Heurísticas com Paralelismo através do Problema do Caixeiro Viajante

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

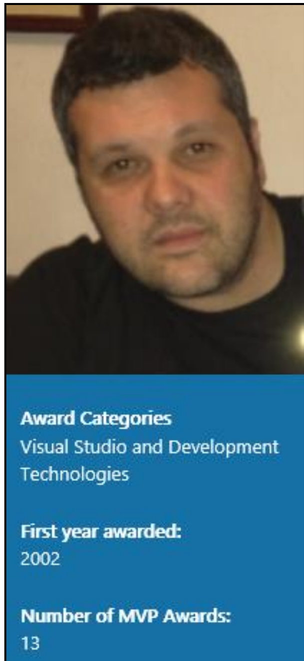
fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

https://mvp.microsoft.com/en-us/PublicProfile/9529

http://bit.ly/pcv-mack-abril-2016



Award Categories
Visual Studio and Development Technologies

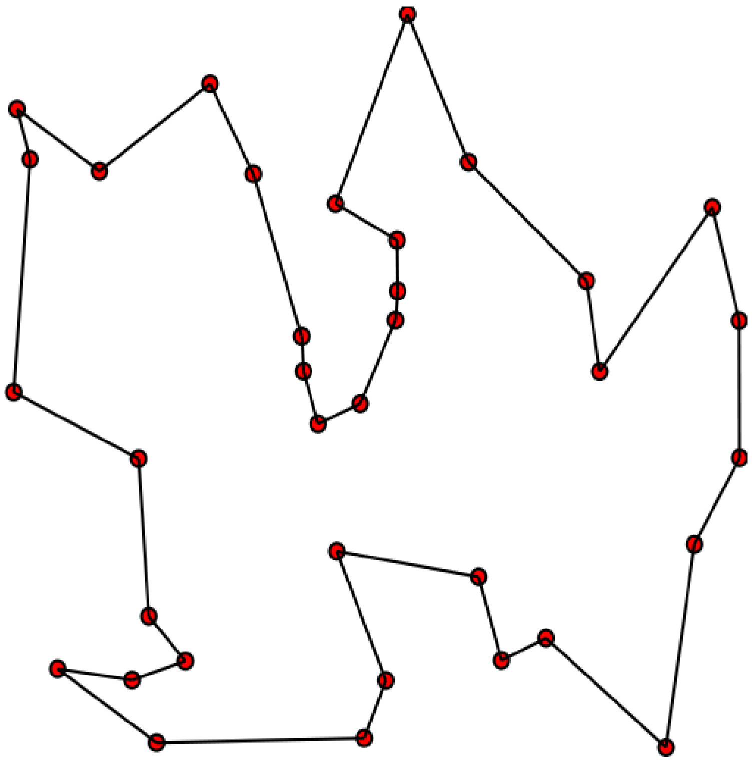
First year awarded:
2002

Number of MVP Awards:
13

Fabio Razzo Galuppo, M.Sc.

Novembro 1973

- Mestrado em Engenharia Elétrica (Universidade Presbiteriana Mackenzie)
  - Ciência da Computação - Inteligência Artificial
- Por mais de 10 anos premiado com Microsoft MVP em Visual C++
- Engenheiro de Software (Programador)
- Matemática Aplicada
- Linguagens de programação prediletas:
  - C++
  - F#
  - Haskell
- Rock'n'Roll
  - E boa música em geral
- http://fabiogaluppo.com
- https://twitter.com/FabioGaluppo
- https://github.com/fabiogaluppo
- http://simplycpp.com



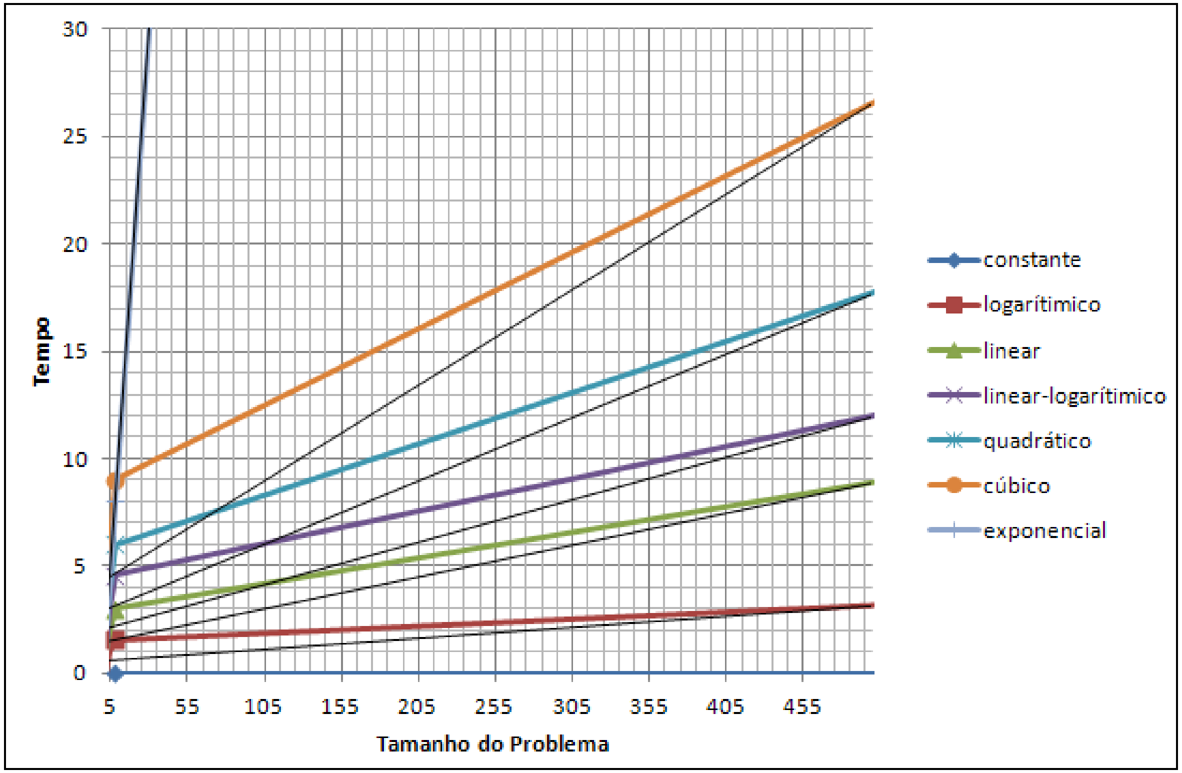
UMA VIAGEM AOS PROBLEMAS INTRATÁVEIS VIA HEURÍSTICAS COM PARALELISMO

http://en.wikipedia.org/wiki/Travelling_salesman_problem

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Problemas Intratáveis

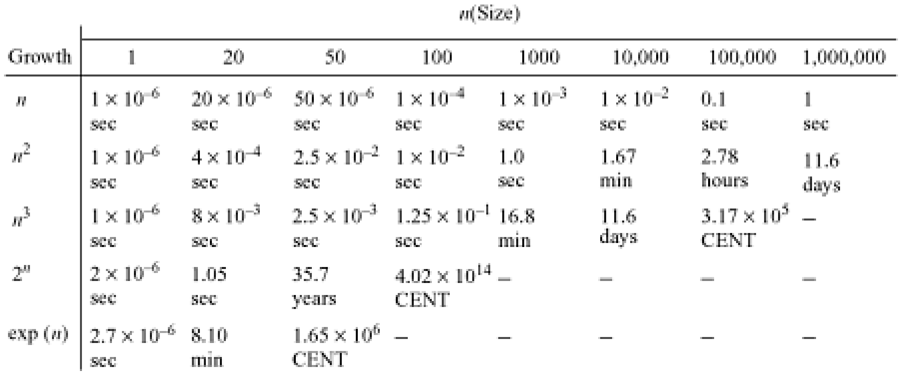
- Um problema é considerado intratável quando não existe um algoritmo conhecido que o resolva deterministicamente em tempo polinomial.



- Este tipo de problema é denominado NP.
  - Aquele que possui tempo polinomial não determinístico.

Problemas Intratáveis

Complexidade Computacional



	n(Size)							
Growth	1	20	50	100	1000	10,000	100,000	1,000,000
n	1×10^{-6} sec	20×10^{-6} sec	50×10^{-6} sec	1×10^{-4} sec	1×10^{-3} sec	1×10^{-2} sec	0.1 sec	1 sec
n^2	1×10^{-6} sec	4×10^{-4} sec	2.5×10^{-2} sec	1×10^{-2} sec	1.0 sec	1.67 min	2.78 hours	11.6 days
n^3	1×10^{-6} sec	8×10^{-3} sec	2.5×10^{-3} sec	1.25×10^{-1} sec	16.8 min	11.6 days	3.17×10^{5} CENT	–
2^n	2×10^{-6} sec	1.05 sec	35.7 years	4.02×10^{14} CENT	–	–	–	–
exp (n)	2.7×10^{-6} sec	8.10 min	1.65×10^{6} CENT	–	–	–	–	–

Figure 3.5 Sample Growth of Problem Complexity (one step = 10^{-6} sec.)

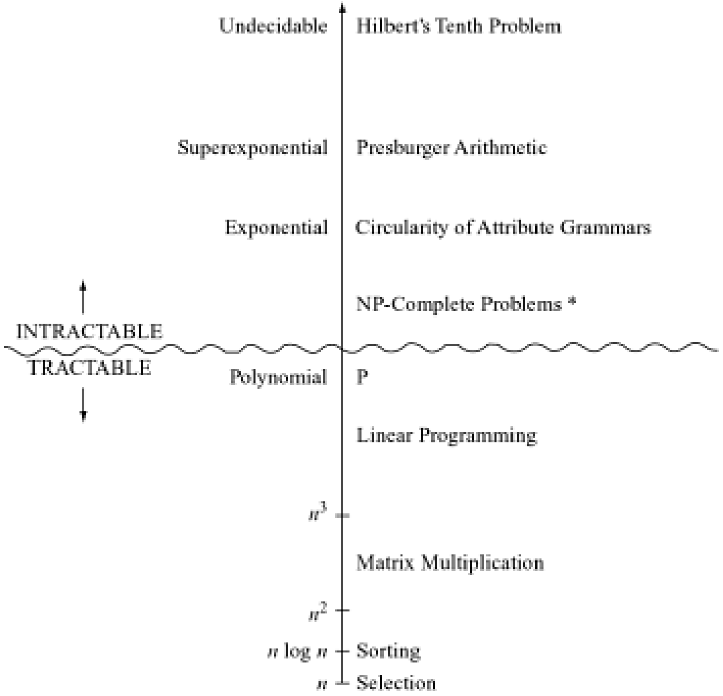
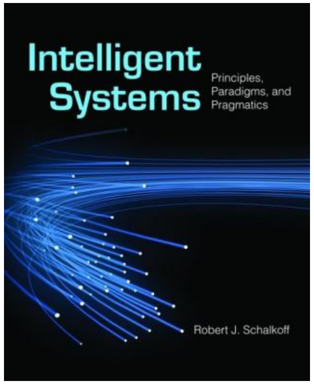


Figure 3.4 The Spectrum of Computational Complexity



http://www.jblearning.com/catalog/9780763780173/preview/

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

P = NP ?

http://www.claymath.org/millennium-problems/p-vs-np-problem

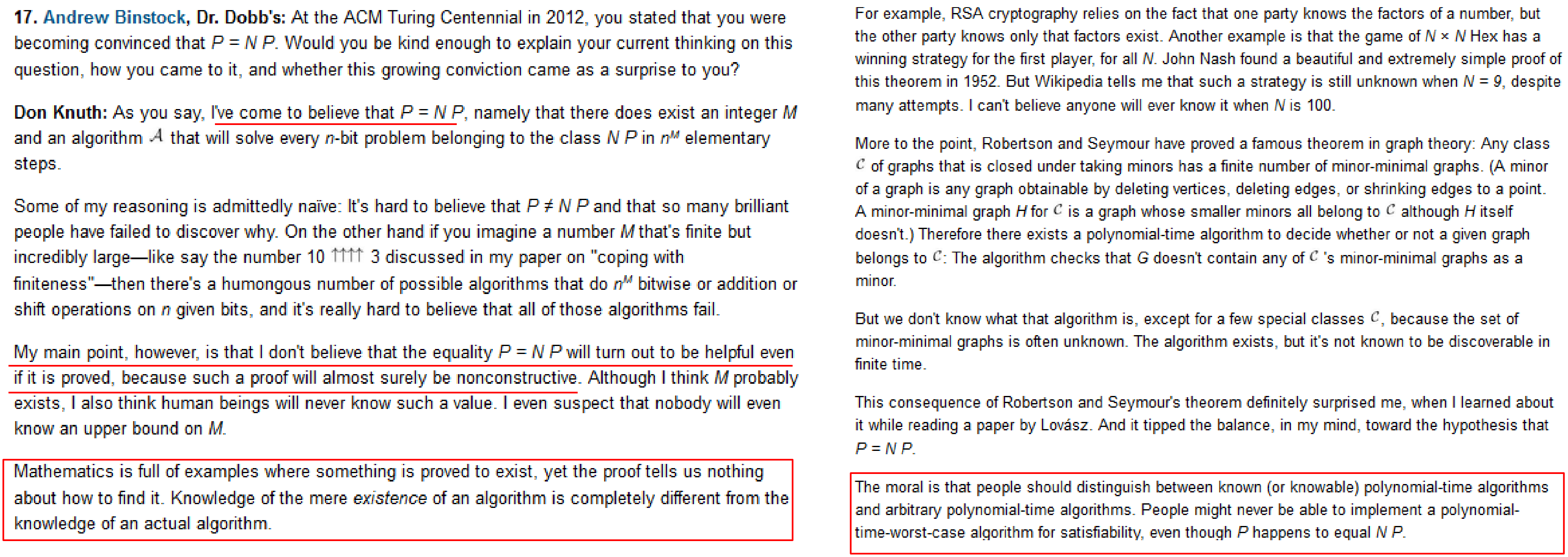
P=NP?
50 3D 4E 50 3F
ASCII encoding in hexadecimal

01010000
00111101
01001110
01010000
00111111

a conjecture encoded in binary form — P=NP?

https://www.cs.princeton.edu/news/article/princeton-cs-30-years-and-still-roll

Donald Knuth, qual é a sua opinião sobre P vs. NP?



17. Andrew Binstock, Dr. Dobb's: At the ACM Turing Centennial in 2012, you stated that you were becoming convinced that $P = NP$. Would you be kind enough to explain your current thinking on this question, how you came to it, and whether this growing conviction came as a surprise to you?

Don Knuth: As you say, I've come to believe that $P = NP$, namely that there does exist an integer M and an algorithm $\mathcal{A}$ that will solve every n-bit problem belonging to the class NP in n^M elementary steps.

Some of my reasoning is admittedly naïve: It's hard to believe that $P \neq NP$ and that so many brilliant people have failed to discover why. On the other hand if you imagine a number M that's finite but incredibly large—like say the number $10 \uparrow\uparrow\uparrow\uparrow 3$ discussed in my paper on "coping with finiteness"—then there's a humongous number of possible algorithms that do n^M bitwise or addition or shift operations on n given bits, and it's really hard to believe that all of those algorithms fail.

My main point, however, is that I don't believe that the equality $P = NP$ will turn out to be helpful even if it is proved, because such a proof will almost surely be nonconstructive. Although I think M probably exists, I also think human beings will never know such a value. I even suspect that nobody will even know an upper bound on M.

Mathematics is full of examples where something is proved to exist, yet the proof tells us nothing about how to find it. Knowledge of the mere *existence* of an algorithm is completely different from the knowledge of an actual algorithm.

For example, RSA cryptography relies on the fact that one party knows the factors of a number, but the other party knows only that factors exist. Another example is that the game of $N \times N$ Hex has a winning strategy for the first player, for all N. John Nash found a beautiful and extremely simple proof of this theorem in 1952. But Wikipedia tells me that such a strategy is still unknown when $N = 9$, despite many attempts. I can't believe anyone will ever know it when N is 100.

More to the point, Robertson and Seymour have proved a famous theorem in graph theory: Any class $\mathcal{C}$ of graphs that is closed under taking minors has a finite number of minor-minimal graphs. (A minor of a graph is any graph obtainable by deleting vertices, deleting edges, or shrinking edges to a point. A minor-minimal graph H for $\mathcal{C}$ is a graph whose smaller minors all belong to $\mathcal{C}$ although H itself doesn't.) Therefore there exists a polynomial-time algorithm to decide whether or not a given graph belongs to $\mathcal{C}$: The algorithm checks that G doesn't contain any of $\mathcal{C}$'s minor-minimal graphs as a minor.

But we don't know what that algorithm is, except for a few special classes $\mathcal{C}$, because the set of minor-minimal graphs is often unknown. The algorithm exists, but it's not known to be discoverable in finite time.

This consequence of Robertson and Seymour's theorem definitely surprised me, when I learned about it while reading a paper by Lovász. And it tipped the balance, in my mind, toward the hypothesis that $P = NP$.

