Detectando o menor caminho de uma aplicação para o desenvolvimento de aplicações RPA

Yury Alencar Lima[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

yuryalencar19@gmail.com

1. Problema escolhido

O problema escolhido para este estudo é a seleção do menor caminho dentro de uma aplicação a partir de um grafo ponderado.

1.1. Justificativa

Em aplicações grandes e complexas é comum que seja possível realizar a mesma ação através de diversos caminhos diferentes. Desse modo, para aumentar a performance de um usuário com pouco tempo para determinada tarefa é necessário encontrar o caminho com o menor esforço possível para realizar determinada ação. No cenário de desenvolvimento voltado ao RPA (Robotic Process Automation) [Van der Aalst et al. 2018] é importante para o programador, saber o menor e mais eficiente caminho para implementar o robô a fim de melhorar a performance do usuário.

2. Algoritmo base

A fim de solucionar o problema escolhido e apresentado na Seção 1, será utilizado o algoritmo de Djikstra. Este algoritmo tem como entrada um grafo ponderado, desse modo a aplicação deverá ser estruturada utilizando um grafo [Skiena 2020]. Assim, o grafo baseado na aplicação seguirá o respectivo formato: 1) Cada página um nó. 2) Cada aresta uma ação, o peso da aresta será formado pela complexidade da ação. A partir dessa representação é possível chegar ao resultado esperado. Com o intuito de detectar o menor caminho possível, o algoritmo é composto por três passos simples: 1) Iniciam-se os valores para cada variável. 2) Utiliza-se um conjunto Q, cujos vértices ainda não contém o custo do menor caminho determinado. 3) Por fim, é realizado uma série de verificações das arestas, definindo o menor caminho [Skiena 2020].

2.1. Justificativa

O algortimo de Dijkstra tem como objetivo encontrar o subconjunto de arestas com o menor esforço necessário para conectar dois nós em um grafo ponderado. Tendo em vista que a aplicação a ser automatizada, pode ser representada com um grafo, o algoritmo de Dijkstra para resolver o problema escolhido.

3. Projeto de avaliação

A fim de realizar uma avaliação da solução implementada foram criados alguns parâmetros e valores. Abaixo estas informações foram apresentadas em forma de subseções.

3.1. Conjunto de parâmetros

:

- **Quantidade de estados**: A quantidade de estados representam a quantidade de páginas de uma aplicação desse modo esse parâmetro é importante a ser introduzido.
- **Sementes aleatórias**: Decorrente a necessidade do uso de uma gerador para testar com altas quantidades de estados, a semente utilizada na criação ser aleatória é um fator importante para não apresentar viés na análise realizada.
- **Densidade do grafo**: A fim de avaliar o tempo do algoritmo tanto em grafos esparsos quanto grafo densos.

3.2. Conjunto de valores para avaliação experimental

:

- **Quantidade de estados**: A quantidades de estados que foram utilizadas na avaliação foram de 100 em 100 até a quantidade de 2000 estados. Tendo em vista que a quantidade de estados representam a quantidade de páginas em uma aplicação o grafo de 2000 estados consegue simular uma aplicação de grande porte.
- **Sementes aleatórias**: As sementes utilizadas foram: 10, 20 e 30.
- **Densidade do grafo**: Foram executados em um grafo denso e outro esparço. O grafo denso possui todos os estados interconectados. O grafo esparso gerado possui 7% dos estados convertidos em arestas.

3.3. Métrica de desempenho

:

- **Tempo de execução do algoritmo de Dijkstra**: O tempo em milisegundos da execução do algoritmo foi adicionado a fim de verificar se a complexidade do algoritmo implementado é condizente com a proposta por Dijkstra.

4. Avaliação do algoritmo

A fim de avaliar o resultado e complexidade do algoritmo de Dijkstra foram criados dois geradores, um de grafos esparsos e outro de grafos densos. Assim, foi possível analisar o tempo de execução de cada algoritmo de acordo com a densidade do grafo. A seguir são apresentados os resultados e como foram criados o gráfico de execução final.

4.1. Criação do gráfico

Com o intuito de criar o gráfico foram realizadas 6 execuções do algoritmo de Dijkstra. Sendo estas execuções, três para grafos densos e três para grafos esparsos, com a semente 10, 20 e 30 respectivamente. Ao final das execuções foram calculadas as médias, desvios padrões e gerado o gráfico presente na Figura 1. Na Figura 1 os desvios padrões e resultados das medições foram representados com ponto e variação, enquanto o resultado analítico com a linha contínua.

