Verslag BDA opdracht 1

Sil Vaes en Maarten Evenepoel

October 12, 2021

Voor dit project was het de bedoeling om een frequent itemset mining algoritme te implementeren op de DBLP dataset. Het doel was om de maximal frequent itemsets uit de DBLP databank te berekenen. De moeilijkheid ligt er bij dit project in om op een goede manier gebruik te maken van het beschikbare werkgeheugen van de computer om zo de gehele dataset, die van aard al relatief groot is, verwerkt te krijgen. In dit verslag zullen we de resultaten van ons algoritme bespreken alsook een blik werpen op de aanpak die we gevolgd hebben in de ontwikkeling van onze implementatie en de moeilijkheden waar we onderweg op botsten.

1 Aanpak en moeilijkheden

We hebben doorheen de loop van het project verschillende iteraties van onze implementatie moeten uitproberen. Initieel zijn we uitgegaan van een naïef textbook a-priori algoritme. Al snel bleek dat dit niet zou voldoen bij een te grote dataset. Deze eerste implementatie bestond eruit om simpelweg voor een gegeven k author sets te genereren van grootte k uit een lijst van alle voorkomende auteurs in de brondata. Vervolgens werd geteld of de gegenereerde author sets ook voldoende support hadden in de brondata. Deze implementatie moest het qua performantie al snel laten afweten wanneer de brondata boven de 10000 entries bevatte. Dit kwam doordat de stap waar de van candidate author sets berekend werden veel te kostelijk is, zeker voor het berekenen van paren (k=2). Achteraf gezien is dit ook logisch. De brondata bevat immers immens veel auteurs, maar een heel grote portie van die auteurs komen nooit samen voor als auteur van een publicatie. Wanneer we toch **alle** sets candidate sets genereren, dan bouwen we een zeer grote verzameling candidate sets die toch nooit kans maken om voor te komen in de brondata. We hebben met deze kennis een nieuwe implementatie gemaakt.

Onze tweede iteratie bestond eruit om voor het berekenen van candidate sets met grootte k=2 de paren meteen af te leiden uit de brondata. De nieuwe implemen-

tatie ging voor het construeren van candidate paren gaat in de brondata kijken wanneer een publicatie 2 of meer auteurs had. Indien dit het geval is, zal hij alle mogelijke subsets van auteurs van grootte 2 uit die publicaties genereren. Door enkel paren te generen uit publicaties die ook echt 2 of meer auteurs hebben worden er een hele hoop candidate sets niet gegenereerd die anders gegarandeerd nooit samen zouden voorkomen. Dit was een zeer grote tijds- en geheugenbesparing in het algoritme. Onze nieuwe implementatie was nu veel sneller, maar had nog steeds moeite met te grote brondata.

Een laatste optimalisatie die we hebben toegevoegd is het hashen van candidate sets naar buckets zoals beschreven in het PCY algoritme. Met deze laatste optimalisatie kregen we het algoritme gedraaid op een dataset ter grootte van 1.5 miljoen publicaties. Dit is ongeveer de helft van de volledige DBLP dataset. Het algoritme gebruikt hier op zijn piek 13GB geheugen. Aangezien we slechts met 8GB ramgeheugen zitten en 5GB swapgeheugen, wordt het proces voornamelijk sterk vertraagd door door het swappen van data tussen de swap-space en het RAM geheugen.

2 Resultaten

Om onze resultaten te reproduceren kan het programma op volgende manier worden aangeroepen:

```
$ python3 main.py {{dataset.xml}} {{chunk_size}}
```

Waar {{dataset}} en {{chunk_size}} het bestand dat de dataset bevat en de chunk size waarmee het bestand wordt ongelezen zijn respectievelijk. Ook kan men een optionele `--make_testfile` flag toevoegen om een kleinere testfile van de grotere {{dataset}} file te maken.

We hebben op bovenstaande manier aan kleinere dataset aangemaakt uit de dblp dataset van grootte 1.5 miljoen publicaties. met volgend commando:

```
$ python3 main.py --make_testfile dblp.xml 1000
```

Vervolgens kan het algoritme worden uitgevoerd op de gegenereerde kleinere dataset met behulp van volgend commando:

```
$ python3 main.py testfile.xml
```

In de output van ons algoritme krijgen we vervolgens te zien dat de maximal frequent itemsets grootte k=9 hebben. Als support hebben we s=15 gebruikt. Een voorbeeld van een dergelijke itemset is de volgende verzameling auteurs:

```
{
    'Robert M. Kuhn',
    'David Haussler',
    'Galt P. Barber',
    'Mark Diekhans',
    'W. James Kent',
    'Hiram Clawson',
    'Brian J. Raney',
    'Kate R. Rosenbloom',
    'Ann S. Zweig'
}
```

Concreet betekent dit dat bovenstaande set geen enkele superset heeft die vaker voorkomt dan deze set zelf. Bovendien komt deze set 15 keer voor in de brondata.

3 Conclusie

Het ontwikkelen van een algoritme dat voldoende grote datasets kon verwerken bleek een heel karwei te zijn. In het verleden van onze opleiding lag de moeilijkheid bij gelijkaardige projecten en opdrachten vaak in het snel genoeg krijgen van een implementatie door een tijds-efficiëntere implementatie te maken. In dit geval was een andere mindset nodig om niet alleen te focussen op de performantie in termen van runtime, maar ook in termen van geheugengebruik. We denken niet dat we het onderste uit de kan hebben kunnen halen op dit gebied, maar we hebben wel nieuwe inzichten opgedaan over hoe we kunnen omgaan met algoritmes die moeten werken op grote datasets. We denken overigens ook er zeker nog ruimte is voor optimalisaties waar we verder niet toe gekomen zijn wegens tijdsgebrek.