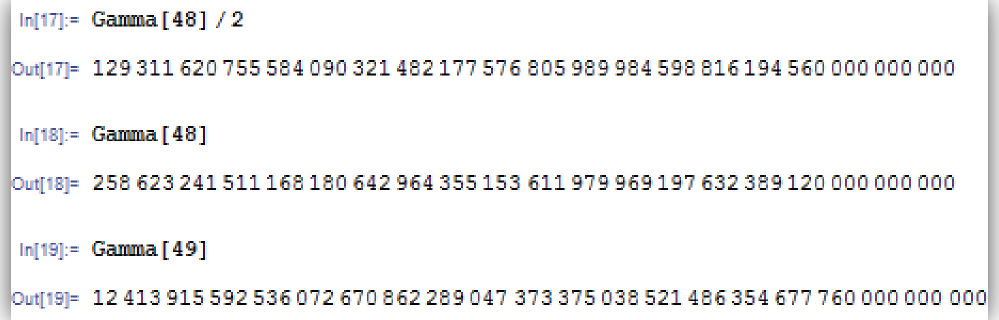
The moral is that people should distinguish between known (or knowable) polynomial-time algorithms and arbitrary polynomial-time algorithms. People might never be able to implement a polynomial-time-worst-case algorithm for satisfiability, even though P happens to equal NP.

http://www.informit.com/articles/article.aspx?p=2213858

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

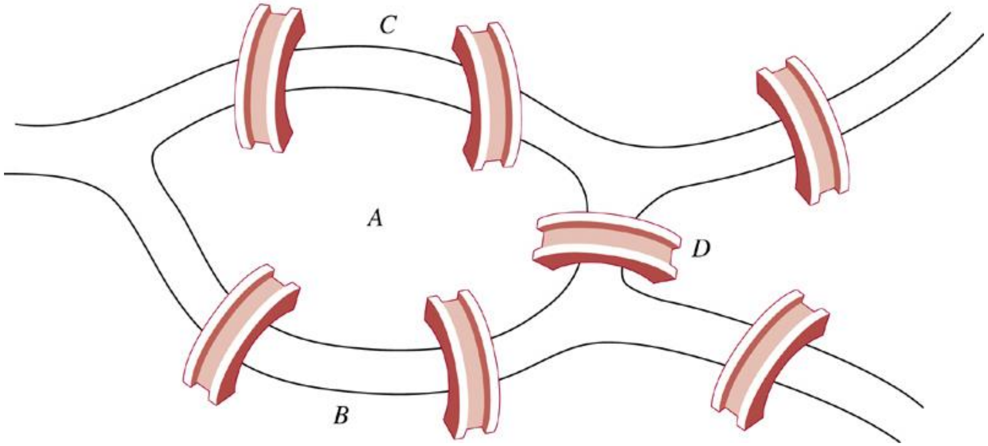
O Problema do Caixeiro Viajante (PCV)

- Problema de Minimização
  - Encontrar o menor ciclo para um conjunto de cidades a serem visitadas obrigatoriamente e retornando a origem
  - Uma de suas instâncias considera a função objetivo como a distância euclidiana entre as cidades
    - Problema NP, inerentemente intratável
- Intratabilidade
  - Simétrica = $\frac{\Gamma(N)}{2}$
  - Assimétrica = $\Gamma(N)$

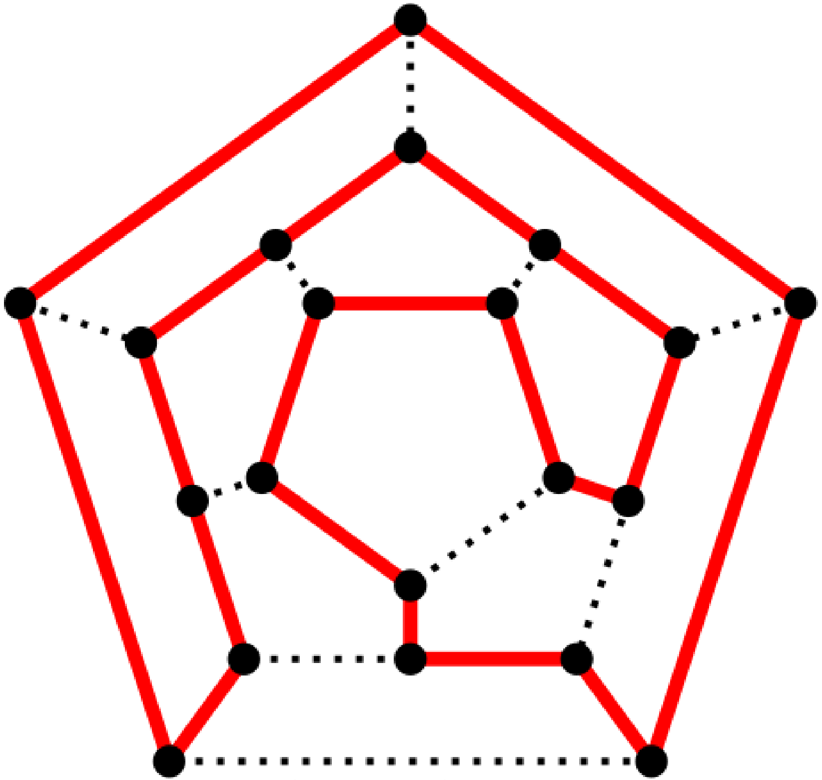


```
In[17]:= Gamma[48] / 2

Out[17]= 129 311 620 755 584 090 321 482 177 576 805 989 984 598 816 194 560 000 000 000

In[18]:= Gamma[48]

Out[18]= 258 623 241 511 168 180 642 964 355 153 611 979 969 197 632 389 120 000 000 000

In[19]:= Gamma[49]

Out[19]= 12 413 915 592 536 072 670 862 289 047 373 375 038 521 486 354 677 760 000 000 000
```

Origens do PCV



Ciclo Euleriano



Ciclo Hamiltoniano

- No ano de 1930, o problema se caracterizou como PCV por Merrill Flood da Universidade de Princeton e da RAND Corporation.

O Problema do Caixeiro Viajante (PCV)

$$f_{objetivo} = minimizar \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij}$$

Onde:

$$c_{ij} \rightarrow custo\ ou\ distância\ da\ cidade\ i\ para\ cidade\ j$$

$$N \rightarrow quantidade\ de\ cidades\ a\ serem\ visitadas$$

$$i, j, N\ \in\ \mathbb{N}^{+}, i \neq j$$

$$x_{ij} = \begin{cases} 1, & vai\ da\ cidade\ i\ para\ cidade\ j \\ 0, & não\ vai\ da\ cidade\ i\ para\ cidade\ j \end{cases}$$

$$x_{ij} \rightarrow variáveis\ binárias\ de\ decisão$$

Restrições:

$$\sum_{i=1}^{N} x_{ij} = 1 \qquad \forall_{j} \in \mathbb{N}^{+}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \qquad \forall_{i} \in \mathbb{N}^{+}$$

$$\sum_{i.j\ \in S} x_{ij} \leq\ |S| - 1 \qquad \forall_{\mathbf{S}} \subset \mathbb{N}^{+}$$

Solution of a large-scale traveling-salesman problem

1954

G Dantzig , R Fulkerson , S Johnson

http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.9319

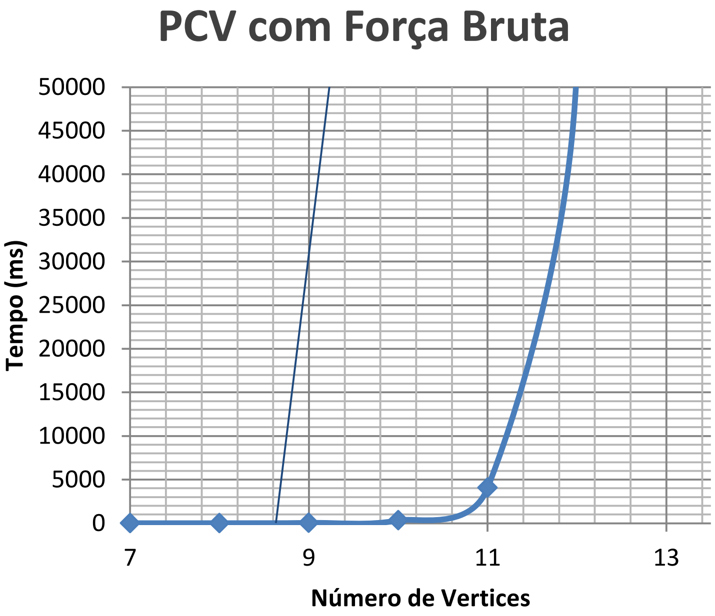
Modelagem Matemática
programação binária

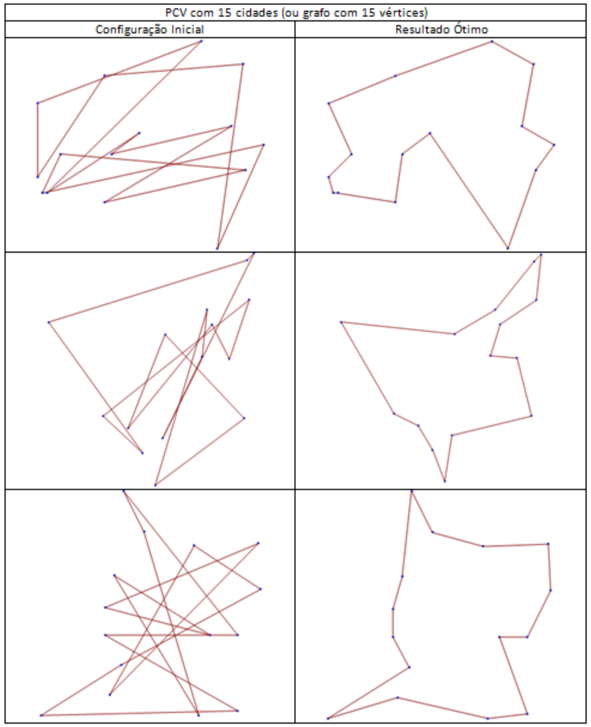
O Problema do Caixeiro Viajante (PCV)

Complexidade

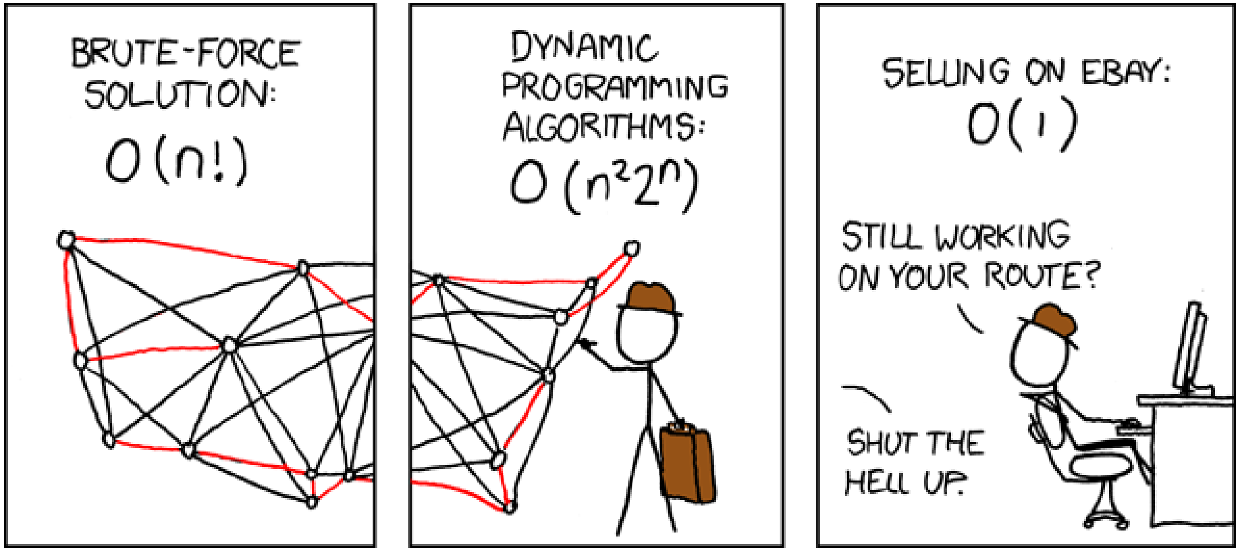
	Tempo (ms)
Número de Cidades	Força Bruta
13	743691
12	53093
11	4056
10	331
9	39
8	2
7	1

PCV com 15 Cidades		
	Força Bruta	
Solução Ótima	Tempo (ms)	Tempo (h)
359,399	165340592	45,928
317,232	165590540	45,997
368,79	165517424	45,977

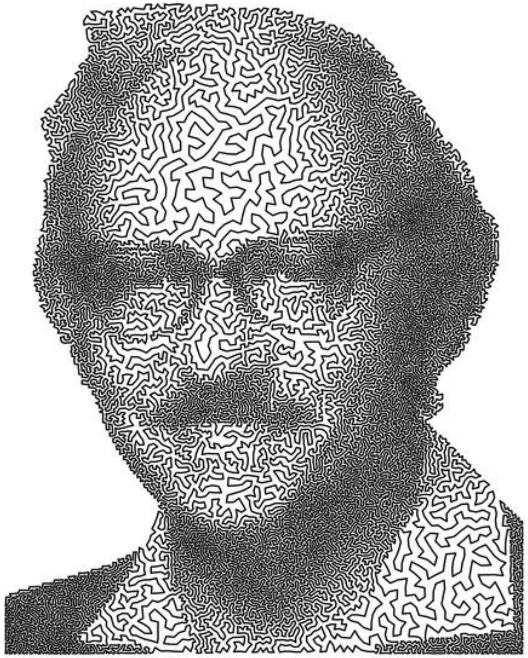




$47!/2 = 1{,}29311 \times 10^{59}$ $47! = 2{,}58623 \times 10^{59}$ $48! = 1{,}24139 \times 10^{60}$



Travelling Salesman Problem XKCD
http://xkcd.com/399/

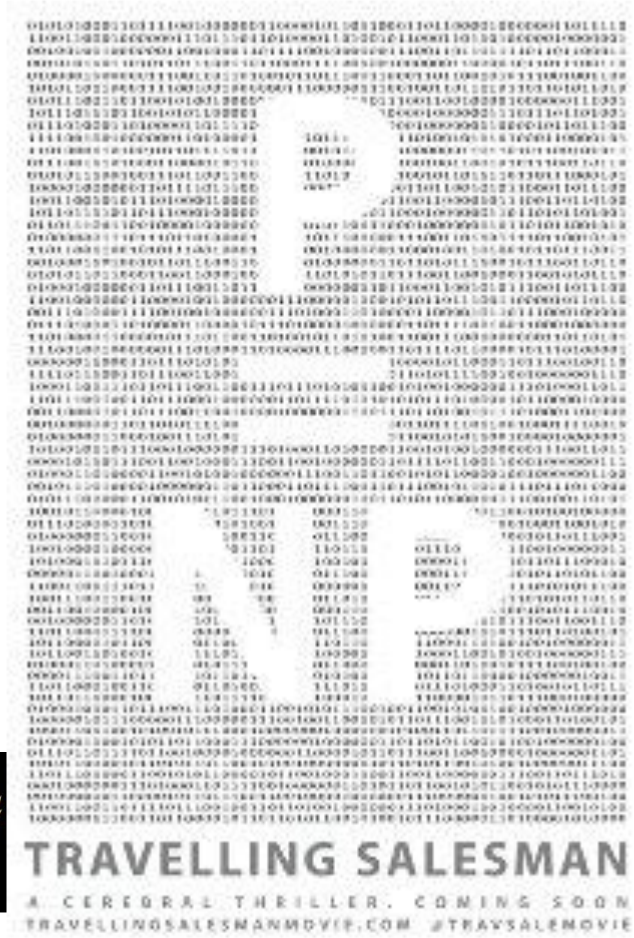


George Dantzig (25K vértices)
http://www.oberlin.edu/math/faculty/bosch/tspart-page.html

Mona Lisa (100K vértices)
http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html

Travelling Salesman is a 2012 intellectual thriller film about four mathematicians solving the **P versus NP problem**, one of the most challenging mathematical problems in history.

http://en.wikipedia.org/wiki/Travelling_Salesman_(2012_film)



Silicon Valley Film Festival
2012
WINNER
BEST FEATURE FILM
BEST ACTOR
BEST EDITOR

http://www.travellingsalesmanmovie.com/

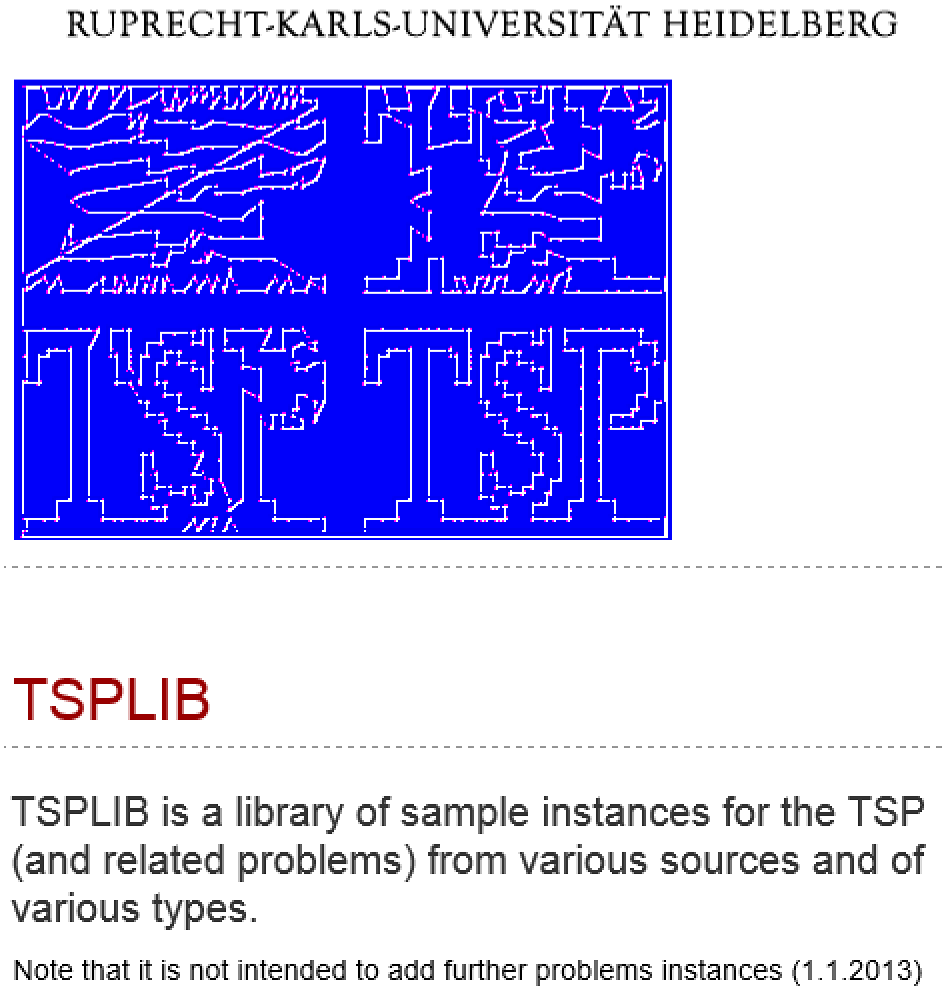
JULIAN LETHBRIDGE

Traveling Salesman, 1995
Lithograph
43 3/4 x 42 in
Edition 50

http://www.larissagoldston.com/artists/julianlethbridge/index.aspx

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

TSPLIB



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

TSPLIB

TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types.

