ocupadas por parasitas no genoma; c) resultado da recombinação P1 e P2; d) parasitas descendentes.

No caso 3, figura 7-3, em que os parasitas compartilham algumas posições no genoma hospedeiro, os alelos na zona de sobreposição são combinados através da recombinação uniforme. Além disso, a descendência terá diferentes tamanhos de genoma em comparação com seus pais. A figura 7-3 b) apresenta a recombinação uniforme dos parasitas através de uma máscara gerada de forma aleatória. O símbolo *1* na máscara significa que há uma troca de alelos na zona de sobreposição, e o símbolo *0* significa o oposto. A figura 7-3 c) e a 7-3 d) mostram o resultado da recombinação dos progenitores *p1* e *p2*. O descendente *o1* herda de ambos os pais as partes que não são sobrepostas, bem como o genoma recombinado pela máscara de recombinação. O descendente *o2* herda apenas a parte comum que resulta da aplicação da máscara. Este operador altera o tamanho de ambos os parasitas em relação aos seus progenitores: o descendente *o1* tem um comprimento maior e o *o2* tem um comprimento menor.

No caso 4, figura 7-4, onde um dos parasitas, *p1*, ocupa todas as posições do outro, *p2*, no genoma do hospedeiro, a zona de sobreposição é recombinada utilizando recombinação uniforme nos moldes definidos no caso anterior. Na figura 7-4 d) o descendente *o1* herda do progenitor *p1* a primeira parte, não comum a ambos os pais, e a parte comum recombinada; e o descendente *o2* herda o resultado da recombinação da parte sobreposta e a última parte não comum de *p1*. Neste caso, o parasita mais pequeno vai atuar como faca de corte de grandes parasitas.

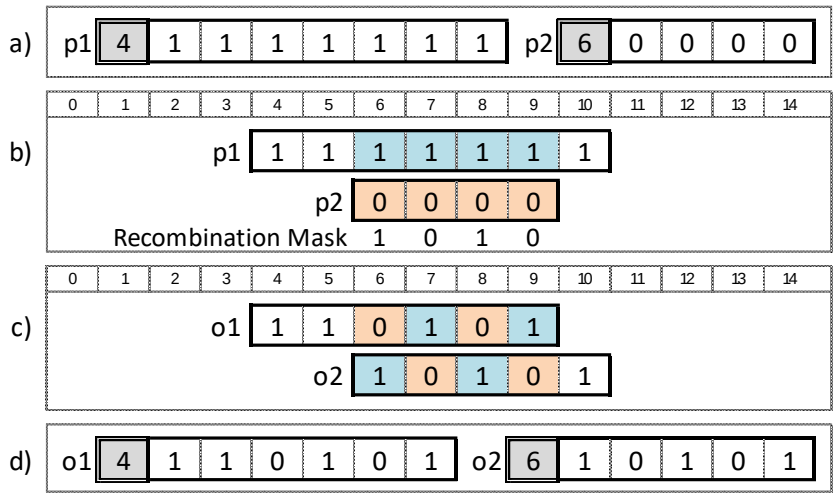


figura 7-4- Recombinação quando um dos parasitas ocupa todas as posições de outros: a) parasitas selecionados; b) posições ocupadas por parasitas no genoma; c) resultado da recombinação de p1 e p2; d) parasitas descendentes.

7.2.2 Mutação do parasita

O operador da mutação altera aleatoriamente as características de um parasita, a saber: a posição, o comprimento e o seu material genético. Assim, definimos três tipos de mutação do parasita:

1. Mudança na posição de ancoragem;
2. Mudança no genoma;
3. Quebra do genoma do parasita.

Na primeira situação, os parasitas mudam a posição da infeção no hospedeiro de forma aleatória. Na figura 7-5, o parasita *p1*, que infecta a quarta posição do hospedeiro, gera o mutante *m1*, que infeta a posição 10. Na figura 7-5 c), podemos observar que o parasita *m1* afeta o genoma hospedeiro de forma circular, pois os últimos três bits do parasita infetam as três primeiras posições do hospedeiro.

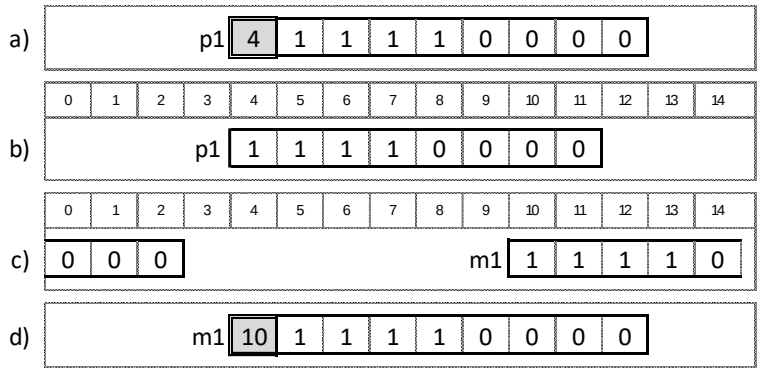


figura 7-5 - Mutação mudando de posição: a) parasita original; b) posições ocupadas pelo original; c) posições ocupadas pelo parasita mutante; d) parasita mutante.

No segundo caso, o valor dos alelos é alterado por uma distribuição de probabilidade uniforme que gera uma máscara de mutação, figura 7-6 b), mas nas posições em que a máscara tem o valor *1*, o valor de bit do parasita é invertido. Nesta situação, apenas o valor do genoma do parasita é modificado, o que favorece o aparecimento de parasitas na população com novos genomas.

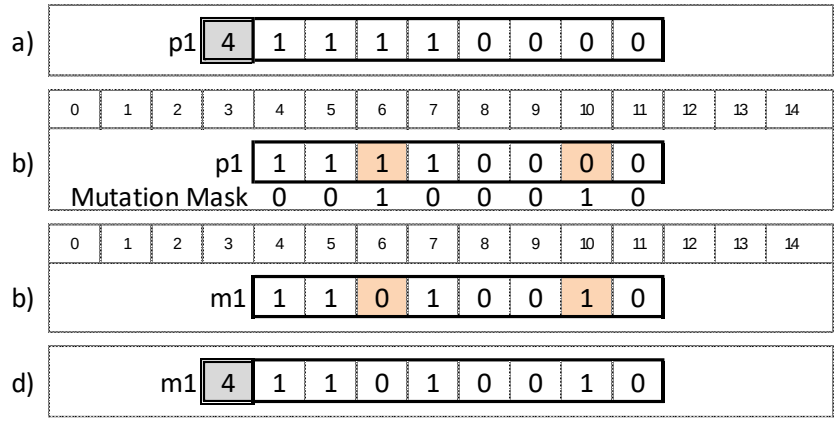


figura 7-6 - Mutação alterando o genoma: a) parasita original; b) posições ocupadas por máscara original e de mutação; c) posições ocupadas pelo parasita mutante; d) parasita mutante.

Na última situação, figura 7-7, o genoma do parasita é dividido em duas partes, dando origem a dois novos parasitas. A probabilidade de dividir um genoma é proporcional ao seu comprimento em bits.

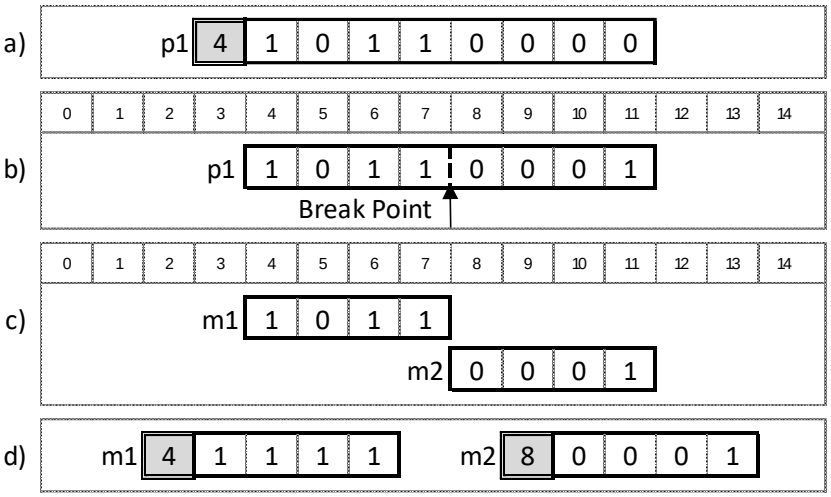


figura 7-7 - Mutação pela quebra do genoma: a) parasita original; b) posições ocupadas pelo parasita original e pelo ponto de rutura; c) posições ocupadas por parasitas mutantes; d) parasitas mutantes.

A equação 7.1 mostra a fórmula para calcular a probabilidade de divisão do parasita, p_{break}. O parâmetro *k* controla a dimensão a partir da qual a divisão de um parasita é inevitável, ou seja, nos casos em que a proporção entre a potência do tamanho do genoma do parasita, *parasite.size*, é maior do que o tamanho do hospedeiro, *host.size*.

Já o parâmetro *n* controla a probabilidade de divisão nos outros casos. O ponto de divisão do genótipo é selecionado por uma distribuição de probabilidade uniforme sobre o genótipo do parasita.

$$p_{break}(parasite) = \max\left(\left(\frac{parasite.size^{k}}{host.size}\right)^{n}, 1\right) \quad (7.1)$$

Este tipo de mutação evita o crescimento desproporcionado do parasita e a possível captura do genoma do anfitrião pelo parasita. Na figura 7-7 o parasita *p1* cria dois parasitas, *m1* e *m2*, através do corte no quarto alelo do genoma do progenitor. O mutante *m1* recebe a posição e a primeira parte do genoma do progenitor, enquanto o mutante *m2* assume a posição correspondente ao ponto de corte no hospedeiro e a segunda parte do genoma.

7.2.3 Avaliação de parasitas

Os parasitas representam solução parciais do problema de otimização, e a sua avaliação não pode ser obtida diretamente a partir da função de avaliação dos hospedeiros. Sendo assim, a avaliação da população de parasitas é obtida indiretamente através dos genomas presentes na população de hospedeiros. Este procedimento permite que os parasitas sejam avaliados, sem a necessidade de os aplicar aos hospedeiros para aferir o resultado da colaboração. Substituímos as chamadas de função de avaliação por um procedimento que consiste em verificar se o genoma do parasita está presente no genoma dos hospedeiros, e usamos a sua aptidão para inferir a do parasita. Este tipo de avaliação permite uma economia significativa das chamadas da função de aptidão, bem como dos recursos computacionais que seriam gastos para gerar colaborações entre as duas populações.

Definimos três objetivos para os parasitas:

1. Promover o aparecimento de parasitas com novo material genético, necessário para evoluir a população e prevenir a sua estagnação;
2. Propiciar a variabilidade do ponto de ancoragem de bons parasitas no genoma hospedeiro, a fim de permitir que diferentes regiões sejam infetadas;
3. Fomentar a disseminação de parasitas com bons genótipos na população hospedeira para que todos os indivíduos tenham o parasita.

O último objetivo, a manutenção dos bons parasitas, é incompatível com os anteriores, que promovem o aparecimento de novos elementos com genótipo diferente ou em posições distintas no genoma. A função de avaliação deve fazer um

balanceamento entre todos os objetivos de forma a ajudar a população dos hospedeiros a evoluir. A função de avaliação também deve fomentar o crescimento do comprimento do parasita e descobrir BB de grandes dimensões, para acelerar o processo evolutivo. Os valores produzidos pela função devem ser independentes do valor da aptidão dos hospedeiros, evitando assim enviesamentos produzidos por indivíduos especialmente bem-adaptados e dando oportunidade de ser introduzido novo material genético na população.

A função de avaliação deve, portanto, ser proporcional ao tamanho do parasita, para suscitar o seu crescimento, quanto à aptidão do hospedeiro, ela deve ser inferida pela sua posição no ranking da população. Os hospedeiros são ordenados por ordem decrescente do seu grau de aptidão e recebem uma classificação de acordo com a sua posição, ou seja, o melhor hospedeiro recebe a classificação que corresponde a tamanho da população, n, e o pior recebe a classificação 1. A avaliação dos parasitas utiliza a mesma posição no ranking dos hospedeiros que estão infetados com o parasita, isto é, daqueles que possuem o genoma do parasita na posição de infeção, para incrementar a sua aptidão. Se um parasita infetar a população toda, vai ter o valor de aptidão máximo; no entanto, o seu contributo na evolução da população dos hospedeiros é nulo, contrariando deste modo o objetivo 1. No entanto, tais parasitas são bons candidatos para a disseminação, o que está em consonância com os objetivos 2 e 3. Por forma a contornar este problema fizemos uma transposição do intervalo para $[n/2-1, -n/2]$, com mudança no ranking da população, e esta fornece um conjunto de vantagens significativas: a aptidão dos parasitas que infetam toda a população é zero; os parasitas presentes apenas nos melhores indivíduos têm aptidão positiva; por oposição, os parasitas que apenas estão presentes nos piores indivíduos têm aptidão negativa.

A fim de premiar indivíduos com um grande genoma, o valor da soma obtido pelo ranking dos hospedeiros é multiplicado pelo tamanho do parasita. Assim, se um parasita tem uma soma positiva obtida pela infeção nos hospedeiros, o seu tamanho é recompensado; no caso inverso, o seu tamanho contribui para a diminuição da sua aptidão. O aumento do tamanho do genoma do parasita faz com que esteja presente em menos hospedeiros e, se a sua aptidão for positiva, tal significa que apenas os bons indivíduos estão infetados.

A função de avaliação definida promove a descoberta de bons parasitas, isto é, daqueles que infetam e estão presentes apenas nos melhores hospedeiros. Esta aptidão faz com que os bons parasitas se disseminem e infetem cada vez mais hospedeiros,

espalhando-se na população e, quando infetam mais de metade da população, a sua aptidão começa a desvanecer, dando oportunidade a que novos parasitas sigam o mesmo caminho. Esta avaliação do parasita é muito eficiente, porque não usa uma única chamada à função da aptidão.

Quando a evolução descobre um novo parasita, cujo genótipo não existe na população, a função de avaliação deve premiar a sua descoberta com uma aptidão que lhe permita sobreviver, infetar os hospedeiros e reproduzir-se, caso a sua qualidade o permitir. Por outro lado, a dimensão de um novo parasita deve reduzir a sua aptidão, por forma a evitar o surgimento de grandes parasitas com genomas aleatórios que, por sua vez, contrastam com os grandes parasitas que evoluíram a partir de bons BB. Decidimos atribuir ao novo parasita um valor de aptidão igual ao tamanho da população, dividido pelo seu comprimento em bits, como recompensa pela descoberta de novos genomas parasitas. A função de avaliação permite que pequenos parasitas com novos genótipos apareçam na população e se recombinem com os já existentes, fomentando assim o seu crescimento, caso possuam material genético útil para a evolução.

7.3 Algoritmo de evolução dos parasitas

A evolução dos parasitas é feita pelo algoritmo 7-1, que recebe como parâmetros a população de parasitas para evoluir, *parasitePop*, a população de hospedeiros para realizar a avaliação dos parasitas, *hostPop*, e o número de parasitas que serão selecionados para evoluir, *n*.

```
ParasiteEvolution (parasitePop, hostPop, n)
  selectPop = selecionar n parasitas de parasitePop
  offspringPop = recombinar selectedPop
  enquanto offspringPop.size < parasitePop.size
      parasite = Selecionar um parasita aleatório de offspringPop
      mutant = Mutar parasite
      Adicionar mutant a offspringPop
  fim enquanto
  Avaliar offspringPop em hostPop
  parasitePop = selecionar parasitePop.size parasitas de
                parasitePop e offspringpop
fim
```

algoritmo 7-1 – Algoritmo de evolução dos parasitas

O algoritmo começa por selecionar *n* parasitas de população dando origem à população dos progenitores, select*Pop*, que serão recombinados, de acordo com as regras definidas anteriormente. A população *offspringpop***,** é gerada pela remoção de um par de indivíduos da população selecionada, que vai gerar dois descendentes, aplicando um operador de recombinação de forma probabilística. Se o operador de recombinação for aplicado, os descendentes partilharão os atributos dos pais; caso contrário, passam diretamente para a população tal como os seus clones. De seguida, o algoritmo introduz novos parasitas na população *offsringpop* até esta atingir o tamanho da população original através do operador de mutação. Este operador seleciona aleatoriamente um elemento de *offsringpop*, clona-o, introduz-lhe mutações e adiciona o clone à população. O método de mutação aplicado ao parasita é selecionado aleatoriamente dentre as que foram propostas: mutação do genoma, mutação da posição de ancoragem ou quebra do genoma. Optamos por usar a distribuição de probabilidade uniforme para selecionar o método de mutação. Esta forma de completar uma população permite que um parasita se possa submeter a várias mutações numa uma única geração, porque um parasita mutante pode ser selecionado e clonado várias vezes.

No final, a população *Offspringpop* é avaliada através dos hospedeiros da população *hostPop* e o algoritmo termina com o cálculo da nova geração, através do operador de substituição aplicado à população de parasitas original, *parasitePop***,** e à população de seus descendentes, *offspringpop*.

7.4 Algoritmo de evolução dos hospedeiros

A população de hospedeiros é evoluída com o MuGA, algoritmo 4-1, que usa alguns operadores genéticos adaptados a multipopulações. O MuGA usa o operador *MTournamentSelection*, algoritmo 3-1, para fazer a seleção dos progenitores e o operador de cruzamento com 1 ponto de corte para fazer a sua recombinação. Tivemos de fazer a adaptação dos operadores genéticos de mutação e de substituição, para tratar do número de cópias dos multi-indivíduos, de forma a que o MuGA seja capaz de resolver problemas difíceis.

7.4.1 Multiset Wave Mutation - MWM

Para resolver problemas em que a solução não pode ser encontrada pela recombinação dos genes dos progenitores, o operador de mutação realiza a missão crítica de gerar novos alelos no genoma dos descendentes. Geralmente, o operador

utiliza taxas de mutação muito baixas, responsáveis por pequenas mudanças nos descendentes, e, se estas estas novas características forem benéficas, elas são propagadas na população, aumentando gradualmente a evolução da população. Uma taxa de mutação alta, responsável por várias mudanças num único indivíduo, pode ocultar as novas características benéficas com outras prejudiciais e o próprio processo evolutivo, tornando-se uma busca aleatória. No entanto, por vezes é necessário fazer a mudança de um conjunto significativo de alelos, e tal só é possível com uma taxa de mutação mais elevada. O balanceamento entre a capacidade de refinar as características da população com baixas de mutação e a exploração do espaço na busca de novas soluções pode ser feito pelos vários clones que um multi-indivíduo representa. Os clones de um multi-indivíduo podem ser numerados, e esta numeração de cópia ser utilizada para alterar a taxa de mutação de cada um dos clones.

$$waveFunction(copy) = \left[\frac{\sin\left(\frac{\pi}{2}+\frac{copy-1}{roughness}\right)}{2}\right]^{thickness} \qquad 7.1$$

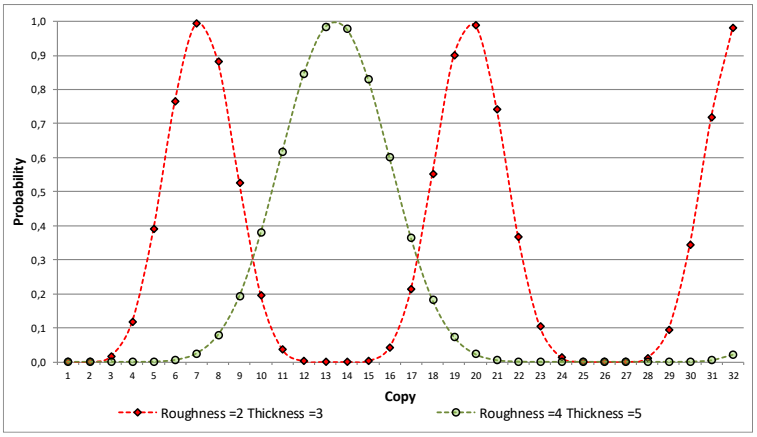


figura 7-8 - Gráfico da função waveFunction

Em (Manso e Correia, 2013a) apresentamos a equação 7.1, que utiliza o número de cópia dos clones para calcular a probabilidade de mutação de um gene com vista à otimização de problemas enganosos. A forma da função é controlada pelo parâmetro *roughness*, que altera a rugosidade da função, e pelo parâmetro *thickness*, que controla a fineza dos seus picos. A figura 7-8 mostra duas representações gráficas da função para um indivíduo com 32 cópias. A linha verde representa a função com uma rugosidade igual 3 e uma fineza de 5; no gráfico, podemos ver dois picos suaves separados por duas planícies. A linha vermelha revela a mesma função com a rugosidade 1 e uma fineza de 3, apresentando 5 picos separados por vales profundos. Os vales fornecem taxas de

mutação baixas e são utilizadas para a exploração dos genes nos descendentes; os picos fornecem taxas de mutação que favorecem a sua exploração.

```
MultisetWaveMutation (mi, minProb, mutOperator)
  mutantes = lista vazia
  Para copy = 1 até mi.copies
    prob = min ( minprob + wavefunction(copy), 1)
    individual = mi.genotype
    mutOperator(individual, prob)
    adicional individual a mutants
  próximo
  retornar mutants
fim
```

algoritmo 7-2 – Algoritmo de mutação MultisetWaveMutation

O operador de mutação MultisetWaveMutation, algoritmo 7-2, foi projetado para aplicar um operador de mutação tradicional, *mutOperator***,** num multi-individuo, *mi*, usando a função *waveFunction* para calcular a probabilidade de mutação de cada clone. O operador recebe um multi-indivíduo e devolve uma lista com a mutação de cada um dos seus clones alterados. A probabilidade é calculada adicionando uma probabilidade mínima, *minprob*, ao resultado de *waveFunction* e truncando o resultado para 1, se a soma for maior do que 1. A primeira cópia utiliza a probabilidade de mutação mínima, *minprob*, o que equivale à mutação de um indivíduo simples, mas as cópias seguintes recebem taxas de mutação diferenciadas. Quando a taxa de mutação atinge o valor unitário, todos os genes do individuo são alterados.

