Resolução de Problemas Através de Métodos de Busca Informada

Pedro Vasconcelos Rodrigues

1 Objetivos

O objetivo do trabalho foi exercitar e fixar conhecimentos adquiridos sobre resolução de problemas através de métodos de busca informada, mais especificamente a busca greedy e a busca A*. A implementação dos algoritmos foi feita em Python.

2 Atividades

2.1 Descrição

2.1.1 Primeira Atividade

A primeira atividade foi encontrar o menor caminho entre as cidades Alice Springs e Yulara da Austrália, baseando-se em um grafo específico conectando as cidades com base num arquivo australia.csv, dado pelo professor, com os seguintes campos: ID da cidade, coordenada x, coordenada y, estado e população.

A distância em linha reta entre as cidades pode ser calculada a partir das coordenadas cartesianas (x,y), mas a distância pela estrada é 10% maior que a distância em linha reta.

Os IDs das cidades são correspondentes à sua posição na ordenação alfabética e uma cidade com ID x se conecta com as cidades x+2 e x-1, se x>1 e x é par, e com as cidades x-2 e x+1, se X é ímpar e x>2.

2.1.2 Segunda Atividade

Já a segunda atividade foi criar um agente capaz de resolver o problema dos blocos deslizantes com dimensões 9x9, com uma interface gráfica que podia ser bastante simples para permitir ao usuário perceber qual a situação inicial do problema e a evolução dos movimentos até a solução completa. A situação inicial dos blocos deslizantes é definida aleatoriamente, gerada a partir da configuração objetivo que sofre um determinado número de movimentos aleatórios.

2.2 Classificação dos sistemas

2.2.1 Primeira Atividade

O sistema da primeira atividade, caracterizado pelo grafo proposto, é um sistema parcialmente observável, visto que o agente só é capaz de saber com que cidades uma cidade se conecta quando ele está nela. Ele é estático, visto que o grafo não muda com o tempo. Ele é determinístico, visto que não há eventos probabilísticos no sistema. Ele é discreto, visto que há um número finito de nós no grafo e de caminhos entre eles. Por fim, ele contém apenas um único agente.

2.2.2 Segunda Atividade

Já o sistema da segunda atividade, caracterizado pelo jogo clássico dos blocos deslizantes, é um sistema observável, visto que o estado do sistema é definido unicamente pela configuração do tabuleiro. Ele é estático, visto que o tabuleiro não muda com o tempo e só se avalia a quantidade de movimentos. Ele é determinístico, visto que não há eventos probabilísticos no sistema. Ele é discreto, visto que há um número finito de blocos no tabuleiro e de movimentos possíveis. Por fim, ele contém apenas um único agente.

2.3 Implementação

Para ambas as atividades, implementou-se a estrutura de fila de prioridade, para a es-

colha dos candidatos a expansão. Além disso, implementaram-se os algoritmos de busca greedy e A* e um procedimento para se obter o caminho encontrado a partir da lista de nós expandidos.

A fila de prioridade foi implementada usando uma lista ordenada com inserções e remoções usando algoritmo de busca binária. Caso a inserção e remoção se mostrasse um bottleneck do processo de busca, considerou-se uma implementação em heap, mas isso não se viu necessário.