Note that it is not intended to add further problems instances (1.1.2013)

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Resolução exata do PCV com 24.978 cidades

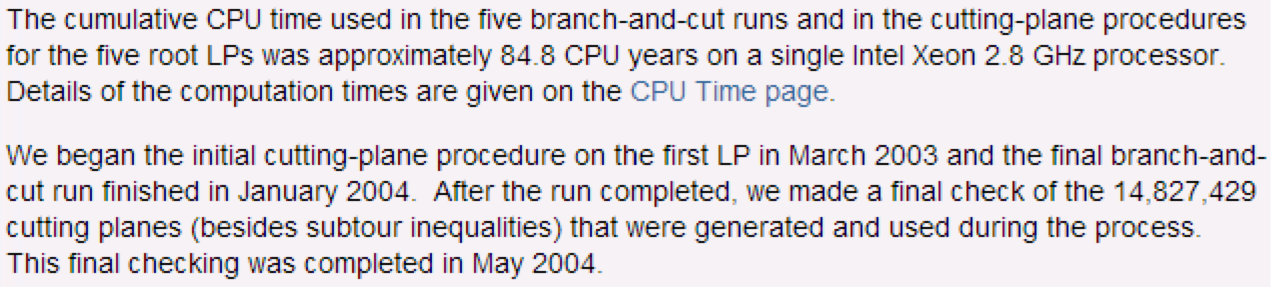
Optimal Tour of Sweden

In May 2004, the traveling salesman problem of visiting all 24,978 cities in Sweden was solved: a tour of length 855,597 TSPLIB units (approximately 72,500 kilometers) was found and it was proven that no shorter tour exists. At the time of the computation, this was the largest solved TSP instance, surpassing the previous record of 15,112 cities through Germany set in April 2001. The current record an 85,900-city tour that arose in a chip-design application.

- The 24,978 cities
- Pictures of the Sweden Tour
- How was the tour found?
- Details of the computation
- Data sets for Sweden TSP and tour

Research Team

- David Applegate, AT&T Labs - Research
- Robert Bixby, ILOG and Rice University
- Vašek Chvátal, Rutgers University
- William Cook, University of Waterloo
- Keld Helsgaun, Roskilde University



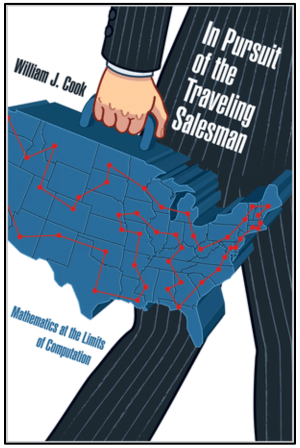
The cumulative CPU time used in the five branch-and-cut runs and in the cutting-plane procedures for the five root LPs was approximately 84.8 CPU years on a single Intel Xeon 2.8 GHz processor. Details of the computation times are given on the CPU Time page.

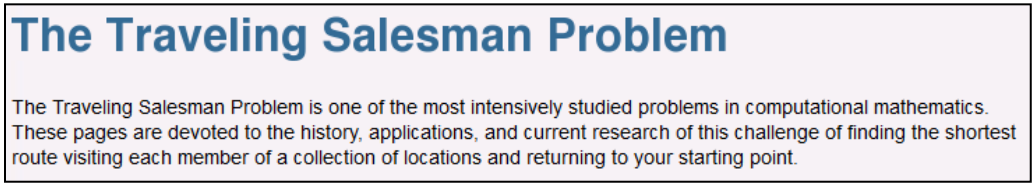
We began the initial cutting-plane procedure on the first LP in March 2003 and the final branch-and-cut run finished in January 2004. After the run completed, we made a final check of the 14,827,429 cutting planes (besides subtour inequalities) that were generated and used during the process. This final checking was completed in May 2004.

http://www.math.uwaterloo.ca/tsp/sweden/index.html

Quer saber mais sobre PCV?

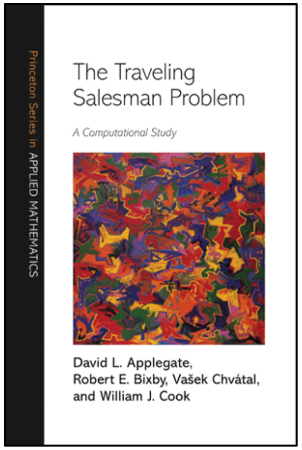
- http://www.math.uwaterloo.ca/tsp/



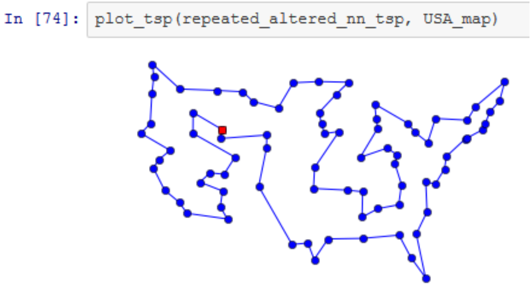


The Traveling Salesman Problem

The Traveling Salesman Problem is one of the most intensively studied problems in computational mathematics. These pages are devoted to the history, applications, and current research of this challenge of finding the shortest route visiting each member of a collection of locations and returning to your starting point.

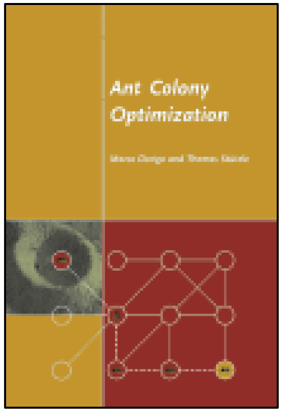


- http://nbviewer.ipython.org/url/norvig.com/ipython/TSPv3.ipynb



```
In [74]: plot_tsp(repeated_altered_nn_tsp, USA_map)
```

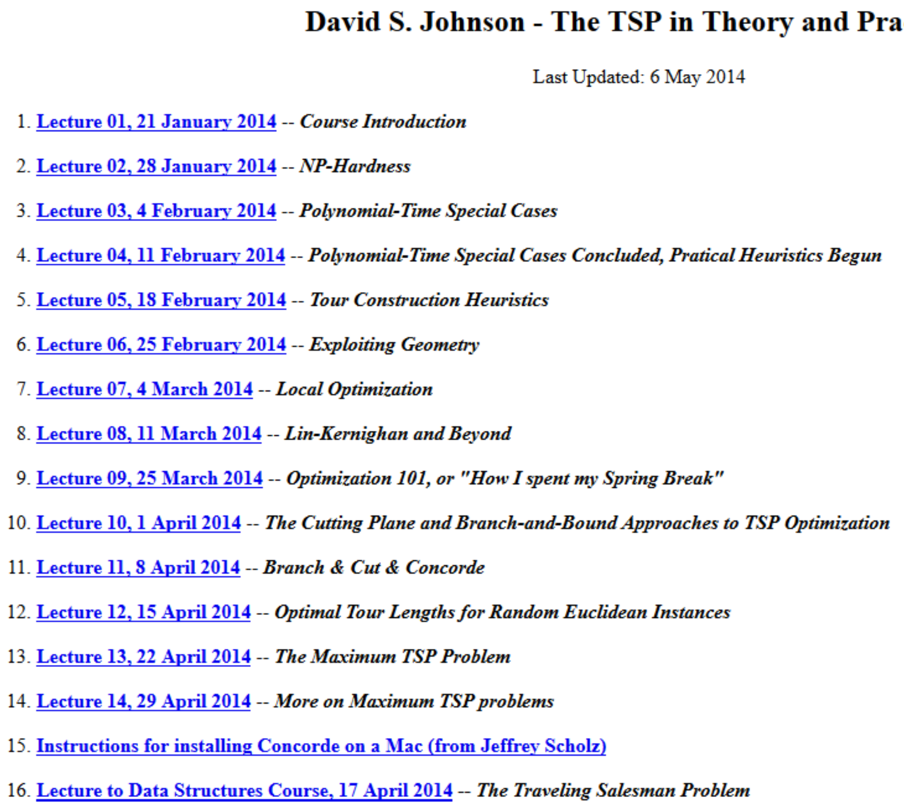
- http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html
  - http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html



Quer saber ainda mais sobre PCV?

- Recomendo fortemente!

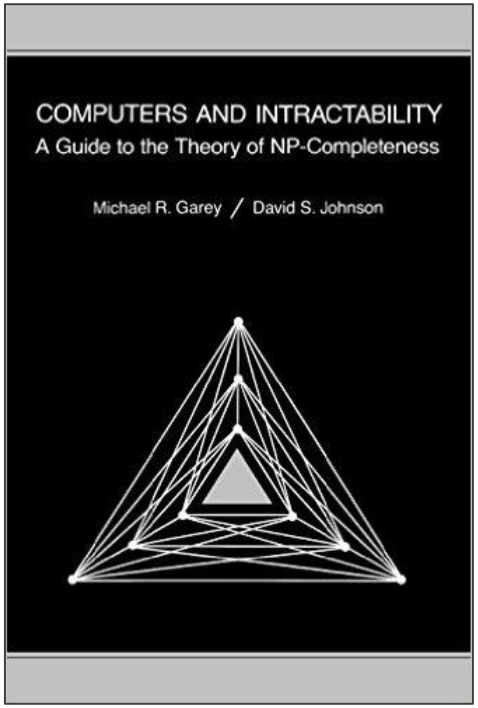
http://davidsjohnson.net/TSPcourse.html



David S. Johnson - The TSP in Theory and Practice

Last Updated: 6 May 2014

1. Lecture 01, 21 January 2014 -- *Course Introduction*
2. Lecture 02, 28 January 2014 -- *NP-Hardness*
3. Lecture 03, 4 February 2014 -- *Polynomial-Time Special Cases*
4. Lecture 04, 11 February 2014 -- *Polynomial-Time Special Cases Concluded, Pratical Heuristics Begun*
5. Lecture 05, 18 February 2014 -- *Tour Construction Heuristics*
6. Lecture 06, 25 February 2014 -- *Exploiting Geometry*
7. Lecture 07, 4 March 2014 -- *Local Optimization*
8. Lecture 08, 11 March 2014 -- *Lin-Kernighan and Beyond*
9. Lecture 09, 25 March 2014 -- *Optimization 101, or "How I spent my Spring Break"*
10. Lecture 10, 1 April 2014 -- *The Cutting Plane and Branch-and-Bound Approaches to TSP Optimization*
11. Lecture 11, 8 April 2014 -- *Branch & Cut & Concorde*
12. Lecture 12, 15 April 2014 -- *Optimal Tour Lengths for Random Euclidean Instances*
13. Lecture 13, 22 April 2014 -- *The Maximum TSP Problem*
14. Lecture 14, 29 April 2014 -- *More on Maximum TSP problems*
15. Instructions for installing Concorde on a Mac (from Jeffrey Scholz)
16. Lecture to Data Structures Course, 17 April 2014 -- *The Traveling Salesman Problem*

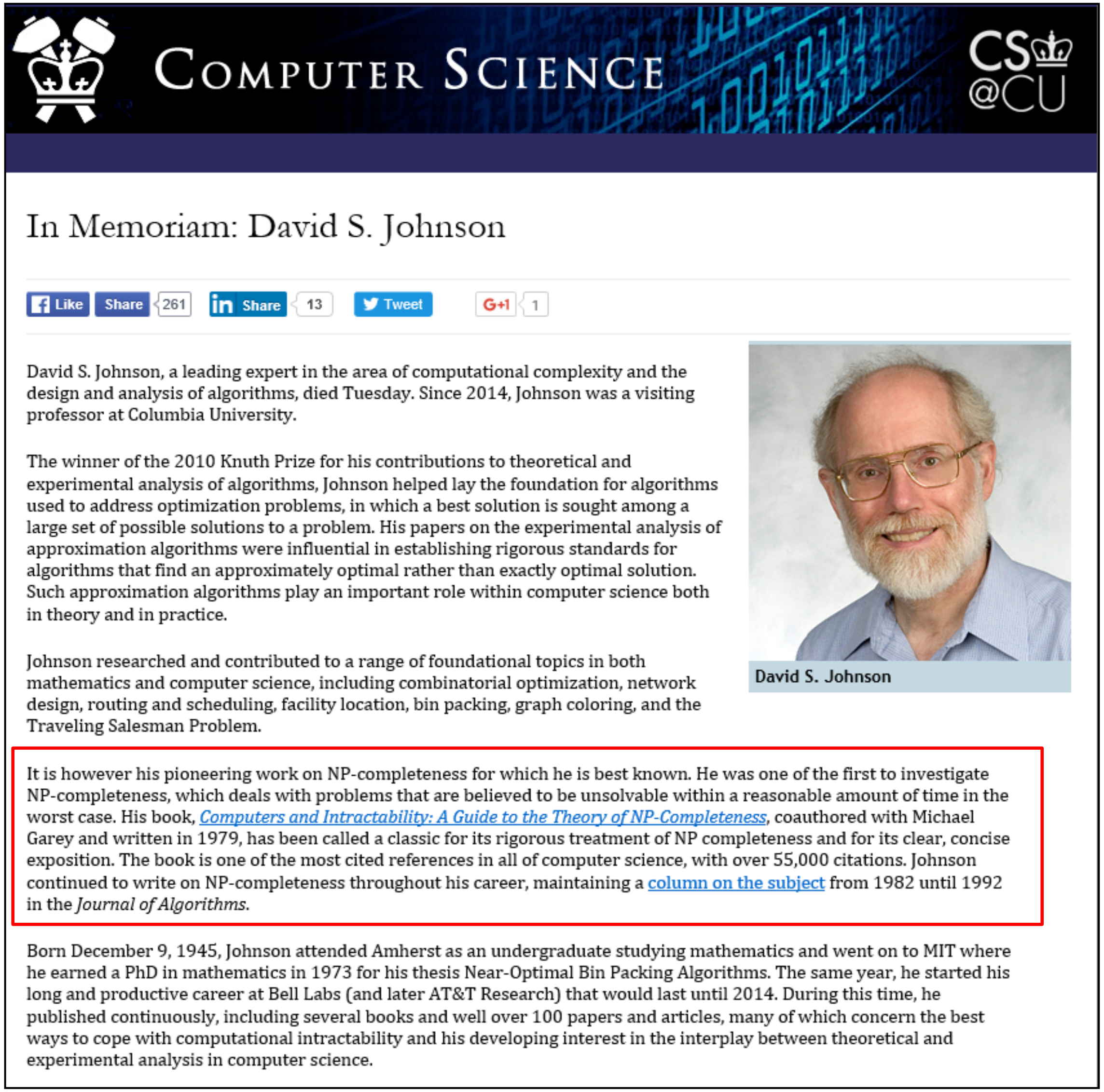


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson

http://davidsjohnson.net

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

In Memoriam: David S. Johnson



In Memoriam: David S. Johnson

David S. Johnson, a leading expert in the area of computational complexity and the design and analysis of algorithms, died Tuesday. Since 2014, Johnson was a visiting professor at Columbia University.