7.4.2 MDR - MultiSet Decimation Replacement

O operador de substituição tem a tarefa de formar a população que continuará o processo evolutivo. O operador *MultiSet Decimation Replacement* (MDR), algoritmo 7-3 foi projetado para introduzir os descendentes produzidos pelos operadores genéticos, *offspringPop*, na multipopulação principal, *parentsPop***,** preservando o máximo de multi-indivíduos. O MDR começa por juntar as duas multipopulações, e esta operação produz grande multipopulação e geralmente multi-indivíduos com várias cópias. O passo seguinte vai dizimá-los de forma a restabelecer o tamanho original da população. Neste passo o algoritmo seleciona um grupo de ***n*** multi-indivíduos e elimina o mais fraco.

```
MultisetDecimation (parentsPop, offspingPop, n)
  parentsSize = parentspop.size
  allPop = parentspop + offspringpop
  Enquanto parentsPop.size > parentsSize
    group = selecione n indivíduos aleatórios de allPop
    weak = indivíduo mais fraco de group
    remover weak de allPop
  fim enquanto
fim
```

algoritmo 7-3 – Algoritmo de substituição MultisetDecimationReplacement

7.5 Co-evolução dos hospedeiros e dos parasitas

O algoritmo SMuGA é um algoritmo evolutivo que utiliza duas populações cooperantes para resolver problemas difíceis: a população hospedeira, que contém as soluções do problema, e a população parasita, que a ajuda a evoluir. A população de parasitas, que representa soluções parciais do problema, infeta a população dos hospedeiros, substituindo os seus genes. A interação entre hospedeiros e parasitas produz uma nova população usando simbiose, à semelhança do que ocorre no mundo natural.

7.5.1 Colaboração

Definimos a colaboração como o resultado de um hospedeiro infetado por um ou mais parasitas através da simbiose. Um parasita pode infetar vários hospedeiros e os hospedeiros podem ser infetados por vários parasitas. Esta colaboração é obtida copiando os alelos do parasita para o hospedeiro caso seja possível, figura 7-9.

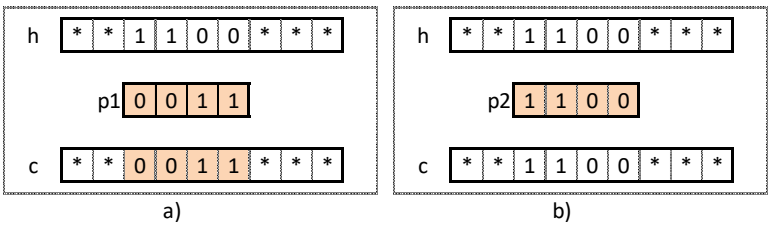


figura 7-9 - Colaboração entre um hospedeiro e um parasita: a) colaboração bem-sucedida b) cooperação rejeitada.

A colaboração de um parasita com um hospedeiro só é possível se o hospedeiro não tiver todos os bits do parasita, figura 7-9 a), e, com esta premissa, definimos que o parasita pode infetar o hospedeiro apenas uma vez. Este procedimento permite a

eliminação de colaborações que não adicionam nada de novo ao indivíduo, abrindo espaço a outras que adicionam material genético novo.

Restringimos a aplicação de múltiplos parasitas aos casos em que os parasitas não têm bits incompatíveis. Isto significa que os parasitas se podem sobrepor, desde que o segmento sobreposto não contenha bits diferentes, figura 7-10.

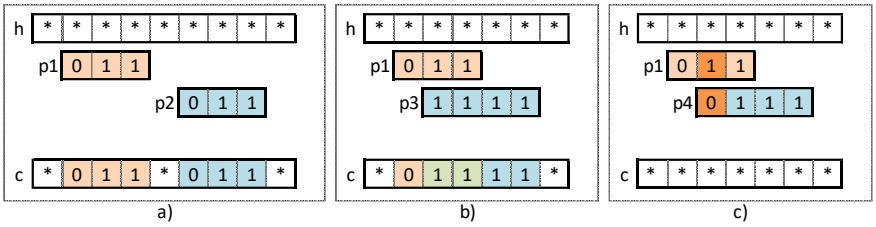


figura 7-10 - Infeção de um hospedeiro por dois parasitas: a) parasitas não sobrepostos, b) parasitas sobrepostos compatíveis c) parasitas sobrepostos incompatíveis

Na figura 7-10 a), os parasitas *p1* e *p2* podem infetar o hospedeiro *h* porque infetam regiões disjuntas do genoma. Na figura 7-10 b), os parasitas *p1* e *p3* podem infetar o hospedeiro *h* porque, embora compartilhem dois genes, eles têm o mesmo valor e, portanto, a infeção não causa ambiguidade no seu valor. Na figura 7-10 c), os parasitas *p1* e *p2* não podem ser usados simultaneamente, porque têm dois genes sobrepostos com alelos distintos. Neste último caso, o hospedeiro pode ser infetado por qualquer um deles, mas não por ambos simultaneamente.

O algoritmo 7-4 controla a formação de colaborações entre uma população de hospedeiros e uma população de parasitas. Ele tem como parâmetros uma multipopulação de hospedeiros, *sortedhostpop*, ordenada por ordem decrescente do seu grau de aptidão, uma população de parasitas, *parasitepop***,** e um parâmetro *n*, que controla a probabilidade de infeção. A ordenação da população dos hospedeiros é importante, porque a sua ordem determina a probabilidade de este receber parasitas. Os piores hospedeiros têm maior probabilidade de receber parasitas relativamente aos mais bem-adaptados.

O algoritmo começa pela inicialização de uma nova população, *symbpop*, que vai suportar o resultado da colaboração entre as populações, passadas por parâmetro. Na fase seguinte, os hospedeiros são selecionados sequencialmente, e a probabilidade de infeção é calculada com base na sua posição na população. Como os hospedeiros são multi-indivíduos, o algoritmo faz a sua expansão em clones e aplica parasitas a cada um deles de forma independente. Assim, os indivíduos mais bem-adaptados são aqueles

que costumam ter mais cópias e, deste modo, podem sofrer várias combinações de parasitas.

```
Colaboration (parasitesPop, sortedHostsPop, n)
    symPop = nova MultiPopulacão
    para index = 1 para Sortedhostspop.size
        host = sortedHostsPop.get(index)
        pInfection = (index/hosts.size)^n
        para copy = 1 até host.copies
            simbiont = host.genotype
            baralhar parasitas na parasitesPop
            para cada parasita de parasitesPop
                se compatível(parasita, simbiont) e
                       uniformRandom (0,1) < pInfection
                    simbiont = simbiont + parasita
                    adicionar clone de simbiont a symPop
              fim se
          próximo parasita
      próxima cópia
    próximo índice
    retornar symbPop
fim
```

algoritmo 7-4 –colaboração entre hospedeiros e parasitas

Depois de selecionar o hospedeiro e calcular a probabilidade de infeção, o algoritmo continua com a aplicação de parasitas a cada um dos seus clones: os parasitas são dispostos aleatoriamente dentro da sua população para garantir que não há preferência na sua aplicação. Na etapa seguinte, o algoritmo tenta aplicar cada parasita ao hospedeiro selecionado, usando as regras de compatibilidade. A fim de preservar os bons indivíduos da população de uma infeção generalizada, originando uma mudança súbita do seu genoma, os parasitas são introduzidos de forma probabilística. Um hospedeiro é particularmente vulnerável a parasitas quando a sua classificação na população é menor, pelo que estão sujeitos a uma infeção generalizada acomodando vários parasitas. Em sentido inverso, os hospedeiros mais aptos recebem poucos parasitas, preservando assim seus genes. Este processo é semelhante ao descrito em (Dumeur 1996).

$$p_{infection}(h, pop) = \left(\frac{rank(h,pop)}{pop.size}\right)^n \qquad 7.2$$

A equação 7.2 mostra a fórmula para calcular a probabilidade de infeção de um hospedeiro, ***h,*** contido dentro de uma população, *pop*. A função *rank* retorna a posição do indivíduo dentro da população, ordenada por ordem decrescente de aptidão, e *pop.size* representa o número de hospedeiros que a população tem. Finalmente, o parâmetro ***n*** controla a forma da proporção descrita acima através da exponenciação.

A população de *symbPop* é construída por clones de hospedeiros infetados por parasitas. Quando um parasita é aplicado a um hospedeiro, o seu genoma é copiado para o do hospedeiro gerando um novo indivíduo através da simbiose. Um clone dessa colaboração é adicionado à população simbionte, e a simbiose continua o processo de infeção por outros parasitas.

7.5.2 SMuGA - SymbioGenetic MultiSet Algorithm

O algoritmo 7-5, SMuGA, evolui duas multipopulações representativas dos hospedeiros e dos parasitas através de simbiogénese. Este algoritmo tem seis parâmetros: *h* representa o tamanho da população hospedeira; *p*, o tamanho da população dos parasitas; *problema*, o problema a ser resolvido; *iterações*, o número de iterações em que hospedeiros e parasitas evoluem sem colaboração; *k*, o número de hospedeiros selecionados para participar na colaboração; e *n* controla a probabilidade de infeção.

```
SMuGA (h, p, problem, iterations, k, n)
    hPop = gerar e avaliar h hospedeiros de problem
    pPop = gerar e avaliar p parasitas de problem
    Repetir
      //fase da evolução
      Repetir iterations vezes
            Evoluir hPop
            Evoluir pPop
      //fase da colaboração
      selPop = selecionar k hospedeiros de hPop
      symbPop = colaboration(pPop, selPop, n)
      hPop = selecionar h hospedeiros de symbPop e hpop
  Até atingir critério de paragem
End Function.
```

algoritmo 7-5 –SMuGA - SymbioGenetic MultiSet Algorithm

O algoritmo começa por gerar e avaliar a população dos hospedeiros, *hPop*, com *h* elementos aleatórios de *problema*, e uma população de parasitas aleatórios, *pPop*, com *p* parasitas.

O processo iterativo começa com a fase de evolução, onde as duas populações evoluem os seus elementos, seguida pela fase de colaboração, onde os parasitas infetam os hospedeiros, até que um critério de paragem seja atingido. Na fase da evolução os hospedeiros e os parasitas evoluem através do algoritmo MuGA durante o número de gerações definidas no parâmetro *iterações*. As duas populações evoluem de forma autónoma; no entanto, os parasitas necessitam dos hospedeiros para fazerem a avaliação da sua aptidão, figura 7-11.

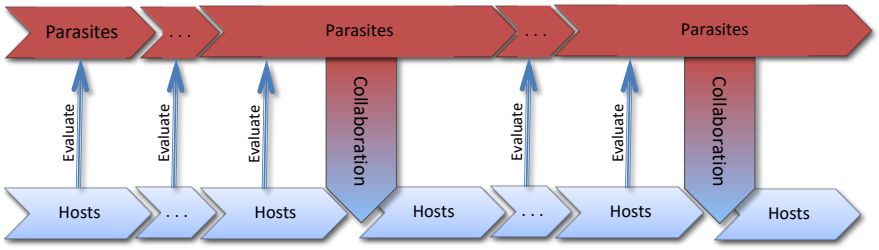


figura 7-11 Interação entre as populações dos hospedeiros e dos parasitas no SMuGA

A fase de colaboração é realizada pelo algoritmo 7-4 entre a população de parasitas, *pPop*, e os *k* hospedeiros, *selPop*, selecionados na população *hPop*, utilizando o parâmetro *n* para controlar a probabilidade de infeção. O resultado do algoritmo 7-4 é uma população de simbiontes, *symbPop*, que contém os clones dos hospedeiros selecionados, depois de infetados pelos parasitas. Como um hospedeiro pode ser infetado por muitos parasitas e o algoritmo guarda os seus clones, o número de simbiontes é enorme comparativamente ao número de parasitas e ao número de hospedeiros. Esta fase é computacionalmente dispendiosa, mas o esforço é aliviado pelo uso de multipopulações, pois o processo de colaboração produz muitos genótipos iguais que são armazenados no número de cópias dos multi-indivíduos, facilitando o processo de avaliação. Esta fase termina com a seleção de *h* hospedeiros, escolhidos da população de *symbPop* e de *hPop*.

Esta fase da evolução é utilizada para estabilizar os indivíduos nas populações e assimilar, nos hospedeiros, o material genético introduzido pela fase de colaboração.

7.6 Estudo experimental

Para examinar a influência da simbiose na otimização de problemas difíceis realizamos um conjunto de experiências com o algoritmo SMuGA. O objetivo deste

estudo é aferir a capacidade de o algoritmo lidar com problemas enganosos, *deceptive*, e em comparar os resultados com o estado da arte.

7.6.1 Configuração experimental

O MuGA foi configurado com 128 MI na população principal. A seleção é feita por torneios de dimensão 3, e o operador seleciona o dobro dos indivíduos da população, *MTournamentSelection* (*3, 2.0*). A dimensão do torneio aliada ao grande número de progenitores selecionados, 256, faz com que a população dos progenitores possua MI com várias cópias. A recombinação é feita pelo operador de cruzamento tradicional, com um ponto de corte, e é utilizado com probabilidade de 0.6. A mutação é realizada em onda e configurada com rugosidade 2 e fineza 3, equação 7.1. A probabilidade de mutação mínima, parâmetro *minprob* do algoritmo 7-2 , é igual a *1/l* , onde *l* representa o tamanho em bits do genoma do indivíduo. Para controlar o número de cópias dos MI, foi usado o operador *AdaptiveCeiling* com uma proporção máxima de 2.

O SMuGA foi configurado com 32 MI na população dos hospedeiros e 32 MI na população dos parasitas. Neste caso, podemos usar uma população menor em relação ao MuGA, graças ao aumento da variedade genética introduzida pelos parasitas. O tamanho da população selecionada para fazer a colaboração é de 16 MI, e o parâmetro que controla a probabilidade de infeção, parâmetro *n* da equação 7.2, tem o valor 1. Definimos o número de iterações em isolamento na evolução das populações em 16. A evolução dos hospedeiros usa a seleção por torneio com dimensão 3 e seleciona 32 indivíduos, *MTournamentSelection* (3, 1.0). A recombinação é feita pelo operador de cruzamento uniforme tradicional, com probabilidade 0,6. A mutação, substituição e redimensionamento são feitos da mesma forma nos dois algoritmos. Estes parâmetros, tabela 7-1, foram obtidos e afinados de forma experimental para tirar o melhor partido de cada um dos algoritmos.

Para obter significância estatística realizámos 128 execuções independentes para cada simulação. Em cada simulação, foram geradas populações iniciais de forma aleatória nas várias populações, e utilizámos como critério de paragem da evolução o número de chamadas à função de avaliação. O valor do número de avaliações foi ajustado à dificuldade de otimização de cada problema, de forma a permitir o sucesso do processo evolutivo. Para cada simulação, calculámos a média do número de avaliações para encontrar o ótimo. Atribuímos o número máximo de avaliações às simulações em que o ótimo não foi encontrado. Também calculámos o resultado mais

revelador, que é a taxa de sucesso, ou seja, a percentagem de simulações que atingem o ótimo.

	MuGA		SMuGA	
	Parâmetro	valores	Parâmetro	Valor
População (MI)	Principal	128	Hospedeiros	32
			Parasitas	32
Seleção	MTournamentSelection	3 , 2.0	MTournamentSelection	3 , 1.0
Recombinação	Recombinação 1 corte	0.6	Recombinação uniforme	0.6
Mutação	MWM	2 , 3	MWM	2 , 3
Substituição	Decimation	2	Decimation	2
Redimensionamento	AdaptiveCeiling	2	AdaptiveCeiling	2

tabela 7-1- Configuração dos algoritmos MuGA e do SMuGA

Para comparar os algoritmos, usámos o teste t-student com intervalo de confiança de 95%. Devido ao grande número de simulações, assumimos a normalidade das variáveis. Para cada problema, também comparámos os resultados com outros algoritmos, quando disponíveis na literatura, mas diversas vezes os resultados publicados para esses problemas são imprecisos e as dimensões são geralmente menores. Em alguns casos, apenas os gráficos logarítmicos são impressos, e os resultados aqui apresentados são fruto do nosso melhor esforço de leitura e nunca apresentam a percentagem de execuções que atingem o ótimo.

7.6.2 Resultados experimentais com funções enganosas

A chave para o sucesso dos algoritmos evolutivos é a combinação de blocos construtores de baixa ordem (BB) para formar BB de ordem superior, que eventualmente levarão ao ótimo. Quando a solução não pode ser construída através desta combinação incremental de BB, estamos na presença de problemas enganosos e precisamos de melhorar o processo evolutivo, se pretendemos resolver esses problemas. O conceito de deceção foi introduzido pela primeira vez por Goldberg (David E. Goldberg 1987) e muito trabalho tem sido feito para resolver esta classe de problemas. Ora, o MuGA, o SMuGA e o SymbCoev são algoritmos evolutivos que são capazes de otimizar funções enganosas. Nas próximas seções, apresentamos resultados experimentais em diferentes funções enganosas.

Função totalmente enganosa F3

Goldberg concebeu uma função de 3 bits, ***F3***, apresentada na equação 7.3, que é totalmente enganosa, uma vez que os blocos de construção da ordem *n* são enganosos para construir blocos de ordem *n + 1* (David E. Goldberg 1989).

F3 (000) = 28, F3 (001) = 26, F3 (010) = 22, F3 (011) = 0 (4) 7.3

F3 (100) = 14, F3 (101) = 0, F3 (110) = 0, F3 (111) = 30

A função *F3* é facilmente resolvida pelo SGA devido ao seu pequeno tamanho de três bits. Para obter um problema desafiador, definimos a função *F3 10* como dez cópias consecutivas de *F3*. Este procedimento é usual na otimização deste tipo de problemas e adequa-se a ser resolvido através da simbiogénese com a migração de parasitas dentro do genoma do hospedeiro.

	SMuGA		MuGA	
F3 10	Média	Desv.Pad.	Média	Desv.Pad.
Evals. to find Best	3309,79	1273,36	6074,30	2516,68
Best value found	300,00	0,00	300,00	0,00
Sucess rate (%)	100,00	0,00	100,00	0,00

tabela 7-2- Resultados da otimização da função F3 10 com os algoritmos SMuGA e o MuGA

A otimização do problema *F3 10* alcançou uma taxa de sucesso de 100% nos dois algoritmos com populações baseadas em multisets, tabela 7-2; no entanto, podemos reparar que a abordagem simbiótica acelera significativamente o processo evolutivo.

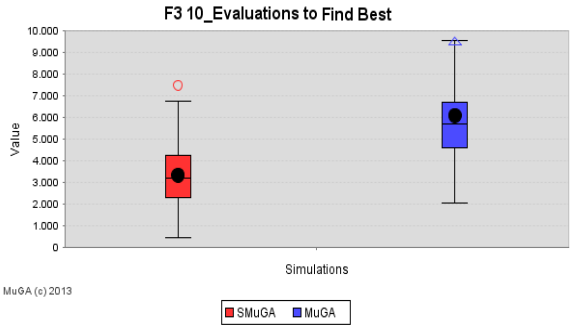


figura 7-12 - Gráfico boxplot do número de avaliações necessárias para otimizar a função F3 10.

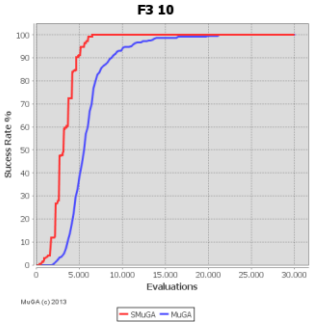


figura 7-13 - Evolução da taxa de sucesso na otimização da função F3 10.

