Transações de SGBDs

Ramon Duarte de Melo

Universidade Federal do Rio de Janeiro

ramonduarte@poli.ufrj.br

29 de novembro de 2018

Sumário

Glossário

Recurso

Abstração-chave de um pedaço coeso de informação. Dentro do contexto da disciplina de *Bancos de Dados*, consideraremos como um subconjunto qualquer de dados que possa ser nomeado, agrupado, processado ou referenciado: um registro, ou uma coluna, ou mesmo toda a base de dados.

Cadeado (ou *lock*)

Estrutura de dados implementada atomicamente pelo sistema operacional que oferece controle de acesso concorrente a um determinado recurso através da exclusão mútua.

Problema do Caixeiro Viajante (TSP)

O problema de roteamento de veículos (VRP) é uma generalização do problema do caixeiro viajante, onde o número k de veículos representa o número máximo de circuitos admitido como resposta.

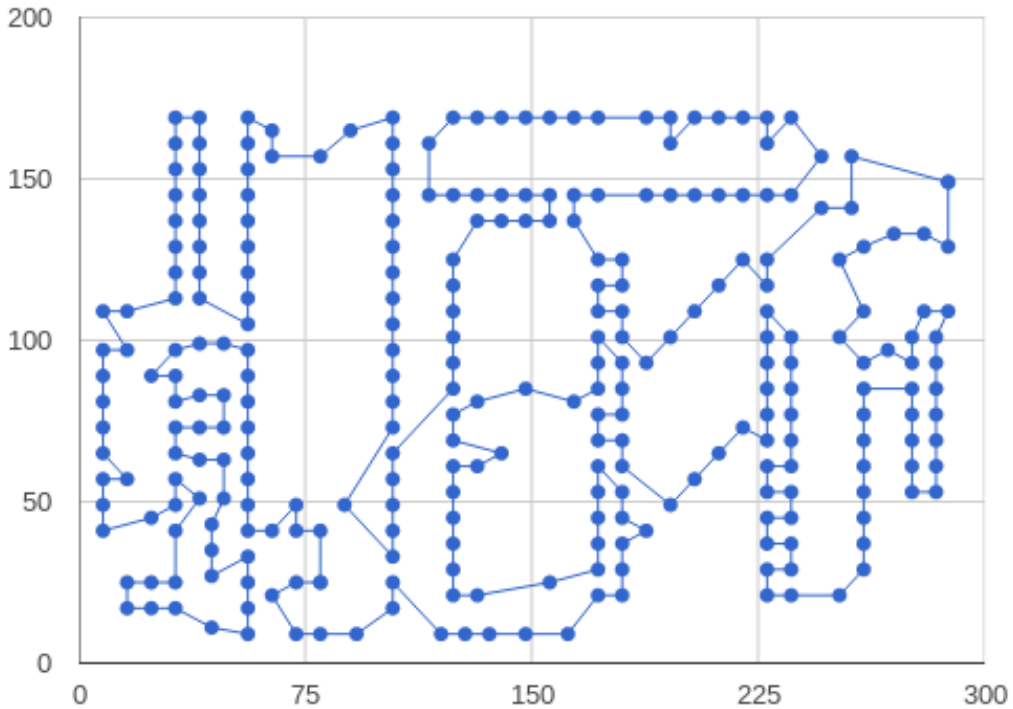

Figura: TSP com $n = 280$ e $k = 1$.

Problema de Roteamento de Veículos (VRP)

No roteamento de veículos, há um vértice especial - o *armazém* - que deve ser a origem e o destino de todas as rotas, e que não será incluído na cardinalidade n. Os demais vértices serão chamados de *clientes*.

