Relatório Final - Meta-Heurísticas

Pedro Henrique Kroll Nascimento[1], **Gabriel Daher Monteiro Bastos Nascimento**[2], **Prof. Igor Machado Coelho**[3]

[1 23] Universidade Federal Fluminense (UFF)
[123] Instituto de Computação

pedrokroll@id.uff.br[1], gabrieldaher@id.uff.br[2], imcoelho@ic.uff.br[3]

Resumo. *Este trabalho aplica as meta-heurísticas Simulated Annealing (SA) e GRASP com Busca Local ao problema de posicionamento de torres em um ambiente Tower Defense, buscando maximizar o dano ao inimigo sob restrições de custo. O modelo utiliza uma grade para representar o mapa e um vetor para descrever a alocação das torres. Os experimentos mostram que o SA produz soluções de maior qualidade por escapar de ótimos locais, enquanto o GRASP+LS é mais rápido e eficiente em encontrar soluções satisfatórias. Conclui-se que o SA é mais adequado quando a prioridade é qualidade, e o GRASP+LS quando o tempo de execução é fator determinante.*

1. Introdução

Nesse trabalho, usaremos meta-heurísticas aplicadas ao contexto de um Tower Defense em que haverá um mapa, com início e fim, torres, que conseguem dar dano em tropas em que elas atiram e possuem um custo para serem colocadas no mapa, e um inimigo que começará no início do mapa e tentará chegar até o fim e as torres devem infligir a maior quantidade de dano possível nesse inimigo. A ideia central do problema é **maximização de dano**. Além disso, há outro fator importante a ser considerado: há diferentes possibilidades de mapas a serem escolhidos para serem utilizados, cada mapa terá uma certa quantidade de casas, um caminho específico a ser seguido pelo inimigo e, principalmente, cada mapa possui um **budget**, ou seja, um *orçamento* limitado para que sejam compradas as torres a serem utilizadas, sendo esse mais um fator para ser pensado e mais um pilar da otimização do problema: **causar a maior quantidade de dano possível respeitando o orçamento limitado pré-estabelecido pelo mapa escolhido**.

Elementos principais do problema:

Mapa: Basicamente será o ambiente em que se passará o contexto do Tower Defense, é nele que o inimigo se deslocará e também será nele que as torres serão colocadas para atirarem no inimigo que estiver se locomovendo.

Torres: São colocadas nas casas disponíveis do mapa e causam dano ao inimigo se ele estiver dentro do alcance delas. São de 3 tipos:

• **Fraca**: Possuem menor custo, o que possibilita que sejam colocadas em maior quantidade, maior alcance, menor dps;

• **Média**: Possuem custo, alcance e dps intermediários, maiores do que os das torres fracas e menores do que os das torres fortes;

• **Forte**: Possuem o maior custo, e dps, porém com alcance menor que as outras, permitindo que deem maior dano ao ini-

migo que esteja em sua área de cobertura, porém sacrificando mais o orçamento.

Inimigo: O inimigo é um *NPC* que apenas caminhará pela trajetória do mapa até chegar ao final. Um ponto importante a se ressaltar é que ele possui vida ”infinita”, sendo sua função unicamente receber os ataques das torres, a fim de mostrar quais configurações conseguiram maximizar o *output* de dano total.

Esse artigo tem como objetivo principal utilizar algumas Heurísticas para ajudar a resolver problemas de otimização. Heurísticas são muito importantes em contextos de problemas complexos, uma vez que esses porblemas muitas vezes não possuem maneiras fáceis/rápidas de serem resolvidos de maneira ótima, portanto, as Heurísticas, com seu objetivo de não obter a solução 100% ótima, mas sim uma solução relativamente próxima da ótima, se mostram muito eficientes na hora de alcançar resultados satisfatórios sem necessitar de gasto excessivo de tempo.

