Dicas para Trabalho Final: Aplicação e Comparação de Metaheurísticas

Disciplina: Metaheurísticas e Aplicações
Professor: Franklin Angelo Krukoski

3 de maio de 2025

1 Dicas Iniciais

O objetivo deste trabalho é aplicar os conceitos das metaheurísticas **GA, PSO, TS e ACO** em problemas mais complexos que os vistos em aula. A estratégia recomendada é **adaptar e estender** as implementações base (TSP e QAP) fornecidas, em vez de começar do zero. O "esqueleto"dos algoritmos pode ser reutilizado, mas as principais modificações ocorrerão nos seguintes pontos:

1. **Representação da Solução:** Como codificar uma solução válida para o novo problema (TSPWT, Escala ou CARP)? Esta é uma decisão crucial que impactará todo o resto.
2. **Função de Custo/Fitness:** Como avaliar a qualidade (objetivo) de uma solução, incorporando as novas restrições específicas do problema? Esta função precisará ser cuidadosamente reescrita.
3. **Operadores/Vizinhança:** Como gerar novas soluções (crossover, mutação, movimentos de vizinhança) que sejam válidas ou façam sentido para a nova representação e restrições?
4. **Tratamento de Restrições:** Como garantir que as soluções sejam viáveis ou como penalizar violações? Pense em:
   - **Penalização:** Adicionar termos ao custo/fitness. Requer ajuste de pesos.
   - **Reparo: Corrigir soluções inviáveis. Pode ser complexo.**
   - **Geração Restrita: Operadores que só geram soluções viáveis. Pode limitar a busca.**
   - **Rejeição: Descartar inviáveis. Geralmente ineficiente.**
5. **Adaptações Específicas: Como aplicar PSO a problemas discretos (e.g., Random Keys)? Qual a heurística η mais adequada para o ACO no novo problema?**

Dicas Gerais de Implementação

- **Comece Simples:** Implemente a metaheurística focando no objetivo principal. Adicione as restrições uma a uma.
- **Modularize seu Código:** Use funções separadas para cálculo de custo, operadores, verificação de restrições, etc. Facilita a depuração e adaptação.
- **Representação é Fundamental: Justifique a escolha da sua representação da solução no relatório.**
- **Ajuste de Parâmetros: Os parâmetros que funcionaram para TSP/QAP provavelmente precisarão de reajuste para os novos problemas. Experimente!**
- **Teste com Instâncias Pequenas: Valide sua lógica com exemplos onde você possa verificar o resultado manualmente ou com mais facilidade.**

2 Adaptações Específicas por Problema

2.1 Opção 1: Caixeiro Viajante com Janelas de Tempo (TSPWT)

Base Sugerida: Código do TSP.

- **Representação: Manter Permutação.** A solução ainda é uma sequência de cidades, como no TSP (`VectorInt`).
- **Função de Custo/Fitness (*Principal Alteração*):**
  - **Reescrever Completamente!** A função (`calculate_tspwt_cost`) deve simular a rota passo a passo.
  - Para cada cidade i: calcular tempo de chegada $chegada_i$ (considerando viagem $t_{i-1,i}$ + serviço s_{i-1} + espera $espera_{i-1}$).
  - **Verificar Viabilidade:** Se $chegada_i > l_i$ (latest time), a rota é **inviável**. Retornar `Inf` ou alta penalidade.
  - Calcular tempo de espera: $espera_i = \max(0, e_i - chegada_i)$.
  - Atualizar tempo atual para saída: $tempo = chegada_i + espera_i + s_i$.
  - O **valor retornado** deve ser a distância total (se viável) + (opcionalmente) penalidades por espera ou duração total.
- **Tratamento de Restrições:** Integrado na **função de custo**. Uma rota pode ser estruturalmente válida (permutação), mas temporalmente inviável.
- **Adaptação das Metaheurísticas:**

  GA: Representação e operadores de permutação (`pmx_crossover_corrected`, `mutate!` com swap/inversão) **mantêm-se**. Apenas a chamada à função de fitness muda.

  PSO (RK): Representação (Random Keys) e mapeamento (`keys_to_permutation`) **mantêm-se**. A função de custo usada para avaliar a permutação decodificada é a nova `calculate_tspwt_cost`.

TS: Representação (permutação) e vizinhança (e.g., 2-opt, swap) **mantêm-se**. A avaliação do `neighbor_cost` usa a nova função com verificação temporal. A lista tabu pode continuar com movimentos (preferencial) ou rotas. **Aspiração** pode ser importante para aceitar movimentos que levem a soluções viáveis.

ACO: Representação (construção de permutação) **mantém-se**. **Grande mudança na Regra de Transição:** Ao escolher o próximo nó j a partir de i, a formiga só pode considerar nós j que sejam **temporalmente alcançáveis** a partir de i (i.e., $chegada_j \leq l_j$). A probabilidade p_{ij} só é calculada para esses vizinhos viáveis. A heurística η_{ij} pode ser adaptada ($1/D_{ij}$ ainda é útil, mas pode incluir termos relacionados à janela de tempo). Deposição de feromônio apenas por rotas **viáveis**.

2.2 Opção 2: Escala de Funcionários (Max Satisfação)

Base Sugerida: Código do QAP (conceitualmente, pela alocação), mas requer **adaptações significativas**.