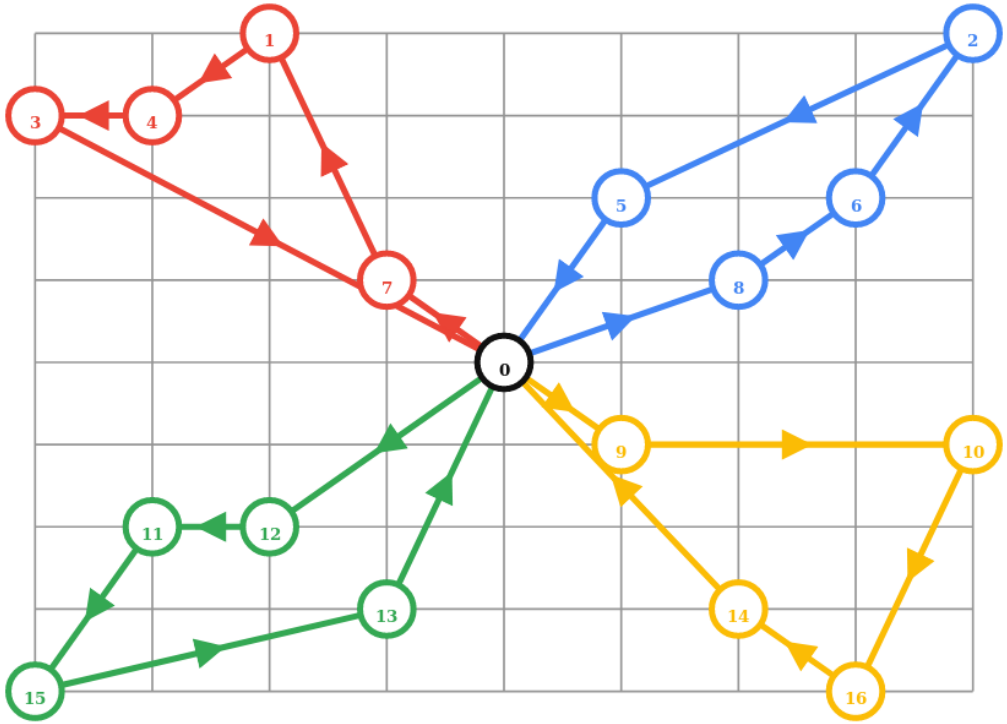

Figura: VRP com $n = 16$ e $k = 4$.

Problema de Roteamento de Veículos Capacitado (CVRP)

Além da limitação de distância e de veículos disponíveis, cada cliente possui uma demanda $q_i \mid i \in \{1, \ldots, n\}$ que deve ser atendida por um dos veículos de capacidade $c_i \mid i \in \{1, \ldots, k\}$.

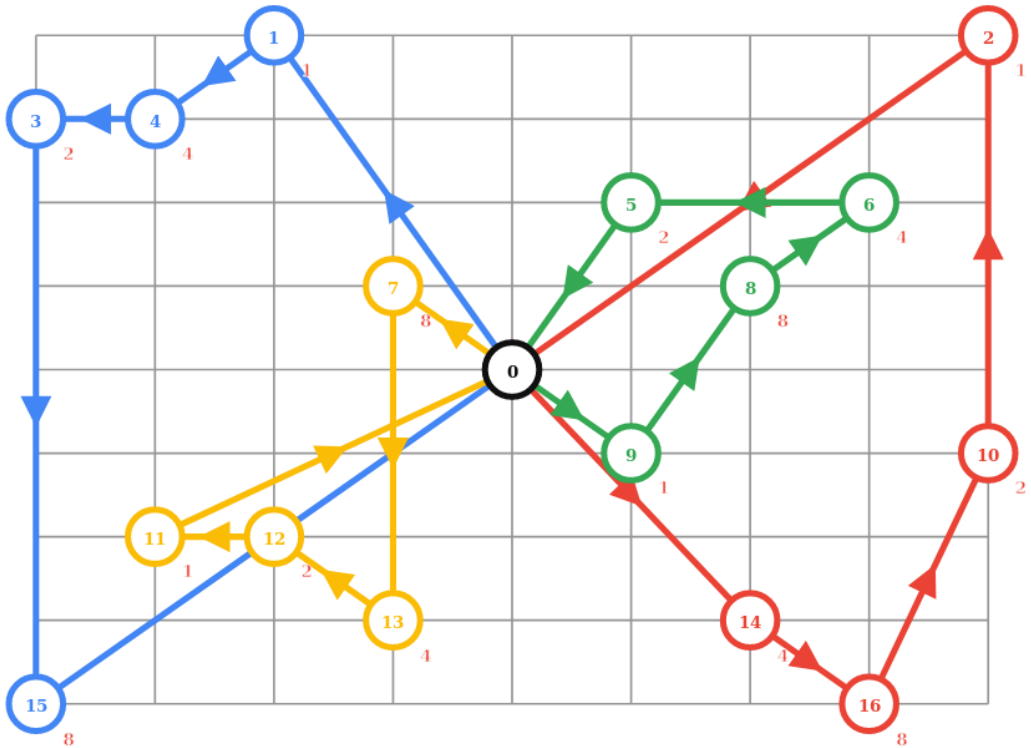

Figura: CVRP com $n = 16$, $k = 4$ e $\sum_{i=1}^{n} q_i = 60$.