O algoritmo de busca greedy foi implementado com os seguintes passos, sendo os passos relativos à medição de performance só implementados para a segunda atividade.

```
função BuscaGreedy():
    inicializar a árvore de busca;
    inicializar o caminho a ser retornado;
    inicializar a lista de nós expandidos;
    inicializar a fila com prioridade de
        ↪ candidatos;
    inicializar as variáveis relativas à
        ↪ medição de performance;

    adicionar o nó raiz da árvore de busca à
        ↪ lista de candidatos

    iterar por um número grande de
        ↪ tentativas:
        se a lista de candidatos está vazia:
            retornar -1 (falha de caminho
                ↪ não encontrado)

        escolher o candidato a ser expandido
            ↪ e retirá-lo da fila de
            ↪ prioridade (medindo-se o tempo
            ↪ para isso)
        adicionar ao caminho a tupla:
            (pai do nó escolhido,
                ↪ custo/movimento necessário
                ↪ para chegar do pai ao nó,
                ↪ nó escolhido)
        expandir o nó escolhido e adicionar
            ↪ seu identificador à lista de
            ↪ nós expandidos (medindo-se o
            ↪ tempo para isso)

        se o nó escolhido é o objetivo:
            retornar o caminho gerado

        para cada filho do nó escolhido que
            ↪ não seja seu pai:
            se o filho não está na lista dos
                ↪ nós expandidos (medindo-se
                ↪ o tempo para essa
                ↪ checagem):
                adicionar o filho ao nível
                    ↪ seguinte da árvore de
                    ↪ busca
                inserir o filho na lista de
                    ↪ candidatos, usando-se
                    ↪ a heurística como
                    ↪ prioridade (medindo-se
                    ↪ o tempo para isso)
    se chegar ao fim da iteração, retornar
        ↪ -2 (timeout)
```

Já algoritmo de busca A* foi implementado com os seguintes passos, sendo os passos relativos à medição de performance só implementados para a segunda atividade.

```
função BuscaA*():
    inicializar a árvore de busca;
    inicializar o caminho a ser retornado;
    inicializar a lista de nós expandidos;
    inicializar a fila com prioridade de
        ↪ candidatos;
    inicializar as variáveis relativas à
        ↪ medição de performance;

    adicionar o nó raiz da árvore de busca à
        ↪ lista de candidatos

    iterar por um número grande de
        ↪ tentativas:
        se a lista de candidatos está vazia:
            retornar -1 (falha de caminho
                ↪ não encontrado)

        escolher o candidato a ser expandido
            ↪ e retirá-lo da fila de
            ↪ prioridade (medindo-se o tempo
            ↪ para isso)
        adicionar ao caminho a tupla:
            (pai do nó escolhido,
                ↪ custo/movimento necessário
                ↪ para chegar do pai ao nó,
                ↪ nó escolhido)
        expandir o nó escolhido e adicionar
            ↪ seu identificador à lista de
            ↪ nós expandidos (medindo-se o
            ↪ tempo para isso)

        se o nó escolhido é o objetivo:
            retornar o caminho gerado

        para cada filho do nó escolhido que
            ↪ não seja seu pai:
            se o filho não está na lista dos
                ↪ nós expandidos (medindo-se
                ↪ o tempo para essa
                ↪ checagem):
                atualizar o custo do filho
                adicionar o filho ao nível
                    ↪ seguinte da árvore de
                    ↪ busca
                inserir o filho na lista de
                    ↪ candidatos, usando-se
                    ↪ a projeção como a
                    ↪ heurística (medindo-se
                    ↪ o tempo para isso)
    se chegar ao fim da iteração, retornar
        ↪ -2 (timeout)
```

Já o procedimento que identifica o caminho da raiz ao objetivo a partir da lista de tuplas

retornada pela função de busca é o seguinte:

```
função TrimPath(nó inicial, passos):
    p := passo que chega ao nó objetivo
    adiciona p ao fim do caminho
    nó := início do passo p
    até que o início do passo p seja o nó
        ↪ inicial:
        procura em passos o passo que chega
            ↪ em nó a partir de seu pai
        p := passo achado
        adiciona p ao início do caminho
        nó := início do passo p
    retorna caminho
```

2.3.1 *Primeira Atividade*

Na primeira atividade, criou-se uma estrutura de dados de grafo com uma lista de adjacências, visto que cada um dos mais de 200 nós tem no máximo grau 3. Além disso, cada nó possui como sua heurística a distância em linha reta do nó até o nó final, que só é calculada uma vez, no momento da criação do grafo. Além disso, há a estrutura de dados da árvore de busca, em que cada nó da árvore tem uma referência a um nó do grafo, uma referência ao seu nó pai, o seu custo e sua projeção (soma do custo com a heurística).

