Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

SCC0272 – Introdução à Computação Bioinspirada

Trabalho 3 – Algoritmo ACO para resolver TSP

Elias Italiano Rodrigues – 7987251
Vinicius Katata Biondo – 6783972

São Carlos, 03 de julho de 2015

Sumário

1 Introdução

Este trabalho implementa o algoritmo ACO (*Ant Colony Optimization*) para resolver TSP (*Travelling Salesman Problem*), também conhecido em português como "O Problema do Caixeiro Viajante".

O algoritmo implementado pode ser executado com uma quantidade arbitrária de *threads*. Os parâmetros do algoritmo também podem ser definidos pelo usuário como descrito na Seção 2.2.

O objetivo desse relatório é mostrar e comentar a execução do algoritmo para diferentes valores de *formigas* e *threads* para um escolhido caso de teste.

2 O Programa `aco-tsp`

O programa implementa o algoritmo ACO em linguagem C especificamente para resolver TSPs do tipo `EUC_2D` obtidos do TSPLIB [1].

2.1 Estrutura

O diretório do projeto que acompanha este documento está organizado da seguinte maneira:

`./data`
- `|-- ./tsp_euc-2d` : diretório com os arquivos `.tsp` para teste.
- `|-- README` : informação sobre como foram obtidos e tratados os casos de teste.
- `|-- optimal.result` : arquivo com os resultados ótimos conhecidos.

`./doc` : arquivos-fonte deste relatório.
`Makefile` : arquivo para automação do processo de compilação do programa.
`README` : instruções de como compilar e executar o programa.
`RELATORIO_pt-br.pdf` : este relatório compilado a partir de `./doc`.
`aco-tsp` : programa compilado em sistema operacional Linux.
`main.c` : código-fonte do programa em linguagem C.
`test.sh` : *shell-script* para testar o programa compilado.

2.2 Parâmetros

O programa foi feito para aceitar *flags* pelas quais podem ser informados parâmetros de execução. Seguem as descrições para cada *flag* disponível (pode-se consultá-las executando `./aco-tsp --help` na linha de comando):

```
-i, --in          input .tsp EUC_2D file from TSPLIB
                  (default ./data/tsp_euc-2d/a280.tsp)
-e, --expected    expected results (default ./data/optimal.result)
-t, --threads     number of threads to be used (default 1)
-k, --ants        number of ants (default 10)
-a, --alpha       alpha constant (default 1.000000)
-b, --beta        beta constant (default 1.000000)
```

```
  -r, --rho          rho constant (default 0.100000)
  -p, --pheromone    initial pheromone value (default 1.000000)
  -w, --weight       weight for depositing pheromone (default 1.000000)
  -n, --times        number of runs (default 1)
  --print            print progress of best distances for each iteration
  --help             print this help information
  --version          print version number
```

Exemplo: o seguinte comando executa o programa 100 vezes sobre o caso de teste padrão (`./data/tsp_euc-2d/a280.tsp`) utilizando 2 *threads*, 20 formigas e os valores 2, 2 e 0.15 para as constantes α, β e ρ respectivamente.

```
./aco-tsp -t 2 -k 20 --alpha 2 --beta 2 --rho 0.15 -n 100
```

O resultado obtido é impresso na saída padrão no seguinte formato:

```
input path            : ./data/tsp_euc-2d/a280.tsp
input filename        : a280.tsp
input dimension       : 280
expected results file : ./data/optimal.result

threads   : 2
ants      : 20
alpha     : 2.000000
beta      : 2.000000
rho       : 0.150000
pheromone : 1.000000
weight    : 1.000000
times     : 100

time (s)        : 12.740000
ants result     : 2778
expected result : 2579
error           : 0.077162

ants path       :
0 279 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 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101
102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158
159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253
254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273 274 275 276 277 278
```

3 Execuções e Comentários

Escolhido o caso de teste `./data/tsp_euc-2d/a280.tsp`, definiu-se executar o programa 10 vezes variando-se o número de *threads* em 1, 2 e 4, e o número de formigas em 10, 100, 1000. As constantes escolhidas foram $\alpha = 2$, $\beta = 2$, $\rho = 0.15$.

Tabela 1: Resultados obtidos nas execuções.

Threads	**Dados**	**Formigas**		
		10	**100**	**1000**
	Esperado	2579		
1	**Obitdo**	2967	2847	2784
	Erro	0.150446	0.103916	0.079488
	Tempo (s)	0.540838	5.341067	54.789295
2	**Obtido**	2952	2916	2784
	Erro	0.144630	0.130671	0.079488
	Tempo (s)	0.271209	2.714911	27.986094
4	**Obtido**	2991	2929	2789
	Erro	0.159752	0.135712	0.081427
	Tempo (s)	0.317286	2.706635	27.962556

As execuções foram realizadas no seguinte computador:

```
Hardware:
Processor: Intel Core 2 Duo CPU T6600 @ 2.20GHz (Total Cores: 2)
Motherboard: clevo M7x0S, Chipset: SiS
System Memory: 2 x 2048 MB DRAM
Graphics: SiS [SiS] 771/671 PCIE VGA Display Adapter 256MB

Software:
OS: Ubuntu 10.04, Kernel: 2.6.32-74-generic-pae (i686)
Compiler: GCC 4.4.3
```

Como pode-se observar pelo gráfico da Figura 1, independentemente do número de *threads*, o erro obtido diminuiu conforme aumentou-se o número de formigas. Esse comportamento é esperado, pois sabe-se do algoritmo que quanto maior o número de formigas, maiores são as chances de um menor caminho ser percorrido.

