Relatório Final – Problema de Instalação de Software com Dependências

Grupo 1(maratona) - Gabriel Daher Monteiro, Pedro Henrique Kroll

1) Descrição do Problema

O problema consiste em selecionar um subconjunto de pacotes de software a serem instalados, **maximizando o benefício total** e **respeitando a capacidade máxima de armazenamento (b)** em MB.
Cada pacote pode depender de um ou mais outros pacotes, e essas dependências também ocupam espaço.

Formalmente, trata-se de um problema de **otimização combinatória com restrição de capacidade**, semelhante ao *Knapsack Problem*, porém com **relações de dependência entre os itens**.

A entrada define:

- `m`: número de pacotes;
- `n`: número de dependências;
- `ne`: número de relações pacote → dependência;
- `b`: capacidade máxima em MB;
- `c[i]`: benefício do pacote *i*;
- `a[j]`: peso (em MB) da dependência *j*;
- Relações: matriz indicando quais dependências são necessárias para cada pacote.

2) Métodos Propostos

Foram utilizados **dois métodos meta-heurísticos distintos** para resolver o problema:

Método 1: GRASP + Local Search (First Improvement)

GRASP (Greedy Randomized Adaptive Search Procedure) combina construção gulosa e aleatoriedade:

1. **Construção:** monta soluções adicionando pacotes segundo uma lista de candidatos restrita (RCL), priorizando a melhor razão benefício/peso, mas com variação aleatória.

2. **Busca local:** refina a solução usando o método **First Improvement**, que troca/remova pacotes até não encontrar mais melhora.

Parâmetros calibrados:

Parâmetro	Valor	Justificativa
`iters`	140 a 180	Quantidade de reconstruções completas; reduzido para limitar o tempo.
`rcl_size`	10–14	Maior diversidade na construção inicial.
`local_search`	First Improvement	Metade do custo do Best, mesma qualidade.

Método 2: Simulated Annealing (SA)

O **SA** é inspirado no recozimento térmico. Ele parte de uma solução inicial (vinda do GRASP) e faz pequenas alterações (flips e swaps) aceitando **pioras temporárias** com probabilidade:

$$P = e^{\Delta/T}$$

onde `Δ` é a diferença de benefício e `T` é a temperatura atual.
O algoritmo **esfria gradualmente**, reduzindo a chance de aceitar pioras e buscando estabilidade em torno de ótimos globais.

Parâmetros calibrados (preset SA_FAST):

Parâmetro	Valor	Significado
`T0`	90	Temperatura inicial (nível de exploração).
`α`	0.93	Fator de resfriamento (velocidade da queda de T).
`SAmax`	140	Número de vizinhos testados por temperatura.
`Tfinal`	1e-3	Temperatura mínima (critério de parada).

`max_neighbor_trials`	8–10	Tentativas para gerar vizinho viável antes de pular.

A vizinhança usa:

- **flip (60%)**: adiciona/remove um pacote;
- **swap (40%)**: troca um pacote da solução por outro fora dela.

3) Técnicas de Calibração

Durante o processo, foram aplicadas **técnicas de calibração empírica**:

- **Teste de convergência de valor vs. tempo:** reduzir iterações e ajustar α até o tempo ficar < 1min30s.
- **Exploração adaptativa:** ajustar a probabilidade flip/swap conforme a ocupação de espaço (mais swap quando próximo do limite).
- **Controle de temperatura:** α entre 0.9–0.94 mostrou melhor equilíbrio entre qualidade e tempo.
- **Sementes fixas** para reprodutibilidade.

O objetivo era obter soluções próximas ao ótimo sem ultrapassar o limite de tempo imposto (1m30s).

4) Métricas Utilizadas

As métricas obrigatórias salvas para cada execução:

1. **Benefício total (`best_value`)**
2. **Espaço consumido (`total_weight`)**
3. **Tempo de execução (s)**
4. **Solução em formato binário**
5. **Parâmetros da meta-heurística**
6. **Semente aleatória usada**

A análise final utilizou as médias das 3 execuções por instância (extraídas de `metricas-finais.txt`).

5) Resultados e Comparação

Instância	Método	Média Benefício	Média Peso	Média Tempo (s)
prob-software8.txt	GRASP+LS	12813.3	11991.7	0.5283
	SA_FAST	12813.0	11982.3	0.4219
prob-software9.txt	GRASP+LS	11333.3	38872.0	2.7756
	SA_FAST	**11435.3**	38865.0	**2.1973**
prob-software10.txt	GRASP+LS	**9453.7**	71592.7	7.9122
	SA_FAST	9337.3	**71600.3**	**7.0653**

7) Análise dos Resultados

- Em **instâncias pequenas (prob-software8)**, ambos os métodos obtêm praticamente o mesmo benefício, com SA sendo 20% mais rápido.
- Em **instâncias médias (prob-software9)**, o SA superou o GRASP em +0.9% no benefício e ainda reduziu o tempo em 25%.
- Em **instâncias grandes (prob-software10)**, o GRASP manteve ligeira vantagem em benefício, mas o SA executou 10% mais rápido.

Conclusão parcial:

- O **GRASP+LS** é mais estável e garante boas soluções rapidamente.
- O **Simulated Annealing** é melhor em instâncias médias, pois continua explorando depois da construção inicial.
- Ambos ficaram **bem abaixo do limite de 1m30s** imposto para a maratona.

8) Conclusões Gerais

A comparação entre **GRASP+Local Search** e **Simulated Annealing** evidencia que, embora ambos os métodos sejam capazes de produzir soluções de alta qualidade dentro do tempo limite imposto, eles possuem comportamentos distintos ao explorar o espaço de busca. O **GRASP+LS** mostrou-se **mais estável e previsível**, produzindo resultados semelhantes entre execuções e convergindo rapidamente para boas soluções. Isso ocorre porque sua construção gulosa orienta o algoritmo diretamente para regiões promissoras, enquanto o mecanismo de First Improvement refina rapidamente a solução sem grande custo computacional.

Em contrapartida, o **Simulated Annealing** demonstrou um caráter mais **exploratório**, sendo capaz de aceitar temporariamente soluções piores no início da busca. Essa estratégia permitiu ao SA escapar de ótimos locais produzidos pelo GRASP, especialmente em instâncias de tamanho intermediário, onde o espaço de soluções é diversificado o suficiente para que a exploração probabilística gere ganhos reais. Contudo, essa maior liberdade de exploração vem acompanhada de maior sensibilidade aos parâmetros de temperatura e de um comportamento menos determinístico, fazendo com que o SA apresente **variação maior entre execuções**, ainda que mantenha tempos competitivos.

Em instâncias maiores, o GRASP+LS recuperou vantagem, possivelmente por já capturar de maneira eficiente as combinações de pacotes com melhor razão benefício/peso, enquanto o SA não se beneficia tanto da exploração adicional. Assim, a análise conjunta sugere que o GRASP+LS é mais adequado quando se busca **robustez, velocidade e reprodutibilidade**, enquanto o SA tende a superar o GRASP em cenários onde **exploração adicional tem maior impacto**, ressaltando a complementaridade natural entre os dois métodos em um pipeline híbrido.