2.3.2 *Segunda Atividade*

Na segunda atividade, há a estrutura de dados da árvore de busca, em que cada nó da árvore tem uma matriz de estado, uma referência ao seu nó pai, a sua heurística, calculada por um método próprio no momento da criação do nó da árvore, o seu custo e sua projeção (soma do custo com a heurística) e uma lista dos seus filhos, preenchida apenas quando o nó é expandido via um método do nó.

2.4 Resposta da atividade

A resposta da primeira atividade é dada via *console*, com um detalhamento auxiliar nos arquivos "debugG1.txt"e "debugA1.txt", que contém a matriz de adjacências, a evolução da árvore de busca e a lista final do percurso em forma de tabela.

A resposta da segunda atividade, no entanto, é dada apenas pelos arquivos "debugG2.txt"e "debugA2.txt", pois neles está o passo a passo da resolução do problema. No *console*, é dado apenas o número de passos exigido e as medições de performance.

3 Resultado Experimental

3.1 Primeira Atividade

Trabalhou-se com distâncias usando os valores numéricos da latitude e longitude, visto que foi assim que as cidades foram localizadas. O comprimento do menor caminho encontrado, portanto, só traz informação quando comparado com o comprimento de outros caminhos calculados pelo programa. Ou seja, a informação real que o programa oferece é a sequência de cidades que representa o menor caminho. A sequência de cidades obtidas pelo algoritmo greedy está na tabela 1 e a obtida pelo A*, na tabela 2, ambas no Apêndice

Pode-se comparar a eficiência dos dois algoritmos usando a distância numérica dos dois caminhos acumulada ao final. Usando-se o algoritmo greedy, obteve-se uma distância de 2400,555 e, com o A*, 1647,355. Assumindo-se que o caminho obtido pelo A* é o ótimo, o caminho greedy é 45% maior que o ótimo.

3.2 Segunda Atividade

Para a segunda atividade, gerou-se tabuleiros 9x9 embaralhados a partir de um certo número de movimentos a partir do estado objetivo, considerando-se que a complexidade do algoritmo é exponencial com o número de passos de para o estado objetivo, sendo a base dessa exponencial o número de nós que cada nó gera, no caso um número de 1 a 3.

Na Figura 1, há exemplos de tabuleiros embaralhados de modo fácil e não tão fácil, um que pode ser obtido com em torno de 30 movimentos e outro com 300. Inicialmente, tentou-se embaralhar o tabuleiro com centenas de movimentos e o programa não terminou de rodar, com a memória estourando depois de horas de execução.

Fácil	Não tão fácil
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 0, 61, 62, 63, 64, 65, 66, 67, 68, 60, 71, 79, 72, 73, 74, 75, 76, 77, 69, 78, 70, 80	10, 44, 27, 28, 61, 8, 14, 17, 0, 22, 6, 16, 43, 48, 51, 36, 2, 68, 24, 38, 37, 45, 18, 41, 70, 34, 46, 55, 4, 1, 30, 50, 58, 32, 12, 9, 3, 23, 60, 56, 40, 15, 72, 54, 20, 7, 25, 11, 47, 5, 74, 29, 35, 26, 52, 57, 73, 65, 49, 42, 77, 78, 21, 31, 67, 13, 53, 62, 66, 80, 33, 69, 39, 75, 64, 19, 59, 76, 63, 79, 71

Figura 1: Exemplos de tabuleiros embaralhados de modo fácil e não tão fácil.

Em seguida, se ateve a um número de movimentos entre 10 e 20. Os resultados obtidos estão nas Figuras 2, 3 e 4.

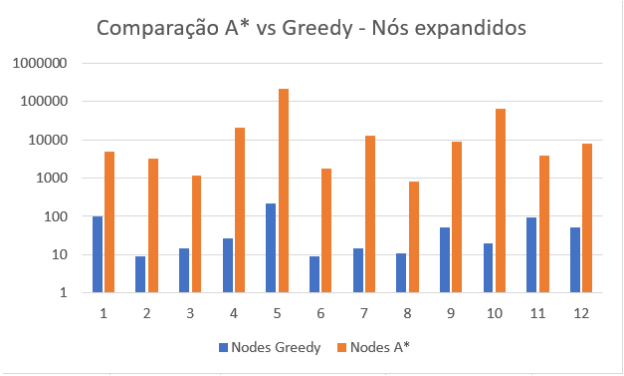

Figura 2: Comparação entre os dois algoritmos quanto ao número de nós expandidos.