The winner of the 2010 Knuth Prize for his contributions to theoretical and experimental analysis of algorithms, Johnson helped lay the foundation for algorithms used to address optimization problems, in which a best solution is sought among a large set of possible solutions to a problem. His papers on the experimental analysis of approximation algorithms were influential in establishing rigorous standards for algorithms that find an approximately optimal rather than exactly optimal solution. Such approximation algorithms play an important role within computer science both in theory and in practice.

Johnson researched and contributed to a range of foundational topics in both mathematics and computer science, including combinatorial optimization, network design, routing and scheduling, facility location, bin packing, graph coloring, and the Traveling Salesman Problem.

David S. Johnson

It is however his pioneering work on NP-completeness for which he is best known. He was one of the first to investigate NP-completeness, which deals with problems that are believed to be unsolvable within a reasonable amount of time in the worst case. His book, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, coauthored with Michael Garey and written in 1979, has been called a classic for its rigorous treatment of NP completeness and for its clear, concise exposition. The book is one of the most cited references in all of computer science, with over 55,000 citations. Johnson continued to write on NP-completeness throughout his career, maintaining a column on the subject from 1982 until 1992 in the *Journal of Algorithms*.

Born December 9, 1945, Johnson attended Amherst as an undergraduate studying mathematics and went on to MIT where he earned a PhD in mathematics in 1973 for his thesis Near-Optimal Bin Packing Algorithms. The same year, he started his long and productive career at Bell Labs (and later AT&T Research) that would last until 2014. During this time, he published continuously, including several books and well over 100 papers and articles, many of which concern the best ways to cope with computational intractability and his developing interest in the interplay between theoretical and experimental analysis in computer science.

http://www.cs.columbia.edu/2016/david-johnson-in-memoriam/

Minha Abordagem ao Problema

- Inteligência Artificial + Computação Paralela + Modelo Composicional (Teoria das Categorias e Programação Funcional) + Problema Complexo + Pesquisa Operacional

$$Computação\ Paralela \circ IA\ (problema) = \ Metaheurística\ Paralela$$

$$f\ \circ g\ (problema)$$

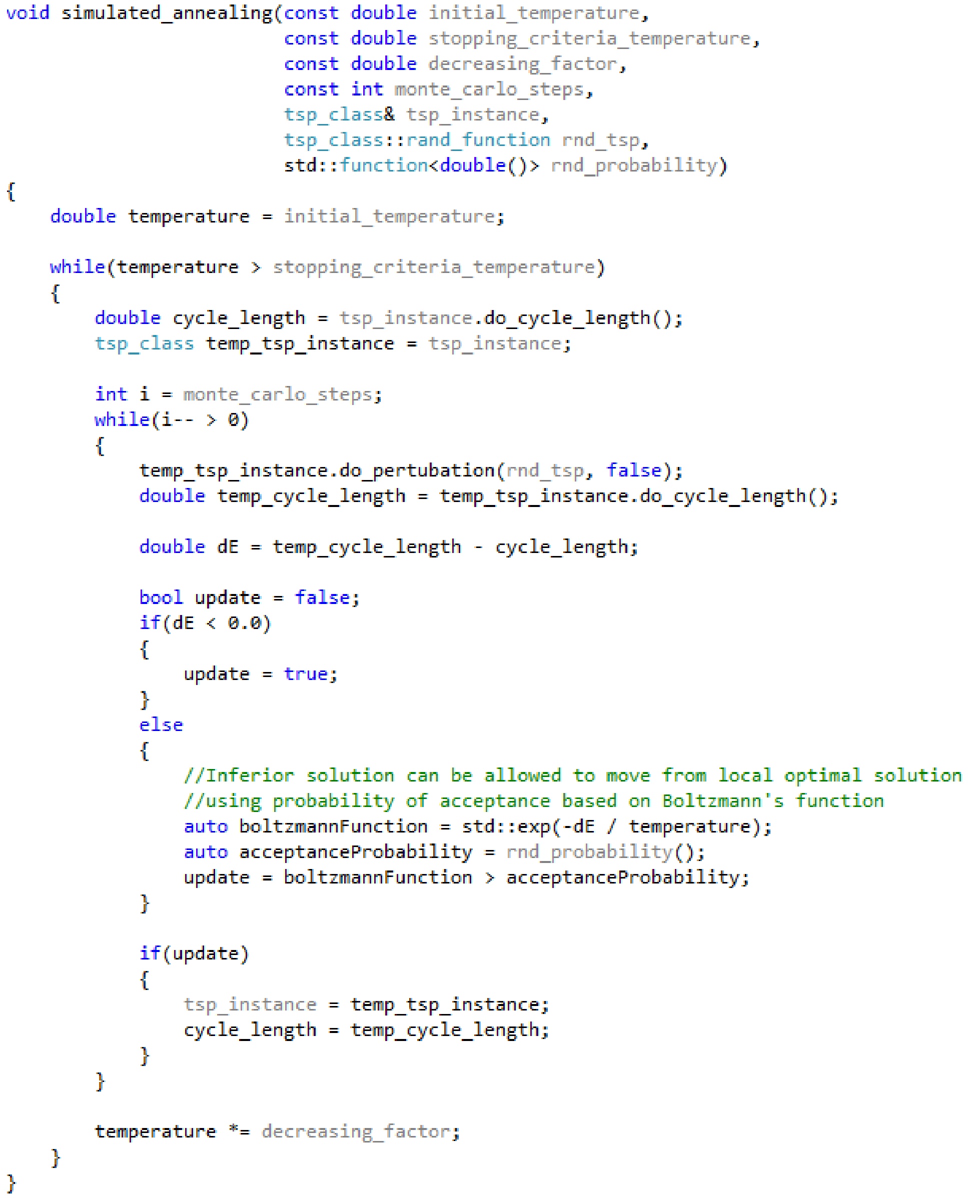
$$M(TSP) \xrightarrow{TSP \xrightarrow{f} M(TSP^{*})} M\ (TSP^{*})$$

$$f\ \equiv SA \vee AG \vee ACO \vee OBL$$

$$M(TSP) \xrightarrow{TSP \xrightarrow{SA} M(TSP)} M\ (TSP) \xrightarrow{TSP \xrightarrow{2-Opt} M(TSP)} M\ (TSP^{*})$$

Resolução de Problemas através de Busca

Simulated Annealing

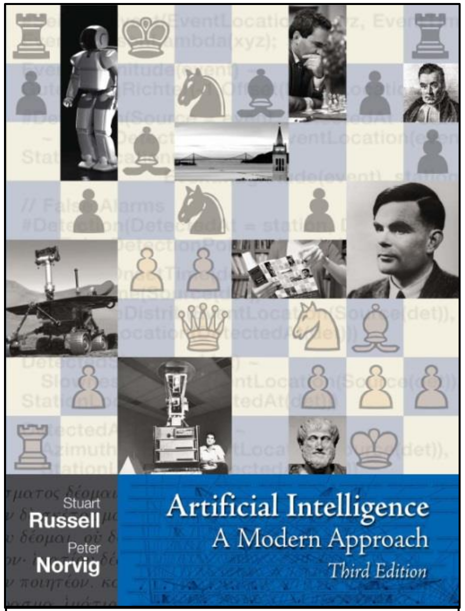


```
void simulated_annealing(const double initial_temperature,
                         const double stopping_criteria_temperature,
                         const double decreasing_factor,
                         const int monte_carlo_steps,
                         tsp_class& tsp_instance,
                         tsp_class::rand_function rnd_tsp,
                         std::function<double()> rnd_probability)
{
    double temperature = initial_temperature;

    while(temperature > stopping_criteria_temperature)
    {
        double cycle_length = tsp_instance.do_cycle_length();
        tsp_class temp_tsp_instance = tsp_instance;

        int i = monte_carlo_steps;
        while(i-- > 0)
        {
            temp_tsp_instance.do_pertubation(rnd_tsp, false);
            double temp_cycle_length = temp_tsp_instance.do_cycle_length();

            double dE = temp_cycle_length - cycle_length;

            bool update = false;
            if(dE < 0.0)
            {
                update = true;
            }
            else
            {
                //Inferior solution can be allowed to move from local optimal solution
                //using probability of acceptance based on Boltzmann's function
                auto boltzmannFunction = std::exp(-dE / temperature);
                auto acceptanceProbability = rnd_probability();
                update = boltzmannFunction > acceptanceProbability;
            }

            if(update)
            {
                tsp_instance = temp_tsp_instance;
                cycle_length = temp_cycle_length;
            }
        }

        temperature *= decreasing_factor;
    }
}
```

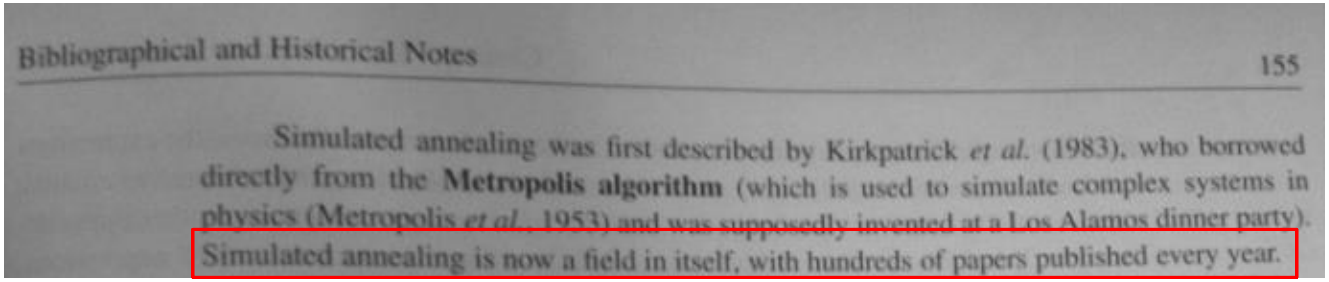
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Resolução de Problemas através de Busca

Simulated Annealing



http://aima.cs.berkeley.edu



Bibliographical and Historical Notes 155

Simulated annealing was first described by Kirkpatrick *et al.* (1983), who borrowed directly from the **Metropolis algorithm** (which is used to simulate complex systems in physics (Metropolis *et al.*, 1953) and was supposedly invented at a Los Alamos dinner party). Simulated annealing is now a field in itself, with hundreds of papers published every year.

Optimization by simulated annealing

1983

S. Kirkpatrick , C. D. Gelatt , M. P. Vecchi

http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.123.7607&rank=1

13 May 1983, Volume 220, Number 4598

SCIENCE

“The purpose of computing is insight, not numbers”

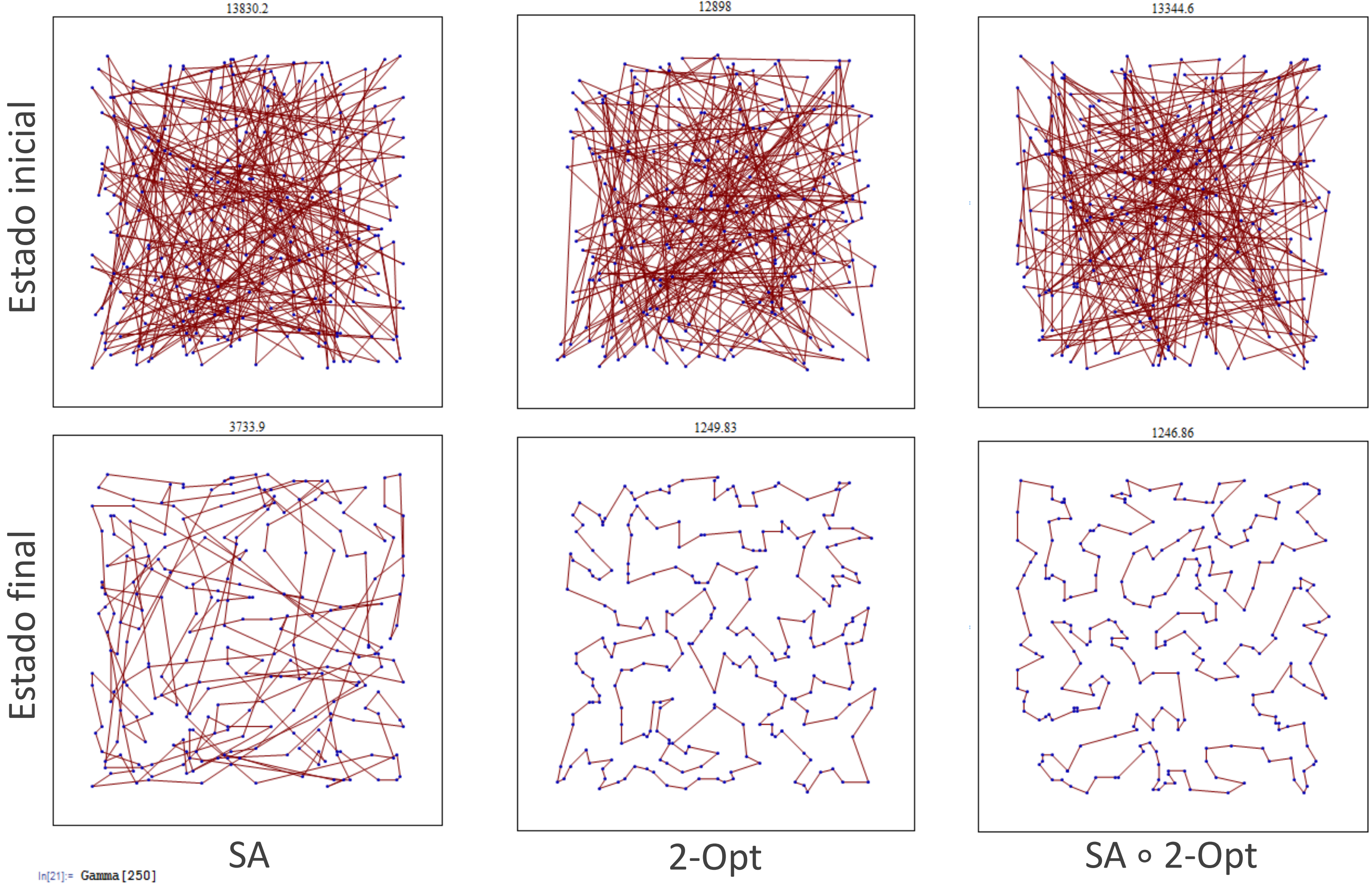
Numerical Methods for Scientists and Engineers

Richard Wesley Hamming

Turing Award 1968

Composição de soluções

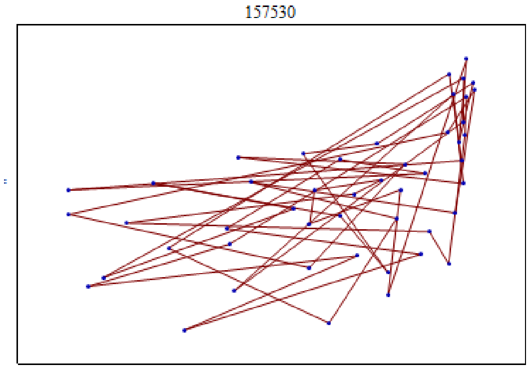
random250 (250 cidades)