2. Formulação do Problema

2.1. Representação do mapa

O mapa do Tower Defense pensado para esse trabalho consiste de uma grade bidimensional, onde cada célula dessa grade representa uma potencial posição do ambiente. Um ponto importante de implementação é que, para fins de visualização e depuração, o mapa foi renderizado utilizando a bibilioteca **Matplotlib**, o que permitiu distinguir-se os diferentes tipos de células.

Cada uma das células, coordenadas (i, j), pertence a uma das seguintes categorias:

• **Células construíveis** (**.**): Posições em que as torres podem ser colocadas;

• **Células não construíveis** (**X**): Obstáculos ou regiões inválidas para construção;

• **Caminho do inimigo** (**P**): Sequência fixa de células por onde o inimigo percorre o mapa;

• **Células de início e fim** (**S** e **E**): Início e término do trajeto que o inimigo percorre.

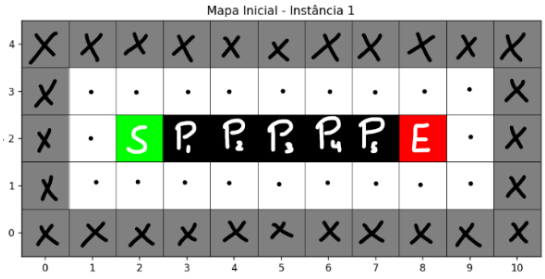

Figura 1. Instância genérica

O caminho percorrido pelo inimigo é representado pela sequência ordenada

$$\mathcal{P} = \{P_1, P_2, \ldots, P_L\}$$

2.2. Modelagem da Torres

Cada torre disponível pertence a um conjunto finito $T = \{T_0, T_1, T_2\}$, correspondendo, respectivamente, às categorias **Fraca**, **Média** e **Forte**. Cada torre T é caracterizada por três atributos principais:

• **Dano por segundo (dmg(t))**: Quantidade de dano aplicado ao inimigo enquanto ele permanece em seu alcance;

• **Alcance (rng(t))**: Raio euclidiano dentro do qual a torre pode atingir o inimigo;

• **Custo (cst(t))**: Valor monetário descontado do orçamento total disponível.

O balanceamento das torres segue um comportamento típico de jogos do gênero Tower Defense:

• **Torre Fraca**: baixo dano, **longo alcance**, custo reduzido. Costuma ser mais eficiente em corredores longos, onde seu alcance estendido maximiza o número de acertos. (*Tile Azul*);

• **Torre Média**: compromisso entre dano, custo e alcance. (*Tile Roxo*);

• **Torre Forte**: alto dano, **baixo alcance** e maior custo. Costuma ser eficiente em curvas do caminho, onde consegue atingir o inimigo repetidas vezes enquanto este faz a manobra. (*Tile Amarelo*).

2.3. Representação da Solução

Uma solução é representada por um vetor $x = \{x_0, x_1, x_2\}$, onde $n = |B|$ é o número de células construíveis.
Cada elemento assume valores
$x_i \in \{-1, 0, 1, 2\}$, onde:

• -1 indica que nenhuma torre foi construída na célula *i*.
• 0, 1 e 2 indicam, respectivamente, torres dos tipos **Fraca**, **Média** e **Forte**.

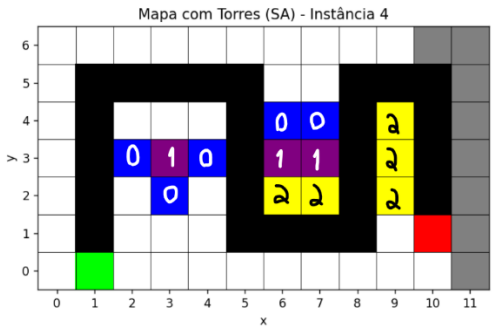

Figura 2. Torres na Solução

A cada célula construída com torre *i*, o dano total infligido ao inimigo é calculado somando-se todas as contribuições das torres que atingem as posições do caminho.
O inimigo permanece **1 segundo** em cada célula do caminho, o que torna equivalente à contagem de dano por hit ou por segundo.

2.4. Função Objetivo

O objetivo do problema é: **Maximizar** $D(x)$, onde $D(x)$ representa o dano total causado ao inimigo ao longo de toda sua trajetória P.

