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. Um critério de poda preciso e uma mescla de busca em largura e profundidade foram obtidos em busca de uma rápida aproximação do valor ótimo.

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.

Iterative deepening depth-first search (IDDFS) é feita para a parte mais ao topo da árvore com o objetivo de já obter uma boa aproximação do valor ótimo. Assim, começa-se da profundidade 1 e vai-se até um certo limite (definido aqui como duas vezes o número de classes). Essa heurística deu bons resultados na prática, mostrando um bom compromisso entre tempo e aproximação do valor ótimo.

`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

Para o branch and bound, estes são os resultados para o conjunto de testes inicial:

arquivo	tempo-limite	sol-valor	tempo (s)	timeout
1000_50_800.in	15	2177	16.008	1
100_5_500.in	15	763	0.003	0
5_2_20.in	15	0	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:

arquivo	tempo-limite	sol-valor	tempo (s)	timeout
10_1_3.in	15	0	0.003	0
50_8_32.in	15	0	0.003	0
100_26_33.in	15	0	0.003	0
200_78_68.in	15	183	0.004	0
300_97_276.in	15	529	0.048	0
350_68_134.in	15	298	0.008	0
350_98_254.in	15	447	0.023	0
400_192_267.in	15	458	0.037	0
400_51_205.in	15	556	0.039	0
450_70_354.in	15	847	2.320	0
450_81_308.in	15	612	0.367	0
500_204_280.in	15	609	0.140	0
500_7_434.in	15	1566	0.006	0
550_173_341.in	15	574	0.698	0
550_236_493.in	15	769	2.025	0
550_82_483.in	15	1023	16.003	1
600_124_486.in	15	906	4.650	0
600_125_574.in	15	1064	16.003	1
600_259_480.in	15	775	6.644	0
700_104_423.in	15	991	8.662	0
700_272_529.in	15	802	4.321	0
700_321_353.in	15	770	0.925	0
800_265_536.in	15	995	16.003	1
800_271_532.in	15	792	8.363	0
800_298_642.in	15	1014	16.004	1
900_225_564.in	15	1067	16.010	1
900_322_386.in	15	721	5.451	0
900_396_476.in	15	951	16.003	1
1000_129_355.in	15	852	4.066	0
1000_458_557.in	15	1022	16.003	1
1000_72_967.in	15	2241	16.004	1