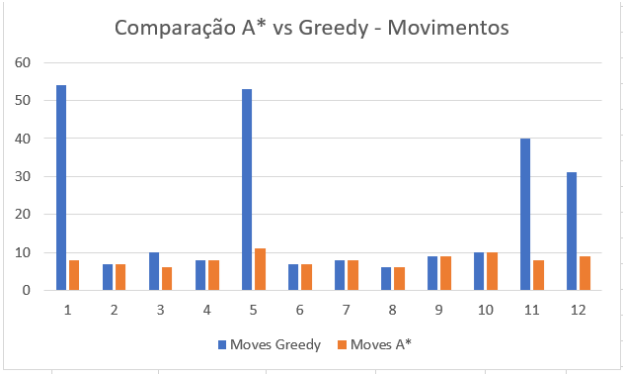

Figura 3: Comparação entre os dois algoritmos quanto ao número de movimentos.

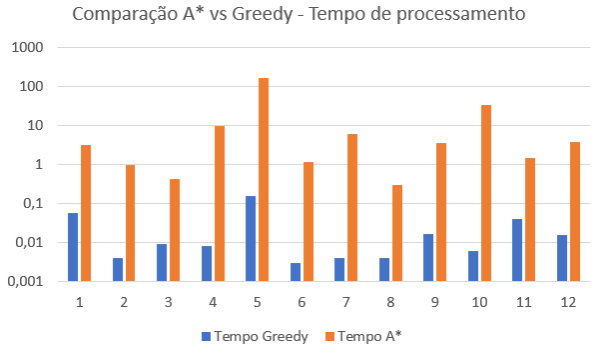

Figura 4: Comparação entre os dois algoritmos quanto ao tempo de execução.

Assim, foi possível verificar que, embora o algoritmo A* seja ótimo, em média ele expande na ordem de 100 vezes mais nós, o que resulta em tempos de processamento entre 100 e 1000 vezes maior.

4 Conclusão

Em conclusão, pode-se verificar na primeira atividade a diferença no desempenho dos algoritmos e, na segunda, a diferença entre seus tempos de execução. O A* sendo mais efetivo, o que era esperado, visto que ele é o algoritmo ótimo para o problema. No entanto, embora eles tenham a mesma complexidade de pior caso, observou-se que em média o A* expande mais nós e, portanto, é mais lento que o greedy. Essa diferença não foi significante para a primeira atividade, que só comparava valores escalares, mas foi muito diferente para a segunda atividade, visto que ela envolvia operações matriciais, que são mais custosas.

5 Apêndice

Tabela 1: Sequência de cidades obtidas com o algoritmo Greedy.

Origem	Distância	Destino	Cumulativo
Alice Springs	17,955	Ararat	17,955
Ararat	22,196	Atherton	40,151
Atherton	3,308	Ayr	43,459
Ayr	20,168	Ballarat	63,627
Ballarat	15,489	Barcaldine	79,116
Barcaldine	11,835	Bathurst	90,951
Bathurst	6,893	Bendigo	97,843
Bendigo	14,627	Bedourie	112,470
Bedourie	10,980	Berri	123,450
Berri	15,401	Biloela	138,851
Biloela	12,390	Birdsville	151,242
Birdsville	11,557	Bordertown	162,798
Bordertown	8,898	Bourke	171,696
Bourke	10,317	Boulia	182,013
Boulia	9,710	Bowen	191,723
Bowen	15,124	Broken Hill	206,847
Broken Hill	28,430	Bunbury	235,277
Bunbury	31,392	Burketown	266,669
Burketown	31,861	Busselton	298,530
Busselton	41,992	Caboolture	340,522
Caboolture	13,690	Cairns	354,212
Cairns	9,034	Camooweal	363,246
Camooweal	27,454	Carnarvon	390,700
Carnarvon	40,663	Canberra	431,364
Canberra	17,352	Ceduna	448,715
Ceduna	15,184	Charleville	463,899
Charleville	11,715	Clare	475,614
Clare	8,067	Cobram	483,681
Cobram	3,491	Colac	487,172
Colac	8,943	Cowell	496,115
Cowell	10,403	Cranbourne	506,518
Cranbourne	2,554	Currie	509,072
Currie	33,441	Darwin	542,512
Darwin	27,697	Dalby	570,210
Dalby	11,512	Deniliquin	581,722
Deniliquin	5,406	Dubbo	587,128
Dubbo	6,029	Echuca	593,157
Echuca	14,376	Emerald	607,533
Emerald	37,477	Exmouth	645,010
Exmouth	39,339	Forbes	684,349
Forbes	10,272	Gawler	694,621
Gawler	18,701	Georgetown	713,323
Georgetown	33,626	Gingin	746,949