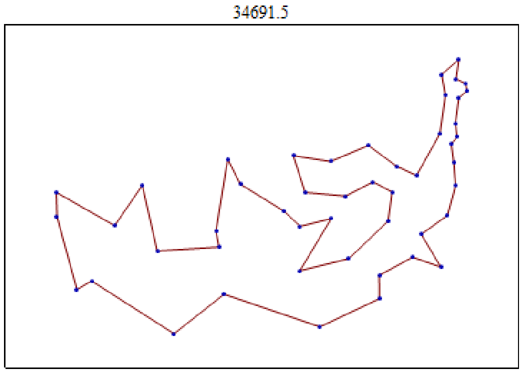
```
In[21]:= Gamma[250]

Out[21]= 12 931 425 043 636 430 929 283 258 208 097 473 883 979 374 870 695 122 666 991 769 708 451 294 990 220 444 837 955 271 661 484 112 797 803 714 029 412 757 731 706 923 726 851 235 963 291 164 121 \
          116 285 127 686 706 108 960 757 058 932 872 164 745 044 027 331 701 460 534 715 756 358 279 742 854 120 700 162 052 715 040 308 991 732 261 609 356 024 659 302 208 995 277 746 290 344 229 596 890 \
          565 019 331 928 819 396 550 887 106 602 565 107 435 228 612 515 915 110 691 807 655 963 377 509 914 810 356 691 893 020 601 132 967 149 282 632 753 928 249 914 330 639 235 395 302 195 200 000 000 \
          000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

Resultados

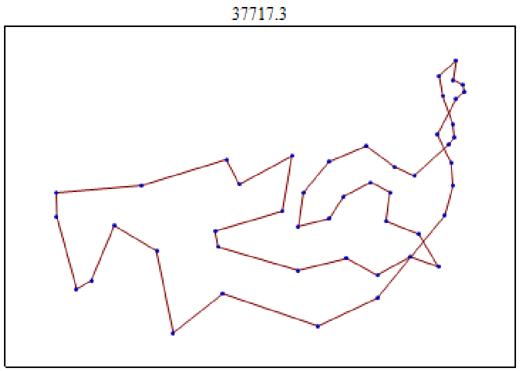
att48.tsp (48 cidades)



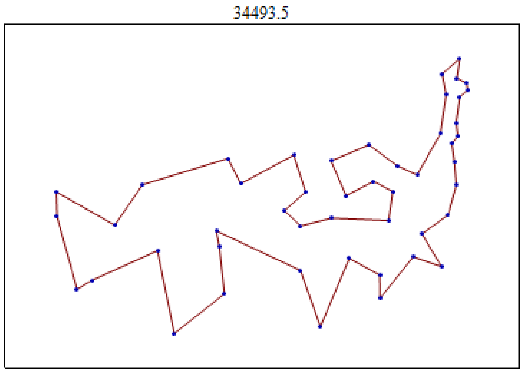
Estado Inicial



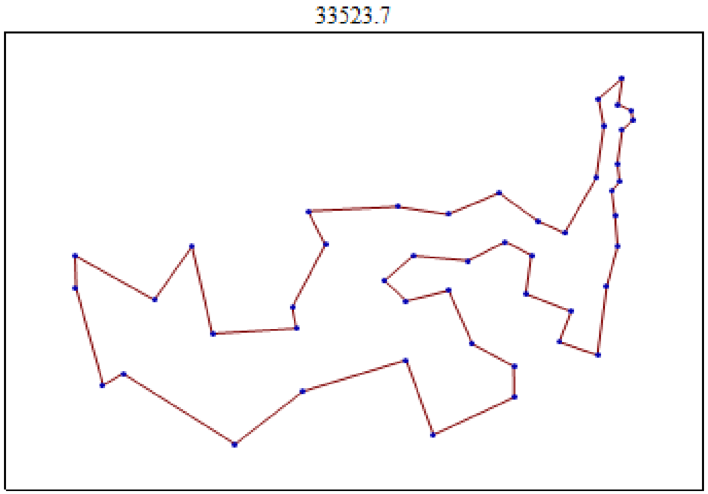
2-opt (8 tarefas)



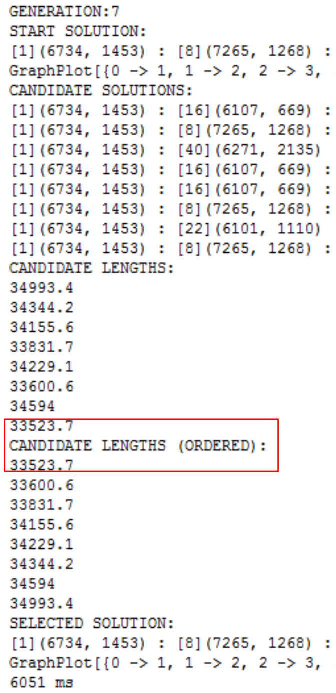
SA (8 tarefas)



SA ∘ 2-Opt (8 tarefas)
1 geração



SA ∘ 2-Opt (8 tarefas)
10 gerações – resultado ótimo
encontrado na geração 7

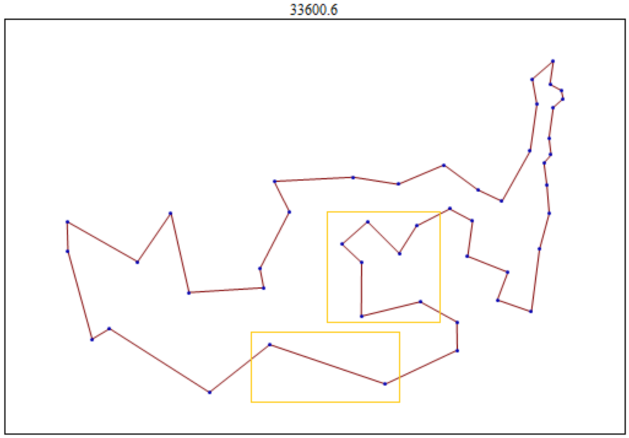


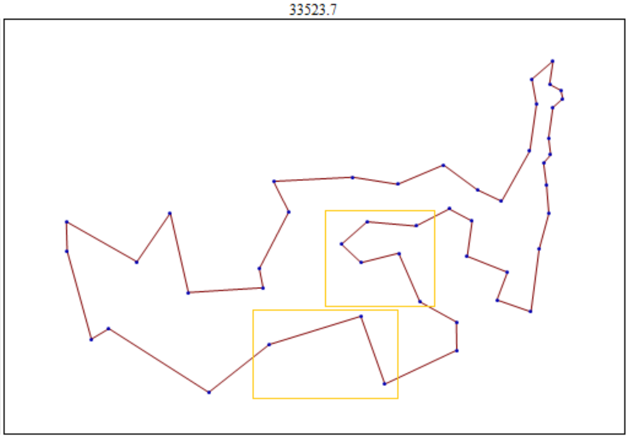
```
GENERATION:7
START SOLUTION:
[1](6734, 1453) : [8](7265, 1268) :
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3,
CANDIDATE SOLUTIONS:
[1](6734, 1453) : [16](6107, 669) :
[1](6734, 1453) : [8](7265, 1268) :
[1](6734, 1453) : [40](6271, 2135)
[1](6734, 1453) : [16](6107, 669) :
[1](6734, 1453) : [16](6107, 669) :
[1](6734, 1453) : [8](7265, 1268) :
[1](6734, 1453) : [22](6101, 1110)
[1](6734, 1453) : [8](7265, 1268) :
CANDIDATE LENGTHS:
34993.4
34344.2
34155.6
33831.7
34229.1
33600.6
34594
33523.7
CANDIDATE LENGTHS (ORDERED):
33523.7
33600.6
33831.7
34155.6
34229.1
34344.2
34594
34993.4
SELECTED SOLUTION:
[1](6734, 1453) : [8](7265, 1268) :
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3,
6051 ms
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Resultados

att48.tsp (48 cidades) – ajuste fino

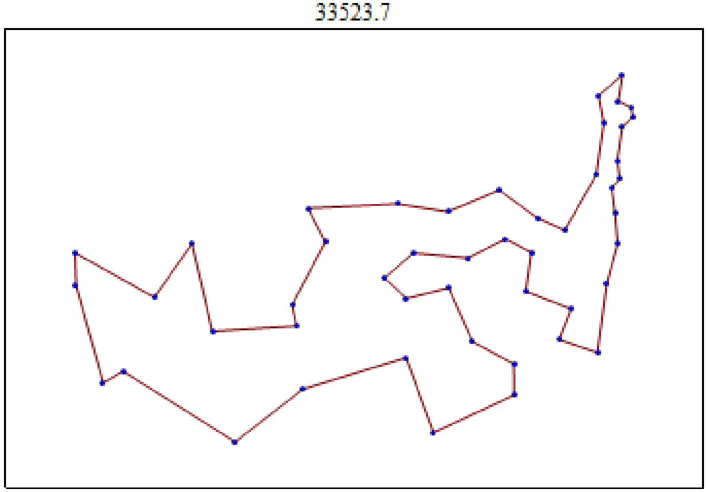




```
simulated_annealing(100.0, 0.0001, 0.999, 200, …);
simulated_annealing(1000.0, 0.00001, 0.999, 400, …);
```

```
CANDIDATE LENGTHS (ORDERED):
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
33600.6
```

18487 ms

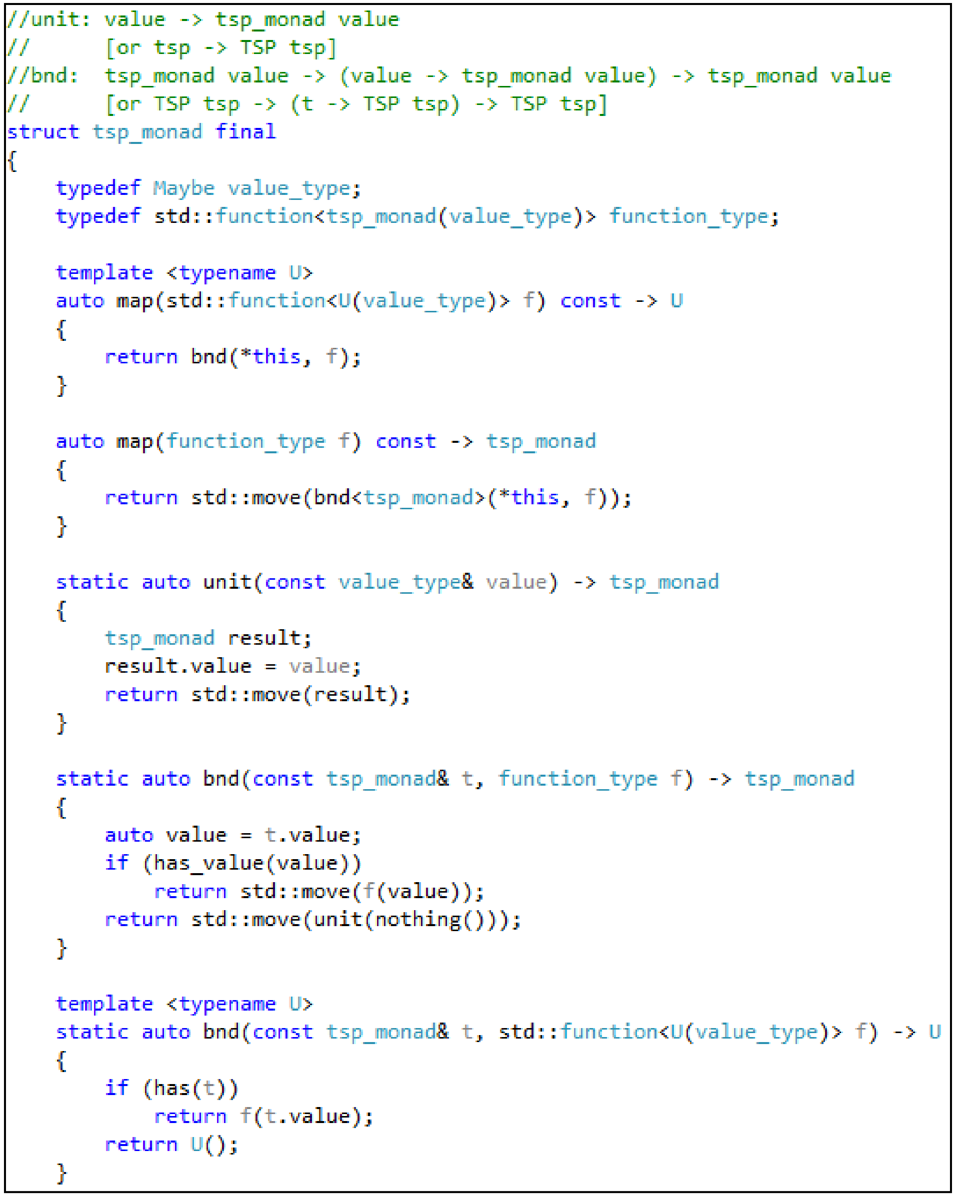


SA ∘ 2-Opt

```
CANDIDATE LENGTHS (ORDERED):
33523.7
33523.7
33600.6
33600.6
33632.1
33632.1
33753.4
33857.5
33861.8
33899.9
33956.5
34015.8
34049.4
34066.9
34076.6
34076.6
34081
```

24375 ms

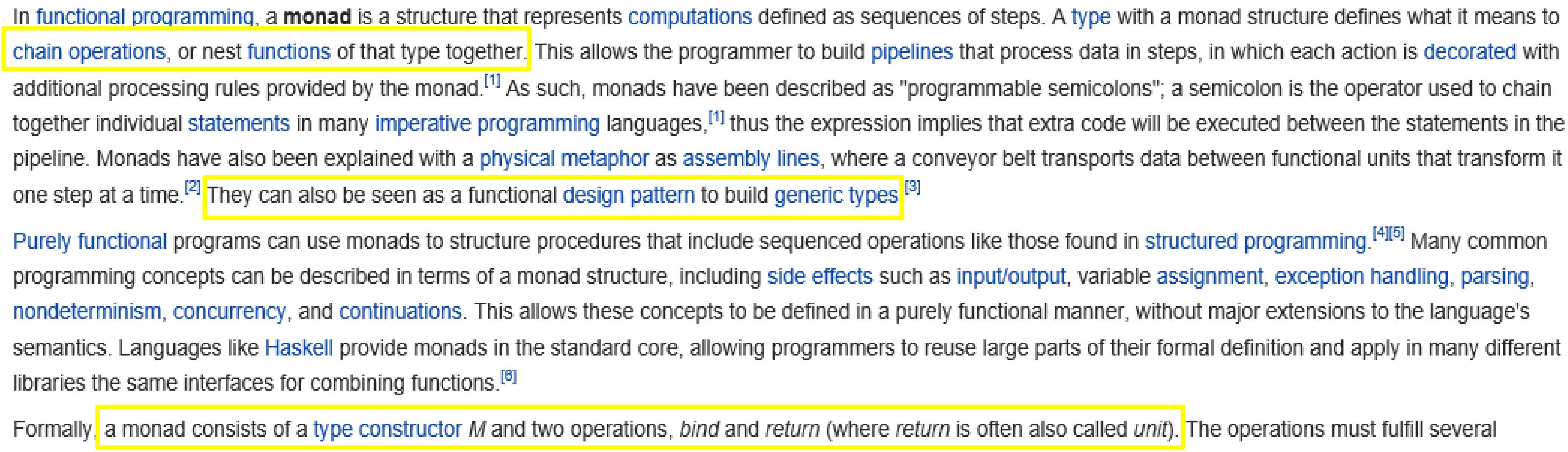
TSP Monad



```
//unit: value -> tsp_monad value
//      [or tsp -> TSP tsp]
//bnd:  tsp_monad value -> (value -> tsp_monad value) -> tsp_monad value
//      [or TSP tsp -> (t -> TSP tsp) -> TSP tsp]
struct tsp_monad final
{
    typedef Maybe value_type;
    typedef std::function<tsp_monad(value_type)> function_type;

    template <typename U>
    auto map(std::function<U(value_type)> f) const -> U
    {
        return bnd(*this, f);
    }

    auto map(function_type f) const -> tsp_monad
    {
        return std::move(bnd<tsp_monad>(*this, f));
    }

    static auto unit(const value_type& value) -> tsp_monad
    {
        tsp_monad result;
        result.value = value;
        return std::move(result);
    }

    static auto bnd(const tsp_monad& t, function_type f) -> tsp_monad
    {
        auto value = t.value;
        if (has_value(value))
            return std::move(f(value));
        return std::move(unit(nothing()));
    }

    template <typename U>
    static auto bnd(const tsp_monad& t, std::function<U(value_type)> f) -> U
    {
        if (has(t))
            return f(t.value);
        return U();
    }
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Monad



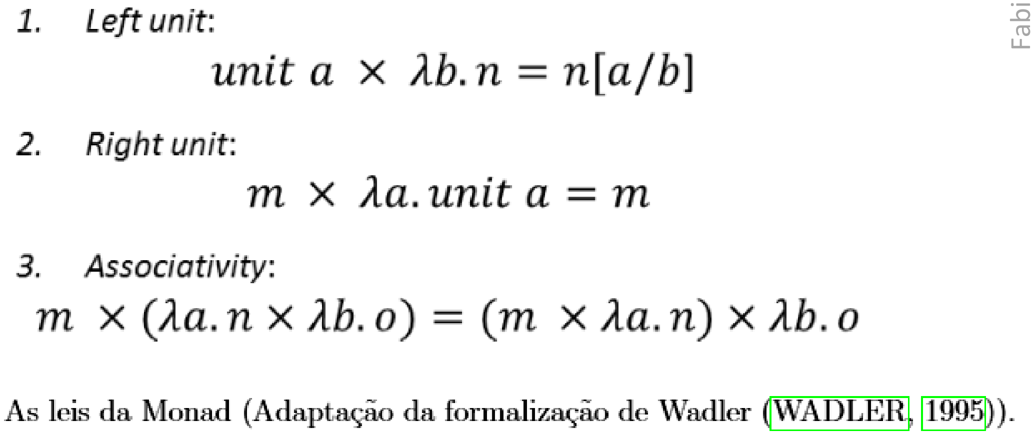
In functional programming, a **monad** is a structure that represents computations defined as sequences of steps. A type with a monad structure defines what it means to chain operations, or nest functions of that type together. This allows the programmer to build pipelines that process data in steps, in which each action is decorated with additional processing rules provided by the monad.[1] As such, monads have been described as "programmable semicolons"; a semicolon is the operator used to chain together individual statements in many imperative programming languages,[1] thus the expression implies that extra code will be executed between the statements in the pipeline. Monads have also been explained with a physical metaphor as assembly lines, where a conveyor belt transports data between functional units that transform it one step at a time.[2] They can also be seen as a functional design pattern to build generic types.[3]

Purely functional programs can use monads to structure procedures that include sequenced operations like those found in structured programming.[4][5] Many common programming concepts can be described in terms of a monad structure, including side effects such as input/output, variable assignment, exception handling, parsing, nondeterminism, concurrency, and continuations. This allows these concepts to be defined in a purely functional manner, without major extensions to the language's semantics. Languages like Haskell provide monads in the standard core, allowing programmers to reuse large parts of their formal definition and apply in many different libraries the same interfaces for combining functions.[6]

Formally, a monad consists of a type constructor *M* and two operations, *bind* and *return* (where *return* is often also called *unit*). The operations must fulfill several

Fonte: http://en.wikipedia.org/wiki/Monad_(functional_programming)

