1. Descrição do Problema

Esta seção detalha o desafio abordado pelo projeto, seus objetivos e como o sucesso da solução é medido.

O Problema Real

Em ambientes de datacenter modernos, a virtualização é uma prática padrão. Administradores de sistemas enfrentam o desafio constante de alocar de forma eficiente os recursos de hardware (CPU e RAM) de servidores físicos para um número crescente de máquinas virtuais (VMs). Uma alocação ineficiente leva ao desperdício de capacidade, aumentando os custos com energia, refrigeração e manutenção de hardware subutilizado.

Novos projetos precisam ser analisados em termos de capacidades do datacenter para saber se os mesmos irão aumentar o número de servidores físicos necessários para implementação da solução.

Este desafio do mundo real é uma instância do clássico problema de otimização combinatória conhecido na ciência da computação como Problema do Empacotamento (Bin Packing Problem). Neste problema:

- As Máquinas Virtuais (VMs) são os “itens” a serem empacotados, cada uma com um “tamanho” bidimensional (requisitos de CPU e RAM).
- Os Servidores Físicos são as “caixas” (bins), cada uma com uma capacidade finita (CPU e RAM totais).

O projeto DRE (Datacenter Resource Emulator) se propõe a resolver este problema, não como uma ferramenta de balanceamento em tempo real, mas como um instrumento de planejamento estratégico para otimizar a infraestrutura de um datacenter.

Objetivos

O projeto possui os seguintes objetivos, alinhados com o problema descrito:

- Objetivo Principal: Minimizar o número de servidores físicos ativos necessários para hospedar uma dada carga de trabalho de VMs. Este é o foco principal da otimização, visando a máxima consolidação e redução de custos operacionais.
- Objetivo Secundário (Restrição): Garantir que a alocação de VMs em cada servidor físico não exceda sua capacidade máxima de CPU e RAM. A validade da solução é um pré-requisito para a otimização.

Critérios de Sucesso

O sucesso do algoritmo genético implementado é medido pelos seguintes critérios:

- Validade da Solução: Uma solução é considerada válida somente se todas as VMs forem alocadas a um servidor e nenhuma restrição de capacidade (CPU e RAM) for violada. Soluções inválidas são descartadas ou recebem uma penalidade máxima durante o processo de fitness.
- Eficácia da Otimização: A eficácia é demonstrada pela capacidade do algoritmo de encontrar uma solução que utilize um número de servidores

significativamente menor do que métodos de solução convencionais (como a heurística First Fit Decreasing ou a alocação inicial Round-Robin).

- Convergência e Estabilidade: O algoritmo é considerado bem-sucedido se, em múltiplas execuções, ele consistentemente converge para a mesma solução ótima (ou para soluções com fitness muito próximo), demonstrando que o resultado não é fruto do acaso, mas de uma busca inteligente e robusta.

2. Detalhes da Implementação do Algoritmo

A solução foi desenvolvida em Python, utilizando uma arquitetura modular para separar o modelo de dados, os operadores do algoritmo genético e a visualização. A abordagem central foi criar um sistema que adere ao paradigma “Lousa Limpa”, onde o estado do datacenter é reconstruído a cada passo, garantindo a consistência dos dados e prevenindo bugs de estado.

2.1. Modelo de Dados e Representação

O ambiente do datacenter é modelado por duas classes principais no arquivo datacenter_model.py:

- MaquinaVirtual: Representa um “item” a ser alocado. Armazena seus requisitos de cpu_req e ram_req, além de um id único e seu nome_real para relatórios.

- ServidorFisico: Representa uma “caixa” ou bin. Possui uma capacidade total de cpu_total e ram_total e uma lista vms_hospedadas que rastreia as VMs alocadas. Métodos como .pode_hospedar(), .alocar_vm(), .desalocar_vm() e .resetar() encapsulam a lógica de manipulação de estado do servidor.

- O “cromossomo”, ou a representação de uma solução individual, é uma lista de inteiros. O índice da lista corresponde ao id da MaquinaVirtual, e o valor naquela posição corresponde ao id do ServidorFisico onde ela está alocada. Ex: **solucao[10] = 5** *significa que a VM de ID 10 está alocada no Servidor de ID 5*. Esta estrutura garante por design que cada VM tenha apenas uma alocação.

2.2. Geração da População Inicial

Para iniciar a busca, a função generate_round_robin_population cria uma população homogênea. Ela aloca as VMs sequencialmente entre todos os servidores disponíveis (Round-Robin). O objetivo desta abordagem é criar deliberadamente uma solução inicial subótima, com um fitness alto (muitos servidores em uso), para que a melhoria proporcionada pelo algoritmo genético ao longo das gerações seja claramente visível durante a demonstração.