A figura 7-13 mostra a evolução da taxa de sucesso dos algoritmos nas primeiras 30.000 chamadas de função de avaliação, e a figura 7-12 apresenta uma visão estatística do número de chamadas de função de avaliação necessárias para atingir o ótimo em

ambos os algoritmos. Os resultados do SMuGA na função *F3 10*, em várias ordens de magnitude, são melhores do que aqueles apresentados por (Yang 2004) e (Chen et al. 2008)

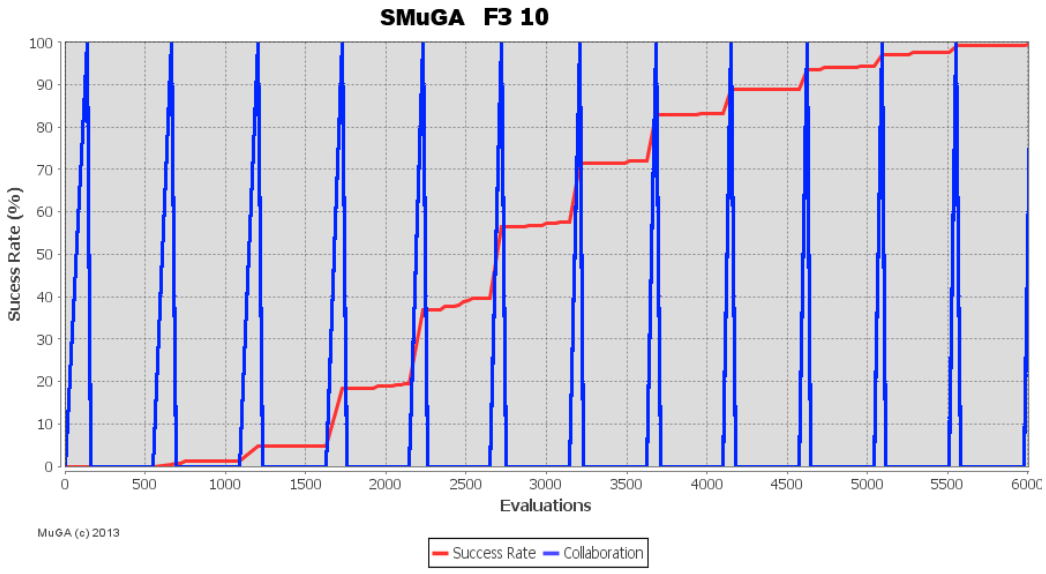


figura 7-14 Detalhe da evolução da taxa de sucesso na otimização de 10 cópias da função F3 com o algoritmo SMuGA . A linha azul representa o evento de colaboração entre hospedeiros e parasitas.

A figura 7-14 mostra detalhadamente a evolução da taxa de sucesso, observando apenas as primeiras 6.000 chamadas de função de avaliação. Nesta figura podemos ver claramente o efeito da incorporação periódica de parasitas nos hospedeiros, quando novas colaborações são formadas e integradas na população. A evolução isolada da população dos hospedeiros ao longo de algumas gerações permite a disseminação de bom material genético introduzido por simbiontes nos indivíduos da população. A população dos parasitas evolui paralelamente, e de forma isolada, utilizando a população dos hospedeiros para estimar o valor de aptidão dos seus elementos. Este processo de avaliação dos parasitas economiza muito poder computacional, porque não precisa de colaborações entre as duas populações.

A Função *F3 10* é resolvida facilmente pelo SMuGA através da utilização da simbiose entre hospedeiros e parasitas. Se um parasita que representa um BB da função for encontrado, este pode migrar para a posição onde outro BB começa. A busca de BB e as suas posições não são fáceis de determinar, porque não é fornecida qualquer informação sobre a estrutura da função ao algoritmo. De modo notável, o SMuGA encontra o comprimento adequado dos BB e as suas posições e usa a simbiose de uma forma muito eficiente.

A fim de verificar a escalabilidade do SMuGA na otimização de genomas de grande dimensão, realizamos um conjunto de testes com a composição de 10, 20, 40, 80, e 160 blocos da função F3, que correspondem a problemas com 30, 60, 120, 240 e 480 bits, respetivamente. Para estes testes só apresentamos resultados para o SMuGA, já que, em problemas de grande dimensão, a MuGA não encontra a solução em tempo razoável, e a literatura não fornece resultados de algoritmos com estas dimensões.

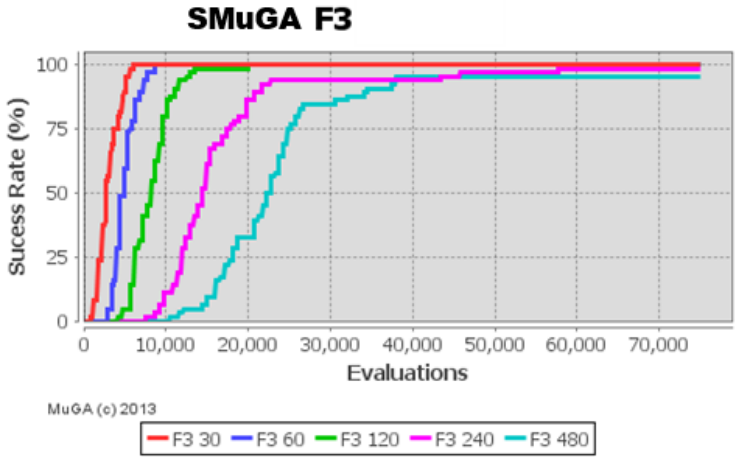


figura 7-15- Evolução da taxa de sucesso na otimização de 10, 20, 40, 80 e 160 cópias da função F3 com o SMuGA

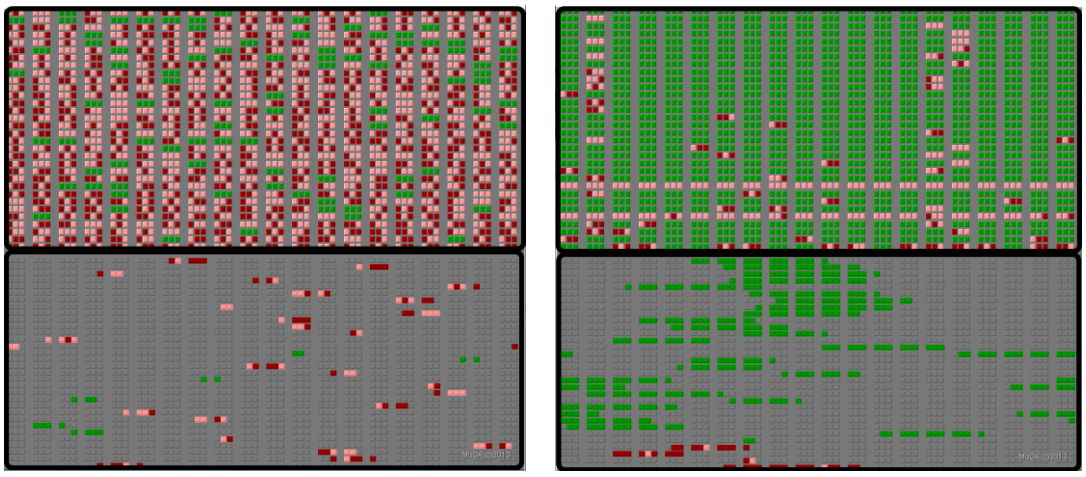
A figura 7-15 mostra a evolução da taxa de sucesso na otimização de função F3 com vários blocos concatenados ao longo de 75.000 chamadas à função de avaliação. A tabela 7-3 mostra o resultado da simulação, apresentando o número de chamadas necessárias para otimizar as funções e a taxa de sucesso da otimização.

SMuGA	Evals. to find best		Sucess %	
F3	Média	Desv.Pad.	Média	Desv.Pad.
30 bits	3088,69	1363,68	100,00	0,00
60 bits	5054,30	1495,08	100,00	0,00
120 bits	8457,08	3021,80	100,00	0,00
240 bits	17500,25	12182,22	98,44	12,40
480 bits	24960,44	13143,26	95,31	21,14

tabela 7-3- Estatísticas do SMuGA na otimização da função de F3 concatenada com diferentes comprimentos.

O algoritmo tem um comportamento linear na otimização desta função, pois o número de chamadas é função de avaliação é proporcional ao tamanho do cromossoma. O SMuGA consegue encontrar pequenos BB através dos parasitas que infetam a população dos hospedeiros. Estes parasitas movimentam-se no genoma mudando a posição de ancoragem, permitindo-lhes infetar o hospedeiro em várias zonas. Os pequenos parasitas crescem através da recombinação entre eles e, se forem benéficos

para a população, infetam regiões cada vez maiores. Se a solução puder ser construída através da descoberta e movimentação de BB no genoma, o SMuGA executa estas tarefas de forma exímia.



a) b)

figura 7-16- Evolução simbiótica da função F3 20: a) Populações iniciais b) Populações finais

A figura 7-16 mostra um exemplo de um sistema simbiótico simulado no SMuGA, no início e no final da sua evolução. O sistema é constituído por uma população de hospedeiros, no topo da figura, e de parasitas, em baixo. Os bits a verde representam os blocos construtores, e os avermelhados os restantes bits. No início da evolução, os parasitas ocupam apenas alguns bits, mas, no final da evolução, o seu tamanho é consideravelmente maior. A figura 7-18 mostra a evolução do tamanho médio dos parasitas que representam BB ao longo da evolução para as simulações da tabela 7-3. Como podemos ver na figura, os problemas com genomas longos são também resolvidos por parasitas com genomas longos, que infetam os hospedeiros através da colaboração, acelerando deste modo a sua evolução. Mais uma vez, notamos que o algoritmo não recebe qualquer informação sobre o BB.

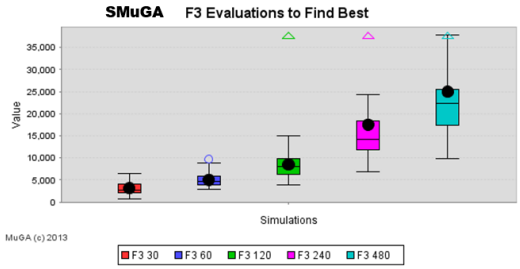


figura 7-17.Gráfico boxplot do número de avaliações necessárias para otimizar a função F3 10.

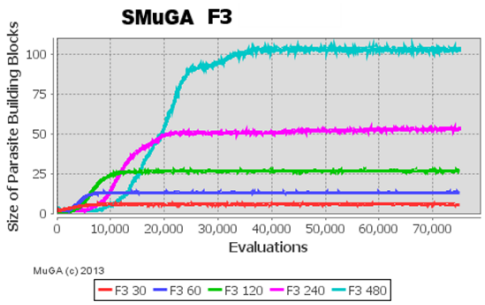


figura 7-18 Evolução do tamanho dos parasitas que representam blocos construtores (BB).

Na tabela 7-3 podemos verificar a diminuição da eficácia do SMuGA nas configurações de F3 com 240 bits (98%) e 480 bits (95%). Esta diminuição pode ser explicada pelo tamanho diminuto da população dos parasitas (32 indivíduos) quando comparado com o imenso genoma dos hospedeiros. Neste caso, a probabilidade de agrupar pequenos parasitas, que representam BB, é muito menor por causa do comprimento do genoma onde eles se movimentam, uma vez que eles só se agrupam em posições contíguas, ou se partilharem posições no genoma. Nas simulações das secções seguintes, o SMuGA apresenta um comportamento semelhante ao descrito e, por isso, omitimos esses dados.

Função enganosa F3 totalmente separada

A composição das funções de forma sequencial é resolvida pelo SMuGA usando a propriedade da mobilidade de parasitas presentes no algoritmo. A aplicação de um bom parasita, que representa um BB, numa posição onde o outro BB começa, contribui para o sucesso do algoritmo, devido à natureza da composição da função. O problema é mais complicado quando os bits de cada função são separados. O caso mais difícil de separação ocorre quando os bits são uniformemente distribuídos no cromossoma e em distância máxima. Chamamos a essas funções *F3S n*, onde *n* representa o número de funções *F3* no cromossoma. No caso de *F3S 10*, cada bit de uma função está localizado nas posições *i*, *i + 10* e *i + 20*.

Estas funções são difíceis de otimizar, porque o problema não é separável e a formação do BB não é possível com uma estratégia simples. Desta forma, os bits das funções estão espalhados no cromossoma e a aplicação de um parasita em posições diferentes não é suficiente para resolver o problema. O SMuGA contorna esta situação, combinando vários parasitas num único hospedeiro. Com esta experiência, verificamos a eficácia de SMuGA em problemas não separáveis.

	SMuGA		MuGA	
F3S 10	Média	Desv.Pad.	Média	Desv.Pad.
Evals. to Find Best	9419,20	5604,92	34063,77	18018,62
Best value found	300,00	0,00	299,95	0,30
Sucess rate (%)	100,00	0,00	99,22	8,80

tabela 7-4- Estatísticas do SMuGA e do MuGA na otimização da função de F3S 10

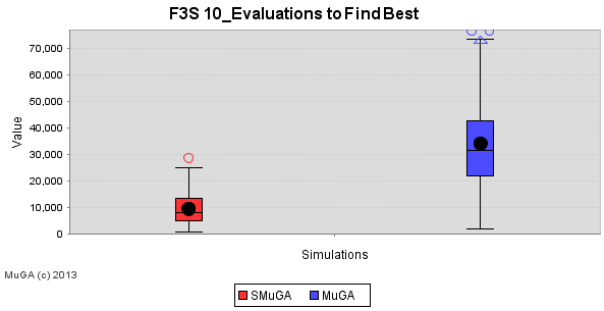


figura 7-19.Gráfico boxplot do número de avaliações necessárias para otimizar a função FS3 10

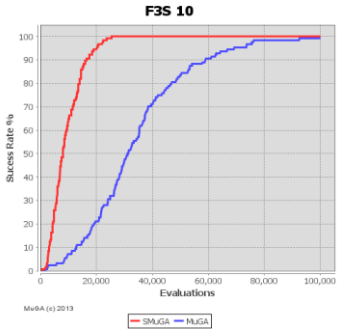


figura 7-20 Evolução da taxa de sucesso na otimização da função FS3 10.

A tabela 7-4 apresenta os resultados da otimização de *F3S 10*. Ambos os algoritmos conseguem resolver o problema com uma eficácia notável e, novamente, a simbiogénese acelera o processo evolutivo. A figura 7-19 mostra o resultado do número de avaliações necessário para otimizar a função, enquanto a figura 7-20 apresenta a sua evolução ao longo de 100.000 avaliações. O SMuGA obtém uma taxa de sucesso de 100%, com um número de avaliações várias vezes menor do que as do MuGA, cuja taxa de sucesso é 99,2%. O SMuGA não é só mais eficiente, como é também muito mais eficaz.

SMuGA	Evals. to find best		Sucess %	
F3S	Média	Desv.Pad.	Média	Desv.Pad.
30 bits	7651,08	5049,29	100,00	0,00
60 bits	26429,98	19408,04	100,00	0,00
120 bits	184549,02	145359,55	95,31	21,14
240 bits	381741,92	148172,39	51,56	49,98
480 bits	496565,75	27516,57	1,56	12,40

tabela 7-5- Estatísticas do SMuGA na otimização da função de F3S com diferentes comprimentos.

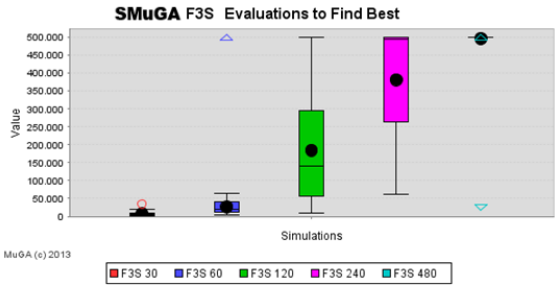


figura 7-21.Gráfico boxplot do número de avaliações necessárias para otimizar a função F3S

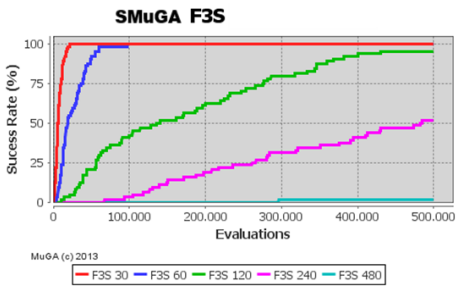


figura 7-22 Evolução da taxa de sucesso na otimização da função F3S.

A figura 7-21 e a figura 7-22 mostram as estatísticas da otimização da função F3S com 10, 20, 40, 80 e 160 blocos espalhados no cromossoma da função F3. Como

descrito anteriormente, os bits de cada bloco da função F3 são distribuídos no cromossoma com a distância máxima entre eles. Quanto mais blocos a função tiver, mais separados estarão os bits de cada bloco. O SMuGA consegue otimizar com sucesso pleno as funções F3S com 10 e 20 blocos. Na otimização da versão com 40 blocos, cujo cromossoma tem 120 bits e os bits de cada função F3S são separados por 40 bits, o SMuGA é bem sucedido em 95% de simulações, necessitando, todavia, de mais gerações para otimizar todas as funções, como se pode inferir ao analisar a velocidade de convergência da simulação na figura 7-22.

Na simulação com cromossomas de 480 bits, a pequena população de parasitas e o grande genoma dos hospedeiros dificultam a otimização, devendo, por isso, ser ajustados os parâmetros do algoritmo para melhorar o seu desempenho.

Funções enganosas unitárias

As funções enganosas unitárias, referidas na literatura como *Trap* ou *Deceptive*, foram introduzidas por (Ackley 1987) e contam o número de alelos com o valor 1 no cromossoma, não obstante a ordem. A equação 7.4 apresenta a fórmula de uma função enganosa, onde $\boldsymbol{x}$ é o cromossoma, $u\ (x)$ é o número de uns no cromossoma x,e l representa o comprimento de cromossoma x. Embora existam outros modelos com resultados publicados, a equação 7.4 permite-nos testar os algoritmos de funções enganadores com blocos de vários bits.

$$deceptive(x) = \begin{cases} u(x) & if\ u(x) > 0 \\ l + 1 & if\ u(x) = 0 \end{cases} \qquad 7.4$$

Nesta experiência, usamos a função *Deceptive 16 4*, que concatena 16 blocos de 4 bits da função enganosa 7.4, representando um cromossoma com 64 bits. A tabela 7-6 mostra os resultados do MuGA e SMuGA na otimização da função após 100.000 avaliações. O SMuGA otimiza todas as simulações com um número de chamadas da função de avaliação menor, quando comparado ao MuGA. A figura 7-23 apresenta a evolução da taxa de sucesso ao longo da evolução, e podemos notar que o MuGA tem grandes dificuldades em otimizar a função, enquanto o SMuGA a otimiza facilmente. Mais uma vez, a evolução simbiótica torna o algoritmo mais eficiente e eficaz.

Deceptive 16 4	SMuGA		MuGA	
	Mean	Std	Mean	Std
Evals. to Find Best	5431,04	3724,07	84749,20	30440,50
Best value found	80,00	0,00	78,34	1,43
Sucess rate (%)	100,00	0,00	27,34	44,57

tabela 7-6- Estatísticas do SMuGA e do MuGA na otimização da função de Deceptive 4 16.

Comparando os resultados com SymbCoev apresentado em (Wallin, Ryan, e Azad 2005), o SymbCoev precisa de centenas de milhares de avaliações de função, enquanto o SMuGA necessita apenas de 5.431. A capacidade do SMuGA na manipulação do conteúdo e do tamanho do genoma dos parasitas é a chave para resolver este tipo de problemas. O SymbCoev não tem essa propriedade, e o tamanho estático dos parasitas retarda a evolução.

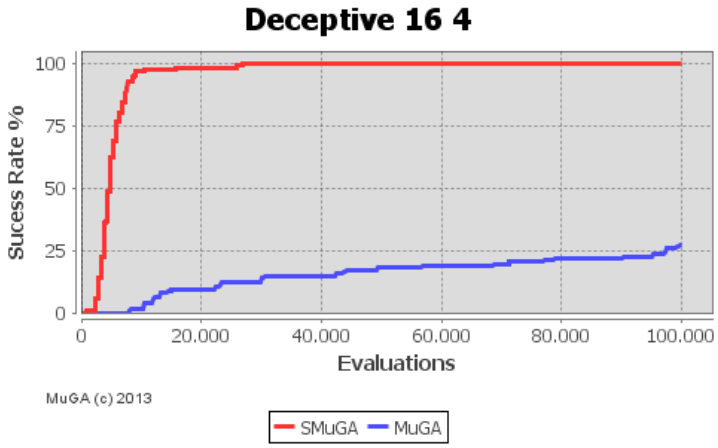


figura 7-23 Evolução da taxa de sucesso na otimização da função Deceptive 16 4

SMuGA	Evals. to find best		Sucess %	
Deceptive 4	Mean	Std	Mean	Std
64 bits	4243.00	1645.40	100.00	0.00
128 bits	7061.89	3215.18	100.00	0.00
256 bits	11180.36	4897.55	100.00	0.00
512 bits	21458.61	13696.15	96.88	17.40

tabela 7-7- Estatísticas do SMuGA na otimização da função Deceptive 4 com diferentes comprimentos.