```
class Monad m where
    return :: a → m a
    (>>=)  :: m a → (a → m b) → m b
```



1. *Left unit*:

$$unit\ a \times \lambda b.n = n[a/b]$$

2. *Right unit*:

$$m \times \lambda a.unit\ a = m$$

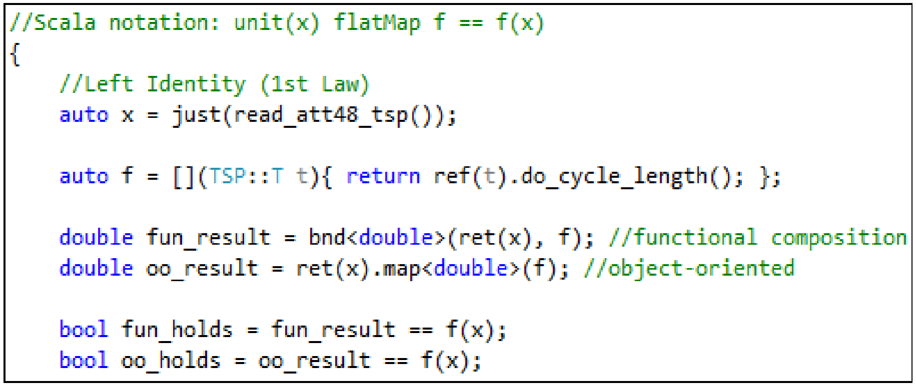
3. *Associativity*:

$$m \times (\lambda a.n \times \lambda b.o) = (m \times \lambda a.n) \times \lambda b.o$$

Figura 26: As leis da Monad (Adaptação da formalização de Wadler (WADLER, 1995)).

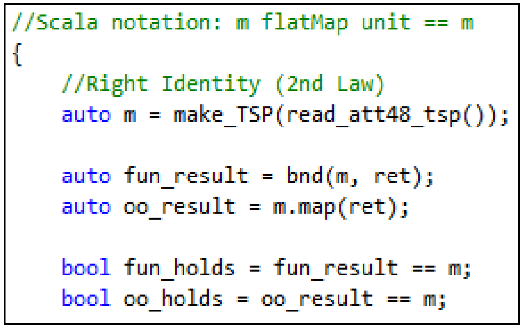
http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.9674&rank=1

TSP Monad
Leis



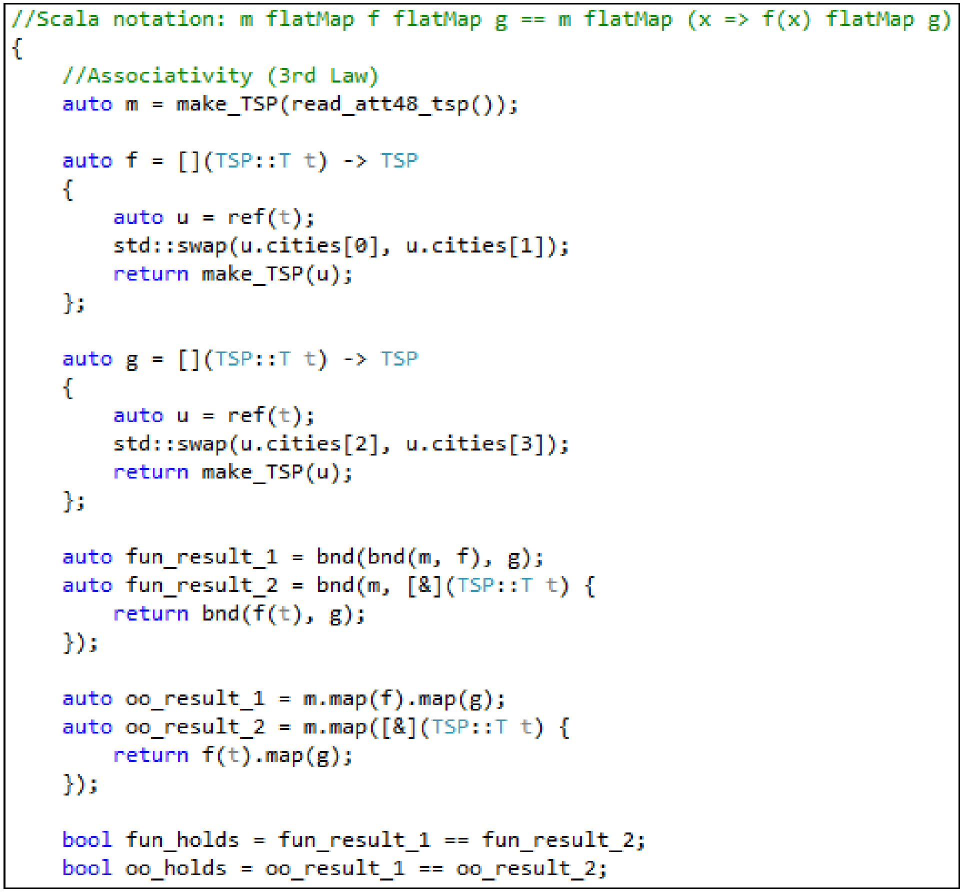
```
//Scala notation: unit(x) flatMap f == f(x)
{
    //Left Identity (1st Law)
    auto x = just(read_att48_tsp());

    auto f = [](TSP::T t){ return ref(t).do_cycle_length(); };

    double fun_result = bnd<double>(ret(x), f); //functional composition
    double oo_result = ret(x).map<double>(f); //object-oriented

    bool fun_holds = fun_result == f(x);
    bool oo_holds = oo_result == f(x);
```

1ª Lei



```
//Scala notation: m flatMap unit == m
{
    //Right Identity (2nd Law)
    auto m = make_TSP(read_att48_tsp());

    auto fun_result = bnd(m, ret);
    auto oo_result = m.map(ret);

    bool fun_holds = fun_result == m;
    bool oo_holds = oo_result == m;
```

2ª Lei



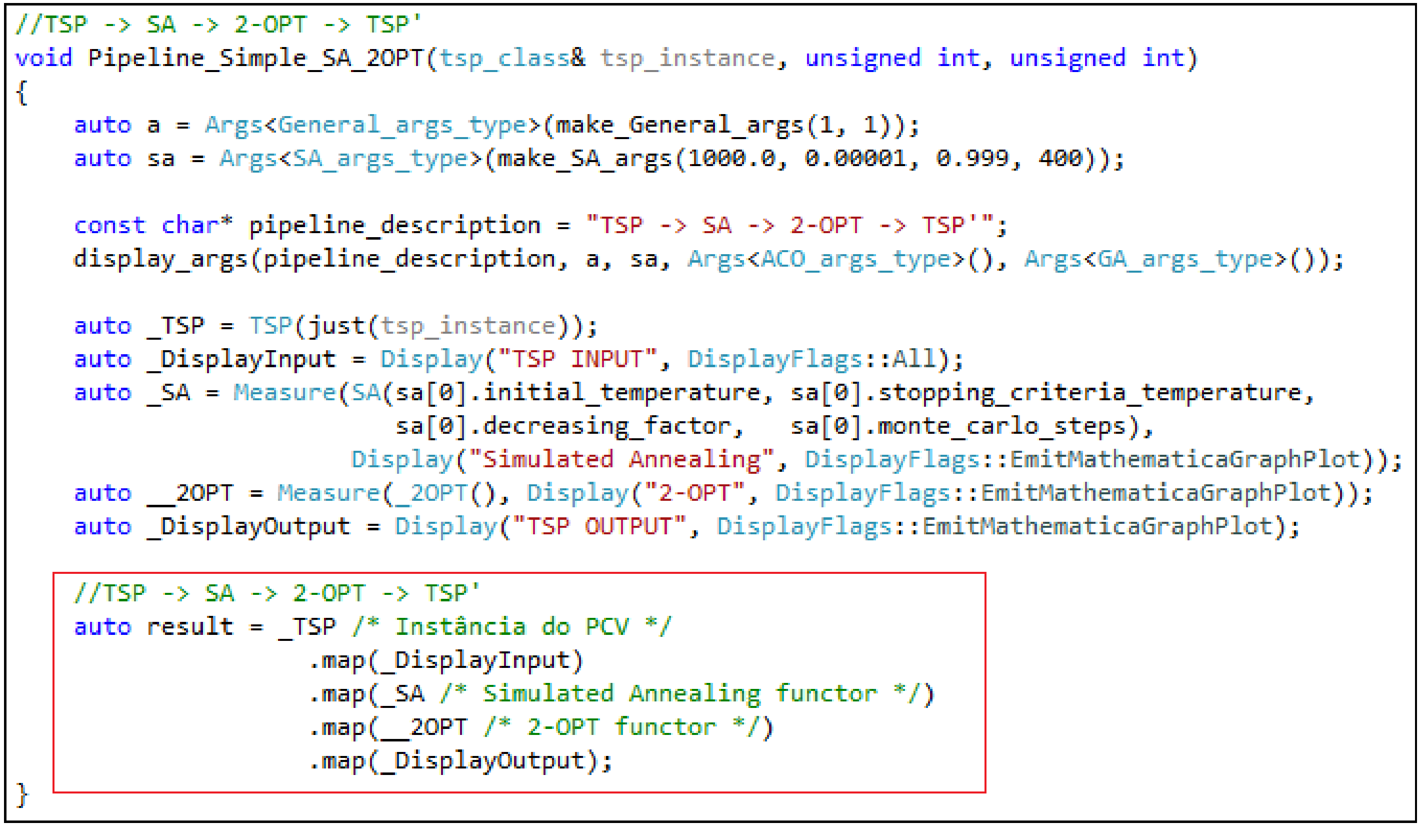
```
//Scala notation: m flatMap f flatMap g == m flatMap (x => f(x) flatMap g)
{
    //Associativity (3rd Law)
    auto m = make_TSP(read_att48_tsp());

    auto f = [](TSP::T t) -> TSP
    {
        auto u = ref(t);
        std::swap(u.cities[0], u.cities[1]);
        return make_TSP(u);
    };

    auto g = [](TSP::T t) -> TSP
    {
        auto u = ref(t);
        std::swap(u.cities[2], u.cities[3]);
        return make_TSP(u);
    };

    auto fun_result_1 = bnd(bnd(m, f), g);
    auto fun_result_2 = bnd(m, [&](TSP::T t) {
        return bnd(f(t), g);
    });

    auto oo_result_1 = m.map(f).map(g);
    auto oo_result_2 = m.map([&](TSP::T t) {
        return f(t).map(g);
    });

    bool fun_holds = fun_result_1 == fun_result_2;
    bool oo_holds = oo_result_1 == oo_result_2;
```

3ª Lei

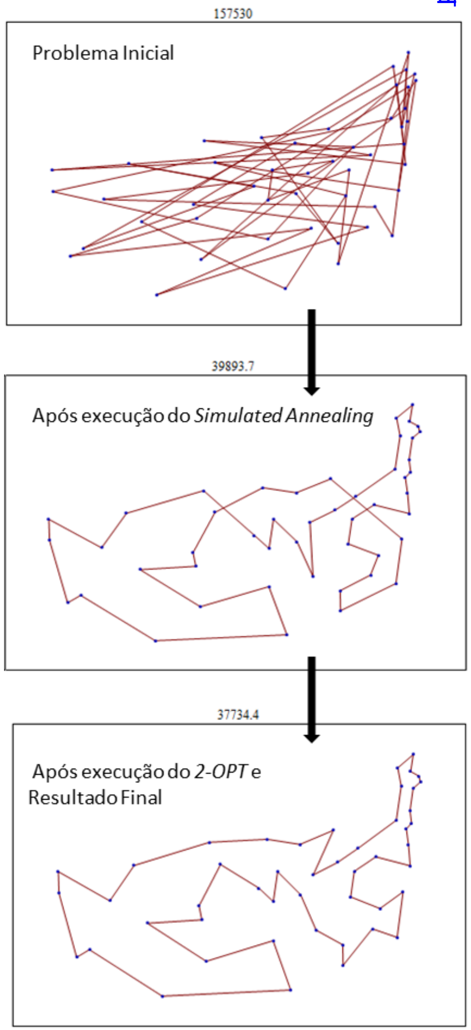
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Pipeline com *Simulated Annealing* e *2-OPT*

TSP -> SA -> 2-OPT -> TSP’

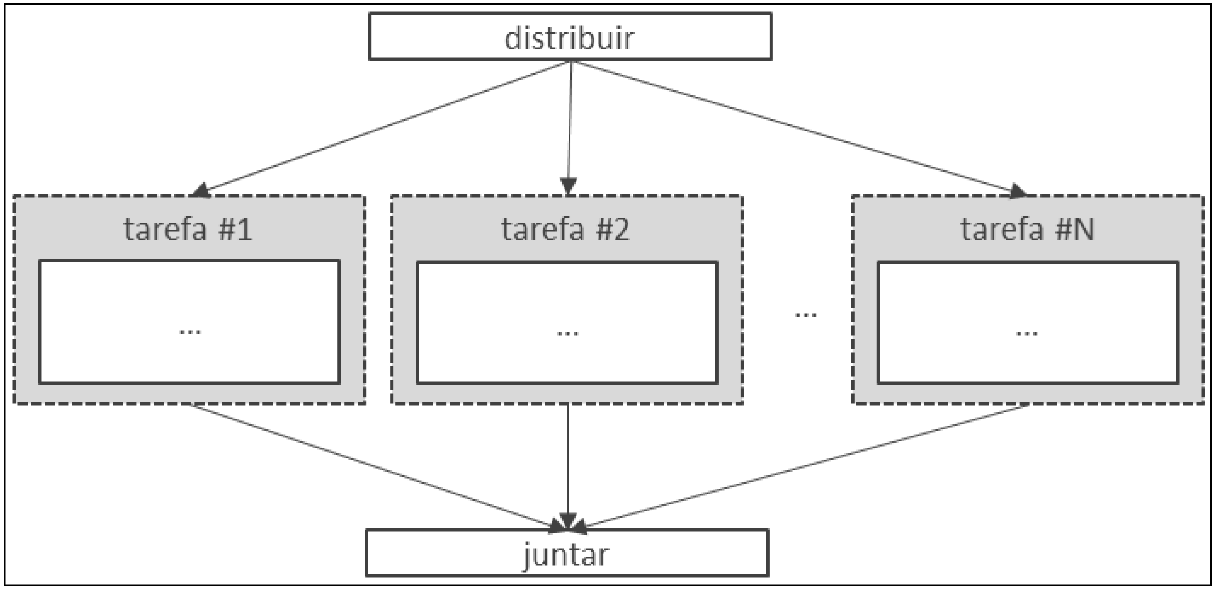


```
//TSP -> SA -> 2-OPT -> TSP'
void Pipeline_Simple_SA_2OPT(tsp_class& tsp_instance, unsigned int, unsigned int)
{
    auto a = Args<General_args_type>(make_General_args(1, 1));
    auto sa = Args<SA_args_type>(make_SA_args(1000.0, 0.00001, 0.999, 400));

    const char* pipeline_description = "TSP -> SA -> 2-OPT -> TSP'";
    display_args(pipeline_description, a, sa, Args<ACO_args_type>(), Args<GA_args_type>());

    auto _TSP = TSP(just(tsp_instance));
    auto _DisplayInput = Display("TSP INPUT", DisplayFlags::All);
    auto _SA = Measure(SA(sa[0].initial_temperature, sa[0].stopping_criteria_temperature,
                          sa[0].decreasing_factor,   sa[0].monte_carlo_steps),
                       Display("Simulated Annealing", DisplayFlags::EmitMathematicaGraphPlot));
    auto __2OPT = Measure(_2OPT(), Display("2-OPT", DisplayFlags::EmitMathematicaGraphPlot));
    auto _DisplayOutput = Display("TSP OUTPUT", DisplayFlags::EmitMathematicaGraphPlot);