2.3. Função de Fitness

A função calculate_fitness é o “juiz” do algoritmo. Seguindo a arquitetura “Lousa Limpa”, sua primeira ação é sempre chamar o método .resetar() em todos os servidores para garantir que a avaliação comece de um estado limpo. Em seguida, ela simula a alocação do cromossomo recebido e:

- Verifica a validade: Se a alocação sobrecarregar qualquer servidor, a função retorna um fitness infinito (float(‘inf’)), penalizando e descartando efetivamente a solução inválida.

- Calcula a qualidade: Se a solução for válida, o fitness retornado é simplesmente o número total de servidores que foram utilizados. **O objetivo do AG é minimizar este valor**.

2.4. Operadores Genéticos: Crossover e Mutação

O núcleo da inteligência do projeto reside nos seus operadores, que foram projetados para explorar o espaço de soluções de forma eficaz.

Crossover Convencional - crossover_por_consenso (CPC): Utilizado como método de comparação, o Crossover de Particionamento por Consenso primeiro identifica as alocações em que ambos os pais concordam (o “consenso”) e as copia diretamente para o filho. Em seguida, para as VMs restantes (o “conflito”), ele tenta herdar a alocação de cada um dos pais e, se ambas as opções forem inválidas (por exemplo, por falta de capacidade), ele recorre a uma heurística “First Fit” para encontrar o primeiro servidor disponível. Esta abordagem preserva as boas características compartilhadas, ao mesmo tempo que introduz variação ao resolver os conflitos.

Crossover Principal - doac_cross (Dominant Optimal Anti-Cancer). Este é o crossover heurístico customizado e principal do projeto. Sua filosofia, inspirada na biologia, é mais agressiva e direcionada:

- **Gene Dominante**: A função primeiro identifica qual dos pais é o melhor. Em seguida, ela encontra o “gene dominante” desse pai, definido como o servidor mais potente (maior soma de CPU e RAM) em uso. O filho já nasce herdando todo o conjunto de VMs que o melhor pai havia alocado neste servidor, garantindo a preservação de uma característica de alta qualidade.

- **Construção Alternada**: As VMs restantes são alocadas de forma semelhante ao CPC, herdando alternadamente dos pais.

- **Tratamento “Anti-Câncer”**: Como uma etapa final de otimização, a função identifica o servidor “câncer” — o menos potente em uso no filho recém-criado. Ela então tenta realocar todas as VMs desse servidor para os outros servidores mais fortes já ativos. Se for bem-sucedida, o filho é aprimorado, utilizando um servidor a menos.

Mutação - swap_mutation: Para introduzir novas variações e evitar a convergência prematura, o operador de mutação seleciona aleatoriamente duas VMs em servidores diferentes e verifica se a troca entre elas é válida (se a VM1 cabe no servidor da VM2 e vice-versa). Se a troca for possível sem violar as restrições de capacidade, ela é efetuada no cromossomo.

3. Análises de Resultados

Para demonstrar a eficácia do algoritmo genético e, em particular, do operador de crossover customizado, foram realizados múltiplos testes comparativos. O desempenho do crossover principal, D.O.A.C. (Dominant Optimal Anti-Cancer), foi comparado com o de um método de crossover convencional e bem estabelecido para problemas de particionamento, o Crossover por Consenso (CPC).

Ambos os algoritmos foram executados sob as mesmas condições, utilizando o cenário de dados reais do VMware, que consiste em 730 Máquinas Virtuais e 31 Servidores Físicos. A população inicial para todos os testes foi gerada pelo método Round-Robin, resultando em um fitness inicial de 31 servidores em uso.

3.1. Desempenho do Método Convencional (Crossover por Consenso)

O Crossover por Consenso foi testado com diferentes taxas de mutação para avaliar sua performance e estabilidade. Os resultados foram os seguintes:

- Taxa de Mutação (0.5): O algoritmo demonstrou capacidade de otimização, convergindo consistentemente para uma solução final que utilizava 18 servidores.
- Taxa de Mutação (0.8): Com uma taxa de mutação mais alta, o algoritmo apresentou um comportamento mais instável, oscilando entre soluções de 18 e 20 servidores, indicando uma dificuldade em reter as melhores características encontradas.

O CPC provou ser um método válido, melhorando significativamente a solução inicial. No entanto, sua performance final ficou aquém do ótimo e mostrou-se sensível a variações na taxa de mutação.

3.2. Desempenho do Algoritmo Otimizado (D.O.A.C.)

O crossover D.O.A.C. foi desenvolvido com heurísticas específicas para o problema, como a herança do “Gene Dominante” e o refinamento “Anti-Câncer”. Seus resultados demonstram uma superioridade marcante:

- Taxa de Mutação (0.2 a 0.5): Independentemente da taxa de mutação utilizada, o D.O.A.C. convergiu de forma estável e robusta para a solução ótima de 16 servidores em todas as execuções.
- A consistência do D.O.A.C. em encontrar a melhor solução, mesmo com taxas de mutação mais baixas, indica que sua eficácia deriva da inteligência incorporada em sua lógica de crossover, e não apenas da exploração aleatória da mutação.