Gingin	3,161	Geraldton	750,110
Geraldton	40,669	Gladstone	790,780
Gladstone	5,273	Goondiwindi	796,053
Goondiwindi	7,862	Griffith	803,915
Griffith	5,887	Gunnedah	809,802
Gunnedah	28,539	Halls Creek	838,341
Halls Creek	28,850	Gympie South	867,190
Gympie South	17,244	Hamilton	884,434
Hamilton	18,164	Hervey Bay	902,599
Hervey Bay	17,195	Horsham	919,794
Horsham	21,525	Innisfail	941,319
Innisfail	17,023	Ivanhoe	958,342
Ivanhoe	25,231	Kalgoorlie	983,573
Kalgoorlie	25,820	Karumba	1009,393
Karumba	26,636	Karratha	1036,029
Karratha	14,266	Katanning	1050,294
Katanning	36,029	Katoomba	1086,323
Katoomba	1,207	Kiama	1087,530
Kiama	15,965	Kimba	1103,495
Kimba	2,756	Kingoonya	1106,251
Kingoonya	8,224	Kingston South East	1114,475
Kingston South East	26,957	Kwinana	1141,433
Kwinana	8,290	Laverton	1149,722
Laverton	1,210	Leonora	1150,932
Leonora	25,912	Longreach	1176,844
Longreach	33,118	Manjimup	1209,961
Manjimup	30,509	Maryborough	1240,471
Maryborough	30,055	Meekatharra	1270,526
Meekatharra	31,185	Melton	1301,711
Melton	29,735	Merredin	1331,446
Merredin	23,628	Meningie	1355,074
Meningie	3,519	Mildura	1358,593
Mildura	29,285	Morawa	1387,878
Morawa	6,232	Mount Barker	1394,110
Mount Barker	28,472	Mount Isa	1422,582
Mount Isa	17,140	Mudgee	1439,723
Mudgee	1,481	Muswellbrook	1441,204
Muswellbrook	37,091	Narrogin	1478,295
Narrogin	10,861	Newman	1489,156
Newman	37,385	North Lismore	1526,541
North Lismore	14,945	North Scottsdale	1541,486
North Scottsdale	7,633	Nowra	1549,119
Nowra	41,669	Onslow	1590,787
Onslow	33,379	Ouyen	1624,166
Ouyen	32,189	Pannawonica	1656,355
Pannawonica	32,044	Penola	1688,399
Penola	5,318	Peterborough	1693,717
Peterborough	22,432	Pine Creek	1716,149