    //TSP -> SA -> 2-OPT -> TSP'
    auto result = _TSP /* Instância do PCV */
                  .map(_DisplayInput)
                  .map(_SA /* Simulated Annealing functor */)
                  .map(__2OPT /* 2-OPT functor */)
                  .map(_DisplayOutput);
}
```



Onde entrou o Paralelismo?

- *Fork/Join pattern*



- Paralelismo e Composição

$$TSP \rightarrow NN \rightarrow Generations(g, ForkJoin(n, SA \rightarrow 2 - OPT)) \rightarrow TSP^{*}$$

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Considerações sobre o Modelo Composicional

$$TSP \rightarrow NN \rightarrow Generations(g, ForkJoin(n, \mathbf{Circular}(ACO, SA \rightarrow 2 - OPT, GA)) \rightarrow k - OPT \rightarrow TSP^{*}$$

$$f_1 : TSP \rightarrow Generations(g', ForkJoin(n', GA)) \rightarrow TSP^{*}$$

$$f_2 : TSP \rightarrow Generations(g'', ForkJoin(n'', SA)) \rightarrow TSP^{*}$$

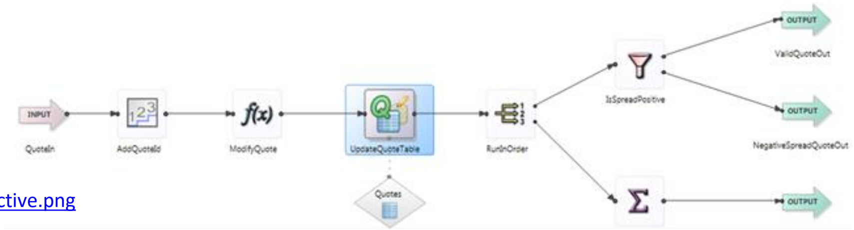
$$f_3 : TSP \rightarrow Generations(g''', ForkJoin(n''', ACO)) \rightarrow TSP^{*}$$

$$TSP \rightarrow NN \rightarrow Generations(g'''', ForkJoin(n'''', Circular(f_1, f_2, f_3)) \rightarrow 2 - OPT \rightarrow TSP^{*}$$

$$TSP \rightarrow NN \rightarrow Generations(g, ForkJoin(n, Circular(ACO_{GPU}, GA_{GPU}, 3 - OPT_{GPU})) \rightarrow TSP^{*}$$

$$TSP \rightarrow NN \rightarrow Generations(g, ForkJoin(n, AdaptiveExecution(ACO_{GPU}, GA_{Cloud}, \dots)) \rightarrow TSP^{*}$$

Empresas investem em produtos que é baseado neste conceito de composição



http://www.streambase.com/wp-content/gallery/streambase-screen-shots/streambase_studio_authoring_perspective.png

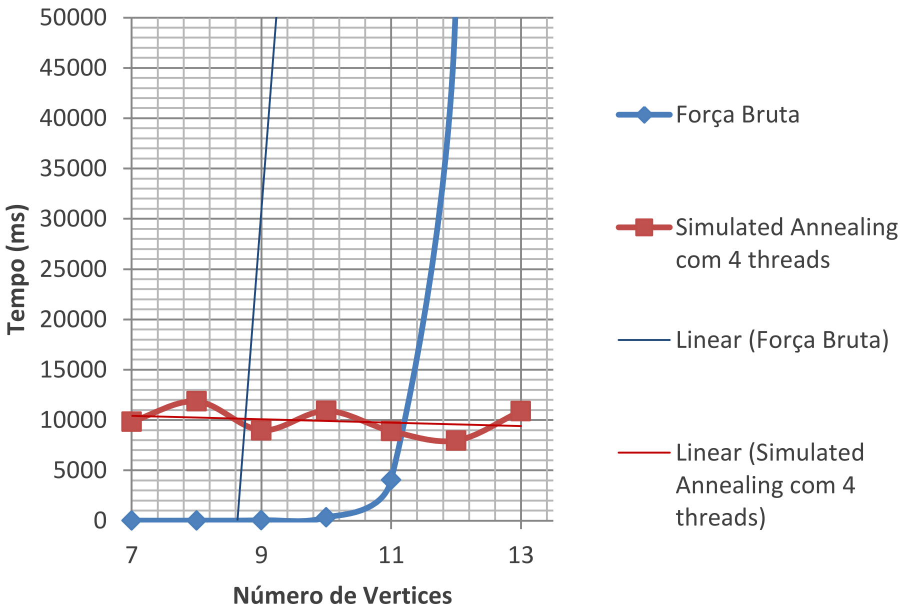
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

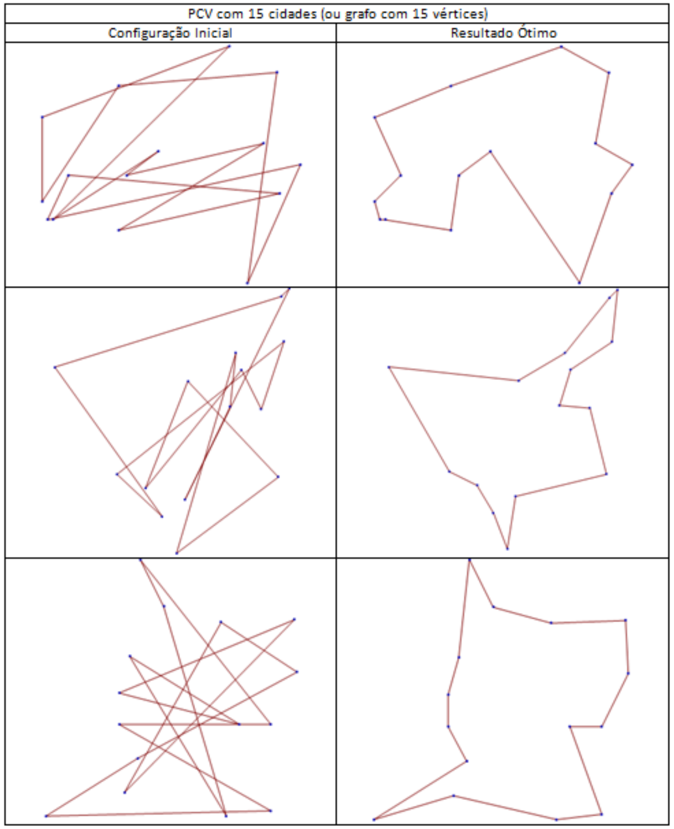
Resultados

Revisitando a complexidade

	Tempo (ms)	
Número de Vertices	Força Bruta	Simulated Annealing com 4 threads
13	743691	10885
12	53093	7964
11	4056	8901
10	331	10908
9	39	8979
8	2	11852
7	1	9843

PCV com 15 Cidades		
	SA	
Solução Ótima	Tempo (ms)	Tempo (s)
359,399	9749	9,749
317,232	13735	13,735
368,79	13735	13,735





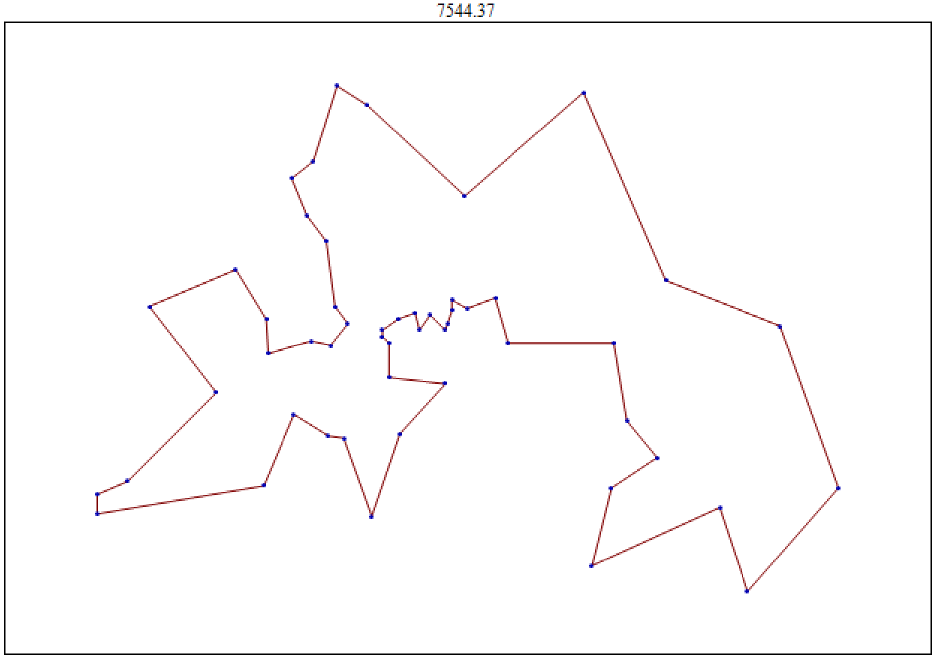
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

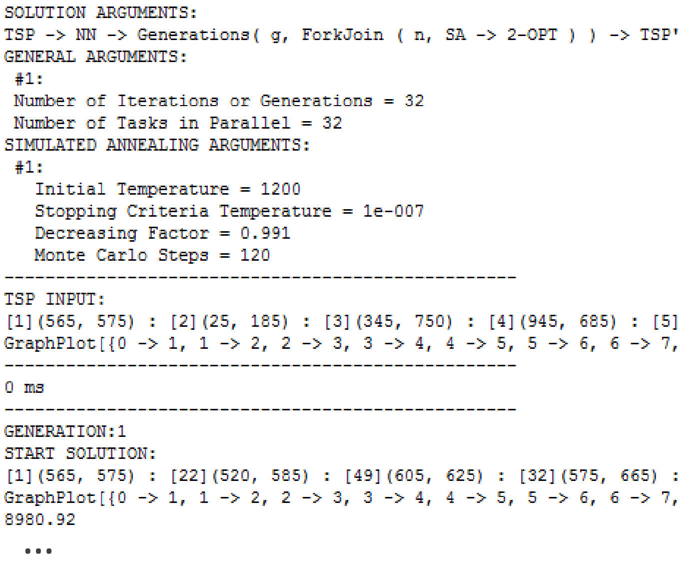
Resultados

Instância da TSPLIB berlin52.tsp com resultado ótimo

```
In[4]:= GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7, 7 -> 8, 8 -> 9, 9 -> 10, 10 -> 11, 11 -> 12, 12 -> 13, 13 -> 14, 14 -> 15,
   15 -> 16, 16 -> 17, 17 -> 18, 18 -> 19, 19 -> 20, 20 -> 21, 21 -> 22, 22 -> 23, 23 -> 24, 24 -> 25, 25 -> 26, 26 -> 27, 27 -> 28, 28 -> 29,
   29 -> 30, 30 -> 31, 31 -> 32, 32 -> 33, 33 -> 34, 34 -> 35, 35 -> 36, 36 -> 37, 37 -> 38, 38 -> 39, 39 -> 40, 40 -> 41, 41 -> 42, 42 -> 43,
   43 -> 44, 44 -> 45, 45 -> 46, 46 -> 47, 47 -> 48, 48 -> 49, 49 -> 50, 50 -> 51, 51 -> 0}, PlotLabel -> "7544.37", Frame -> True,
  VertexLabeling -> False,
  VertexCoordinateRules -> {0 -> {565, 575}, 1 -> {605, 625}, 2 -> {575, 665}, 3 -> {555, 815}, 4 -> {510, 875}, 5 -> {475, 960}, 6 -> {525, 1000},
    7 -> {580, 1175}, 8 -> {650, 1130}, 9 -> {875, 920}, 10 -> {1150, 1160}, 11 -> {1340, 725}, 12 -> {1605, 620}, 13 -> {1740, 245},
    14 -> {1530, 5}, 15 -> {1465, 200}, 16 -> {1170, 65}, 17 -> {1215, 245}, 18 -> {1320, 315}, 19 -> {1250, 400}, 20 -> {1220, 580},
    21 -> {975, 580}, 22 -> {945, 685}, 23 -> {880, 660}, 24 -> {845, 680}, 25 -> {845, 655}, 26 -> {835, 625}, 27 -> {830, 610}, 28 -> {795, 645},
    29 -> {770, 610}, 30 -> {760, 650}, 31 -> {720, 635}, 32 -> {685, 610}, 33 -> {685, 595}, 34 -> {700, 580}, 35 -> {700, 500}, 36 -> {830, 485},
    37 -> {725, 370}, 38 -> {660, 180}, 39 -> {595, 360}, 40 -> {560, 365}, 41 -> {480, 415}, 42 -> {410, 250}, 43 -> {25, 185}, 44 -> {25, 230},
    45 -> {95, 260}, 46 -> {300, 465}, 47 -> {145, 665}, 48 -> {345, 750}, 49 -> {415, 635}, 50 -> {420, 555}, 51 -> {520, 585}}]
```

Out[4]=

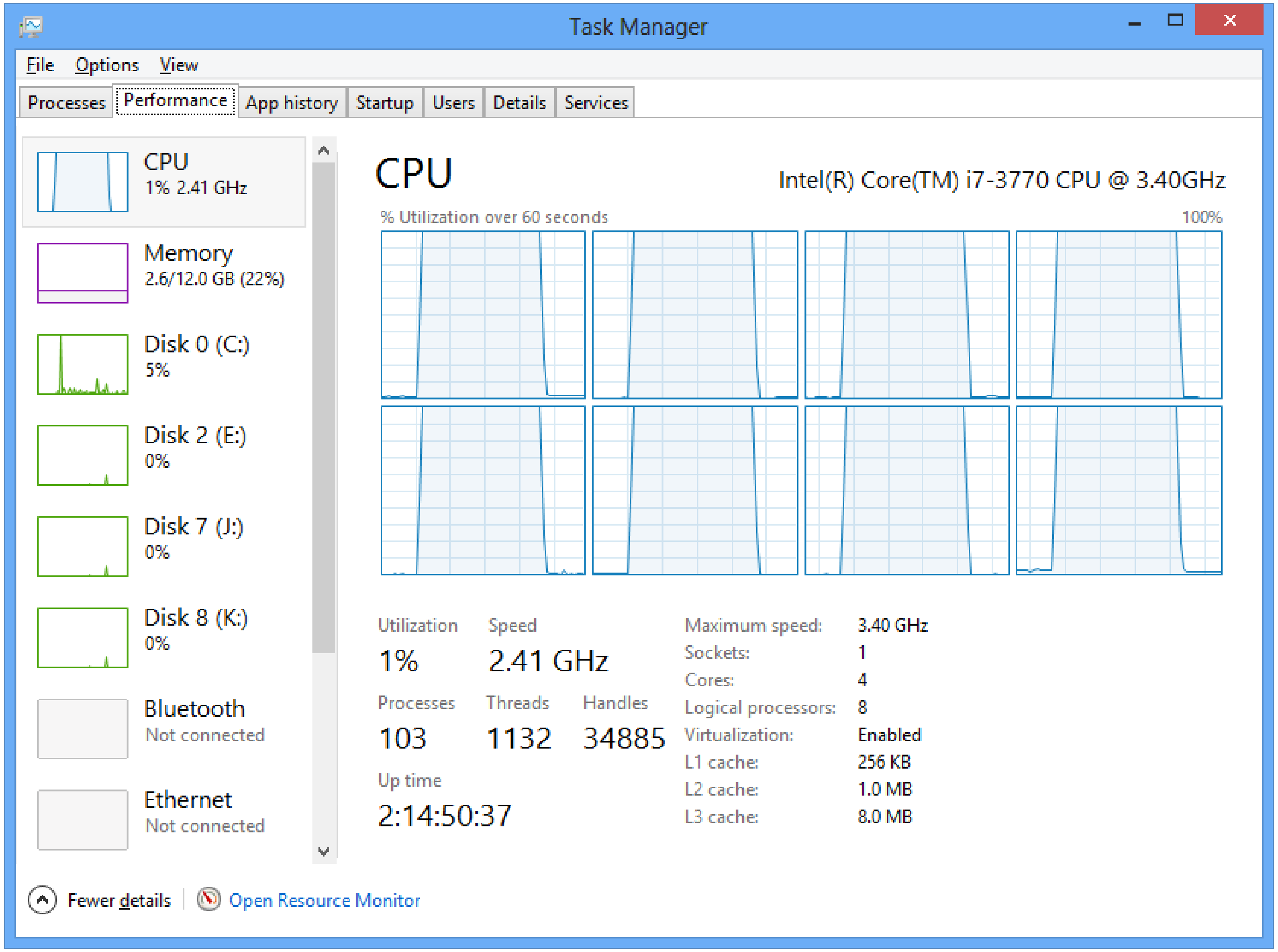