3.3. Comparativo e Tabela de Resultados

A comparação direta dos melhores resultados obtidos por cada método evidencia a eficácia da abordagem customizada:

Método de Crossover	Fitness Inicial	Melhor Fitness Final	Redução de Servidores
Crossover por Consenso (CPC)	31	18	41.9%
D.O.A.C. (Implementação Principal)	**31**	**16**	**48.4%**

O D.O.A.C. não apenas encontrou uma solução que utiliza dois servidores a menos que o método convencional, como também demonstrou maior estabilidade e confiabilidade em seus resultados, cumprindo com sucesso o objetivo principal do projeto.

```
[ Crossover: Por Consenso ]
    |-> Cenário: Real em vmware.
    |-> População Inicial = generate_round_robin_population.
    |   \-> Distriuiçao igual das vms nos servidores.
    |
    |-> Mutação: swap_mutation
    |       \-> Taxa: 0.5
```

```
    |
    |-> Finaliza no número máximo de gerações sem evoluir = 200
    |
    \-> Resultados
        |-> Geração 210: Melhor Fitness = 20
        |-> Geração 212: Melhor Fitness = 20
        \-> Geração 278: Melhor Fitness = 18



[ Crossover: Por Consenso ]
    |-> Cenário: Real em vmware.
    |-> População Inicial = generate_round_robin_population.
    |   \-> Distriuiçao igual das vms nos servidores.
    |
    |-> Mutação: swap_mutation
    |       \-> Taxa: 0.8 <-- Alteração.
    |
    |-> Finaliza no número máximo de gerações sem evoluir = 200
    |
    \-> Resultados
        |-> Geração 210: Melhor Fitness = 20
        |-> Geração 237: Melhor Fitness = 18
        \-> Geração 230: Melhor Fitness = 20



[ Crossover: D.O.A.C. - Dominant Optimal Anti-Cancer ] <--- Alteração.
        |   \-> Minha autoria baseada na biologia real.
    |
    |-> Cenário: Real em vmware.
    |-> População Inicial = generate_round_robin_population.
    |   \-> Distriuiçao igual das vms nos servidores.
    |
    |-> Mutação: swap_mutation
    |       \-> Taxa: 0.5 <--- Alteração.
    |
    |-> Finaliza no número máximo de gerações sem evoluir = 200
    |
    \-> Resultados
        |-> Geração 215: Melhor Fitness = 16
        |-> Geração 210: Melhor Fitness = 16
        \-> Geração 215: Melhor Fitness = 16



[ Crossover: D.O.A.C. - Dominant Optimal Anti-Cancer ]
        |   \-> Minha autoria baseada na biologia real.
    |
    |-> Cenário: Real em vmware.
    |-> População Inicial = generate_round_robin_population.
    |   \-> Distriuiçao igual das vms nos servidores.
    |
    |-> Mutação: swap_mutation
    |       \-> Taxa: 0.2 <--- Alteração.
    |
    |-> Finaliza no número máximo de gerações sem evoluir = 200
    |
```

```
\-> Resultados
    |-> Geração 215: Melhor Fitness = 16
    |-> Geração 215: Melhor Fitness = 16
    \-> Geração 215: Melhor Fitness = 16
```

4. Conclusões

O desenvolvimento do projeto DRE (Datacenter Resource Emulator) demonstrou com sucesso a aplicação prática e a eficácia de Algoritmos Genéticos para resolver um problema de otimização complexo e relevante no mundo real: a alocação de máquinas virtuais em servidores físicos.

Principais Conclusões:

Validação da Abordagem Genética: Os resultados obtidos provam que a abordagem evolucionária é não apenas viável, mas altamente eficaz para o Problema do Empacotamento. O algoritmo foi capaz de navegar por um vasto espaço de soluções possíveis e convergir consistentemente para um resultado ótimo, superando métodos convencionais.

A Superioridade da Heurística Customizada: A comparação direta entre o crossover convencional (CPC) e o crossover customizado (DOAC) foi a descoberta mais significativa do projeto. Enquanto o CPC obteve uma melhoria respeitável, o desempenho superior e a estabilidade do DOAC destacam uma conclusão fundamental em computação evolucionária: a incorporação de conhecimento específico do problema (heurísticas) nos operadores genéticos acelera drasticamente a convergência e eleva a qualidade da solução final. Estratégias como o “Gene Dominante” e o “Tratamento Anti-Câncer” não foram apenas metáforas, mas mecanismos eficazes que guiaram a busca para regiões mais promissoras do espaço de soluções.

No fim foi obtido uma ferramenta real para uso em um Datacenter real, visando prever alocações com a entrada de novos projetos no mesmo.