2.5. Restrições

A solução deve respeitar 2 restrições fundamentais:

$$\sum_{i \in \mathcal{B}, x_i \neq -1} \text{cst}(x_i) \leq B,$$

sendo B o orçamemnto (*budget*) máximo permitido para construção de torres, e

$$x_i \in \{-1, 0, 1, 2\}, \quad \forall i,$$

assegurando **que cada célula construível receba no máximo uma torre**.

3. Meta-Heurísticas Aplicadas

3.1. Construtivo inicial

A solução inicial é produzida por uma heurística aleatória simples: começa-se com todas as células vazias e, para cada posição construível, tenta-se inserir uma torre aleatoriamente. Inserções só são aceitas se respeitarem o orçamento. O resultado é uma solução viável e diversa, utilizada como ponto de partida para as metaheurísticas.

3.2. Simulated Annealing

O Simulated Annealing é uma metaheurística de busca local probabilística inspirada no processo físico de recozimento de metais. O método permite a aceitação controlada de soluções piores durante as fases iniciais da busca, evitando que o algoritmo fique preso em ótimos locais. A aceitação é guiada por uma temperatura T que diminui progressivamente ao longo da execução.

Descrição resumida da implementação:

3.2.1. Inicialização:

O algoritmo começa com uma solução viável e uma temperatura inicial T_0 suficientemente alta para permitir a aceitação de movimentos piores.

3.2.2. Iteração Principal (resfriamento):

Enquanto $T > T_{final}$ continua aqui , o algoritmo executa um número fixo de tentativas de vizinhança (SA_{max}), simulando o equilíbrio térmico.

3.2.3. Geração de Vizinhos:

Os vizinhos são gerados através das operações:
• **FLIP**: altera o tipo de torre em uma posição ou remove a torre.
• **SWAP**: troca conteúdos entre duas posições construíveis. Apenas vizinhos viáveis, respeitando o orçamento, são considerados.

3.2.4. Critério de aceitação:

Seja Δ a variação no dano total:
• Se $\Delta \geq 0$, o vizinho é aceito
• Se $\Delta < 0$, o vizinho pode ser aceito com probabilidade: $P = e^{\Delta/T}$

3.2.5. Atualização:

A solução corrente é substituída pela vizinha aceita e a melhor solução global é atualizada sempre que ocorre melhoria.

3.2.6. Resfriamento:

A temperatura é reduzida conforme a equação: $T \leftarrow \alpha T$

3.3. GRASP + Local Search

O GRASP (Greedy Randomized Adaptive Search Procedure) é uma meta-heurística iterativa composta por duas fases:

I. Construção gulosa randomizada e

II. Refinamento por busca local.

Descrição resumida da implementação:

3.3.1. Loop principal:

O processo é repetido por um número fixo de iterações **max_iters**. Cada iteração produz uma solução candidata.

3.3.2. Fase de construção (Gulosa Randomizada):

• Avaliam-se todos os movimentos de inserção de torres que gerem ganho de dano e respeitem o orçamento.
• Os candidatos são ordenados por ganho marginal.
• É construída uma RCL (*Restricted Candidate List*) contendo os melhores candidatos segundo um parâmetro **alpha**.
• Um elemento da RCL é escolhido aleatoriamente e incorporado à solução.
• O processo se repete até não haver mais movimentos que aumentem o dano.

3.3.3. Fase de Busca Local (First Improvement):

• A vizinhança da solução construída é explorada por operações do tipo FLIP.
• A estratégia **Primeira Melhoria** é utilizada: o algoritmo move-se para o primeiro vizinho que apresentar dano estritamente maior.

3.3.4. Atualização:

A solução obtida após a busca local é comparada com a melhor solução global. Havendo melhora, ela é armazenada.

Ao final das iterações, o algoritmo retorna a melhor solução produzida ao longo do processo.