A tabela 7-7 e a figura 7-24 mostram as estatísticas da evolução do SMuGA após 75.000 chamadas à função de avaliação para problemas compostos por 16, 32, 64 e 128 funções *deceptive*, que representam genomas com 64, 128, 256 e 512 bits. O algoritmo foi bem sucedido em todas as simulações; porém, na função com 512 bits, teve algumas dificuldades na otimização devido ao grande genoma do hospedeiro, e seriam

necessárias mais gerações para otimizar a totalidade das simulações, como se pode inferir da figura 7-25.

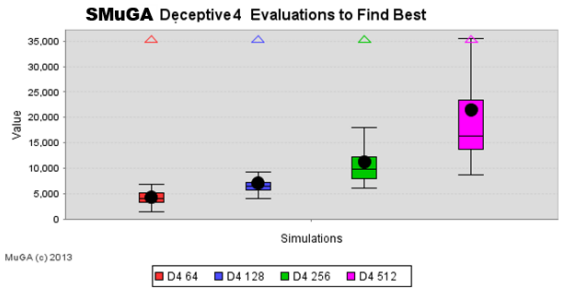


figura 7-24.Gráfico boxplot do número de avaliações necessárias para otimizar a função Deceptive 4

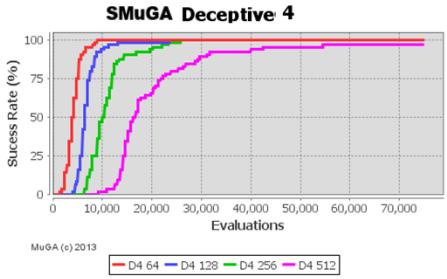


figura 7-25 Evolução da taxa de sucesso na otimização da função Deceptive 4.

O SMuGA escala muito bem para otimizar a função *deceptive*, e os resultados são novamente significativamente melhores do que os apresentados em (Wallin, Ryan, e Azad 2005), usando o Symbiotic Genetic Algorithm, SymbCoev, e por (Dirk Thierens 2010), usando o *Linkage Tree Genetic Algorithm*, LTGA. Os artigos referidos não fornecem valores detalhados das simulações e os valores apresentados na tabela 7-8 são fruto do nosso melhor esforço para ler os gráficos fornecidos.

Algoritmo	Dimensão do problema	Nº. de Avaliações
SymbCoev	16	100000
	128	200000

Algoritmo	Dimensão do problema	Nº. de Avaliações
LTGA	60	40000
	100	75000

tabela 7-8- Número de chamadas de avaliação de funções para resolver a função Deceptive 4 usando o algoritmo SymbCoev e LTGA (aprox.).

Funções enganosas entrelaçadas

A função enganosa entrelaçada DPI, figura 7-26, proposta em (Wallin, Ryan, e Azad 2005), é definida como a combinação de duas funções enganosas, equação 7.4, onde os bits estão entrelaçados na mesma função. A função DPI possui muitos extremos locais introduzidos pela combinação das duas funções enganosas. Nesta simulação, usamos a função D4PI, onde duas funções enganosas de quatro bits se entrelaçam formando um bloco construtor de 8 bits.

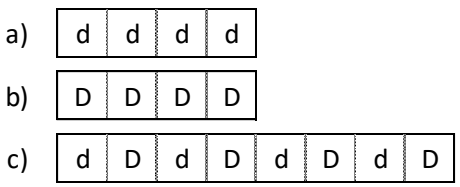


figura 7-26 - Função enganosa entrelaçada D4PI: a) função enganosa d; b) função enganosa D; c) Função enganosa D4PI.

A tabela 7-9 apresenta as estatísticas da otimização de 8 funções D4PI, num total de 64 bits, após 100.000 chamadas de função de avaliação; a figura 7-27 mostra a evolução da taxa de sucesso nos dois algoritmos. O SMuGA otimiza todos os problemas sem grande esforço, devido à capacidade já demonstrada pelo parasita em descobrir BB, combiná-los e espalhá-los pelo cromossoma. O sucesso do MuGA nesta simulação é muito limitado, devido ao grande comprimento do BB e ao tamanho do genoma dos indivíduos. Comparando os resultados com SCA, apresentado em (Wallin, Ryan, e Azad 2005), na tabela 7-10, podemos constatar que o SMuGA continua a ser muito melhor.

	SMuGA		MuGA	
D4PI 8	Mean	Std	Mean	Std
Evals. to Find Best	12401.04	12114.49	97294.48	12106.18
Best value found	80.00	0.00	76.88	1.61
Sucess rate (%)	100.00	0.00	6.25	24.21

tabela 7-9- Estatísticas do SMuGA e do MuGA na otimização da função de D4PI 8.

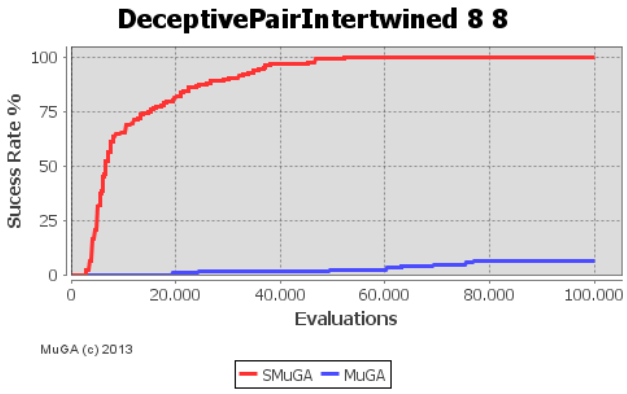


figura 7-27 Evolução da taxa de sucesso na otimização da função D4PI 8.

Algorithm	Size	Evals.
SCA	64	150000
	128	250000

tabela 7-10- número de chamadas de avaliação de funções para resolver a função D4PI 8 usando o algoritmo SCA(aprox.).

Para verificarmos a escalabilidade do SMuGA na função D4PI, aumentamos o número de blocos concatenados. A tabela 7-11 e a figura 7-28 mostram as estatísticas da evolução após 500.000 chamadas à função de avaliação, para os problemas compostos por 8, 16, 32 e 64 funções D4PI, que representam cromossomas com 64,

128, 256 e 512 bits. A figura 7-29 apresenta a evolução da taxa de sucesso ao longo da evolução. O SMuGA foi eficaz em todas as simulações; porém, no problema com 512 bits, o algoritmo mostrou algumas dificuldades na otimização devido à elevada dimensão do genoma. A definição correta do tamanho das populações para um genoma tão largo resolve o problema, como foi constatado experimentalmente.

SMuGA	Evals. to Find best		Sucess %	
D4PI	Mean	Std	Mean	Std
64 bits	9246.20	9045.93	100.00	0.00
128 bits	20050.59	21544.04	100.00	0.00
256 bits	56673.89	84180.82	100.00	0.00
512 bits	204222.19	206614.44	79.69	40.23

tabela 7-11- Estatísticas do SMuGA na otimização da função D4PI com diferentes comprimentos.

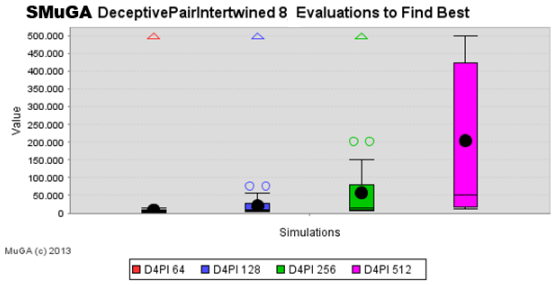


figura 7-28.Gráfico boxplot do número de avaliações necessárias para otimizar a função D4PI

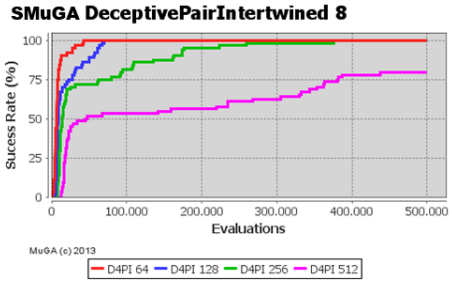


figura 7-29 Evolução da taxa de sucesso na otimização da função D4PI.

Funções enganosas entrelaçadas 0-1

As funções otimizadas nas simulações anteriores possuem um ótimo, composto por BB que contêm sequências de uns. O SMuGA demonstrou ser um algoritmo que lida facilmente com este tipo de funções. Na avaliação da capacidade do SMuGA para evoluir outro tipo de BB, definimos uma nova função entrelaçada, DPI01, onde uma função é avaliada pela equação 7.4 e outra pela equação 7.5. A função *Deceptivez, z(x)* conta o número de zeros no cromossoma ***x***. O ótimo da função DPI01 é composto por uma sequência alternada de zeros e uns. Esta função é mais difícil do que as anteriores, porque os parasitas precisam de estar alinhadas no cromossoma para serem úteis, e a simples deslocação de 1 bit na posição do parasita torna-o inútil.

$$deceptiveZ(x) = \begin{cases} z(x) & if\ z(x) > 0 \\ l+1 & if\ z(x) = l \end{cases} \qquad 7.5$$

A figura 7-30 mostra a evolução da taxa de sucesso ao longo de um milhão de avaliações para os problemas compostos por 8,16, 32 e 64 funções DPI01, funções que representam genomas com 64, 128, 256 e 512 bits.

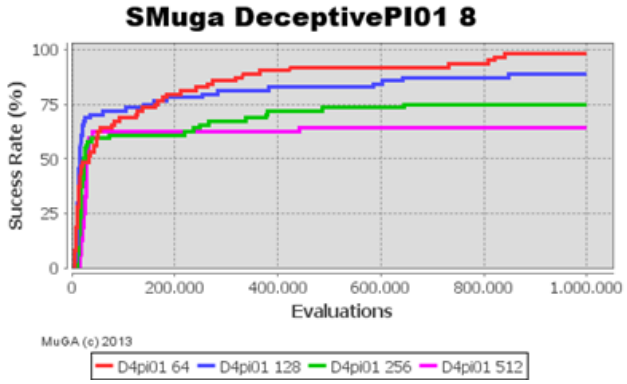


figura 7-30. SMuGA: Evolução da taxa de sucesso na otimização de 8, 16, 32 e 64 cópias da função D4PI01 com 8 bits

Usando os parâmetros definidos, o MuGA manifesta um desempenho fraco e por isso omitimos os resultados nesta experiência. O SMuGA, na maioria das simulações, otimiza a função DeceptivePI01 composta por oito blocos de bits, quatro da equação 7.4 e quatro da equação 7.5 intercaladas, como se pode ver na figura 7-30. Uma razão para as falhas poderia ser encontrada no número reduzido de parasitas na população parasitária.

A necessidade de alinhamento do parasita com o hospedeiro implica uma população maior de parasitas, para evitar os extremos locais introduzidos pelo padrão de bits das funções DeceptivePI0. Os dois extremos locais, ou seja, BB com os bits todos a zero e todos a um, são mais atraentes para os parasitas, porque esse padrão não precisa de alinhamento, e os parasitas são facilmente assimilados pelos hospedeiros.

A tabela 7-12 e a figura 7-32 apresentam a evolução da função D4PI01 na mesma situação da figura 7-30, mas agora com 128 elementos na população parasita, em vez de 32. O sucesso do algoritmo é maior, e as simulações evoluindo funções com 64, 128 e 256 bit têm uma taxa de 100%. A taxa de sucesso da simulação com 512 bits também aumenta, não atingindo, porém, os 100% de sucesso. O ajuste adicional do parâmetro é uma solução possível para conseguir a eficácia plena.

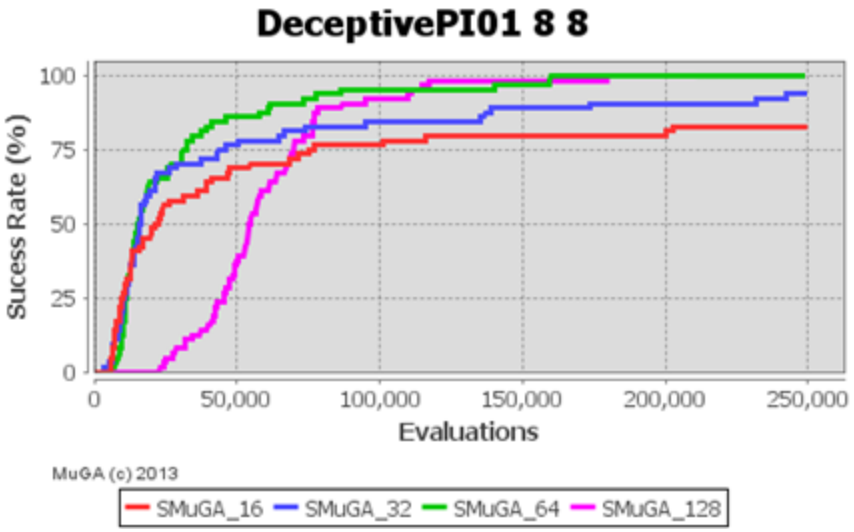


figura 7-31. - SMuGA: Evolução da taxa de sucesso na otimização de 8 cópias da função DeceptivePI01 com 8 bits com 16, 32, 64 e 128 elementos na população dos parasitas.

A figura 7-31 mostra o efeito do tamanho da população dos parasitas na otimização de 8 blocos da função DPI01 com 8 bits. Como se pode verificar, o aumento do número de parasitas no sistema simbiótico aumenta a robustez do algoritmo. O crescimento da população parasitária amplifica a complexidade computacional do algoritmo, mas a população parasita pode evoluir paralelamente à população hospedeira explorando o paralelismo dos computadores.

SMuGA	Evals. to Find best		Sucess %	
D4PI01	Mean	Std	Mean	Std
64 bits	61204.72	29939.75	100.00	0.00
128 bits	54285.36	111617.89	100.00	0.00
256 bits	107131.61	187231.57	100.00	0.00
512 bits	350228.05	421266.94	73.44	44.17

tabela 7-12- Estatísticas do SMuGA na otimização da função D4PI com diferentes comprimentos.

Estes resultados mostram que, quanto maior for o tamanho da população do parasita, mais o SMuGA se mostra robusto na evolução de funções difíceis. Padrões complexos de bits dificultam a otimização com o SMuGA, devido ao alinhamento entre os parasitas e os hospedeiros, mas esta dificuldade pode ser contornada utilizando populações maiores de parasitas.

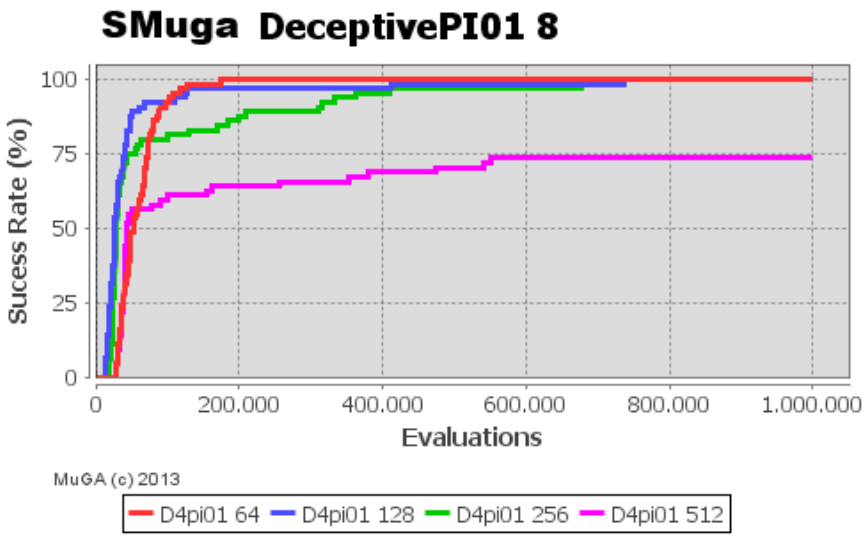


figura 7-32 - SMuGA: Evolução da taxa de sucesso na otimização de 8, 16, 32 e 64 cópias da função DeceptivePI01 com 8 bits entrelaçados com populações de 128 parasitas.

7.7 Comentários finais

Neste capítulo apresentámos o algoritmo simbiogenético com populações baseadas em multiset, SMuGA, uma extensão do algoritmo genético MuGA. Este algoritmo tira partido de uma nova aproximação de simbiogénese artificial, onde um hospedeiro recebe o material genético de vários parasitas, e estes têm comprimento variável. Este é o primeiro modelo evolutivo onde os parasitas não têm um comprimento fixo, pelo contrário, ele varia ao longo do processo evolutivo.

O modelo proposto também introduziu um passo de evolução em duas fases, a de colaboração e a evolutiva. Na fase de colaboração simbiótica são gerados novos indivíduos que competem com os anteriores para formar a geração seguinte de hospedeiros. Na outra fase, a evolutiva, as populações hospedeiras e os parasitas evoluem por conta própria durante algumas gerações, mas os parasitas usam a aptidão dos hospedeiros para estimar a sua própria aptidão. Aferir a avaliação do parasita utilizando a avaliação dos hospedeiros diminui significativamente as chamadas à função de avaliação e evita a necessidade de gerar um número exponencial de colaborações. A fase de evolução separada dos hospedeiros e dos parasitas permite estabilizar e assimilar o material genético introduzido pelas colaborações. Na população hospedeira, a fase de evolução fomenta a exploração através do movimento dos parasitas e do aproveitamento pela recombinação.

Os resultados obtidos ultrapassaram largamente os modelos simbiogenéticos conhecidos, permitindo-nos resolver problemas enganosos de grande dimensão. Deve-

se notar que, apesar de o MuGA obter bons resultados, só o SMuGA resolve problemas com genomas longos e complexos, integrando a simbiogénese no MuGA.

Na verdade, o SMuGA acabou por ser tão eficiente que escalou de forma linear o comprimento dos problemas enganosos utilizados para o teste. A variação do comprimento dos parasitas permitiu que a evolução encontrasse blocos de construção de comprimento adequados (BB) para o problema. Na verdade, a acumulação de múltiplos parasitas num único hospedeiro oferece a oportunidade de usar combinações de parasitas, contribuindo para a resolução de problemas mais complexos.

8 Conclusão

O multiset é um tipo especial de coleção que introduz um novo atributo nos seus elementos para agrupar outros repetidos. Os elementos dos multiset são, portanto, tuplos <*valor,cópias*> em que *cópias* representa a quantidade de elementos iguais a *valor* da estrutura.

Nesta tese explorámos a possibilidade de os multisets serem utilizados como suporte das populações de algoritmos evolutivos de forma a incrementar o seu desempenho. Assim, às populações baseadas em multisets chamamos multipopulações e aos seus elementos multi-indivíduos, que são uma extensão dos indivíduos tradicionais, a quem foi adicionado o atributo cópias, representativas da quantidade de clones existentes na multipopulação.

Estipulámos que o número de multi-indivíduos na multipopulação permanece constante ao longo do processo evolutivo e, por isso, os melhores indivíduos vão incrementando o seu número de cópias, para acomodar os clones gerados pelos operadores genéticos. As multipopulações possuem duas características de grande utilidade para o processo evolutivo: a manutenção de uma quantidade de genomas distintos, que incrementa a sua diversidade genética; a permanência dos clones bem-adaptados na população, aumentando a pressão seletiva sobre os bons genomas.

Os benefícios atrás enunciados só se verificam se o número de cópias dos multi-indivíduos for controlado. Para atingir este propósito, introduzimos um novo operador genético no processo evolutivo, o redimensionamento, que reduz o seu número de cópias. Este operador permite fazer o balanceamento entre a diversidade genética da multipopulação introduzida pelo conjunto de genomas únicos e a pressão seletiva introduzida pelo seu número de cópias e, para tal, criamos os operadores *FixedCeiling* e *AdaptiveCeiling*.

Para poder usar multipopulações como suporte da população dos algoritmos evolutivos é necessário fazer a redefinição dos operadores de seleção e de progenitores e o de substituição de gerações. No primeiro caso, apresentámos o operador *MTournamentSelection*, adaptação do operador tradicional de seleção por torneios que usa o número de cópias dos multi-indivíduos para incrementar a pressão sobre o genoma. No segundo caso, propusemos o operador *MTournamentReplacement*, adaptação do operador de substituição, também por torneios, que garante a

incorporação dos clones no número de cópias e a manutenção do número de genomas constante na multipopulação.

