Heurísticas para um problema NP-Difícils

Algoritmos e Estrutura de Dados III

Gabriel Pereira Soares
Gustavo Fernandez Pascoaleto
Jorran Luka Andrade dos Santos
Letícia Freitas de Oliveira
Lucas Pessoa Oliveira Alves
Pedro Henrique de Almeida

Universidade Federal de Alfenas, Abril de 2024

Conteúdo

Tópicos

Introdução

- O problema de Soma de Subconjuntos é definido como a busca por determinar se existe um subconjunto dentro de um conjunto dado de números inteiros, cuja soma seja igual a um valor alvo específico.
- Este problema é conhecido por sua complexidade computacional.

Introdução

- Para enfrentar o desafio do problema de Soma de Subconjuntos, foram implementadas duas heurísticas distintas:
  - Algoritmo de busca local.
  - Algoritmo evolutivo.
- A comparação entre as heurísticas será realizada com base em dois critérios principais:
  - Tempo de execução.
  - Qualidade das soluções encontradas.

Tópicos

Algoritmos Utilizados

- Algoritmo Evolutivo.
  - O algoritmo evolutivo é uma heurística, onde uma população de soluções evolui ao longo de várias gerações para encontrar uma solução otimizada para um problema.
- Algoritmo de Busca local.
  - O algoritmo de busca local é uma heurística que busca encontrar uma solução melhor a partir de uma solução inicial, explorando as soluções vizinhas.

Algoritmo Evolutivo

- O algoritmo Evolutivo utiliza operações de seleção, cruzamento e mutação para melhorar a qualidade das soluções ao longo do tempo.
- Exemplo de passos do Algoritmo:
  - **Inicialização:** Gera uma população inicial de soluções aleatórias.
  - **Avaliação:** Calcula a aptidão (fitness) de cada indivíduo na população.
  - **Seleção:** Seleciona pares de indivíduos (pais).
  - **Cruzamento:** Gera novos indivíduos (filhos) combinando partes dos pais selecionados.

Algoritmo Evolutivo

- **Mutação:**
  - Aplica mutações aleatórias aos filhos para introduzir variação.
- **Substituição:**
  - Forma uma nova população a partir dos filhos gerados.
- **Repetição:**
  - Repete os passos de avaliação, seleção, cruzamento e mutação por um número definido de gerações.
- **Terminação:**
  - Retorna o melhor indivíduo encontrado ao final das gerações.

Algoritmo Evolutivo

```cpp
vector<int> genetic_algorithm(int target_sum, const vector<int>& elements, double& best_fitness) {
    srand(time(0));
    vector<vector<int>> population = initialize_population(POPULATION_SIZE, elements.size());
    vector<int> best_solution;
    best_fitness = INFINITY;

    for (int generation = 0; generation < MAX_GENERATIONS; ++generation) {
        vector<double> fitnesses(population.size());
        for (size_t i = 0; i < population.size(); ++i) {
            fitnesses[i] = 1.0 / (1 + fitness(population[i], elements, target_sum));
        }

        vector<vector<int>> new_population;
        for (int i = 0; i < POPULATION_SIZE / 2; ++i) {
            pair<vector<int>, vector<int>> parents = selection(population, fitnesses);
            pair<vector<int>, vector<int>> children = crossover(parents.first, parents.second);
            mutate(children.first);
            mutate(children.second);
            new_population.push_back(children.first);
            new_population.push_back(children.second);
        }
        population = new_population;

        //verifica se encontros a solucao
        for (const auto& individual : population) {
            double current_fitness = fitness(individual, elements, target_sum);
            if (current_fitness < best_fitness) {
                best_fitness = current_fitness;
                best_solution = individual;
            }
            int total = 0;
            for (size_t i = 0; i < individual.size(); ++i) {
                if (individual[i] == 1) {
                    total += elements[i];
                }
            }
            if (total == target_sum) {
                best_fitness = 0;
                return individual;
            }
        }
    }
    return best_solution;
}
```