4. Parâmetros e Implementação

4.1. Representação das Instâncias:

As instâncias do problema foram representadas em formato *ASCII*, permitindo leitura simples e visualização direta por meio da biblioteca **Matplotlib**. Cada mapa é composto por um grid retangular, contendo células de diferentes tipos:

Legenda dos caracteres *(ASCII)*:
- **X** : Células não construíveis;
- **.** : Células construíveis;
- **S** : Início do caminho do inimigo;
- **P** : Célula pertencente ao inimigo;
- **E** : Fim do caminho do inimigo.

Figura 3. Exemplo de Instância - Mapa 01

4.2. Instâncias:

Instâncias dos 6 mapas eleborados

- **Instância 1 carregada:**

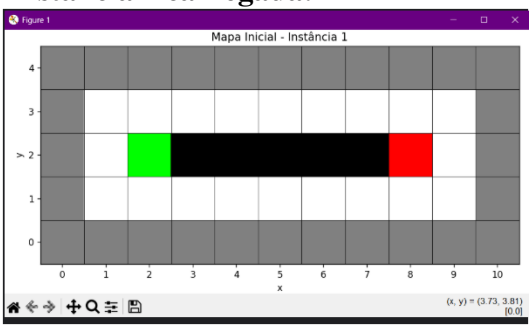

Figura 4. Grid: 11x5
Buildable cells: 20
Path length: 7
Budget: 200

- **Instância 2 carregada:**

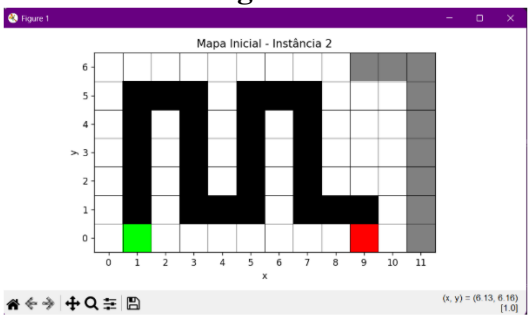

Figura 5. Grid: 12x7
Buildable cells: 48
Path length: 27
Budget: 220

- **Instância 3 carregada:**

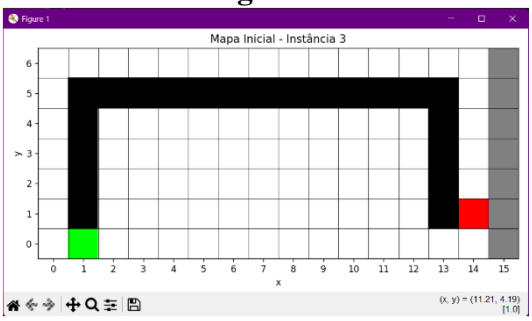

Figura 6. Grid: 16x7
Buildable cells: 82
Path length: 23
Budget: 265

- **Instância 4 carregada:**

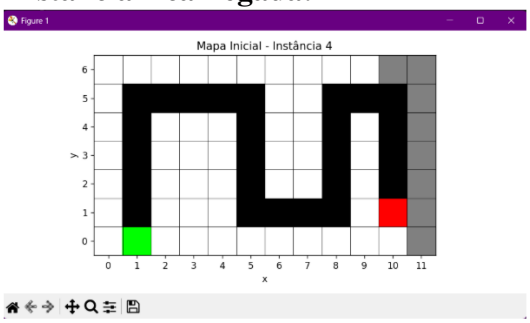

Figura 7. Grid: 12x7
Buildable cells: 49
Path length: 27
Budget: 250

- **Instância 5 carregada:**

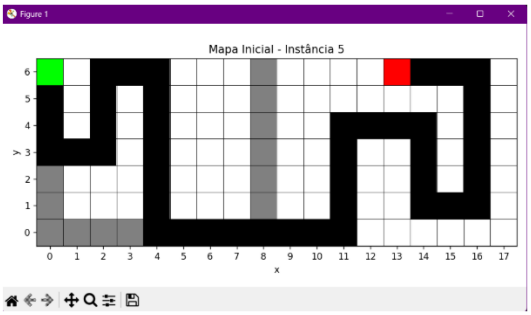

Figura 8. Grid: 18x7
Buildable cells: 70
Path length: 44
Budget: 300

• Instância 6 carregada:

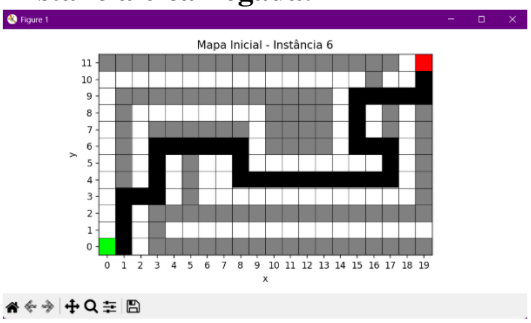

Figura 9. Grid: 20x12
Buildable cells: 100
Path length: 39
Budget: 420

5. Resultados Experimentais

A tabela a seguir resume o desempenho médio das heurísticas em termos de Dano, Custo e Tempo de Execução para todas as instâncias executadas no arquivo de resultados.

Instância	Método	Dano Médio	Custo Médio	Tempo Médio (s)
inst_1	SA	379.88	198	0.3090
inst_1	GRASP+LS	357.67	198	0.0670
inst_2	SA	872.87	218	1.0269
inst_2	GRASP+LS	902.33	220	0.6511
inst_3	SA	711.87	264	1.0483
inst_3	GRASP+LS	599.33	286	1.1295
inst_4	SA	1003.33	247	1.1389
inst_4	GRASP+LS	931.67	246	0.8544
inst_5	SA	1250.87	300	1.8868
inst_5	GRASP+LS	1149.00	297	2.0251
inst_6	SA	1360.33	419	2.3322
inst_6	GRASP+LS	1205.33	420	4.7160

5.1. Análise Comparativa de Desempenho

A análise dos resultados de execução comparando as meta-heurísticas **Simulated Annealing (SA)** e **GRASP + Busca Local (LS)** para o problema de otimização de dano em Tower Defense demonstra um trade-off claro entre **qualidade da solução** e **tempo de execução**.

5.1.1. Qualidade da Solução (Maior Dano)

A heurística **Simulated Annealing (SA)** consistentemente superou o GRASP+LS em todas as instâncias em termos de qualidade média da solução (Dano Médio), com uma margem significativa em alguns casos (Ex: inst_3, inst_6)

• **SA (Busca Global)**: O SA permite **movimentos de piora** (soluções com menor dano) através da probabilidade controlada pela temperatura ($P = e^{\Delta/T}$). Esta característica permite que o algoritmo **escape de ótimos locais** e explore partes mais amplas do espaço de busca (diversificação), o que é crucial em paisagens de otimização complexas (como é o caso do posicionamento de torres).

• **GRASP+LS (Foco Local)**: O GRASP é um procedimento que gera um ponto de partida promissor (construção gulosa-randomizada) e, em seguida, aplica uma **Busca Local (LS)**, que normalmente se detém no primeiro ótimo local encontrado (First Improvement). Embora esta abordagem seja eficaz para encontrar rapidamente bons picos locais, ela tem dificuldade em saltar para regiões globais melhores, resultando em soluções de qualidade inferior na média.

5.1.2. Eficiência (Tempo de Execução)

A heurística **GRASP+LS** demonstrou ser significativamente **mais rápida** que o SA na maioria das instâncias, exceto nas maiores (inst_5 e inst_6), onde a diferença se inverte ou se torna marginal.

• **GRASP+LS (Velocidade):** A busca local *First Improvement* consome menos tempo por iteração do que métodos que insistem em procurar a "melhor melhoria"(*Best Improvement*) ou do que a exploração controlada de pioras do SA. O GRASP é projetado para convergir rapidamente para um ótimo local, repetindo este processo por um número fixo de iterações (max_iters = 10, no caso).