Problema de Roteamento de Veículos Capacitado com Janelas de Tempo (CVRPTW)

Nesta variante, clientes só podem ser atendidos num intervalo de tempo predeterminado. Arestas passam a representar o tempo de deslocamento, em vez da distância, para evitar cálculos desnecessários de velocidade.

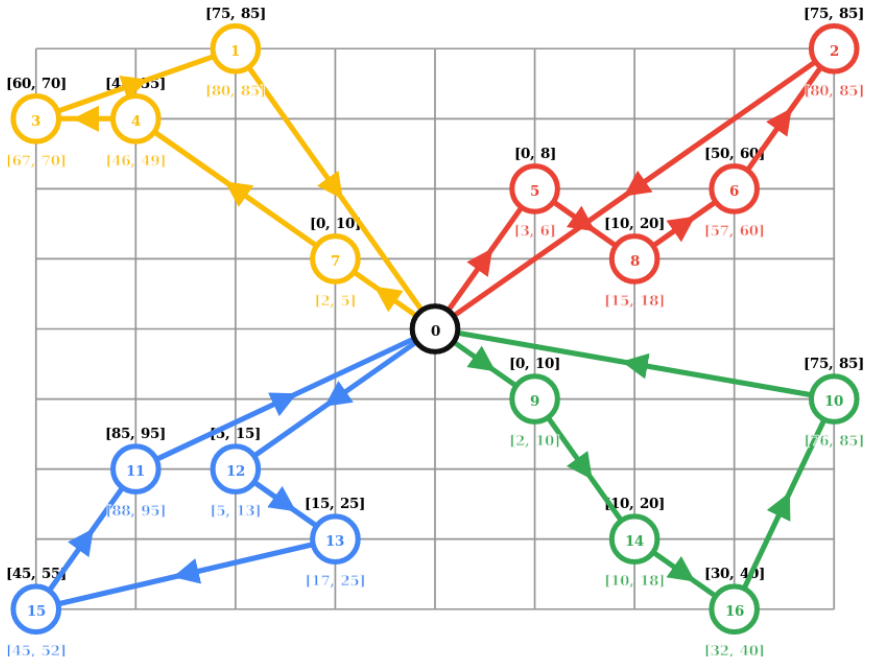

Figura: CVRPTW com $n = 16$, $k = 4$ e $\sum_{i=1}^{n} q_i = 60$.

Problema de Roteamento de Veículos Capacitado com Janelas de Tempo (CVRPTW)

Um veículo pode chegar ao cliente antes do intervalo, mas só poderá seguir adiante quando a janela for aberta.

O problema não fica significativamente mais difícil de resolver. A dificuldade encontrada foi conferir a validade da solução, que diminui muito o *throughput* do programa.

Grafo planar

Como as heurísticas testadas exigem que a *inequação do triângulo* seja satisfeita, e como existem muitas instâncias deste problema já resolvidas, todas as entradas são **grafos planares**.

O grafo planar é **obrigatoriamente completo e não-direcionado** - o que elimina a necessidade de manter a lista de adjacências - e a aresta $e(i, j)$ tem peso $(i_x - j_x)^2 + (i_y - j_y)^2$.

Grafos solucionáveis

Todas as instâncias testadas têm ao menos uma solução conhecida. O objetivo é garantir que existe uma solução plausível de ser encontrada no grafo em questão.

Quando a solução ótima já é conhecida, os valores obtidos para o total de veículos utilizados e para a distância percorrida foram usadas como limites inferiores (*lower bounds*).

Objetivos

A função objetivo que determina a solução ótima do CVRP foi escolhida como a minimização de rotas, isto é, a melhor solução é a que demanda menos veículos.

Todos os veículos são considerados de mesmo custo, mesmo quando possuem capacidades diferentes (*heterogêneos*). Para as heurísticas, essa dinâmica é importante porque os veículos mais pesados serão empregados primeiro.

Objetivos

Dadas duas soluções com mesmo número k de rotas empregadas, a que tiver menor distância total percorrida é considerada melhor.

As duas heurísticas testadas são dependentes da escolha inicial dos vértices. Por isso, foram executadas algumas vezes com valores iniciais distintos e o resultado mostrado é o melhor encontrado.