Pelo gráfico da Figura 2 pode-se concluir que a quantidade de *threads* influenciou diretamente no tempo de execução, pois tornou possível formigas percorrerem o caminho paralelamente. Percebe-se que o ganho de 2 para 4 *threads* é muito pequeno, justificado pelas configurações do computador usado que não possui mais do que 2 *cores* de processamento.

Da Tabela 1, observa-se que o número de *threads* não influenciou na qualidade da solução obtida nesse experimento, pois o erro pouco variou com o aumento do número de *threads*. Portanto, o aumento das *threads* só permitiu uma execução mais rápida.

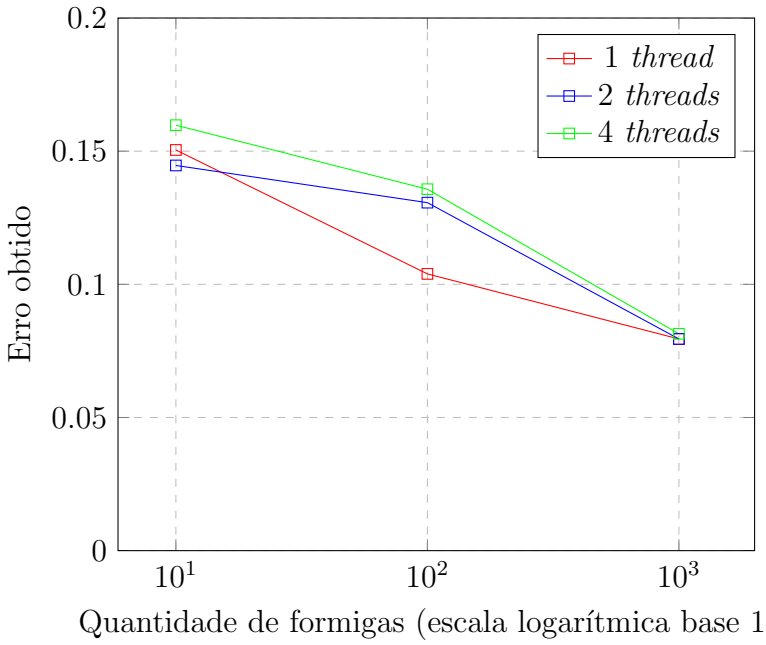

Figura 1: Comparativo dos erros obtidos em relação ao número de formigas.

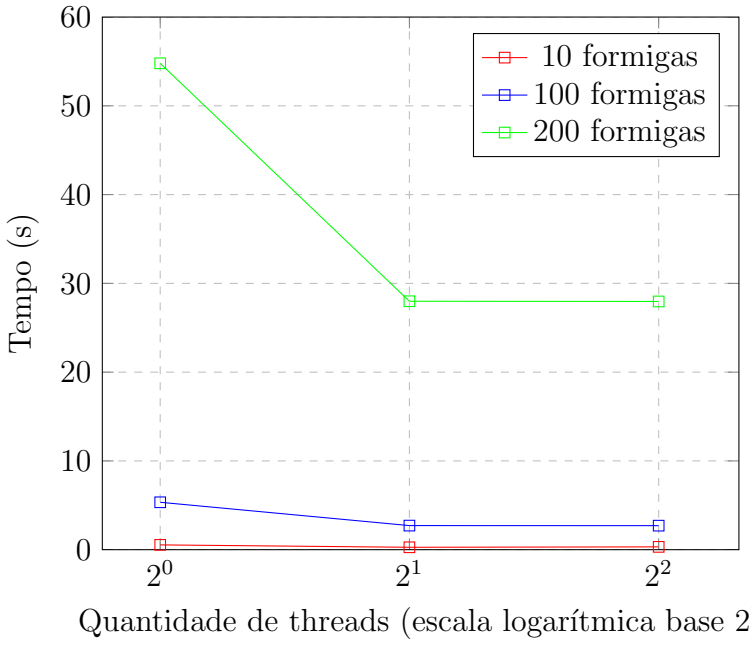

Figura 2: Comparativo dos tempos obtidos de acordo com o número de *threads*.

4 Conclusão

Com este trabalho foi possível aprender mais sobre a algoritmo ACO, analisar seu comportamento na prática com a variação de parâmetros, conhecer a base de casos de teste do TSPLIB e ter uma experiência *hands on* na programação do ACO em linguagem C com pthreads.

Referências

[1] TSPLIB
<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>