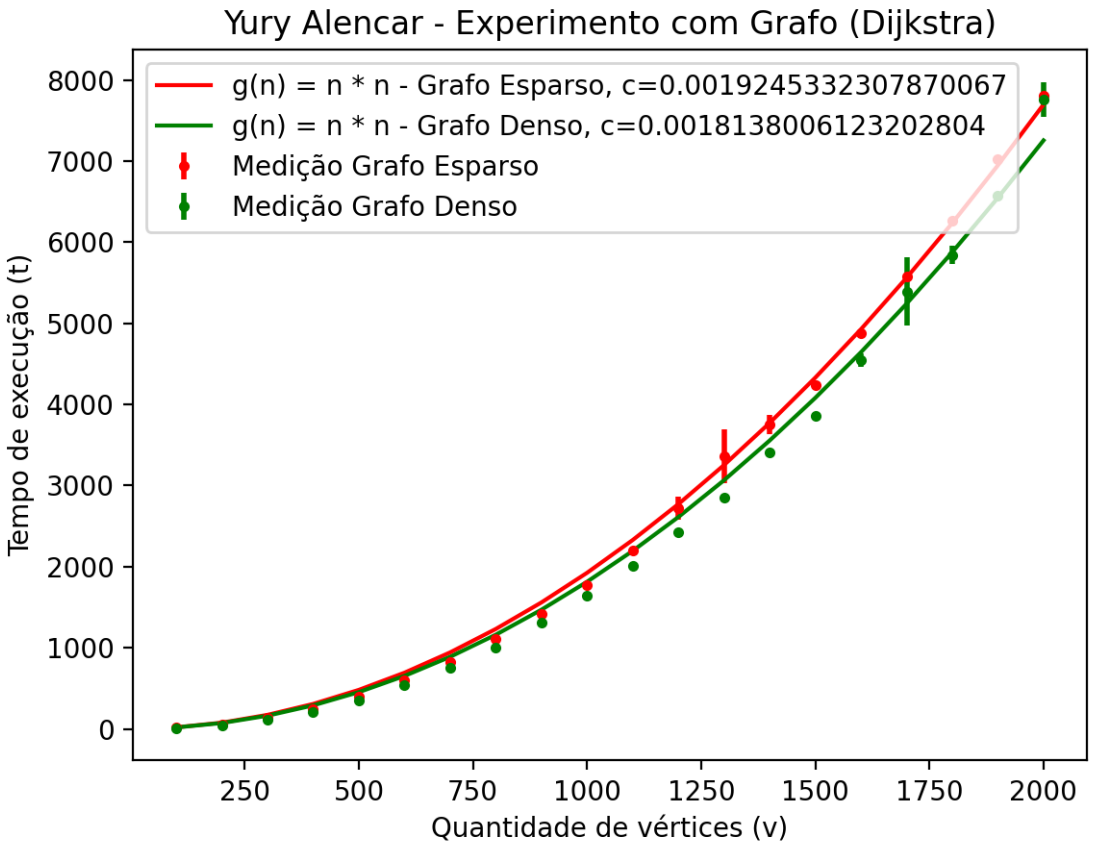

Figure 1. Gráfico de execução com grafos esparços e densos

4.2. Análise dos resultados

A complexidade do algoritmo de Dijkstra é ocasionada por dois fatores: o tempo da procura vértices não marcados com a menor distância d[v], e o tempo do aprimoramento desse valor, por exemplo o tempo de trocar o valor d[to]. Durante a implementação essas operações requerem O(n) e O(1). Desse modo, executamos a primeira operação **n** (quantidade de vértices) vezes, e a segunda **m** (quantidade de arestas) vezes, obtendo a complexidade $O(n^2 + m)$.

De acordo com a Figura 1 fica claro que essa complexidade é otimizada para um grafo denso, por exemplo quando $m \approx n^2$, o que ocorre no *workload* de grafo denso utilizado. Entretanto em grafos esparsos, quando m é bem menor que o número máximo de arestas n^2, a complexidade fica menos otimizada. Assim, é necessário aprimorar o tempo de execução da operação de descobrimento de vértices não marcados com a menor distância.

References

Skiena, S. S. (2020). *The algorithm design manual*. Springer International Publishing.

Van der Aalst, W. M., Bichler, M., and Heinzl, A. (2018). Robotic process automation.