Definimos o *Multiset Genetic Algorithm*, MuGA, um algoritmo geracional que usa as multipopulações e os operadores acima definidos para efetuar a evolução. Para avaliar a influência das multipopulações, aplicamos o MuGA a um conjunto de funções muito diversificado, utilizando os operadores de recombinação e mutação tradicionais. Os resultados obtidos foram comparados com o algoritmo genético canónico, SGA, que utiliza populações simples, numa vasta gama de problemas: o *OneMax*, para verificar o comportamento do MuGA em funções unimodais simples; o *Royal Road*, para a otimização de funções com planaltos; o *NK-Landscapes*, para a otimização de funções rugosas; o *Knapsack*, para a otimização de funções com vários ótimos; e, finalmente, o *ChessQueen*, para otimização de permutações com vários ótimos. Em todos os problemas, o MuGA teve melhor desempenho do que o SGA, pelo simples facto de usar multipopulações.

Com os parâmetros escolhidos para o problema *NK-Landscapes* e *Knapsack*, o SGA teve um desempenho sofrível, por causa da deriva genética da população. Esta situação pode ser contornada com algoritmos de nicho e, por conseguinte, alterámos o processo evolutivo para a sua criação. Estabelecemos os procedimentos para a substituição de um indivíduo dentro de uma multipopulação e, utilizando o algoritmo de *crowding*, definimos o *Multiset Crowding Algorithm*, MuCA. Este é um algoritmo de estado estacionário, selecionando dois progenitores na multipopulação e gerando dois descendentes que competem com os pais por um lugar na multipopulação. Com o algoritmo de *crowding*, o SGA incrementou a sua eficácia na procura das soluções ótimas; todavia, o MuCA continua a ser mais eficiente e mais eficaz.

Criámos *ClearingReplacement*, um operador de substituição de populações, inspirado na partilha de recursos por *clearing*, que tem a particularidade de manter um conjunto de genomas únicos na população. Este operador permite às populações simples conservarem, efetivamente, a sua diversidade genética. Introduzimos este operador no MuGA e no SGA e verificámos que ambos os algoritmos otimizam facilmente os problemas *NK-Landscapes* e *Knapsack*; no entanto, o MuGA necessita de menor esforço computacional para a otimização. Esta eficiência só pode ser devida ao número de cópias dos multi-indivíduos, porque ambos os algoritmos usam populações sem genomas repetidos.

Nas experiências anteriores, as multipopulações serviram como suporte da população principal e utilizamos os operadores tradicionais para a geração dos descendentes. O incremento da eficácia e eficiência do processo evolutivo na descoberta do ótimo deve-se, exclusivamente, ao uso das multipopulações. Além disso, em problema com vários ótimos, a exemplo do *ChessQueen* e do *Knapsack*, as multipopulações permitiram que os algoritmos descobrissem esses ótimos e os retivessem eficazmente na população.

O uso de multipopulações para suporte da população dos progenitores dos seus descendentes foi explorada na otimização de cromossomas codificados por números reais. O operador de seleção gera facilmente multi-indivíduos, ao selecionar várias vezes o mesmo genoma, e exploramos o seu número de cópias para desenhar operadores genéticos que tirem partido desse atributo extra. Nesse sentido, desenvolvemos o operador *Multiset Centroid* Crossover, MuCX, que desloca a região de geração de descendentes na direção do multi-indivíduo que tem mais cópias, e o operador de recombinação aritmética para multi-indivíduos, MuAX, que utiliza esse valor para fazer a expansão da área onde os descendentes vão ser gerados. Além disso, desenvolvemos o operador MuGauss, que utiliza o número de cópias para alterar a dimensão e a frequência da mutação nos multi-indivíduos. Introduzimos estes novos operadores no MuGA e os seus resultados obtidos na otimização de problemas de referência são competitivos com os apresentados na literatura.

Para a otimização de problemas enganosos, *deceptive*, desenvolvemos um algoritmo evolutivo que utiliza duas multipopulações, os hospedeiros e os parasitas, que evoluem num sistema simbiótico. A este algoritmo demos o nome *SymbioGenetic Multiset Algorithm*, SMuGA. Este algoritmo utiliza um operador de mutação adaptado para multi-indivíduos que, por sua vez, usa o número de cópias para alterar a probabilidade de mutação sob a forma de onda, *Multiset Wave Mutation*. O SMuGA foi utilizado para resolver problemas enganosos com genomas longos e complexos, e os resultados obtidos são melhores do que os apresentados na literatura para solucionar tais problemas.

Neste trabalho utilizamos os multi-indivíduos para o agrupamento de indivíduos que têm genomas rigorosamente iguais. Este conceito pode ser estendido para o agrupamento de genomas similares, que levanta algumas questões pertinentes acerca do raio de semelhança e das diferenças do genoma. Esta linha de investigação parece-nos muito pertinente para a otimização de espaços contínuos.

As multipopulações são um suporte eficiente de indivíduos, sendo insensíveis em relação ao processo evolutivo, e neste trabalho exploramos a sua aplicação a algoritmos genéticos e a algoritmos de nicho. Os bons resultados obtidos deixam-nos confiantes de que a sua aplicação a outros modelos evolutivos, baseados em populações, será igualmente bem-sucedido. Esta adaptação a novos modelos de evolução abrirá caminhos para a definição de novos operadores genéticos e para a adaptação dos existentes de forma a lidar com a nova classe de indivíduos.

As multipopulações foram testadas num pequeno conjunto de problemas, com características muito diversificadas e representativos da sua classe. O desempenho obtido pelos algoritmos evolutivos que as utilizam, preveem o seu bom desempenho; no entanto, esta intuição deverá ser confirmada com a otimização de grupos maiores e mais diversificados de problemas.

Em relação ao SMuGA, será necessária testar o seu comportamento na otimização de outras classes de problemas, pois aqueles usados neste trabalho são concatenações repetidas da mesma função. Além disso, a flexibilidade deste modelo é um bom indicador da sua adequação a funções onde o ambiente de avaliação é dinâmico, pelo que deverá ser testado nesse tipo de problemas.

As multipopulações introduzem parâmetros novos no sistema evolutivo que podem ser ajustados para aumentar a sua eficácia, e no futuro deverá ser feito um esforço para a sua otimização e automatização.

E chegamos finalmente à conclusão principal que, apesar de ser a confirmação da nossa intuição inicial, resulta de um árduo trabalho de investigação e de experimentação: as multipopulações são uma forma simples e eficaz de incrementar o sucesso dos algoritmos evolutivos que as utilizam.

Bibliografia

Ackley, David H. 1987. *A Connectionist Machine for Genetic Hillclimbing*. Norwell, MA, USA: Kluwer Academic Publishers.

Alba, E., F. Luna, A. J. Nebro, e J. M. Troya. 2004. «Parallel Heterogeneous Genetic Algorithms for Continuous Optimization». *Parallel Computing*, Parallel and nature-inspired computational paradigms and applications, 30 (5): 699–719. https://doi.org/10.1016/j.parco.2003.12.011.

Aparício, Joaquim N, Luís Correia, e Fernando Moura-Pires. 1999. «Populations are Multisets - PLATO». *W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, eds., Proc. GECCO* Morgan Kaufmann: 1845–1850.

Auger, A., e N. Hansen. 2005. «A Restart CMA Evolution Strategy With Increasing Population Size». Em *2005 IEEE Congress on Evolutionary Computation*, 1769–76. Edinburgh, Scotland, UK. https://doi.org/10.1109/CEC.2005.1554902.

Beyer, H. G, e H. P Schwefel. 2002. «Evolution strategies–A comprehensive introduction». *Natural computing* 1 (1): 3–52.

Blizard, Wayne D. 1991. «The Development of Multiset Theory». *Modern Logic* 1 (4): 319–52.

Brest, Janez, Ales Zamuda, Borko Boskovic, Mirjam Sepesy Maucec, e Viljem Zumer. 2008. «High-dimensional real-parameter optimization using Self-Adaptive Differential Evolution algorithm with population size reduction». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2032–39. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631067.

Cantú-Paz, Erick. 1998. «A Survey of Parallel Genetic Algorithms». *Calculateurs Paralelleles* 10.

Charles Darwin. 1859. *On the origin of species by means of natural selection Or the Preservation of Favoured Races in the Struggle for Life.*

Chen, Yang, Jinglu Hu, K. Hirasawa, e Songnian Yu. 2008. «Solving deceptive problems using a genetic algorithm with reserve selection». Em *IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, 884–89. https://doi.org/10.1109/CEC.2008.4630900.

Cobb, Helen G. 1990. «An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments». NRL-MR-6760. Naval Research Lab Washinghton.

Collingwood, E., D. Corne, e P. Ross. 1996. «Useful Diversity via Multiploidy». Em , *Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, 810–13. IEEE. https://doi.org/10.1109/ICEC.1996.542705.

Correia, Luís. 2010. «Computational Evolution: Taking Liberties». *Theory in Biosciences* 129 (2): 183–91. https://doi.org/10.1007/s12064-010-0099-3.

Correia, Luís, e António Manso. 2015. «A Multiset Model of Multi-Species Evolution to Solve Big Deceptive Problems». Em *Interdisciplinary Evolution Research*, 297–337. Springer Science Business Media. https://doi.org/10.1007/978-3-319-16345-1_11.

Correia, Luís, Fernando Moura-Pires, e Joaquim N Aparício. 1999. «Expressing population based optimization heuristics using PLATO». *P. Barahona, J.J.*

Alferes, eds., LNAI, EPIA'99 - 9th Portuguese Conf. on Artificial Intelligence, Springer, 369–383.

Craighurst, Rob, e Martin Worthy. 1995. «Enhancing GA Performance through Crossover Prohibitions Based on Ancestry». Em *Proceedings of the 6th International Conference on Genetic Algorithms Pages 130-135*. https://dl.acm.org/citation.cfm?id=657905.

Črepinšek, Matej, Shih-Hsi Liu, e Marjan Mernik. 2013. «Exploration and Exploitation in Evolutionary Algorithms: A Survey». *ACM Comput. Surv.* 45 (3): 35:1–35:33. https://doi.org/10.1145/2480741.2480752.

Daida, Jason M., Catherine S. Grasso, Stephen A. Stanhope, e Steven J. Ross. 1996. «Symbionticism and Complex Adaptive Systems I: Implications of Having Symbiosis Occur in Nature». Em *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, 177–186. The MIT Press.

David E. Goldberg. 1989. «Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction». *Complex Systems* 3: 129--152.

De Jong, Kenneth. 1975. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis. Department of Computer and Communication Sciences, University of Michigan.

Diaz-Gomez, Pedro, e Dean Hougen. 2007. «Initial Population for Genetic Algorithms: A Metric Approach.» Em , 43–49.

Doerr, Benjamin, Christian Klein, e Tobias Storch. 2007. «Faster Evolutionary Algorithms by Superior Graph Representation». Em *Foundations of Computational Intelligence, 2007. FOCI 2007.IEEE Symposium.*, 245–50. IEEE. https://doi.org/10.1109/FOCI.2007.372176.

Dumeur, Renaud. 1996. «Evolution through cooperation: The Symbiotic Algorithm». Em *Artificial Evolution*, editado por Jean-Marc Alliot, Evelyne Lutton, Edmund Ronald, Marc Schoenauer, e Dominique Snyers, 145–58. Lecture Notes in Computer Science 1063. Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/3-540-61108-8_36.

Fernandes, Carlos, Rui Tavares, Cristian Munteanu, e Agostinho Rosa. 2001. «Using assortative mating in genetic algorithms for vector quantization problems». Em *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, 361–365. https://doi.org/10.1145/372202.372367.

Fleischer, M. 2003. «The Measure of Pareto Optima Applications to Multi-Objective Metaheuristics». Em *Evolutionary Multi-Criterion Optimization*, editado por Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Lothar Thiele, e Kalyanmoy Deb, 519–33. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-36970-8_37.

Frederick, William G., Robert L. Sedlmeyer, e Curt M. White. 1993. «The Hamming metric in genetic algorithms and its application to two network problems». Em *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, 126–130. SAC '93. Indianapolis, Indiana, USA: Association for Computing Machinery. https://doi.org/10.1145/162754.162835.

Friedrich, Tobias, Pietro S. Oliveto, Dirk Sudholt, e Carsten Witt. 2008. «Theoretical Analysis of Diversity Mechanisms for Global Exploration». Em *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 945–952. GECCO '08. New York, NY, USA: ACM. https://doi.org/10.1145/1389095.1389276.

Ghosh, A., S. Tsutsui, e H. Tanaka. 1998. «Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals». Em *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 666–71. https://doi.org/10.1109/ICEC.1998.700119.

Glibovets, N. N., e N. M. Gulayeva. 2013. «A Review of Niching Genetic Algorithms for Multimodal Function Optimization». *Cybernetics and Systems Analysis* 49 (6): 815–20. https://doi.org/10.1007/s10559-013-9570-8.

Goldberg, D. E. 1989. «Messy Genetic Algorithms : Motivation, Analysis, and First Results». *Complex Systems* 3: 493–530.

Goldberg, D. E., e C. L. Bridges. 1990. «An Analysis of a Reordering Operator on a GA-Hard Problem». *Biological Cybernetics* 62 (5): 397–405. https://doi.org/10.1007/BF00197646.

Goldberg, David E. 1987. «Simple Genetic Algorithms and the Minimal, Deceptive Problem». *Genetic Algorithms and Simulated Annealing* L. Davis, editor, San Mateo, CA: Morgan Kaufmann: 74–88.

Goldberg, David, e Jon Richardson. 1987. «Genetic algorithms with sharing for multimodal function optimization». Em *Genetic algorithms and their applications: Proceedings of the Second International Conference on Ge-netic Algorithms, pp. 41–49*. Hillsdale, NJ: LawrenceErlbaum.

Harik, G. R, F. G Lobo, e D. E Goldberg. 1999. «The Compact Genetic Algorithm». *IEEE Transactions on Evolutionary Computation* 3 (4): 287–97. https://doi.org/10.1109/4235.797971.

Harik, Georges. 1995. «Finding Multimodal Solutions Using Restricted Tournament Selection». *Procedings of Sixth Internationalconference on Genetic Algorithms*, 24--31.

Hartmann, Sönke. 1998. «A Competitive Genetic Algorithm for Resource-constrained Project Scheduling». *Naval Research Logistics (NRL)* 45 (7): 733–50.

Hauschild, Mark, e Martin Pelikan. 2011. «An introduction and survey of estimation of distribution algorithms». *Swarm and Evolutionary Computation*, 111–128.

Herrera, F., M. Lozano, e A. M. Snchez. 2003. «A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study». *International Journal of Intelligent Systems* 18 (3): 309–38. https://doi.org/10.1002/int.10091.

Heywood, Malcolm I., e Peter Lichodzijewski. 2010. «Symbiogenesis as a Mechanism for Building Complex Adaptive Systems: A Review». Em *Applications of Evolutionary Computation*, editado por Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna I. Esparcia-Alcazar, Chi-Keong Goh, et al., 51–60. Lecture Notes in Computer Science 6024. Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/978-3-642-12239-2_6.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan.

Hwang, Sungmin, Benjamin Schmiegelt, Luca Ferretti, e Joachim Krug. 2018. «Universality classes of interaction structures for NK fitness landscapes». *Journal of Statistical Physics* 172 (1): 226–78. https://doi.org/10.1007/s10955-018-1979-z.

Jansen, Thomas, e Ingo Wegener. 2005. «Real royal road functions—where crossover provably is essential». *Discrete Applied Mathematics*, Boolean and Pseudo-Boolean Functions, 149 (1): 111–25. https://doi.org/10.1016/j.dam.2004.02.019.

Kennedy, J., e R. Eberhart. 1995. «Particle swarm optimization». Em *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4:1942–48 vol.4. https://doi.org/10.1109/ICNN.1995.488968.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. 1.ª ed. A Bradford Book.

Lacevic, Bakir, e Edoardo Amaldi. 2010. «On population diversity measures in Euclidean space». Em *IEEE Congress on Evolutionary Computation*, 1–8. https://doi.org/10.1109/CEC.2010.5586498.

Li, Jian-Ping, Marton E. Balazs, Geoffrey T. Parks, e P. John Clarkson. 2002. «A Species Conserving Genetic Algorithm for Multimodal Function Optimization». *Evolutionary Computation* 10 (3): 207–34. https://doi.org/10.1162/106365602760234081.

Lin-Yu Tseng, e Chun Chen. 2008. «Multiple trajectory search for Large Scale Global Optimization». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3052–59. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631210.

Lozano, Manuel, Francisco Herrera, e José Ramón Cano. 2005. «Replacement Strategies to Maintain Useful Diversity in Steady-State Genetic Algorithms». Em *Soft Computing: Methodologies and Applications*, editado por Frank Hoffmann, Mario Köppen, Frank Klawonn, e Rajkumar Roy, 85–96. Advances in Soft Computing. Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-32400-3_7.

Mahfoud, Samir W. 1995. «Niching Methods for Genetic Algorithms». http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.8270.

Manso, A., e L. Correia. 2009. «Genetic algorithms using populations based on multisets». *L. Seabra Lopes, N. Lau, P. Mariano, L. Rocha, eds* EPIA-Portuguese Conference on Artificial Intelligence: 53–64.

———. 2011. «A multiset genetic algorithm for real coded problems». *Genetic and Evolutionary Computation Conference, GECCO'11 - Companion Publication*, 153–54. https://doi.org/10.1145/2001858.2001944.

———. 2011a. «MITree - Multiset Indexed Tree». Em . Coimbra, Portugal.

———. 2011. «MuGA: multiset genetic algorithm». Em *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation - GECCO \textquotesingle11*. Association for Computing Machinery (ACM). https://doi.org/10.1145/2001858.2002093.

———. 2011b. «Preservação da Diversidade Genética no Multiset Genetic Algorithm». Em *Congresso de Métodos Numéricos em Engenharia 2011*. Coimbra.

———. 2013a. «A multiset genetic algorithm for the optimization of deceptive problems». Em *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO 13*. Association for Computing Machinery (ACM). https://doi.org/10.1145/2463372.2463471.

———. 2015. «Parasite diversity in symbiogenetic multiset genetic algorithm - Optimization of large binary problems». *GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, 887–94. https://doi.org/10.1145/2739480.2754780.

Matsui, K. 1999. «New selection method to improve the population diversity in genetic algorithms». Em *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat.*

No.99CH37028), 1:625–30 vol.1. https://doi.org/10.1109/ICSMC.1999.814164.

Mengshoel, Ole J, e David E Goldberg. 2008. «The crowding approach to niching in genetic algorithms». *Evolutionary Computation* 16 (Setembro): 315–354. http://dx.doi.org/10.1162/evco.2008.16.3.315.

Mitchell, M., J. H. (Michigan Univ Holland, e S. (New Mexico Univ Forrest. 1991. «The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance». LA-UR-91-3391; CONF-911266-1. Los Alamos National Lab., NM (United States).

Navalresearch, William Spears, e William M Spears. 1995. «Speciation using tag bits». *Handbook of Evolutionary Computation* IOP Publishing Ltd and Oxford University Press.

Nelder, A, e R Mead. 1965. «A simplex method for function minimization». *Computer Journal*, 1965.

Pelikan, M. 2005. «Bayesian Optimization Algorithm». *Hierarchical Bayesian Optimization Algorithm*, 31–48.

Petrowski, A. 1996. «A Clearing Procedure as a Niching Method for Genetic Algorithms». Em , *Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, 798–803. IEEE. https://doi.org/10.1109/ICEC.1996.542703.

Pisinger, David. 2005. «Where are the hard knapsack problems?» *Computers & Operations Research* 32 (9): 2271–2284.

Potter, Mitchell A., e Kenneth A. De Jong. 2000. «Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents». *Evolutionary Computation* 8 (1): 1–29. https://doi.org/10.1162/106365600568086.

Rosin, Christopher D., e Richard K. Belew. 1997. «New Methods for Competitive Coevolution». *Evolutionary Computation* 5 (1): 1–29. https://doi.org/10.1162/evco.1997.5.1.1.

Rothlauf, Franz. 2006. «Representations for Genetic and Evolutionary Algorithms». Em *Representations for Genetic and Evolutionary Algorithms*, editado por Franz Rothlauf, 9–32. Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-32444-5_2.

Santos, E., & Santos, E. (2001). Effective and Efficient Caching in Genetic Algorithms. International Journal on Artificial Intelligence Tools, 10, 273–301. https://doi.org/10.1142/S0218213001000520

Schaefer, Robert. 2007. *Foundations of Global Genetic Optimization*. 1.ª ed. Springer.

Seredynski, Franciszek. 1994. «Loosely Coupled Distributed Genetic Algorithms». Em *Parallel Problem Solving from Nature — PPSN III*, editado por Yuval Davidor, Hans-Paul Schwefel, e Reinhard Männer, 514–23. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-58484-6_294.