Algoritmo 1: Força bruta

A força bruta é baseada no *powerset* de rotas - o conjunto de todas as rotas possíveis - e no *powerset* de soluções - o conjunto de todas as soluções possíveis.

```
    def powerset(iterable, lb=1):
        "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2)
                               (1,3) (2,3) (1,2,3)"
        s = list(iterable)
        return itertools.chain.from_iterable(
            itertools.combinations(s, r)
            for r in range(lb, len(s)+1))
```

Algoritmo 2: Força bruta com *branch-and-bound*

Para reduzir cada *powerset*, foram aplicadas algumas checagens:

1. limites inferiores de demanda da rota:

todo veículo em rota precisa atender ao menos um cliente

```
lower_bound = min(total_demand % max(capacities),
                  max(demands[1:]))
```

se a capacidade total for igual à demanda total, então cada veículo deve receber uma rota que consuma toda a sua capacidade

```
lower_bound = capacities[i] if (sum(capacities) ==
                                sum(demands))
```

Algoritmo 2: Força bruta com *branch-and-bound*

2. limites superiores de demanda da rota:

rota precisa ter demanda compatível com a capacidade do veículo

```
upper_bound = max(capacities)
```

3. limites inferiores de veículos assinalados:

a capacidade total precisa dar conta da demanda total

```
while sum(capacities) > sum(demands):
    capacities.pop()
for solution in powerset(viable_routes[::-1],
                         lb=len(capacities)):
    (...)
```

Algoritmo 2: Força bruta com *branch-and-bound*

se existe uma solução ótima conhecida da instância, então o número de rotas dela é um limite inferior

4. limites superiores de veículos assinalados:

se existe uma solução heurística viável, então a solução ótima tem que ter um número de rotas obrigatoriamente menor ou igual

```
    upper_bound = heuristics_bounds()["upper"]
```

Algoritmo 2: Força bruta com *branch-and-bound*

O *powerset* gera um conjunto ordenado de rotas. Portanto, se uma rota r_i for incompatível com outra $r_{j>i}$, é abandonar o fio de execução em favor da rota r_{j+1}.

5. checagem de soluções:

se duas rotas possuem um vértice em comum, então as soluções que incluem ambas são inviáveis:

```
if (len(set(i for x in solution for i in x[0]))
    != len([i for x in solution for i in x[0]])):
    continue
```

Algoritmo 2: Força bruta com *branch-and-bound*

se uma rota faz a solução ter distância maior que a encontrada por uma heurística, então ela não leverá a uma solução ótima:

```
    if this_solution_distance > distance_upper_bound:
        continue
```

se a rota introduz uma solução que é pior que a melhor solução já encontrada pelo algoritmo, então ela não levará ao ótimo:

```
    total_weight = sum(
        [G.get_edge_data(x[0][i], x[0][i+1])["weight"]
         for x in solution for i in range(len(x[0])-1)])
    if total_weight > best_route[1]:
        continue
```

Objetivos

Dadas duas soluções com mesmo número k de rotas empregadas, a que tiver menor distância total percorrida é considerada melhor.

As duas heurísticas testadas são dependentes da escolha inicial dos vértices. Por isso, foram executadas algumas vezes com valores iniciais distintos e o resultado mostrado é o melhor encontrado.

Referências bibliográficas


Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database system concepts (Vol. 4). New York: McGraw-Hill.


Rawat, U. (2017). Implementation of Locking in DBMS. Acessado a em https://www.geeksforgeeks.org/implementation-of-locking-in-dbms/.


Porfirio, Alice & Pellegrini, Alessandro & Di Sanzo, Pierangelo & Quaglia, Francesco. (2013). Transparent Support for Partial Rollback in Software Transactional Memories. 8097. 583-594. 10.1007/978-3-642-40047-6_59.


Poddar, Saumendra. (2003). SQL Server Transactions and Error Handling. Acessado a em https://www.codeproject.com/Articles/4451/SQL-Server-Transactions-and-Error-Handling.

Referências bibliográficas


Singhal, Akshay. (2018). Cascading Schedule — Cascading Rollback — Cascadeless Schedule. Acessado a em https://www.gatevidyalay.com/cascading-schedule-cascading-rollback-cascadeless-schedule/.


Pandey, Anand. (2018). Transactions and Concurrency Control. Acessado a em https://gradeup.co/transactions-and-concurrency-control-i-4c5d9b27-c5a7-11e5-bcc4-bc86a005f7ba.


Difference between. (2018). Difference between Deadlock and Starvation. Acessado a em http://www.differencebetween.info/difference-between-deadlock-and-starvation.