• **SA (Custo de Exploração e Eficiência em Grande Escala):** O **SA** executa um número fixo de iterações internas ($SAmax = 120$) por nível de temperatura, que é multiplicado pelo número total de níveis de resfriamento. Em instâncias menores (como a inst_1, onde o SA é $5x$ mais lento), o **tempo é maior** devido à exaustiva exploração da vizinhança em cada nível de temperatura. Contudo, em **instâncias maiores e mais complexas** (inst_6), o **SA torna-se mais rápido que o GRASP+LS** ($2.3322s$ vs. $4.7160s$).Isso ocorre porque, com o aumento da complexidade do problema, o tempo do GRASP+LS passa a ser dominado pelo custo de **varrer a vizinhança densa** durante a Busca Local (*First Improvement*), que se torna proibitivo. Já a estratégia do SA de buscar vizinhos via **FLIP/SWAP aleatórios** e aplicar o critério de Metrópolis consegue manter um **custo por iteração mais estável**, diluindo o custo de exploração ao longo do tempo e resultando em uma convergência mais rápida e com maior qualidade.

5.1.3. Conclusão do Trade-Off

Heurística	Vantagem Principal	Desvantagem Principal	Aplicação Recomendada
SA	Qualidade de solução (*escapa de ótimos locais*)	Maior tempo de execução (especialmente em instâncias grandes)	Quando o **custo da solução ruim é alto** e há tempo para a exploração.
GRASP+LS	Rapidez e eficiência (encontra bom ótimo local rapidamente)	Qualidade de solução inferior (presa em ótimos locais)	Quando é necessário encontrar **uma solução "boa o suficiente" em tempo restrito**.

As meta-heurísticas cumpriram seus papéis: o **SA** performou como uma busca mais abrangente, sacrificando tempo para achar soluções de dano superior, enquanto o **GRASP+LS** atuou como um algoritmo de otimização rápida, fornecendo soluções aceitáveis em um tempo significativamente menor.

6. Conclusão

O presente estudo investigou a aplicação das meta-heurísticas **Simulated Annealing (SA)** e **GRASP combinada com Busca Local (LS)** para o problema de otimização em Tower Defense (TD), cujo objetivo consiste em **maximizar o dano total causado ao inimigo sob um orçamento limitado**. Os experimentos realizados evidenciaram diferenças relevantes no comportamento dos métodos, sobretudo no compromisso entre **qualidade da solução** e **custo computacional**.

6.1. Principais Achados

A implementação e avaliação das meta-heurísticas permitiram observar características fundamentais de cada abordagem:

• Simulated Annealing (SA)
O SA apresentou forte capacidade de **exploração global**, graças ao mecanismo probabilístico que permite a aceitação de movimentos de piora em altas temperaturas. Esse recurso possibilita ao algoritmo escapar de ótimos locais e, como consequência, alcançar soluções de maior qualidade em todas as instâncias testadas.

• GRASP + Local Search (LS)
O GRASP, por sua vez, mostrou desempenho eficaz na **exploração intensiva** do espaço de busca. A construção gulosa randomizada, seguida da Busca Local com estratégia de **First Improvement**, produz soluções competitivas em curto tempo, especialmente em instâncias pequenas e médias. Entretanto, em instâncias maiores,

a busca local apresenta custo computacional elevado devido à necessidade de varrer uma vizinhança extensa.

6.1.1. Desempenho Comparado

Em termos gerais, o **SA produziu as melhores soluções** (maior dano médio) em todas as instâncias avaliadas.

Contudo, o **GRASP+LS foi mais rápido** em mapas pequenos e moderados.

Um resultado notável é que, para instâncias maiores (como inst_6), o **SA tornou-se mais eficiente em tempo** do que o GRASP+LS, sugerindo que o custo por iteração do SA escala melhor em cenários complexos do que a busca local exaustiva empregada pelo GRASP.

6.2. Limitações da Abordagem

A metodologia desenvolvida apresenta algumas limitações importantes:

• **Busca Local Custosa no GRASP+LS**: A estratégia First Improvement exige a verificação contínua da vizinhança, o que se torna o principal gargalo em instâncias grandes.