Sheng-Ta Hsieh, Tsung-Ying Sun, Chan-Cheng Liu, e Shang-Jeng Tsai. 2008. «Solving large scale global optimization using improved Particle Swarm Optimizer». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 1777–84. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631030.

Siarry, Patrick, Alain Pétrowski, e Mourad Bessaou. 2002. «A Multipopulation Genetic Algorithm Aimed at Multimodal Optimization». *Advances in Engineering Software* 33 (4): 207–13. https://doi.org/10.1016/S0965-9978(02)00010-8.

Simões, Anabela, e Ernesto Costa. 2001. «On Biologically Inspired Genetic Operators: Transformation in the Standard Genetic Algorithm». Em *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 584–591. GECCO'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Singh, D., A. Ibrahim, T. Yohanna, e J. Singh. 2007. «An overview of the applications of multisets». *Novi Sad Journal of Mathematics* 37 (3): 73–92.

Sudholt, Dirk. 2020. «The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses». Em *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, editado por Benjamin Doerr e Frank Neumann, 359–404. Natural Computing Series. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-29414-4_8.

Tang, K., X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, e Z. Yang. 2007. «Benchmark functions for the CEC'2008 special session and competition on large scale global optimization». *Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep*.

Thierens, D. 2002. «Adaptive mutation rate control schemes in genetic algorithms». Em *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 1:980–85 vol.1. https://doi.org/10.1109/CEC.2002.1007058.

Thierens, D., e D. Goldberg. 1994. «Elitist recombination: an integrated selection recombination GA». Em *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 508–12 vol.1. https://doi.org/10.1109/ICEC.1994.349898.

Thierens, Dirk. 2010. «Linkage Tree Genetic Algorithm: First Results». Em *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, 1953–1958. GECCO '10. New York, NY, USA: ACM. https://doi.org/10.1145/1830761.1830832.

Toffolo, Andrea, e Ernesto Benini. 2003. «Genetic Diversity As an Objective in Multi-objective Evolutionary Algorithms». *Evol. Comput.* 11 (2): 151–167. https://doi.org/10.1162/106365603766646816.

Ursem, Rasmus K. 2002. «Diversity-Guided Evolutionary Algorithms». *Proceedings of the Congress on Evolutionary Computation* 3: 1633–40.

Ursem, R.K. 1999. «Multinational evolutionary algorithms». Em *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, 3:1633-1640 Vol. 3. https://doi.org/10.1109/CEC.1999.785470.

Wagner, Stefan, e Michael Affenzeller. 2005. «The Allele Meta-Model – Developing a Common Language for Genetic Algorithms». Em *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, editado por José Mira e José R. Álvarez, 202–11. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/11499305_21.

Wallin, D., C. Ryan, e R.M.A. Azad. 2005. «Symbiogenetic coevolution». Em *The 2005 IEEE Congress on Evolutionary Computation, 2005*, 2:1613-1620 Vol. 2. https://doi.org/10.1109/CEC.2005.1554882.

Wiese, Kay C., e Scott D. Goodwin. 2001. «Keep–Best Reproduction: A Local Family Competition Selection Strategy and the Environment It Flourishes In». *Constraints* 6 (4): 399–422. https://doi.org/10.1023/A:1011409029226.

Yang, S. 2004. «Adaptive group mutation for tackling deception in genetic search». *WSEAS Transactions on Systems* 3 (1): 107–112.

Zamuda, Ales, Janez Brest, Borko Boskovic, e Viljem Zumer. 2008. «Large Scale Global Optimization using Differential Evolution with self-adaptation and cooperative co-evolution». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3718–25. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631301.

Zhao, S. Z., J. J. Liang, P. N. Suganthan, e M. F. Tasgetiren. 2008. «Dynamic multi-swarm particle swarm optimizer with local search for Large Scale Global Optimization». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3845–52. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631320.

Zhenyu Yang, Ke Tang, e Xin Yao. 2008. «Multilevel cooperative coevolution for large scale optimization». Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 1663–70. Hong Kong, China. https://doi.org/10.1109/CEC.2008.4631014.

Zhu, Yani, Zhongxiu Yang, e Jiatao Song. 2006. «A Genetic Algorithm with Age and Sexual Features». Em *Intelligent Computing*, editado por De-Shuang Huang, Kang Li, e George William Irwin, 634–40. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/11816157_75.

Anexo A – Lista de Publicações

Capítulo de Livro

Correia, L., Manso, A., 2015. A Multiset Model of Multi-Species Evolution to Solve Big Deceptive Problems, in: Interdisciplinary Evolution Research. Springer Science Business Media, pp. 297–337. https://doi.org/10.1007/978-3-319-16345-1_11

Artigos de Conferência

Manso, A., Correia, L., 2009. Genetic algorithms using populations based on multisets, in: New Trends in Artificial Intelligence, EPIA 2009. Aveiro, Portugal.

Manso, A., Correia, L., 2011. Preservação da Diversidade Genética no Multiset Genetic Algorithm, in: Congresso de Métodos Numéricos Em Engenharia 2011, Coimbra, Portugal.

Manso, A., Correia, L., 2011. MITree - Multiset Indexed Tree. Presented at the Congresso de Métodos Numéricos em Engenharia 2011, Coimbra, Portugal.

Manso, A., Correia, L., 2011. A multiset genetic algorithm for real coded problems, in: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation – GECCO'11. Dublin, Ireland https://doi.org/10.1145/2001858.2002093

Manso, A., Correia, L., 2011. MuGA: multiset genetic algorithm, in: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation - GECCO 11. Dublin, Ireland https://doi.org/10.1145/2001858.2002093

Manso, A., Correia, L. 2013. A multiset genetic algorithm for the optimization of deceptive problems, in: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference – GECCO'13, Amsterdam, The Netherlands https://doi.org/10.1145/2463372.2463471

Manso, A., Correia, L. 2015. Parasite Diversity in Symbiogenetic Multiset Genetic Algorithm: Optimization of Large Binary Problems, in: Proceeding of the Seventh Annual Conference on Genetic and Evolutionary Computation Conference – GECCO'15, Madrid, Spain https://doi.org/10.1145/2739480.2754780

Relatórios técnicos

Manso, A., 2011. Populações Baseadas em Multisets para Algoritmos Evolutivos, Prova de Qualificação - Doutoramento em Informática. Universidade de Lisboa, Portugal

Anexo B – Definição do problema NK 128-3- 128

- 128 – Genes
- 3 – Bits por regra
- 128 – Regras
- 1 Ótimo

```
01111100000110001110010110100001000000111001001011100010001011100011011011010110100101011000011111110110100010110111001000100111
```

Locus dos genes			Valores do fenótipo							
			000	001	010	011	100	101	110	111
0	1	2	0,63	0,97	0,06	**1,00**	0,16	0,51	0,98	0,37
1	2	3	0,04	0,48	0,26	0,72	0,30	0,19	0,08	**1,00**
2	3	4	0,95	0,73	0,19	0,41	0,87	0,13	0,91	**1,00**
3	4	5	0,63	0,04	0,77	0,31	0,66	0,82	0,83	**1,00**
4	5	6	0,34	0,65	0,71	0,55	0,87	0,72	**1,00**	0,33
5	6	7	0,98	0,33	0,18	0,74	**1,00**	0,14	0,08	0,47
6	7	8	**1,00**	0,01	0,51	0,63	0,34	0,92	0,30	0,59
7	8	9	**1,00**	0,88	0,17	0,73	0,74	0,02	0,94	0,71
8	9	10	**1,00**	0,19	0,47	0,12	0,76	0,09	0,07	0,06
9	10	11	0,71	**1,00**	0,45	0,53	0,32	0,34	0,72	0,18
10	11	12	0,09	0,75	0,18	**1,00**	0,34	0,37	0,34	0,58
11	12	13	0,90	0,27	0,32	0,56	0,13	0,90	**1,00**	0,95
12	13	14	0,78	0,25	0,52	0,46	**1,00**	0,46	0,57	0,13
13	14	15	**1,00**	0,28	0,35	0,70	0,58	0,39	0,27	0,22
14	15	16	0,36	**1,00**	0,54	0,99	0,57	0,89	0,66	0,57
15	16	17	0,86	0,15	0,76	**1,00**	0,89	0,91	0,36	0,63
16	17	18	0,53	0,69	0,37	0,96	0,95	0,01	0,40	**1,00**
17	18	19	0,01	0,28	0,79	0,15	0,04	0,65	**1,00**	0,32
18	19	20	0,87	0,60	0,35	0,57	**1,00**	0,41	0,49	0,43
19	20	21	0,51	**1,00**	0,84	0,98	0,48	0,14	0,03	0,06
20	21	22	0,40	0,19	**1,00**	0,46	0,50	0,12	0,56	0,99
21	22	23	0,98	0,66	0,65	0,20	0,93	**1,00**	0,90	0,14
22	23	24	0,41	0,05	0,22	**1,00**	0,06	0,27	0,63	0,51
23	24	25	0,87	0,33	0,15	0,09	0,96	0,86	**1,00**	0,30
24	25	26	0,15	0,07	0,15	0,07	0,90	**1,00**	0,69	0,68
25	26	27	0,77	0,53	**1,00**	0,03	0,71	0,50	0,16	0,46
26	27	28	0,81	0,25	0,35	0,64	**1,00**	0,58	0,65	0,45
27	28	29	**1,00**	0,72	0,30	0,43	0,31	0,37	0,08	0,46
28	29	30	**1,00**	0,37	0,68	0,28	0,91	0,40	0,41	0,06
29	30	31	0,63	**1,00**	0,08	0,30	0,54	0,40	0,22	0,60
30	31	32	0,48	0,82	**1,00**	0,45	0,49	0,60	0,35	0,13
31	32	33	0,74	0,07	0,90	0,75	**1,00**	0,96	0,68	0,25
32	33	34	**1,00**	0,18	0,03	0,29	0,93	0,55	0,88	0,98
33	34	35	**1,00**	0,81	0,18	0,11	0,09	0,23	0,63	0,14
34	35	36	**1,00**	0,97	0,64	0,71	0,67	0,51	0,48	0,97
35	36	37	**1,00**	0,39	0,79	0,01	0,32	0,29	0,75	0,83
36	37	38	0,26	**1,00**	0,89	0,64	0,94	0,67	0,70	0,27
37	38	39	0,11	0,04	0,24	**1,00**	0,55	0,24	0,04	0,83
38	39	40	0,29	0,11	0,52	0,17	0,90	0,44	0,72	**1,00**
39	40	41	0,89	0,08	0,56	0,64	0,64	0,90	**1,00**	0,85
40	41	42	0,32	0,60	0,80	0,99	**1,00**	0,77	0,92	0,38
41	42	43	0,87	**1,00**	0,33	0,86	0,82	0,68	0,46	0,79
42	43	44	0,60	0,92	**1,00**	0,49	0,28	0,64	0,86	0,60
43	44	45	0,25	0,49	0,28	0,10	**1,00**	0,99	0,94	0,76
44	45	46	0,93	**1,00**	0,05	0,29	0,38	0,48	0,41	0,25
45	46	47	0,69	0,78	**1,00**	0,67	0,09	0,75	0,53	0,94
46	47	48	0,35	0,46	0,55	0,45	0,94	**1,00**	0,26	0,54
47	48	49	0,25	0,66	0,47	**1,00**	0,52	0,53	0,31	0,39
48	49	50	0,57	0,20	0,66	0,06	0,79	0,64	0,49	**1,00**
49	50	51	0,55	0,87	0,38	0,30	0,58	0,73	**1,00**	0,70
50	51	52	0,79	0,69	0,33	0,49	**1,00**	0,73	0,61	0,19
51	52	53	**1,00**	0,33	0,41	0,40	0,01	0,47	0,95	0,73
52	53	54	0,25	**1,00**	0,71	0,05	0,83	0,15	0,67	0,45
53	54	55	0,12	0,51	**1,00**	0,23	0,53	0,42	0,99	0,95
54	55	56	0,22	0,11	0,61	0,35	**1,00**	0,50	0,82	0,06
55	56	57	**1,00**	0,23	0,54	0,54	0,45	0,37	0,80	0,05
56	57	58	0,66	**1,00**	0,93	0,01	0,43	0,13	0,85	0,47
57	58	59	0,94	0,45	**1,00**	0,04	0,48	0,82	0,28	0,14
58	59	60	0,34	0,78	0,45	0,09	0,95	**1,00**	0,31	0,92
59	60	61	0,48	0,57	0,28	**1,00**	0,20	0,16	0,33	0,15
60	61	62	0,63	0,85	0,46	0,85	0,03	0,23	0,56	**1,00**
61	62	63	0,96	0,57	0,48	0,06	0,77	0,03	**1,00**	0,83
62	63	64	0,75	0,49	0,84	0,29	**1,00**	0,92	0,39	0,81
63	64	65	**1,00**	0,83	1,00	0,90	0,26	0,18	0,13	0,98

Locus dos genes			Valores do fenótipo							
			000	001	010	011	100	101	110	111
64	65	66	0,74	**1,00**	0,92	0,94	0,32	0,26	0,60	0,68
65	66	67	0,22	0,72	0,96	**1,00**	0,28	0,47	0,91	0,31
66	67	68	0,76	0,19	0,86	0,51	0,08	0,92	**1,00**	0,31
67	68	69	0,43	0,78	0,23	0,15	0,17	**1,00**	0,62	0,42
68	69	70	0,27	0,38	0,73	**1,00**	0,12	0,06	0,87	0,39
69	70	71	0,84	0,87	0,70	0,58	0,49	0,24	**1,00**	0,55
70	71	72	0,43	0,76	0,14	0,06	0,65	**1,00**	0,90	0,99
71	72	73	0,34	0,72	0,22	**1,00**	0,14	0,43	0,14	0,34
72	73	74	0,37	0,91	0,64	0,60	0,65	0,61	**1,00**	0,12
73	74	75	0,28	0,53	0,66	0,95	0,69	**1,00**	0,72	0,40
74	75	76	0,82	0,90	**1,00**	0,89	0,58	0,09	0,30	0,24
75	76	77	0,88	**1,00**	0,61	0,84	0,41	**1,00**	0,16	0,93
76	77	78	0,45	0,31	0,56	**1,00**	0,90	0,34	0,92	0,23
77	78	79	0,75	0,36	0,19	0,36	0,50	0,10	**1,00**	0,05
78	79	80	0,61	0,37	0,00	0,89	0,05	**1,00**	0,96	0,20
79	80	81	0,93	0,83	**1,00**	0,88	0,94	0,88	0,84	0,27
80	81	82	0,86	0,90	0,60	0,41	**1,00**	0,83	0,99	0,72
81	82	83	0,01	**1,00**	0,14	0,54	0,64	0,27	0,15	0,66
82	83	84	0,81	0,43	**1,00**	0,30	0,67	0,48	0,54	0,03
83	84	85	0,84	0,23	0,57	0,26	0,21	**1,00**	0,82	0,11
84	85	86	0,79	0,07	**1,00**	0,78	0,14	0,59	0,30	0,55
85	86	87	0,29	0,63	0,26	0,19	0,94	**1,00**	0,99	0,34
86	87	88	0,84	0,56	0,23	**1,00**	0,55	0,68	0,68	0,43
87	88	89	0,55	0,35	0,76	0,28	0,45	0,09	**1,00**	0,97
88	89	90	0,47	0,10	0,60	0,51	**1,00**	0,22	0,51	0,35
89	90	91	**1,00**	0,48	0,76	0,34	0,58	0,73	0,28	0,35
90	91	92	**1,00**	0,37	0,56	0,76	0,68	0,57	0,16	0,38
91	92	93	0,92	**1,00**	0,84	0,15	0,93	0,32	0,86	0,44
92	93	94	0,53	0,27	0,81	**1,00**	0,85	0,74	0,67	0,19
93	94	95	0,98	0,39	0,25	0,06	0,62	0,05	0,23	**1,00**
94	95	96	0,08	0,88	0,76	0,90	0,44	0,48	0,42	**1,00**
95	96	97	0,36	0,24	0,48	0,75	0,69	0,96	0,93	**1,00**
96	97	98	0,74	0,17	0,51	0,76	0,04	0,22	0,77	**1,00**
97	98	99	0,65	0,19	0,79	0,34	0,75	0,46	**1,00**	0,85
98	99	100	0,01	0,91	0,25	0,31	0,44	**1,00**	0,21	0,51
99	100	101	0,63	0,76	0,50	**1,00**	0,31	0,54	0,64	0,52
100	101	102	0,83	0,28	0,43	0,34	0,54	0,62	**1,00**	0,97
101	102	103	0,55	0,03	0,31	0,25	0,28	**1,00**	0,84	0,69
102	103	104	0,05	0,79	**1,00**	0,90	0,61	0,07	0,60	0,29
103	104	105	0,01	0,78	0,33	0,66	**1,00**	0,25	0,26	0,96
104	105	106	**1,00**	0,79	0,06	0,81	0,53	0,94	0,73	0,98
105	106	107	0,83	**1,00**	0,19	0,33	0,45	0,43	0,91	0,65
106	107	108	0,98	0,34	**1,00**	0,00	0,49	0,08	0,17	0,10
107	108	109	0,60	0,73	0,79	0,73	0,72	**1,00**	0,44	0,04
108	109	110	0,05	0,70	0,53	**1,00**	0,56	0,53	0,13	0,71
109	110	111	0,20	0,13	0,17	0,29	0,88	0,28	**1,00**	0,23
110	111	112	0,47	0,09	0,78	0,64	0,82	**1,00**	0,47	0,32
111	112	113	0,95	0,04	0,04	**1,00**	0,19	0,14	0,70	0,16
112	113	114	0,47	0,14	0,59	0,09	0,28	0,73	0,42	**1,00**
113	114	115	0,76	0,73	0,41	0,88	0,36	0,89	**1,00**	0,39
114	115	116	0,91	0,65	0,03	0,94	**1,00**	0,27	0,74	0,37
115	116	117	0,91	**1,00**	0,86	0,50	0,45	0,38	0,22	0,89
116	117	118	0,99	0,94	**1,00**	0,61	0,33	0,73	0,91	0,27
117	118	119	0,88	0,03	0,92	0,06	**1,00**	0,89	0,09	0,59
118	119	120	**1,00**	0,53	0,78	0,90	0,99	0,87	0,87	0,22
119	120	121	0,72	**1,00**	0,13	0,47	0,12	0,72	0,86	0,07
120	121	122	0,90	0,48	**1,00**	0,87	0,59	0,67	0,64	0,71
121	122	123	0,51	0,34	0,15	0,40	**1,00**	0,90	0,95	0,89
122	123	124	0,95	**1,00**	0,55	0,34	0,95	0,82	0,61	0,12
123	124	125	0,37	0,68	0,57	**1,00**	1,00	0,58	0,43	0,12
124	125	126	0,25	0,73	0,85	0,80	0,61	0,01	0,68	**1,00**
125	126	127	0,59	0,31	0,83	0,77	0,43	0,39	0,91	**1,00**
126	127	0	0,31	0,32	0,67	0,08	0,48	0,59	**1,00**	0,59
127	0	1	0,21	0,52	0,73	0,46	0,66	**1,00**	0,21	0,59

Anexo C – Definição do problema KnapSack 128

- 128 itens
- Pesos aleatórios
    - $w_i = random\ integer\ [1,128]$
- Valores aleatórios
    - $p_i = w_i + random\ integer\ [0, wi/10]$

Item	W_i	P_i
1	89	96
2	26	28
3	17	17
4	2	2
5	58	58
6	102	106
7	72	75
8	107	116
9	45	48
10	10	11
11	4	4
12	10	11
13	27	27
14	67	71
15	72	75
16	10	11
17	119	126
18	101	104
19	104	108
20	78	80
21	62	62
22	122	131
23	88	90
24	83	89
25	125	134
26	83	84
27	49	53
28	123	126
29	86	91
30	87	89
31	109	114
32	110	121

Item	W_i	P_i
33	22	24
34	17	17
35	106	113
36	12	13
37	41	44
38	27	27
39	41	43
40	46	47
41	33	36
42	60	61
43	110	120
44	21	22
45	17	18
46	72	75
47	47	50
48	110	111
49	26	28
50	7	7
51	54	57
52	69	70
53	115	120
54	26	27
55	83	85
56	116	121
57	8	8
58	53	56
59	88	90
60	7	7
61	7	7
62	35	36
63	75	76
64	77	79

Item	W_i	P_i
65	69	70
66	102	108
67	92	98
68	68	71
69	121	131
70	38	38
71	41	45
72	71	72
73	24	26
74	64	69
75	79	83
76	124	127
77	114	123
78	80	86
79	122	131
80	49	51
81	94	99
82	76	77
83	115	119
84	110	121
85	75	78
86	39	41
87	21	23
88	75	80
89	4	4
90	91	99
91	56	60
92	63	69
93	77	83
94	33	36
95	68	72
96	77	83

Item	W_i	P_i
97	66	66
98	115	116
99	29	31
100	55	56
101	109	118
102	120	129
103	117	128
104	47	50
105	39	41
106	106	107
107	88	96
108	25	26
109	44	46
110	91	98
111	120	129
112	92	101
113	28	28
114	110	116
115	63	68
116	5	5
117	85	92
118	93	99
119	51	56
120	49	52
121	43	44
122	86	88
123	118	127
124	56	61
125	108	111
126	18	19
127	126	129
128	58	60

Anexo D – Influência das multipopulações no Restrictive Tournament Selection- RTS

O *Restrictive Tournament Selection*, RTS, está implementado no MuGA-T como um operador de substituição que pode ser utilizado no algoritmo genético, GA.