Pine Creek	25,210	Port Denison	1741,359
Port Denison	10,672	Port Hedland	1752,031
Port Hedland	29,841	Port Douglas	1781,872
Port Douglas	22,672	Port Lincoln	1804,544
Port Lincoln	2,899	Port Pirie	1807,444
Port Pirie	18,247	Proserpine	1825,691
Proserpine	16,465	Queanbeyan	1842,156
Queanbeyan	11,057	Quilpie	1853,213
Quilpie	6,587	Richmond	1859,799
Richmond	8,606	Rockhampton	1868,406
Rockhampton	3,983	Roma	1872,389
Roma	35,384	Roebourne	1907,773
Roebourne	38,040	Sale	1945,813
Sale	2,434	Seymour	1948,247
Seymour	VALOR!	Shepparton	1949,013
Shepparton	4,921	Smithton	1953,934
Smithton	24,442	South Ingham	1978,377
South Ingham	32,607	Southern Cross	2010,984
Southern Cross	16,468	Streaky Bay	2027,452
Streaky Bay	10,648	Swan Hill	2038,100
Swan Hill	2,781	Sunbury	2040,880
Sunbury	8,216	Sydney	2049,097
Sydney	10,372	Thargomindah	2059,468
Thargomindah	30,916	Three Springs	2090,385
Three Springs	7,847	Tom Price	2098,232
Tom Price	32,142	Townsville	2130,373
Townsville	20,367	Tumby Bay	2150,740
Tumby Bay	12,227	Traralgon	2162,967
Traralgon	3,684	Tumut	2166,651
Tumut	2,473	Ulladulla	2169,124
Ulladulla	13,038	Victor Harbor	2182,161
Victor Harbor	23,536	Wagin	2205,697
Wagin	32,051	Wangaratta	2237,748
Wangaratta	9,919	Wallaroo	2247,667
Wallaroo	7,239	Warrnambool	2254,906
Warrnambool	28,342	Weipa	2283,248
Weipa	22,946	Whyalla	2306,194
Whyalla	10,072	Windorah	2316,267
Windorah	3,371	Winton	2319,638
Winton	18,066	Wonthaggi	2337,704
Wonthaggi	7,426	Wollongong	2345,130
Wollongong	15,868	Woomera	2360,998
Woomera	17,693	Yeppoon	2378,691
Yeppoon	21,864	Yulara	2400,555

Tabela 2: Sequência de cidades obtidas com o algoritmo A*.

Origem	**Distância**	**Destino**	**Cumulativo**
Alice Springs	8,257	Andamooka	8,257
Andamooka	15,958	Armidale	24,216
Armidale	12,91	Ayr	37,126
Ayr	20,168	Ballarat	57,294
Ballarat	4,179	Bairnsdale East	61,473
Bairnsdale East	11,815	Ballina	73,288
Ballina	8,4	Batemans Bay	81,688
Batemans Bay	2,602	Bathurst	84,289
Bathurst	6,893	Bendigo	91,182
Bendigo	7,167	Bicheno	98,349
Bicheno	20,154	Birdsville	118,502
Birdsville	11,557	Bordertown	130,059
Bordertown	8,898	Bourke	138,957
Bourke	8,303	Brisbane	147,259
Brisbane	35,444	Broome	182,704
Broome	33,981	Bundaberg	216,684
Bundaberg	19,168	Burnie	235,852
Burnie	16,074	Byron Bay	251,925
Byron Bay	15,513	Cairns	267,438
Cairns	13,69	Caboolture	281,128
Caboolture	0,377	Caloundra	281,505
Caloundra	10,314	Canberra	291,819
Canberra	17,352	Ceduna	309,171
Ceduna	15,184	Charleville	324,355
Charleville	11,715	Clare	336,070
Clare	8,067	Cobram	344,136
Cobram	3,491	Colac	347,628
Colac	8,943	Cowell	356,570
Cowell	10,403	Cranbourne	366,973
Cranbourne	13,693	Dalby	380,666
Dalby	11,512	Deniliquin	392,179
Deniliquin	5,406	Dubbo	397,585
Dubbo	3,323	East Maitland	400,908
East Maitland	8,133	Eidsvold	409,041
Eidsvold	33,492	Esperance	442,533
Esperance	28,742	Forbes	471,276
Forbes	10,272	Gawler	481,548
Gawler	18,701	Georgetown	500,249
Georgetown	33,626	Gingin	533,876
Gingin	3,161	Geraldton	537,037
Geraldton	40,669	Gladstone	577,706
Gladstone	5,273	Goondiwindi	582,979
Goondiwindi	7,862	Griffith	590,842
Griffith	11,471	Gympie South	602,313
Gympie South	17,244	Hamilton	619,557