Figura: Algortimo Evolutivo

Algoritmo de Busca local

- O algoritmo de Busca local realiza pequenas mudanças na solução atual para explorar novas soluções e melhorar a qualidade da solução encontrada.
- Exemplo de passos do Algoritmo:
  - **Inicialização:** Definir uma solução inicial aleatória.
  - **Avaliação:** Calcular a aptidão (fitness) da solução inicial.
  - **Busca Local:** Repetir por um número definido de iterações:
    - Gerar uma nova solução vizinha invertendo um bit aleatório da solução atual.
    - Calcular a aptidão da nova solução.
    - Se a nova solução for melhor, atualizá-la como a melhor solução encontrada.
  - **Terminação:** Retornar a melhor solução encontrada após o número definido de iterações.

Algoritmo de Busca local

```cpp
// Função de busca local
void localSearch(vector<int> &bestSubset, const vector<int> &elements, int target, int maxIterations)
{
    int bestFitness = calculateFitness(bestSubset, elements, target); // calcula a fitness da solucao inicial
    srand(time(0));                                                   // semente para geracao de números aleatorios

    for (int iteration = 0; iteration < maxIterations; ++iteration)
    {
        vector<int> newSubset = bestSubset;                 // cria uma nova solucao vizinha a partir da melhor solucao atual
        int indexToFlip = rand() % elements.size();         // escolhe um indice aleatorio para inverter
        newSubset[indexToFlip] = 1 - newSubset[indexToFlip]; // inverte o bit (0 -> 1 ou 1 -> 0)

        int newFitness = calculateFitness(newSubset, elements, target); // Calcula a fitness da nova solucao
        if (newFitness < bestFitness)
        { // se a nova solucao for melhor atualiza a melhor solucao encontrada
            bestSubset = newSubset;
            bestFitness = newFitness;
        }

        // parar se encontrarmos uma solucao perfeita (fitness == 0)
        if (bestFitness == 0)
        {
            break;
        }
    }

    // Imprime o resultado da busca local
    if (bestFitness == 0)
    {
        cout << "Encontrou um subconjunto que soma exatamente ao valor alvo." << endl;
    }
    else
    {
        cout << "Nao foi possivel encontrar um subconjunto que soma exatamente ao valor alvo." << endl;
        cout << "A solucao mais proxima encontrada tem uma diferenca de: " << bestFitness << endl;
    }

    cout << "Melhor subconjunto encontrado: ";
    for (size_t i = 0; i < bestSubset.size(); ++i)
    {
        if (bestSubset[i] == 1)
        { // Se o bit é 1, o elemento correspondente faz parte do subconjunto
            cout << elements[i] << " ";
        }
    }
    cout << endl;
}
```

Figura: Algoritmo de Busca local

Descrição das Instâncias

- Para avaliar o desempenho das heurísticas implementadas, foram propostas diferentes instâncias de teste.
- As instâncias foram escolhidas para representar uma variedade de cenários, com conjuntos de números de diferentes tamanhos e distribuições

Descrição das Instâncias

- Exemplos de algumas Instâncias:
- **Conjunto 1 original:**
  - 1, 2, 3, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 35, 40, 45, 50, 55

  Conjunto 2:
  - **Tamanho:** 10 elementos.
  - **Características:** Valores ímpares em sequência.
  - **Propósito:** Testar a eficiência do algoritmo em um conjunto pequeno e relativamente uniforme.

  Conjunto 3:
  - **Tamanho:** 10 elementos.
  - **Características:** Valores com variação.
  - **Propósito:** Testar a eficiência do algoritmo em um conjunto pequeno com variação de valores.