O operador substitui um indivíduo na população pesquisando o indivíduo geneticamente mais próximo, utilizando um subconjunto da população principal para fazer o teste. Se o descendente tiver um grau de aptidão superior ou igual ao seu semelhante, a substituição é feita e o descendente ocupa o seu lugar na população.

No caso de ser uma multipopulação, o operador garante que a mesma mantenha o número de genótipos constante e, por isso, se o indivíduo já existir na população, não há substituição, mas sim, o incremento no número de suas cópias.

Nas simulações deste anexo, o operador *RestrictedReplacement* utiliza 25% da população para escolher o indivíduo geneticamente mais próximo, e os restantes parâmetros são os definidos no capítulo 5 desta tese.

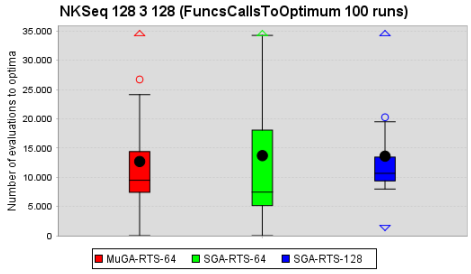
Otimização da função NK-128-3-128 com *RestrictedReplacement*

O quadro seguinte apresenta os resultados da otimização da função NK 128-3-128, sendo possível constatar que nenhum algoritmo conseguiu uma taxa de sucesso de 100%: os algoritmos MuGA-RTS-64 e SGA-RTS-128 otimizaram 99% dos problemas, e o SGA-RTS-64, apenas 78%. Resultados experimentais mostram que um incremento da pressão seletiva dos progenitores ou o aumento da janela de teste de similaridade tornam os algoritmos plenamente eficazes.

	FuncsCallsToOptimum		SucessRate		Individuals		GeneticDiversity		Genotypes	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-RTS-64	12699,3	8688,6	0,99	0,10	127,2	2,5	0,092	0,014	64,0	0,0
SGA-RTS-64	13702,3	16934,4	0,78	0,42	64,0	0,0	0,056	0,012	13,2	3,9
SGA-RTS-128	13605,8	9010,4	0,99	0,10	128,0	0,0	0,108	0,013	36,5	3,8

Resultado da otimização da função NK-128-3-128 após 100.000 chamadas à função de avaliação

Em relação à eficiência, o MuGA-RTS-64 necessita de menos chamadas à função de avaliação; no entanto, esta diminuição não é estatisticamente significativa. A figura seguinte apresenta os resultados estatísticos do número de chamadas à função de avaliação para obter o ótimo. Através da análise da comparação das médias, com significância de 95% no *t-test*, todas as simulações são estatisticamente iguais.



a)

Comparison of means - confidence 95%

	MuGA-RTS-64	SGA-RTS-64	SGA-RTS-128
MuGA-RTS-64	.	.	.
SGA-RTS-64	.	.	.
SGA-RTS-128	.	.	.

b)

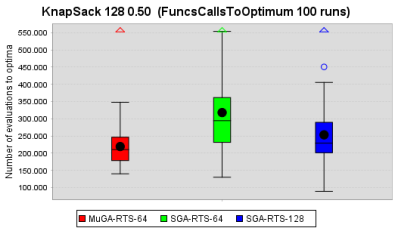
NK-128-3-128 - Número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot; b) Resultado do teste de hipóteses t-test

Otimização da função Knapsack-128 0.5 com *RestrictedReplacement*

A tabela seguinte apresenta os resultados da utilização do operador *RestrictedReplacement* no MuGA para a otimização do problema Knapsack. Na tabela podemos constatar que todas as simulações tiveram uma taxa de sucesso de 100%, descobriram vários ótimos, *NumOptimaFound*, e os retiveram na população, *NumOptimalGenomesPop*. Com a ajuda das multipopulações, o MuGA-RTS-64 conseguiu descobrir mais ótimos e retê-los na população, quando comparado com as simulações envolvendo populações simples.

	FuncsCallsToOptimum		SucessRate		NumOptimaFound		NumOptimalGenomesPop	
Experiência	Média	Desv.Pad.	Média	DesvPad	Média	Desv.Pad.	Média	Desv.Pad.
MuGA-RTS-64	219434,3	53106,5	1,00	0,00	436,1	91,3	55,7	2,3
SGA-RTS-64	318125,3	133569,5	1,00	0,00	132,1	80,0	9,9	2,4
SGA-RTS-128	253242,1	88079,9	1,00	0,00	287,1	98,2	19,9	2,9

Resultado da otimização da função Knapsack -128 0.5 após 1.000.000 chamadas à função de avaliação.



a)

Comparison of means - confidence 95%

	MuGA-RTS-64	SGA-RTS-64	SGA-RTS-128
MuGA-RTS-64	.	+	+
SGA-RTS-64	-	.	-
SGA-RTS-128	-	+	.

b)

Knapsack -128 0.5 - Número de chamadas à função de avaliação para obter o ótimo – 100 simulações: a) gráfico de boxplot; b) Resultado do teste de hipóteses t-test

A figura seguinte apresenta os resultados estatísticos do número de chamadas para obter o primeiro ótimo, *FuncsCallsToOptimum*, e podemos verificar que a simulação MuGA-RTS-64 é a mais eficiente de todas, e que a mesma é estatisticamente significativa.

Anexo E – Otimização do CEC2008 com o MuGA

Características das Funções

		Máximos		Separável		Dominio	
Função	Nome	Um	Vários	Sim	Não	Minimo	Maximo
F1	Shifted Sphere Function	X		x		-100	100
F2	Shifted Schwefel's Problem 2.21	X			x	-100	100
F3	Shifted Rosenbrock's Function		X		x	-100	100
F4	Shifted Rastrigin's Function		X	x		-5	5
F5	Shifted Griewank's Function		X		x	-600	600
F6	Shifted Ackley's Function		X	x		-32	32
F7	FastFractal "DoubleDip" Function		X		x	-1	1

Características das funções do Benchmark CEC2008

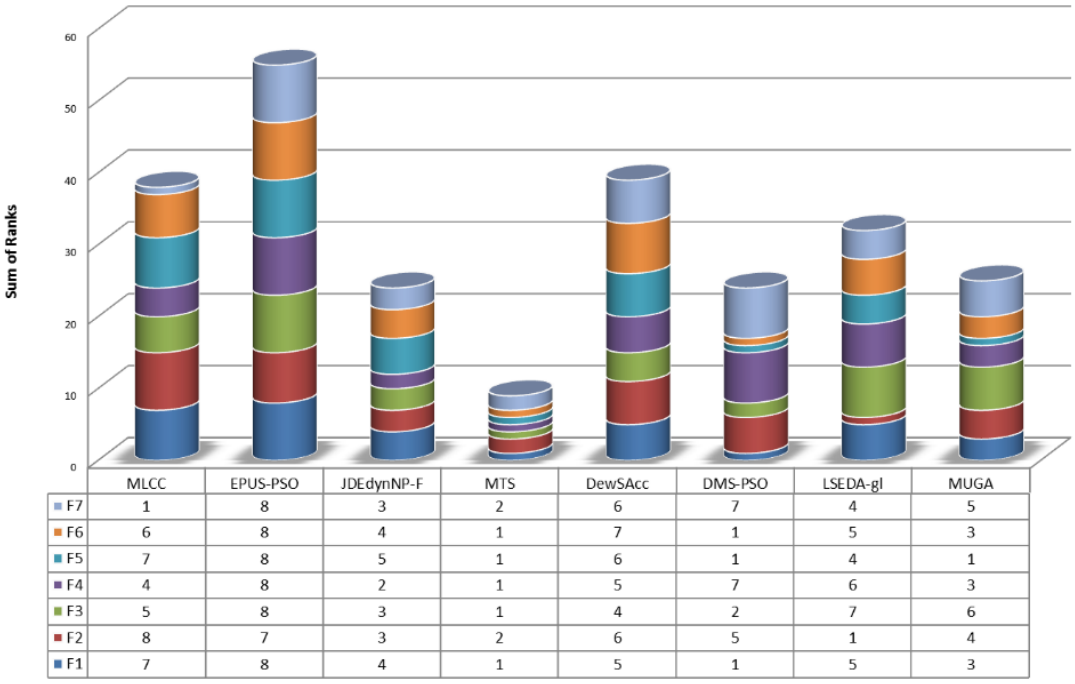
Configuração da experiência para avaliar a influência dos operadores genéticos baseados em multi-indivíduos

Item	**Descrição**
Seleção	*MTournamentSelection (parentsProportion=0.5; toursize=3)*
Recombinação	*MuAX (probCrossover=0.75)*
Mutação	*MuGAuss(* $\propto = 0.1, \partial = 0.1$*)*
Substituição	*Geracional com elitismo*
Redimensionamento	*AdaptiveCeiling(maxProportion=1.5)*

Otimização de problemas com 100 Variáveis

	F1	F2	F3	F4	F5	F6	F7
MLCC	6.82E-14	2.53E+01	1.49E+02	4.39E-13	3.41E-14	1.11E-13	-1.54E+03
EPUS-PSO	7.47E-01	1.86E+01	4.99E+03	4.71E+02	3.72E-01	2.06E+00	-8.55E+02
JDEdynNP-F	5.68E-14	4.29E-01	1.12E+02	5.46E-14	2.84E-14	5.68E-14	-1.48E+03
MTS	0.00E+00	1.44E-11	5.17E-08	0.00E+00	0.00E+00	0.00E+00	-1.49E+03
DewSAcc	5.68E-14	8.25E+00	1.45E+02	4.38E+00	3.07E-14	1.13E-13	-1.37E+03
DMS-PSO	0.00E+00	3.64E+00	2.83E+01	1.83E+002	0.00E+00	0.00E+00	-1.14E+03
LSEDA-gl	5.68E-14	2.21E-13	2.81E+02	1.31E+02	2.84E-14	9.78E-14	-1.46E+03
MUGA	4.77E-29	1.62E+00	2.11E+02	9.37E-14	0.00E+00	7.12E-15	-1.46E+03

Resultado dos algoritmos da competição CEC2008 e do MuGA para prolemas com 100 variáveis após 500.000 chamadas à função de avaliação



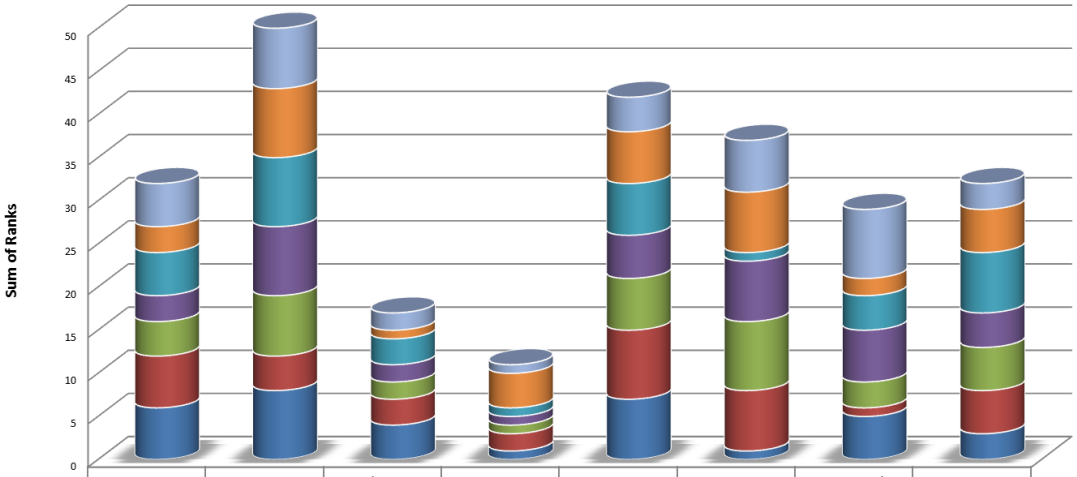
	MLCC	EPUS-PSO	JDEdynNP-F	MTS	DewSAcc	DMS-PSO	LSEDA-gl	MUGA
F7	1	8	3	2	6	7	4	5
F6	6	8	4	1	7	1	5	3
F5	7	8	5	1	6	1	4	1
F4	4	8	2	1	5	7	6	3
F3	5	8	3	1	4	2	7	6
F2	8	7	3	2	6	5	1	4
F1	7	8	4	1	5	1	5	3

Ranking dos algoritmos da competição CEC2008 e do MuGA para prolemas com 100 variáveis após 500.000 chamadas à função de avaliação.

Otimização de problemas com 500 Variáveis

	F1	F2	F3	F4	F5	F6	F7
MLCC	4.30E-13	6.67E+01	9.25E+02	1.79E-11	2.13E-13	5.34E-13	-4.44E+03
EPUS-PSO	8.45E+01	4.35E+01	5.77E+04	3.49E+03	1.64E+00	6.64E+00	-3.51E+03
JDEdynNP-F	9.32E-14	8.46E+00	6.61E+02	1.47E-12	4.21E-14	1.49E-13	-6.88E+03
MTS	0.00E+00	7.32E-06	5.03E-03	0.00E+00	0.00E+00	6.18E-12	-7.08E+03
DewSAcc	2.09E-09	7.57E+01	1.81E+03	3.64E+02	6.90E-04	4.80E-01	-5.75E+03
DMS-PSO	0.00E+00	6.89E+01	4.67E+07	1.61E+003	0.00E+00	2.00E+00	-4.20E+03
LSEDA-gl	2.27E-13	2.72E-10	8.67E+02	8.55E+02	1.14E-13	3.13E-13	4.83E+01
MUGA	**4.11E-26**	**4.41E+01**	**1.09E+03**	**6.02E+01**	**1.48E-03**	**2.07E-11**	**-6.69E+03**

Resultado dos algoritmos da competição CEC2008 e do MuGA para prolemas com 500 variáveis após 2.500.000 chamadas à função de avaliação



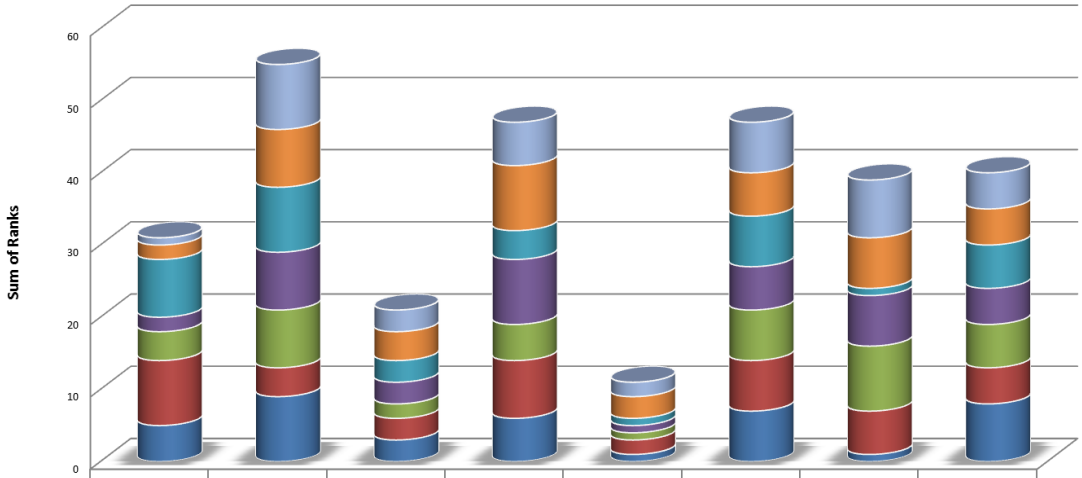
	MLCC	EPUS-PSO	JDEdynNP-F	MTS	DewSAcc	DMS-PSO	LSEDA-gl	MUGA
F7	5	7	2	1	4	6	8	3
F6	3	8	1	4	6	7	2	5
F5	5	8	3	1	6	1	4	7
F4	3	8	2	1	5	7	6	4
F3	4	7	2	1	6	8	3	5
F2	6	4	3	2	8	7	1	5
F1	6	8	4	1	7	1	5	3

Ranking dos algoritmos da competição CEC2008 e do MuGA para prolemas com 500 variáveis após 2.500.000 chamadas à função de avaliação.

Otimização de problemas com 1000 Variáveis

	F1	F2	F3	F4	F5	F6	F7
MLCC	8.46E-13	1.09E+02	1.80E+03	1.37E-10	4.18E-03	1.06E-12	-1.47E+04
EPUS-PSO	5.53E+02	4.66E+01	8.37E+05	7.58E+03	5.89E+00	1.89E+01	-6.62E+03
JDEdynNP-F	1.14E-13	1.95E+01	1.31E+03	2.17E-04	3.98E-14	1.47E-11	-1.35E+04
UEP	5.37E-12	1.05E+02	1.96E+03	1.03E+04	8.87E-14	1.99E+01	-1.18E+04
MTS	0.00E+00	4.72E-02	3.41E-04	0.00E+00	0.00E+00	1.24E-11	-1.40E+04
DewSAcc	8.79E-03	9.61E+01	9.15E+03	1.82E+003	3.58E-03	2.30E+00	-1.06E+04
DMS-PSO	0.00E+00	9.15E+01	8.98E+09	3.84E+03	0.00E+00	7.76E+00	-7.51E+03
LSEDA-gl	3.22E-13	1.04E-05	1.73E+03	5.45E+02	1.71E-13	4.26E-13	-1.35E+04
MUGA	**2.84E-02**	**8.57E+01**	**3.87E+03**	**9.23E+02**	**1.77E-03**	**6.68E-03**	**-1.32E+04**

Resultado dos algoritmos da competição CEC2008 e do MuGA para prolemas com 500 variáveis após 5.000.000 chamadas à função de avaliação



	MLCC	EPUS-PSO	JDEdynNP-F	UEP	MTS	DewSAcc	DMS-PSO	MUGA
F7	1	9	3	6	2	7	8	5
F6	2	8	4	9	3	6	7	5
F5	8	9	3	4	1	7	1	6
F4	2	8	3	9	1	6	7	5
F3	4	8	2	5	1	7	9	6
F2	9	4	3	8	2	7	6	5
F1	5	9	3	6	1	7	1	8

Ranking dos algoritmos da competição CEC2008 e do MuGA para prolemas com 1000 variáveis após 5.000.000 chamadas à função de avaliação.

Anexo F - MuGA-T – Mutiset Genetic Algorithm Tool

O MuGA-T é a interface gráfica para os algoritmos que estão implementados no sistema. A interface foi projetada de forma a ler de forma dinâmica os elementos que são programados, isto é, se for adicionado um elemento qualquer ao sistema a interface gráfica reconhece-o automaticamente e permite usá-lo nas simulações

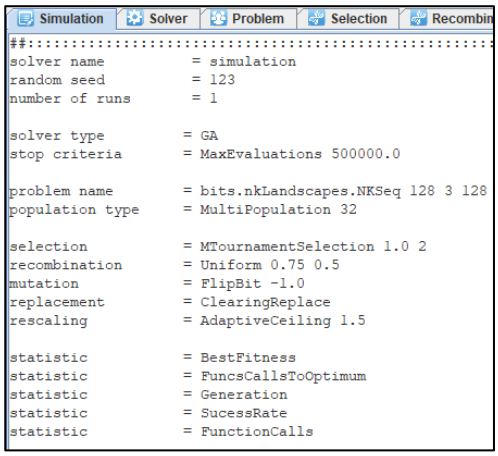
Multiset Genetic Algorithm Tool

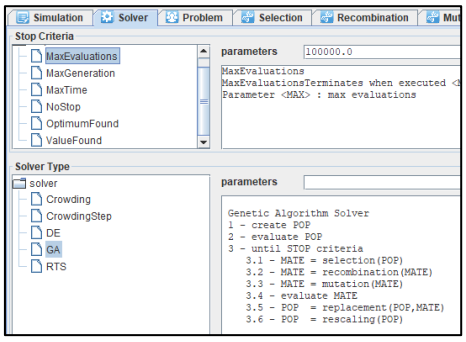
Simulações

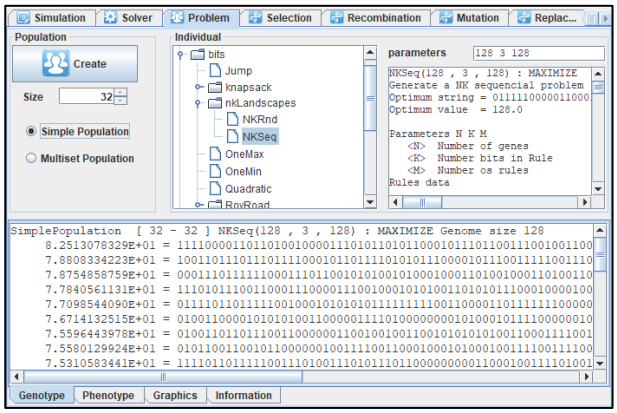
As simulações permitem fazer a configuração dos parâmetros do algoritmo evolutivo. A configuração é texto com pares de <Chave = Valor> que configuram o algoritmo. O utilizador pode editar o texto ou utilizar a interface gráfica para configurar cada um dos parâmetros.