Hamilton	8,123	Hobart	627,680
Hobart	24,487	Hughenden	652,167
Hughenden	12,421	Inverell	664,588
Inverell	40,705	Kalbarri	705,293
Kalbarri	8,215	Karratha	713,508
Karratha	14,266	Katanning	727,773
Katanning	36,029	Katoomba	763,802
Katoomba	1,207	Kiama	765,009
Kiama	15,965	Kimba	780,975
Kimba	2,756	Kingoonya	783,731
Kingoonya	8,224	Kingston South East	791,955
Kingston South East	26,957	Kwinana	818,912
Kwinana	8,29	Laverton	827,202
Laverton	1,21	Leonora	828,411
Leonora	25,912	Longreach	854,323
Longreach	33,118	Manjimup	887,441
Manjimup	30,509	Maryborough	917,950
Maryborough	30,055	Meekatharra	948,005
Meekatharra	31,185	Melton	979,190
Melton	0,44	Melbourne	979,630
Melbourne	6,614	Meningie	986,244
Meningie	3,519	Mildura	989,763
Mildura	29,285	Morawa	1019,048
Morawa	6,232	Mount Barker	1025,280
Mount Barker	28,472	Mount Isa	1053,752
Mount Isa	17,14	Mudgee	1070,893
Mudgee	1,481	Muswellbrook	1072,374
Muswellbrook	2,464	Narrabri West	1074,837
Narrabri West	3,627	Newcastle	1078,464
Newcastle	33,016	Norseman	1111,480
Norseman	32,516	North Mackay	1143,996
North Mackay	9,58	North Lismore	1153,576
North Lismore	14,945	North Scottsdale	1168,521
North Scottsdale	7,633	Nowra	1176,154
Nowra	8,902	Oatlands	1185,056
Oatlands	10,103	Orange	1195,159
Orange	4,101	Pambula	1199,260
Pambula	4,571	Parkes	1203,832
Parkes	35,57	Perth	1239,402
Perth	28,113	Penola	1267,515
Penola	5,318	Peterborough	1272,832
Peterborough	1,315	Port Augusta West	1274,147
Port Augusta West	19,54	Port Douglas	1293,687
Port Douglas	22,672	Port Lincoln	1316,359
Port Lincoln	2,899	Port Pirie	1319,258
Port Pirie	6,914	Portland	1326,173
Portland	9,014	Queanbeyan	1335,186
Queanbeyan	11,057	Quilpie	1346,243

Quilpie	6,587	Richmond	1352,830
Richmond	8,606	Rockhampton	1361,436
Rockhampton	3,983	Roma	1365,419
Roma	6,451	Scone	1371,870
Scone	7,674	Shepparton	1379,545
Shepparton	0,766	Seymour	1380,311
Seymour	8,247	Singleton	1388,557
Singleton	3,715	South Grafton	1392,272
South Grafton	12,536	South Melbourne	1404,809
South Melbourne	2,552	Stawell	1407,360
Stawell	2,215	Sunbury	1409,576
Sunbury	8,216	Sydney	1417,792
Sydney	10,372	Thargomindah	1428,164
Thargomindah	30,916	Three Springs	1459,080
Three Springs	39,872	Toowoomba	1498,951
Toowoomba	13,126	Traralgon	1512,077
Traralgon	3,684	Tumut	1515,761
Tumut	2,473	Ulladulla	1518,234
Ulladulla	3,44	Wagga Wagga	1521,675
Wagga Wagga	10,772	Wallaroo	1532,446
Wallaroo	9,919	Wangaratta	1542,365
Wangaratta	10,939	Warwick	1553,305
Warwick	3,409	West Tamworth	1556,713
West Tamworth	8,301	Wilcannia	1565,014
Wilcannia	10,103	Winton	1575,117
Winton	3,371	Windorah	1578,488
Windorah	13,422	Wollongong	1591,910
Wollongong	15,868	Woomera	1607,778
Woomera	17,693	Yeppoon	1625,471
Yeppoon	21,864	Yulara	1647,335