- **Representação (*Decisão Crítica*):**
  - **Matriz [Funcionário x Slot]:** `MatrixInt` onde `M[f, s]` indica o turno do funcionário `f` no slot `s` (ou 0/1 se for apenas trabalha/não trabalha). Realista, mas operadores complexos.
  - **Vetor Longo [Slots]:** `VectorInt` onde `V[k]` indica o funcionário alocado ao k-ésimo slot (Dia1Turno1, ...). Mais simples para alguns operadores, mas difícil garantir restrições por funcionário.
  - **Lista de Tarefas/Turnos:** Cada elemento representa um turno a ser alocado, e o valor é o funcionário.
  - Justifique sua escolha! Comece com uma representação gerenciável.
- **Função de Custo/Fitness (*Reescrita Total*):**
  - Objetivo: Maximizar $\sum$(Satisfação por alocação) ou Minimizar $\sum$(Penalidades por violação)
  - Calcular satisfação baseada nas preferências dadas.
  - **Calcular Penalidades:** Para CADA restrição (cobertura de turno, horas máx/min, descanso, sequência proibida, preferências violadas), calcular o grau de violação e aplicar uma penalidade ponderada.
  - A soma ponderada das penalidades é subtraída da satisfação (ou adicionada ao custo). **Ajustar os pesos das penalidades é crucial!**
- **Tratamento de Restrições:** Principalmente via **penalidades** no fitness. Operadores de reparo podem ser complexos mas úteis (e.g., se falta cobertura, tentar mover alguém).
- **Adaptação das Metaheurísticas:**

  GA: Representação dita os operadores. Se matriz, crossover pode ser por blocos, mutação pode trocar turnos ou funcionários em slots. Se vetor longo, ope-

radores mais simples podem ser usados, mas geram soluções mais inviáveis. Fitness usa a função de satisfação/penalidade.

PSO: Requer **PSO Discreto** (se representação for discreta) ou **Random Keys** (mapeando chaves para a escala completa). Fitness usa a função de satisfação/penalidade. A adaptação é desafiadora.

TS: Representação é a escala atual. **Definir Vizinhança** é chave: e.g., 'move(func, slot, new_turn)', 'swap_shifts(func1, func2, slot)', 'change_day_off(func)'. Lista Tabu armazena atributos desses movimentos. Fitness usa a função de satisfação/penalidade.

ACO: Complexo de aplicar diretamente. Formigas poderiam "preencher"a escala slot por slot ou funcionário por funcionário. Feromônio τ_{fs} indicaria desejabilidade de alocar funcionário f ao slot s. Heurística η_{fs} poderia incluir preferência, cobertura necessária, etc. Construção deve tentar respeitar restrições rígidas.

2.3 Opção 3: Roteamento em Arcos Capacitado (CARP)

Base Sugerida: Código do TSP, **após transformação do problema**.

- **Transformação CARP → VRP/TSP (*Etapa Crucial*):**
  - Crie um **novo grafo** onde os nós são o **depósito** e cada **arco requerido** (u, v) do grafo original.
  - Calcule as "distâncias"d'_{AB} neste novo grafo: d'_{AB} é o custo do caminho mais curto no grafo *original* para ir do nó final de A até o nó inicial de B, mais o custo de servir o arco B. (Requer algoritmo de caminho mais curto, e.g., Floyd-Warshall ou múltiplos Dijkstras).
  - Cada "nó-arco"herda a demanda do arco original.
  - O problema agora se assemelha a um VRP (visitar "nós-arcos"com veículos capacitados partindo do depósito) ou um TSP gigante (se ignorar capacidade inicialmente).
- **Representação (no Grafo Transformado): Permutação** dos "nós-arcos", potencialmente com marcadores de depósito para separar rotas (VRP) ou uma única permutação (TSP gigante).
- **Função de Custo/Fitness (*Modificação Importante*):**
  - Calcular o custo total usando as "distâncias"d' do grafo transformado.
  - **Incorporar Capacidade:** Essencial! Simular a(s) rota(s) na permutação, acumular a demanda dos arcos. Se a capacidade do veículo for excedida, aplicar **penalidade altíssima** ou retornar `Inf`.
- **Tratamento de Restrições (Capacidade):** Via **penalidade** no fitness é comum. Alternativa: Algoritmos de **Splitting** (pós-processamento) para dividir uma rota gigante inviável em rotas viáveis. Para ACO, a verificação pode ser feita **durante a construção** da rota pela formiga.

- **Adaptação das Metaheurísticas (no Grafo Transformado):**

  GA: Usa permutação dos "nós-arcos". Operadores de permutação (PMX, OX, Inversão, Swap). Fitness penaliza violação de capacidade.

  PSO (RK): Random Keys mapeiam para permutação de "nós-arcos". Fitness penaliza violação de capacidade.

  TS: Vizinhança opera na permutação dos "nós-arcos"(Swap, 2-opt, etc.). Fitness penaliza violação de capacidade. Lista tabu nos movimentos da permutação transformada.

  ACO: Formigas constroem rotas no grafo transformado. Feromônio nas "arestas"d'_{AB}. Heurística $\eta'_{AB} = 1/d'_{AB}$. **Verificação de capacidade durante a construção é recomendada:** se adicionar o próximo "nó-arco"excede a capacidade, a formiga não pode escolhê-lo ou deve retornar ao depósito (se VRP). Deposição apenas por rotas viáveis.

Lembrete Final: A chave é entender o problema, escolher uma representação adequada, adaptar a função de avaliação para incluir restrições e, então, aplicar a lógica central da metaheurística (população/trajetória, operadores/vizinhança, memória). Boa sorte!