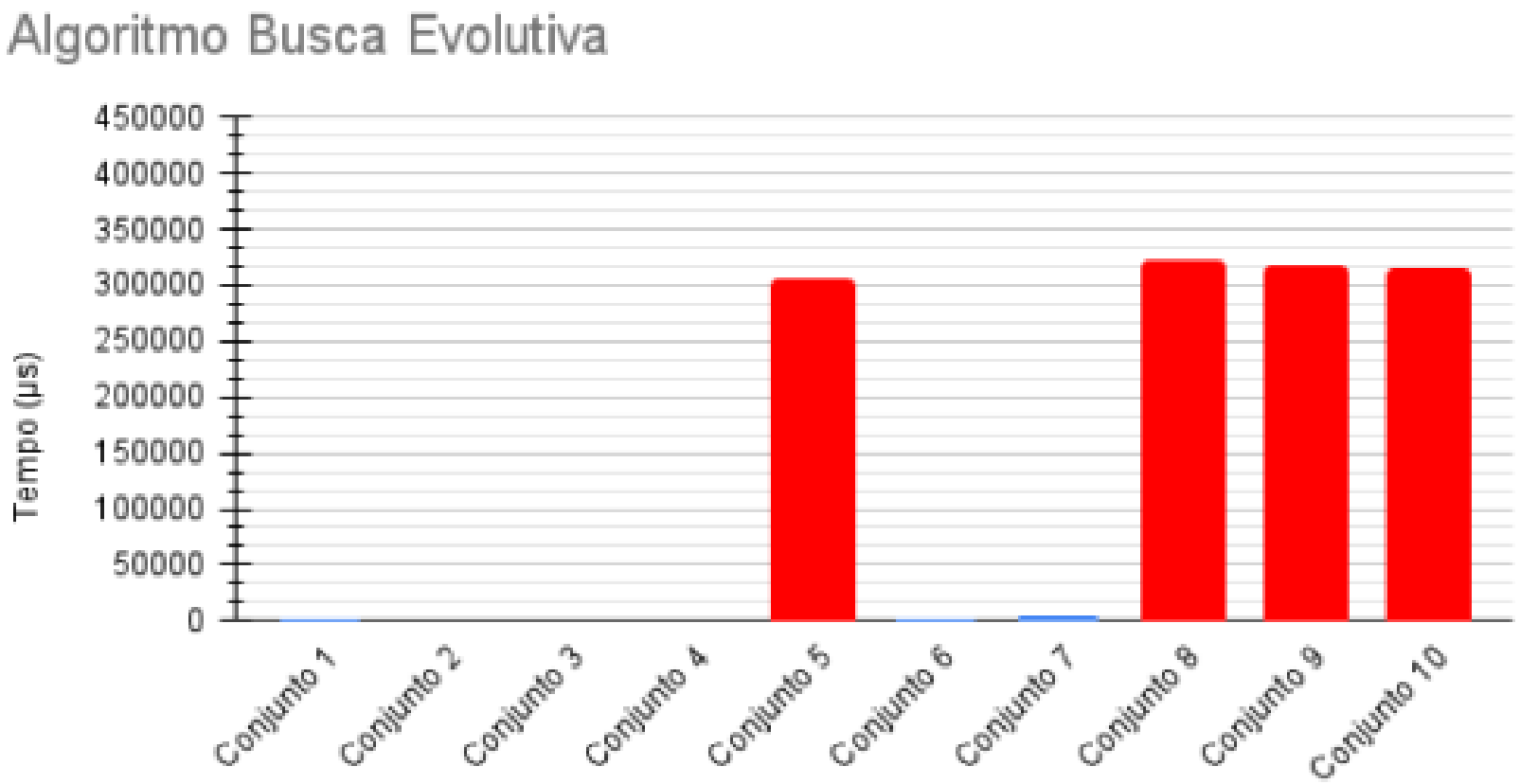
Tópicos

Busca Evolutiva

Algoritmo Busca Evolutiva	
Conjunto	Tempo (µs)
Conjunto 1	807
Conjunto 2	643
Conjunto 3	334
Conjunto 4	712
Conjunto 5	302901
Conjunto 6	1898
Conjunto 7	3416
Conjunto 8	319420
Conjunto 9	316566
Conjunto 10	313360