• **Modelo de dano simplificado**: O cálculo de dano assume efeito puramente aditivo e um único inimigo, não contemplando interações típicas de jogos Tower Defense, como overkill, ataques em área ou sinergias entre torres.

• **Rigidez da restrição orçamentária**: Tanto o SA quanto o GRASP operam exclusivamente dentro do orçamento, sem uso de penalidades ou mecanismos de relaxamento temporário que poderiam favorecer regiões promissoras do espaço de busca.

6.3. Trabalhos Futuros

Nessa seção estão expostos alguns pontos interessantes para serem mais bem pensados e potencialmente melhorados/otimizados.

I. Ajuste Adaptativo de Parâmetros
• **GRASP+LS**: adaptar dinamicamente o número de iterações GRASP e os limites da busca local com base no tamanho da instância (número de células construíveis e extensão do caminho).
• **SA**: ajustar automaticamente a temperatura inicial e a taxa de resfriamento conforme variações observadas no ganho médio entre movimentos.

II. Extensão para Múltiplos Inimigos
Incorporar múltiplos inimigos simultâneos permitirá capturar aspectos essenciais do gênero TD, como:
• Políticas de seleção de alvo;
• Sincronização de ataques;
• Impacto de dano em área.

Essa extensão aumentaria substancialmente o realismo e a complexidade do problema.

III. Reformulação Multi-Objetivo
A análise futura pode considerar múltiplos objetivos simultâneos:
• Maximizar dano;
• Minimizar custo total;
• Minimizar o número total de torres usadas.

Técnicas como *Pareto Front*, *NSGA-II* ou busca multiobjetivo baseada em SA podem ser exploradas nessa direção.

6.4. Considerações Finais

Os resultados demonstram que ambas as meta-heurísticas apresentam desempenho consistente, mas com perfis distintos: **SA destaca-se pela qualidade**, enquanto **GRASP+LS se sobressai pela rapidez** em instâncias menores. A combinação dessas abordagens, bem como suas extensões futuras, oferece um caminho promissor para o desenvolvimento de algoritmos mais eficazes para problemas complexos de otimização em jogos Tower Defense.

7. Referências

Referência para a Busca Local (Refinamento GRASP)

SOUZA, Marcone Jamilson Freitas; PENNA, Puca Huachi Vaz. **Heurísticas de refinamento: fundamentação**. Ouro Preto: Departamento de Computação, Universidade Federal de Ouro Preto, 2025. Notas de aula de Técnicas Meta-heurísticas para Otimização Combinatória. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/HeuristicasRefinamento.pptx>.

Referência para o GRASP (Fase Construtiva)

SOUZA, Marcone Jamilson Freitas; PENNA, Puca Huachi Vaz. **Heurísticas Construtivas**. Ouro Preto: Departamento de Computação, Universidade Federal de Ouro Preto, 2024. Notas de aula de Técnicas Meta-heurísticas para Otimização Combinatória. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/HeuristicasConstrutivas.pptx>

Referência Principal (Visão Geral - GRASP e SA)

SOUZA, Marcone Jamilson Freitas. **Inteligência Computacional para Otimização: meta-heurísticas**. Ouro Preto: Departamento de Computação, Universidade Federal de Ouro Preto, 2024. Notas de Aula. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf>.

Referência para o Simulated Annealing (SA)

SOUZA, Marcone Jamilson Freitas; PENNA, Puca Huachi Vaz. **Simulated Annealing**. Ouro Preto: Departamento de Computação, Universidade Federal de Ouro Preto, 2021. Notas de aula de Técnicas Meta-heurísticas para Otimização Combinatória. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/SimulatedAnnealing.pptx>.

Referência para o Problema de Tower Defense

UMA ANÁLISE DAS CARACTERÍSTICAS DE JOGOS DIGITAIS DO SUB GÊNERO TOWER DEFENSE. Projética, UEL, v. 8, n. 2, p. 1-28, 2022. Artigo de Periódico. Disponível em: <https://ojs.uel.br/revistas/uel/index.php/projetica/article/download/44040/48129/244411>.