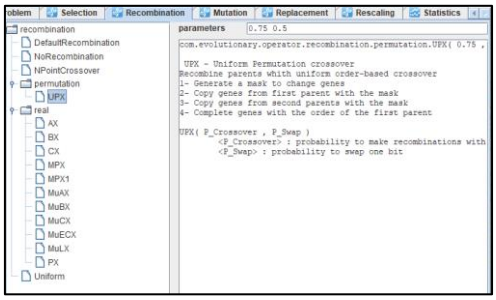
Chave	Valor
solver name	Nome da simulação
random seed	Semente dos números aleatórios
number of runs	Número de execuções do solver
solver type	Tipo de Algoritmo evolutivo
stop criteria	Critério de paragem
population type	Tipo de população
selection	Algoritmo de seleção de reprodutores
recombination	Algoritmo de recombinação
mutation	Algoritmo de mutação
replacement	Algoritmo de substituição
rescaling	Algoritmo de redimensionamento
statistic	Estatísticas do processo evolutivo

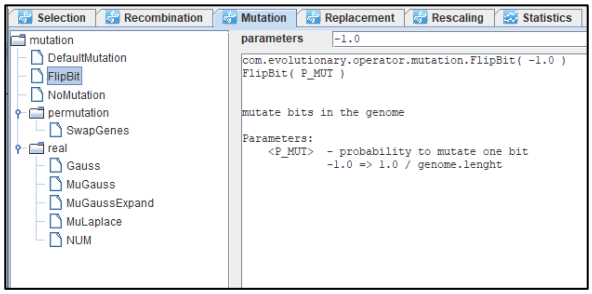
Configuração da simulação

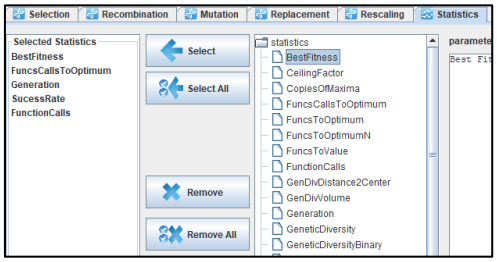










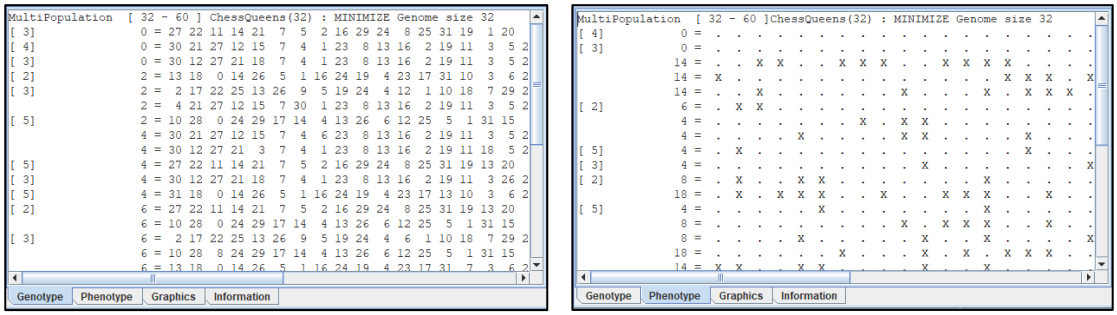


MuGA - Configuração de uma simulação

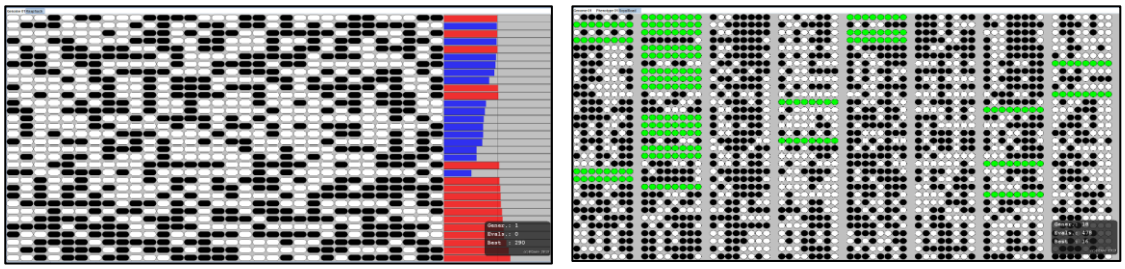
Visualização da população

Todas as simulações permitem ter várias vistas sobre a população principal

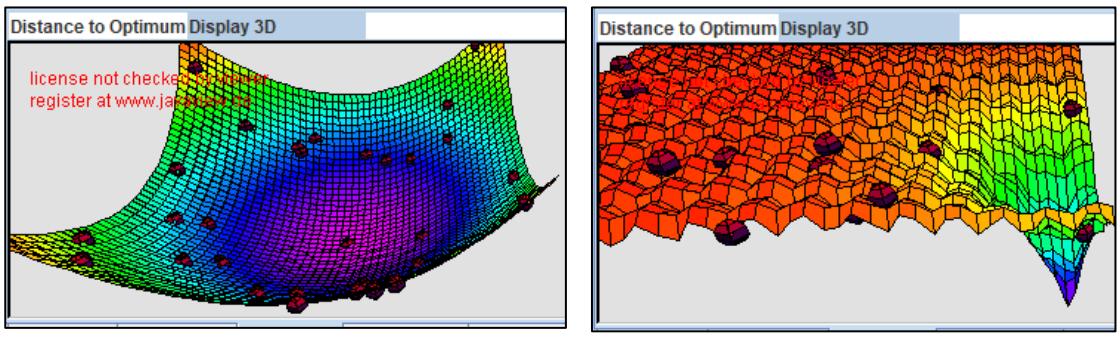
Separador	Descrição
Genotype	Genótipos dos indivíduos
Phenotype	Fenótipo dos indivíduos
Graphics	Representação gráfica dos indivíduos- Dependente dos problemas



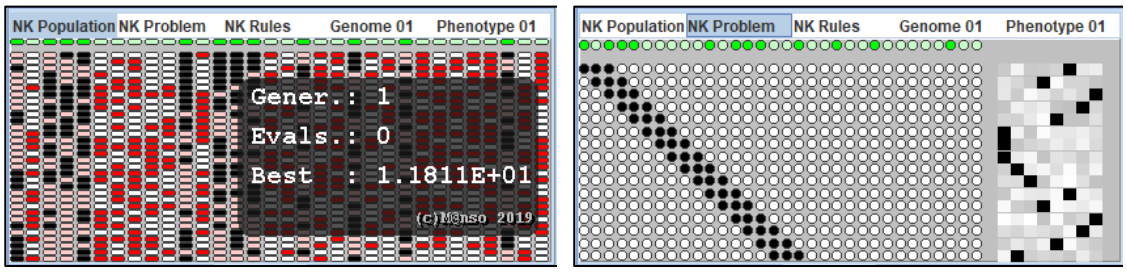
MuGA - Genótipo e Fenótipo de uma população do problema ChessQueen



MuGA – Visualização Gráfica do problema Knapsack e RoyalRoad



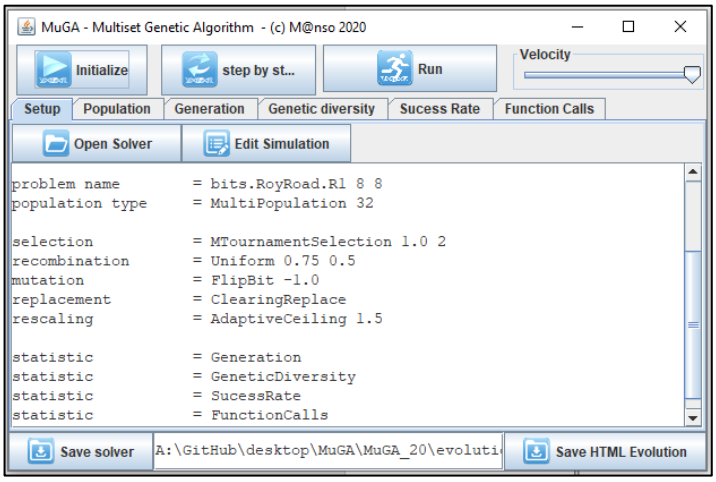
MuGA – Visualização de funções reais 2D



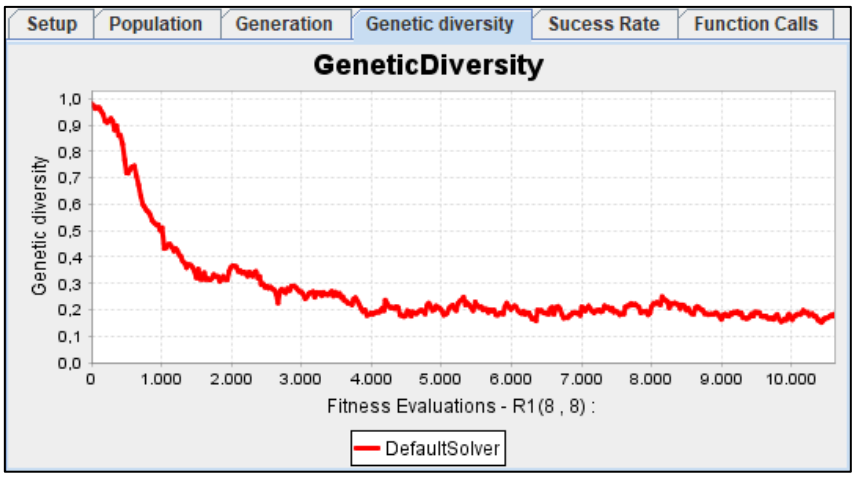
MuGA – Visualização Gráfica do problema NK-Landscapes

Evolução de populações

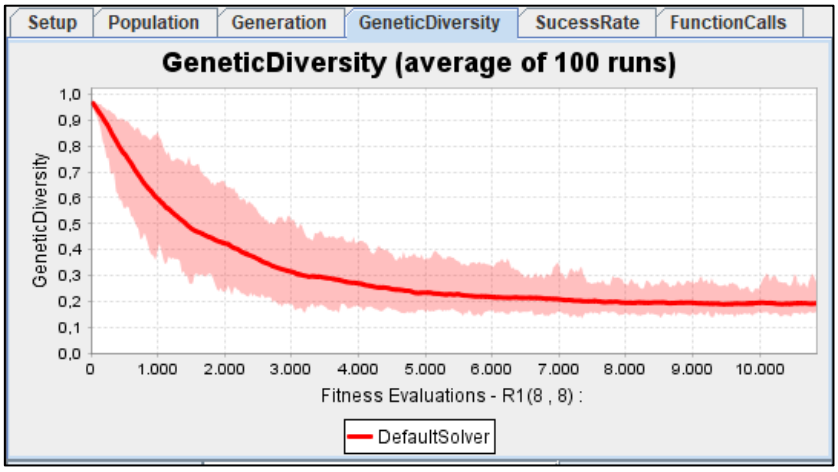
Visualizar a evolução de uma população de indivíduos



MuGA – Interface para a evolução de populações



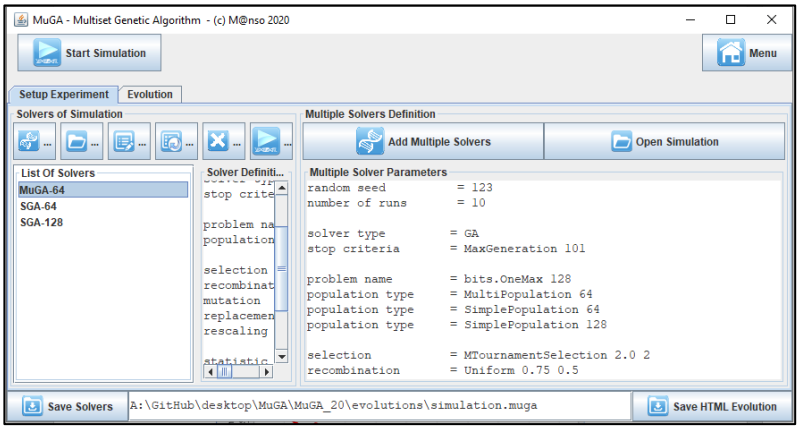
MuGA – Visualização da evolução da diversidade genética da população



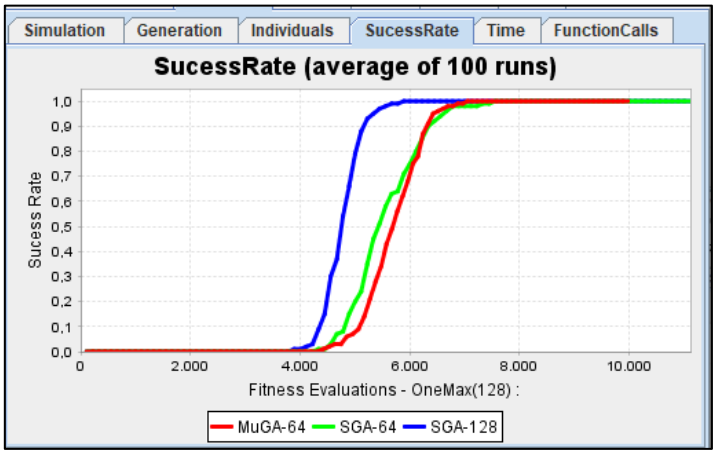
MuGA – Visualização da evolução da diversidade genética de 100 populações:(mínimo, máximo e média)

Testes com algoritmos evolutivos

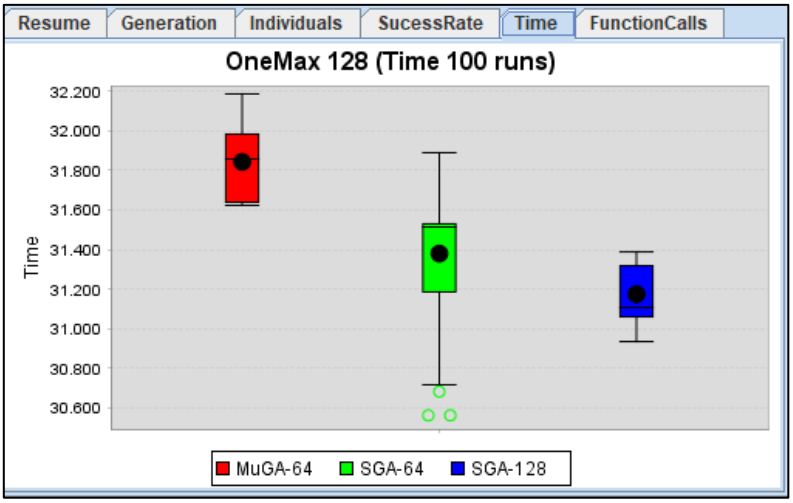
Executar várias simulações em simultâneo.



MuGA – Interface para os testes evolutivos



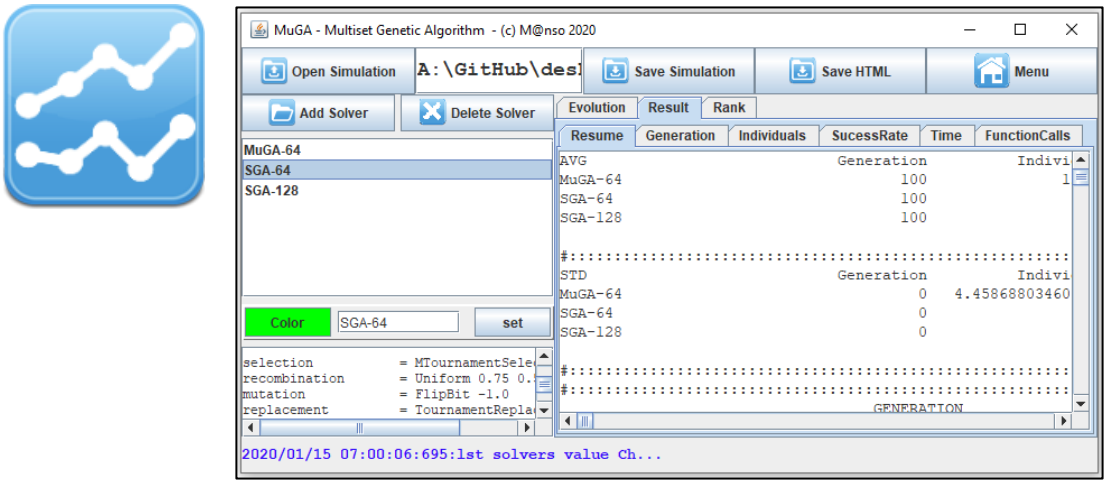
MuGA -Evolução da taxa de sucesso do teste



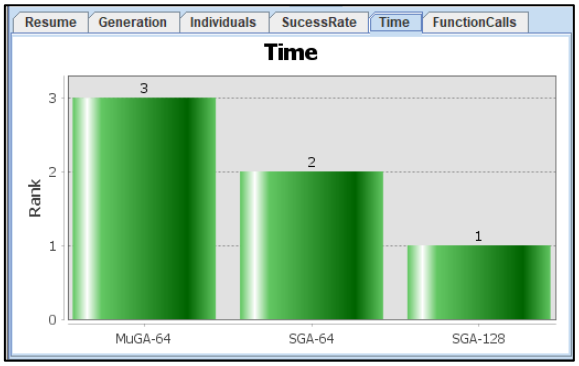
MuGA – Resultado final da experiência

Avaliação de algoritmos evolutivos

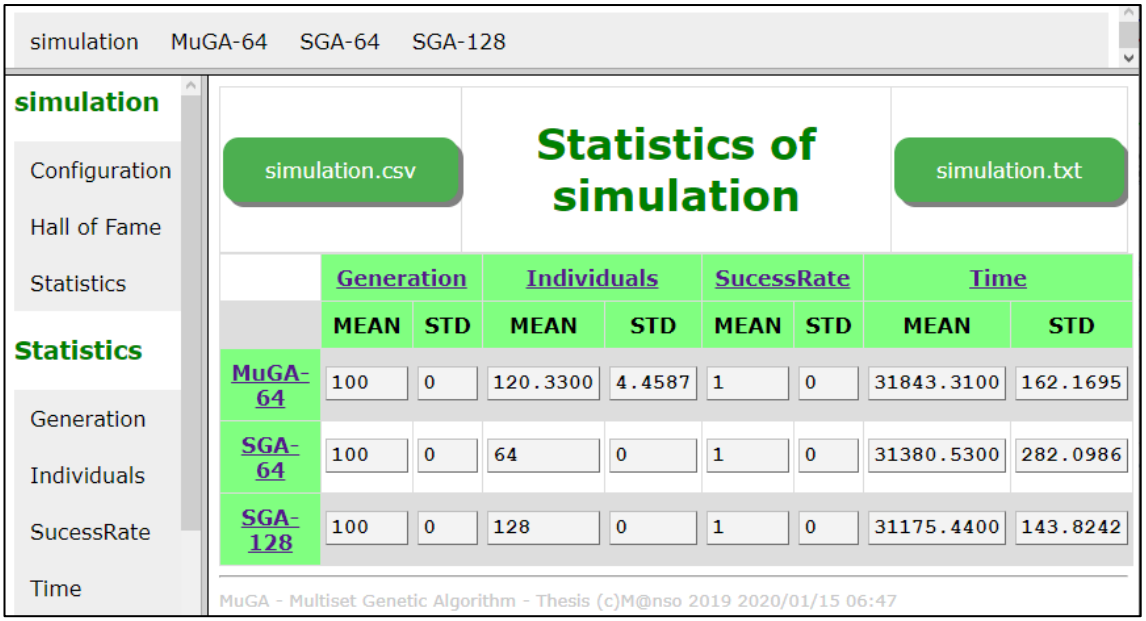
Comparar simulações



MuGA – Interface para a comparação de simulações



MuGA – Ranking dos algoritmos



MuGA – Publicação de resultados