MC658 - Projeto e análise de algoritmos III - Unicamp

Laboratório 2 - Backtracking e Branch-and-Bound

Erik Perillo, RA135582
Kelvin Ronny, RA138645

29 de abril de 2017

1 Introdução

O desafio do Laboratório é resolver o problema do empacotamento com divisórias por meio de algoritmos de *backtracking* e *branch and bound*, analisando quantitativamente seus desempenhos.

2 Backtracking

backtrack_rec é a função que faz recursão dentre as possibilidades de alocação de usuários na banda. Para isso, primeiro simulamos o que acontece se tentarmos colocar mais um usuário na banda. Então, usamos a função validator para validar a possibilidade. Se a possibilidade vale, testamos um backtracking para ela. Senão, apenas tentamos um backtracking sem o usuario, dada a solução candidata melhor até o momento.

3 Branch and Bound

A técnica branch and bound ordena todos os pacotes enfileirados e monta uma árvore com ramificação binária composta pela escolha (ou não) de um certo item.

A ordenação é feita pelo valor relativo, do maior para o menor. O critério de poda usa dos seguintes fatos: Seja K o conjunto de classes presentes na solução atual. Seja r_K o número de pacotes cuja classe pertence ao conjunto K mas não estão na solução atual. A melhor situação possível é que todos os r_K primeiros pacotes (ordenados por valor relativo) ainda não presentes na solução tenham classe no conjunto K, pois isso evita a necessidade de uma divisória nova. Após usar todos esses pacotes, o melhor que pode acontecer é que os pacotes restantes, em ordem de valor relativo, pertençam respectivamente às classes c_1, c_2, ..., c_k, sendo o número de elementos da classe c_1 maior que o da c_2 e assim por diante. Esse cenário garante o menor uso de divisórias possível. Assim, o critério de pode assume esse cenário ideal e calcula o valor máximo que pode ser obtido pela ramificação. Se este ainda for menor que o atual máximo, não faz sentido continuar a busca naquele ramo e então é feita a poda.

`bnb` implementa o algoritmo branch and bound. A rotina ordena os elementos por valor relativo e então chama `_bnb`, rotina recursiva que faz a busca de fato dos valores.

4 Tempos

Tabela 1: Branch and bound para arquivos de teste padrão.

arquivo	tlimite (s)	valor	tempo (s)	timeout
1000_50_800.in	15	2177	16.012	1
100_5_500.in	15	763	0.003	0
5_2_20.in	15	17	0.002	0

Nota-se que foi alcançado o valor ótimo para os três casos e os dois menores terminaram quase instantaneamente. Foram gerados mais valores em um intervalo mais amplo para melhor compreensão do algoritmo:

Tabela 2: Branch and bound para diversos arquivos de teste.

arquivo	tlimite (s)	valor	tempo (s)	timeout
10_1_3.in	15	27	0.003	0
10_2_7.in	15	0	0.003	0
10_5_3.in	15	21	0.003	0
50_13_20.in	15	75	0.002	0
50_6_30.in	15	121	0.003	0
50_8_32.in	15	94	0.003	0
100_16_79.in	15	178	0.004	0
100_26_33.in	15	104	0.002	0
100_45_61.in	15	110	0.003	0
200_11_88.in	15	338	0.006	0
200_1_96.in	15	544	0.004	0
200_78_68.in	15	183	0.004	0
300_114_206.in	15	405	0.025	0
300_71_222.in	15	446	0.029	0
300_97_276.in	15	529	0.055	0
350_124_302.in	15	464	0.028	0
350_68_134.in	15	298	0.009	0
350_98_254.in	15	447	0.037	0
400_192_267.in	15	458	0.067	0
400_29_329.in	15	916	0.079	0
400_51_205.in	15	556	0.055	0
450_192_250.in	15	523	0.084	0
450_70_354.in	15	847	2.380	0
450_81_308.in	15	612	0.375	0
500_201_272.in	15	513	0.151	0
500_204_280.in	15	609	0.143	0
500_7_434.in	15	1566	0.005	0
550_173_341.in	15	574	0.703	0
550_236_493.in	15	769	2.070	0
550_82_483.in	15	1023	16.004	1
600_124_486.in	15	906	4.761	0
600_125_574.in	15	1064	16.014	1
600_259_480.in	15	775	6.515	0
700_104_423.in	15	991	8.519	0
700_272_529.in	15	802	4.737	0
700_321_353.in	15	770	0.911	0
800_265_536.in	15	995	16.003	1
800_271_532.in	15	792	8.380	0
800_298_642.in	15	1014	16.035	1
900_225_564.in	15	1067	16.007	1
900_322_386.in	15	721	5.452	0
900_396_476.in	15	951	16.003	1
1000_129_355.in	15	852	4.094	0
1000_458_557.in	15	1022	16.029	1
1000_72_967.in	15	2241	16.011	1