

# Trabalho prático 0

Victor Pinto / Thiago Theodoro

15 de março de 2005

## 1 Introdução

Neste trabalho serão abordados estruturas auxiliares que auxiliarão na implementação das heurísticas do TSP.

## 2 KDTree

A KDtree é uma estrutura utilizada pra repartir pontos no espaço, em duas ou três dimensões. Foram efetuados os seguintes testes com os seguintes conjuntos de pontos:

Conj. de ptos	Tempo	Mem
<b>att48</b>	0m0.021s	894,936 bytes
<b>kroA100</b>	0m0.028s	1,914,972 bytes
<b>lin105</b>	0m0.030s	1,895,524 bytes
<b>lin318</b>	0m0.073s	8,980,352 bytes
<b>pr1002</b>	0m0.210s	29,024,760 bytes
<b>nrw1379</b>	0m0.504s	92,170,960 bytes

Tabela 1: Tempo de execução e Memória alocada

### 2.1 Conclusões

Para todos os conjuntos de pontos a construção da KDTree foi realizada com sucesso. E ainda foi usada a função `isKdtree` para avaliar se a árvore criada realmente é uma KDTree, e em todos os casos o resultado foi `True`. Por enquanto ela, a KDTree, ainda não teve utilidade, mas mais a frente será de grande utilidade às heurísticas do TSP, devido a sua praticidade, e agilidade na busca por elementos.

## 3 Comparações entre heurísticas

Foram testadas quatro heurísticas para a solução do TSP. Segue abaixo as comparações entre elas, Tempo de execução (Tabela 1), memória alocada (Tabela 2), e taxa entre o peso da heurística e o melhor tour existente (Tabela 3). Para uma maior quantidade pontos ocorreu estouro da pilha (Stack Over Flow), e

só foi comparada a taxa, os conjuntos de pontos que possuem seus respectivos melhores tours.

	Heurística			
Conj. de ptos	NN	FRP	Greedy	PQGreedy
<b>att48</b>	0m0.015s	0m0.013s	0m0.025s	0m0.018s
<b>kroA100</b>	0m0.024s	0m0.022s	0m0.090s	0m0.040s
<b>lin105</b>	0m0.026s	0m0.023s	0m0.105s	0m0.042s
<b>lin318</b>	0m0.084s	0m0.059s	0m1.552s	0m0.196s
<b>pr1002</b>	0m0.578s	0m0.175s	0m36.536s	0m1.268s
<b>nrv1379</b>	0m0.981s	0m0.344s	1m19.034s	0m2.397s

Tabela 2: Tempo de execução

	Heurística			
Conj. de ptos	NN	FRP	Greedy	PQGreedy
<b>att48</b>	1,153,364	1,000,832	2,801,336	2,405,740
<b>kroA100</b>	3,009,992	2,133,260	13,451,524	6,280,300
<b>lin105</b>	2,128,716	15,496,112	3,142,540	6,964,664
<b>lin318</b>	17,631,788	10,082,148	245,442,384	38,961,200
<b>pr1002</b>	142,931,972	32,202,780	5,912,549,036	247,948,728
<b>nrv1379</b>	263,092,244	99,713,164	14,160,907,156	493,422,100

Tabela 3: Memória alocada, em bytes

	Heurística			
Conj. de ptos	NN	FRP	Greedy	PQGreedy
<b>att48</b>	1.20888836171914	1.652023028724	1.19796914883162	1.28012564041996
<b>kroA100</b>	1.26172560101985	1.87304553227699	1.13679974385868	1.27765186805663
<b>lin105</b>	1.41575215007234	1.88181117661256	1.16608712817508	1.15319634247905
<b>pr1002</b>	3.15221792504948	4.68257352011615	2.89946804631958	2.98857390070824

Tabela 4: Qualidade do tour ( somente para os conjuntos de pontos que possuem arquivo com melhor tour)

### 3.1 Conclusões

Analisando os resultados obtidos a partir dos perfis dos testes dos pontos, verificamos que: No quesito tempo a heurística do FRP supera as demais, e assim como no quesito memória alocada, aonde ele troca vantagens com o pqgreedy, já que os dois usam a estrutura da KDtree, o que agiliza a busca pelo ponto mais próximo. Porém se tratando de qualidade de tour, mesmo sendo mais lento e

alocando uma quantidade de memória muito maior ( devido ao tamanho de sua lista de prioridade ), o tour gerado pela Greedy é superior aos demais.