Figura: Tabela Busca Evolutiva

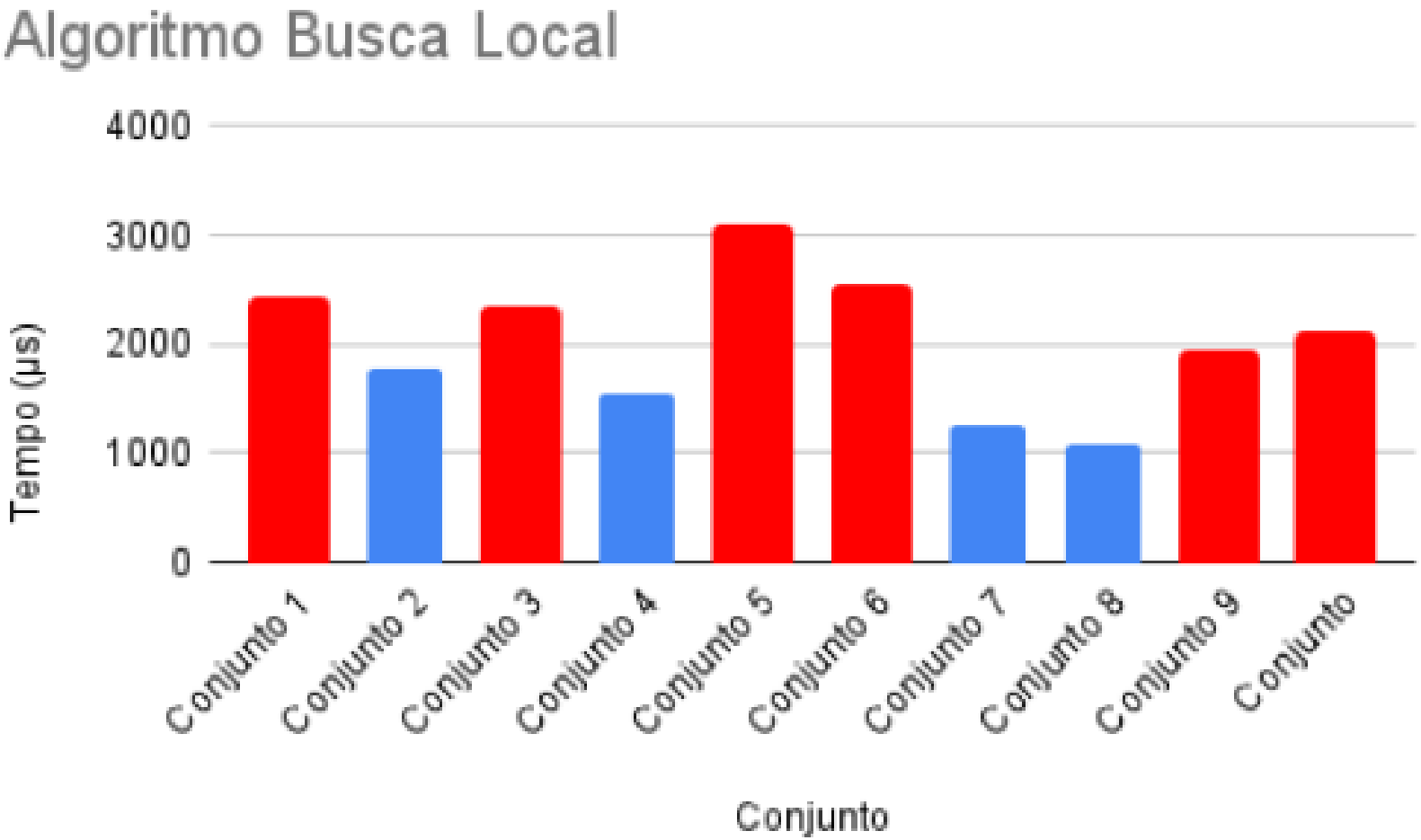
Busca Evolutiva



Busca local

Algoritmo Busca Local	
Conjunto	Tempo (µs)
Conjunto 1	2420
Conjunto 2	1772
Conjunto 3	2329
Conjunto 4	1545
Conjunto 5	3060
Conjunto 6	2518
Conjunto 7	1255
Conjunto 8	1095
Conjunto 9	1916
Conjunto 10	2096

Busca local


Algoritmo Busca Local
4000
3000
2000
1000
0
Tempo (µs)
Conjunto 1
Conjunto 2
Conjunto 3
Conjunto 4
Conjunto 5
Conjunto 6
Conjunto 7
Conjunto 8
Conjunto 9
Conjunto
Conjunto