```
SOLUTION ARGUMENTS:
TSP -> NN -> Generations( g, ForkJoin ( n, SA -> 2-OPT ) ) -> TSP'
GENERAL ARGUMENTS:
 #1:
 Number of Iterations or Generations = 32
 Number of Tasks in Parallel = 32
SIMULATED ANNEALING ARGUMENTS:
 #1:
   Initial Temperature = 1200
   Stopping Criteria Temperature = 1e-007
   Decreasing Factor = 0.991
   Monte Carlo Steps = 120
------------------------------------------------
TSP INPUT:
[1](565, 575) : [2](25, 185) : [3](345, 750) : [4](945, 685) : [5]
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7,
------------------------------------------------
0 ms
------------------------------------------------
GENERATION:1
START SOLUTION:
[1](565, 575) : [22](520, 585) : [49](605, 625) : [32](575, 665) :
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7,
8980.92
  ...
```

Resultados

Desempenho e Paralelismo



CPU
1% 2.41 GHz

Memory
2.6/12.0 GB (22%)

Disk 0 (C:)
5%

Disk 2 (E:)
0%

Disk 7 (J:)
0%

Disk 8 (K:)
0%

Bluetooth
Not connected

Ethernet
Not connected

CPU

Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz

% Utilization over 60 seconds
100%

Utilization
1%

Speed
2.41 GHz

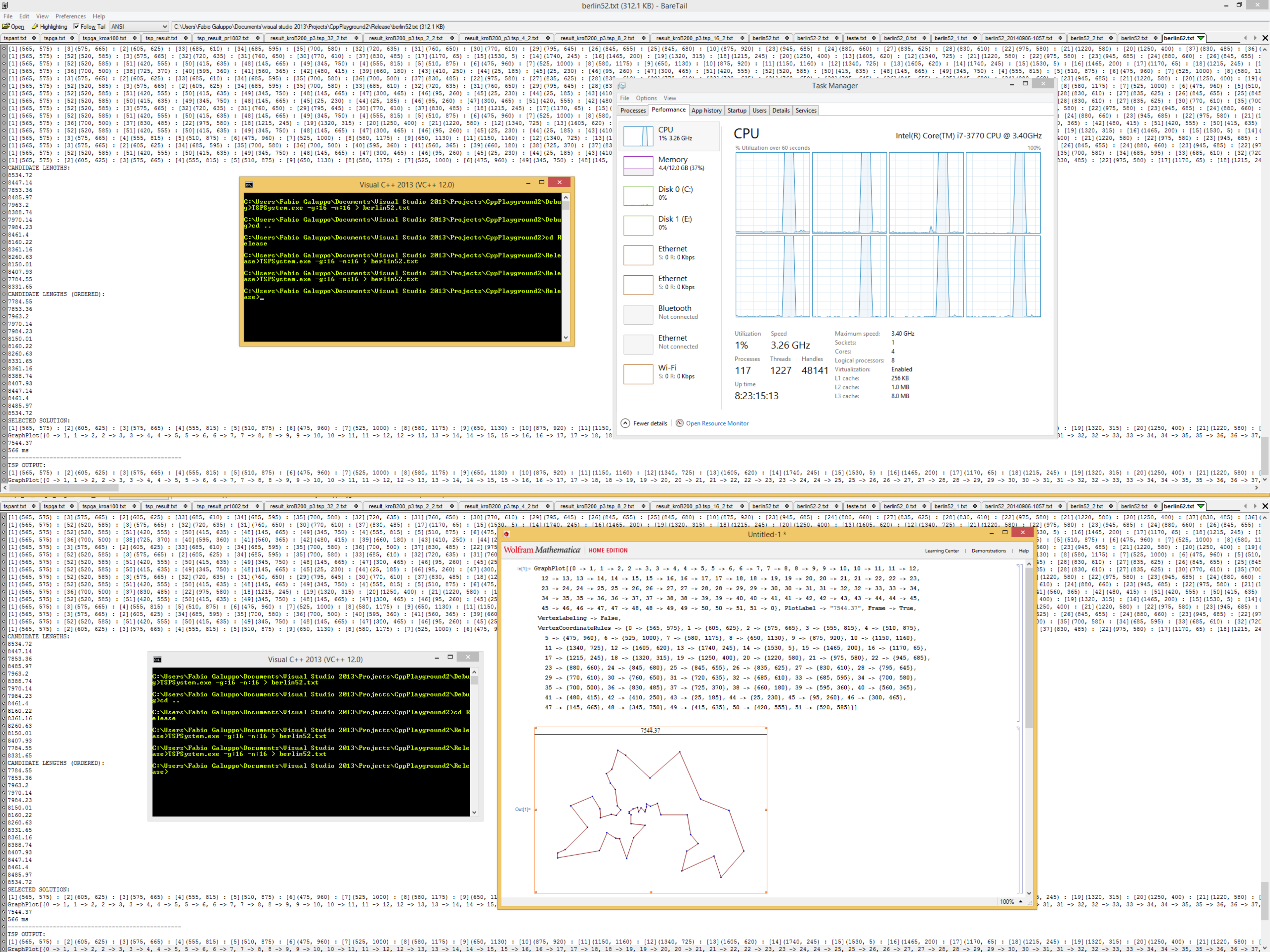
Processes
103

Threads
1132

Handles
34885

Up time
2:14:50:37

Maximum speed:	3.40 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB



berlin52.txt (312.1 KB) - BareTail

C:\Users\Fabio Galuppo\Documents\visual studio 2013\Projects\CppPlayground2\Release\berlin52.txt (312.1 KB)

```
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [48](14
[1](565, 575) : [36](700, 500) : [37](830, 485) : [22](975, 580) : [18](12
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [49](34
[1](565, 575) : [3](575, 665) : [4](555, 815) : [5](510, 875) : [6](475, 9
[1](565, 575) : [3](575, 665) : [2](605, 625) : [34](685, 595) : [35](700,
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [49](34
[1](565, 575) : [2](605, 625) : [3](575, 665) : [4](555, 815) : [5](510, 8
CANDIDATE LENGTHS:
8534.72
8447.14
7853.36
8485.97
7963.2
8388.74
7970.14
7984.23
8461.4
8160.22
8361.16
8260.63
8150.01
8407.93
7784.55
8331.65
CANDIDATE LENGTHS (ORDERED):
7784.55
7853.36
7963.2
7970.14
7984.23
8150.01
8160.22
8260.63
8331.65
8361.16
8388.74
8407.93
8447.14
8461.4
8485.97
8534.72
SELECTED SOLUTION:
[1](565, 575) : [2](605, 625) : [3](575, 665) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580, 1175) : [9](650, 1130) : [10](875, 920) : [11](1150,
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7, 7 -> 8, 8 -> 9, 9 -> 10, 10 -> 11, 11 -> 12, 12 -> 13, 13 -> 14, 14 -> 15, 15 -> 16, 16 -> 17, 17 -> 18, 18
7544.37
566 ms
---------------------------------------------
TSP OUTPUT:
[1](565, 575) : [2](605, 625) : [3](575, 665) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580, 1175) : [9](650, 1130) : [10](875, 920) : [11](1150, 1160) : [12](1340, 725) : [13](1605, 620) : [14](1740, 245) : [15](1530, 5) : [16](1465, 200) : [17](1170, 65) : [18](1215, 245) : [19](1320, 315) : [20](1250, 400) : [21](1220, 580) : [
GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7, 7 -> 8, 8 -> 9, 9 -> 10, 10 -> 11, 11 -> 12, 12 -> 13, 13 -> 14, 14 -> 15, 15 -> 16, 16 -> 17, 17 -> 18, 18 -> 19, 19 -> 20, 20 -> 21, 21 -> 22, 22 -> 23, 23 -> 24, 24 -> 25, 25 -> 26, 26 -> 27, 27 -> 28, 28 -> 29, 29 -> 30, 30 -> 31, 31 -> 32, 32 -> 33, 33 -> 34, 34 -> 35, 35 -> 36, 36 -> 37,
```

Visual C++ 2013 (VC++ 12.0)

```
C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Debu
g>TSPSystem.exe -g:16 -n:16 > berlin52.txt

C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Debu
g>cd ..

C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2>cd R
elease

C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Rele
ase>TSPSystem.exe -g:16 -n:16 > berlin52.txt

C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Rele
ase>TSPSystem.exe -g:16 -n:16 > berlin52.txt

C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Rele
ase>_
```

Task Manager

CPU Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz

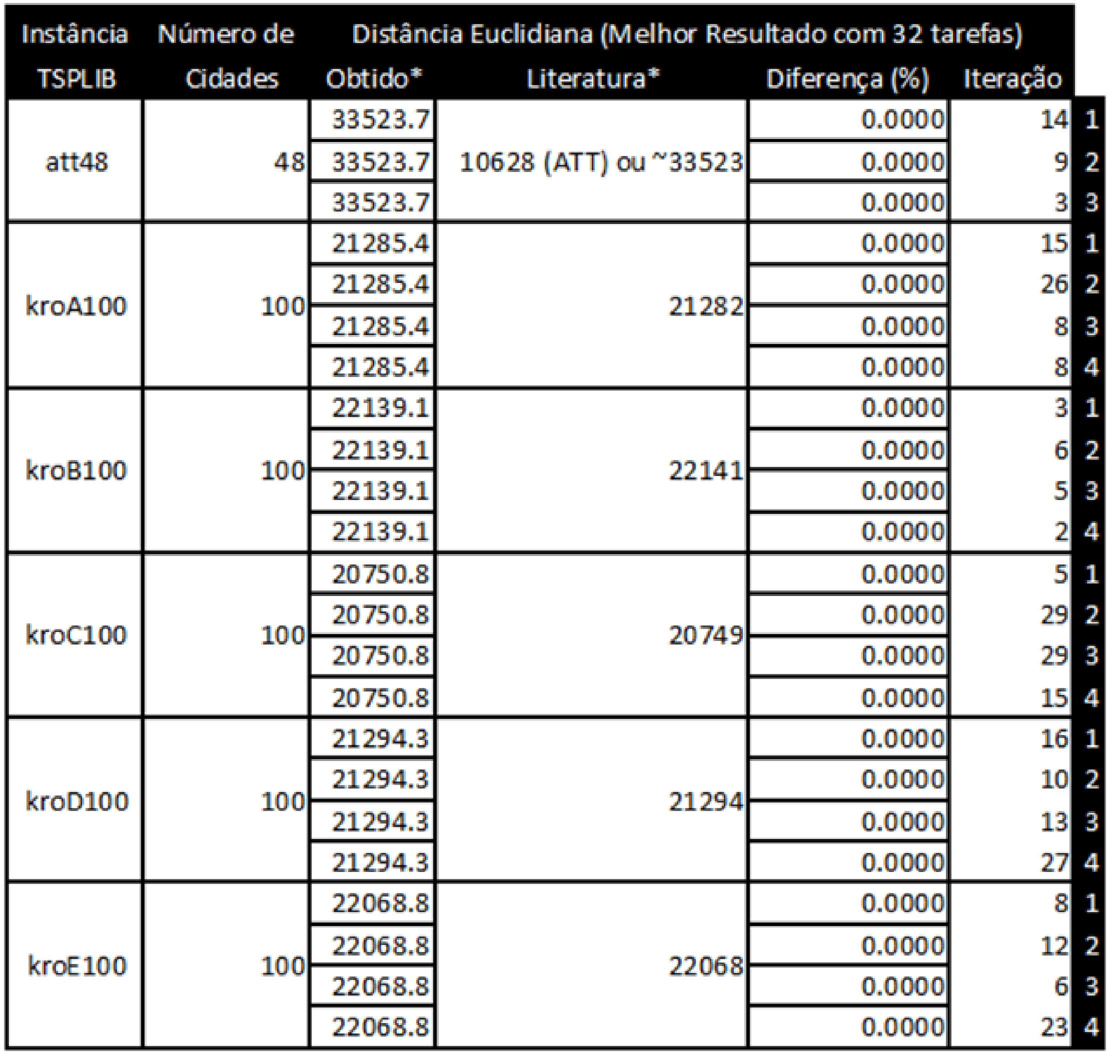
Utilization	1%
Speed	3.26 GHz
Processes	117
Threads	1227
Handles	48141
Up time	8:23:15:13
Maximum speed:	3.40 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB

Untitled-1 * — Wolfram Mathematica HOME EDITION

```
In[1]:= GraphPlot[{0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5, 5 -> 6, 6 -> 7, 7 -> 8, 8 -> 9, 9 -> 10, 10 -> 11, 11 -> 12,
   12 -> 13, 13 -> 14, 14 -> 15, 15 -> 16, 16 -> 17, 17 -> 18, 18 -> 19, 19 -> 20, 20 -> 21, 21 -> 22, 22 -> 23,
   23 -> 24, 24 -> 25, 25 -> 26, 26 -> 27, 27 -> 28, 28 -> 29, 29 -> 30, 30 -> 31, 31 -> 32, 32 -> 33, 33 -> 34,
   34 -> 35, 35 -> 36, 36 -> 37, 37 -> 38, 38 -> 39, 39 -> 40, 40 -> 41, 41 -> 42, 42 -> 43, 43 -> 44, 44 -> 45,
   45 -> 46, 46 -> 47, 47 -> 48, 48 -> 49, 49 -> 50, 50 -> 51, 51 -> 0}, PlotLabel -> "7544.37", Frame -> True,
  VertexLabeling -> False,
  VertexCoordinateRules -> {0 -> {565, 575}, 1 -> {605, 625}, 2 -> {575, 665}, 3 -> {555, 815}, 4 -> {510, 875},
    5 -> {475, 960}, 6 -> {525, 1000}, 7 -> {580, 1175}, 8 -> {650, 1130}, 9 -> {875, 920}, 10 -> {1150, 1160},
    11 -> {1340, 725}, 12 -> {1605, 620}, 13 -> {1740, 245}, 14 -> {1530, 5}, 15 -> {1465, 200}, 16 -> {1170, 65},
    17 -> {1215, 245}, 18 -> {1320, 315}, 19 -> {1250, 400}, 20 -> {1220, 580}, 21 -> {975, 580}, 22 -> {945, 685},
    23 -> {880, 660}, 24 -> {845, 680}, 25 -> {845, 655}, 26 -> {835, 625}, 27 -> {830, 610}, 28 -> {795, 645},
    29 -> {770, 610}, 30 -> {760, 650}, 31 -> {720, 635}, 32 -> {685, 610}, 33 -> {685, 595}, 34 -> {700, 580},
    35 -> {700, 500}, 36 -> {830, 485}, 37 -> {725, 370}, 38 -> {660, 180}, 39 -> {595, 360}, 40 -> {560, 365},
    41 -> {480, 415}, 42 -> {410, 250}, 43 -> {25, 185}, 44 -> {25, 230}, 45 -> {95, 260}, 46 -> {300, 465},
    47 -> {145, 665}, 48 -> {345, 750}, 49 -> {415, 635}, 50 -> {420, 555}, 51 -> {520, 585}}]

Out[1]=
```

Resultados

Diversos

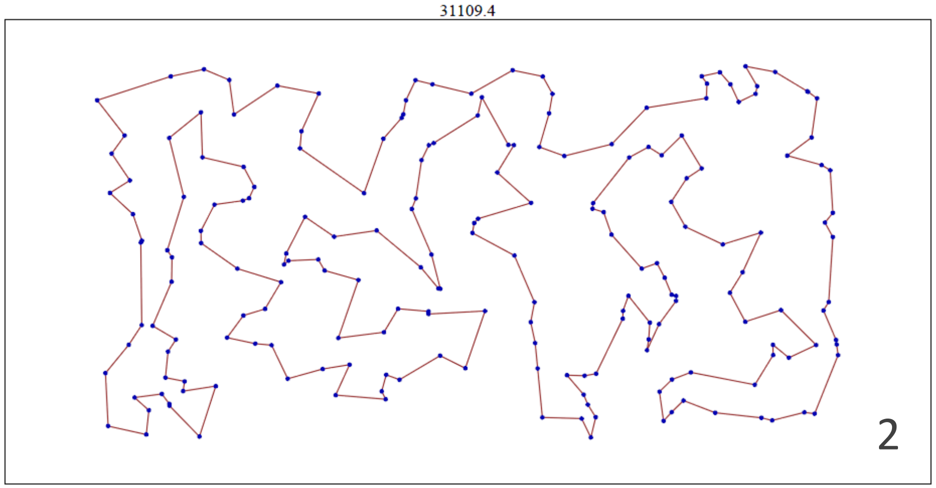


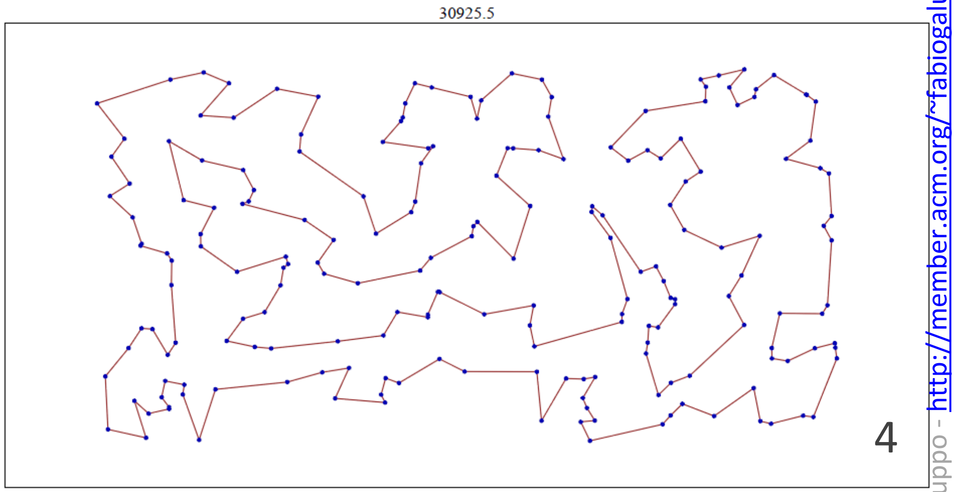
Instância	Número de	Distância Euclidiana (Melhor Resultado com 32 tarefas)				
TSPLIB	Cidades	Obtido*	Literatura*	Diferença (%)	Iteração	
att48	48	33523.7	10628 (ATT) ou ~33523	0.0000	14	1
		33523.7		0.0000	9	2
		33523.7		0.0000	3	3
kroA100	100	21285.4	21282	0.0000	15	1
		21285.4		0.0000	26	2
		21285.4		0.0000	8	3
		21285.4		0.0000	8	4
kroB100	100	22139.1	22141	0.0000	3	1
		22139.1		0.0000	6	2
		22139.1		0.0000	5	3
		22139.1		0.0000	2	4
kroC100	100	20750.8	20749	0.0000	5	1
		20750.8		0.0000	29	2
		20750.8		0.0000	29	3
		20750.8		0.0000	15	4
kroD100	100	21294.3	21294	0.0000	16	1
		21294.3		0.0000	10	2
		21294.3		0.0000	13	3
		21294.3		0.0000	27	4
kroE100	100	22068.8	22068	0.0000	8	1
		22068.8		0.0000	12	2
		22068.8		0.0000	6	3
		22068.8		0.0000	23	4

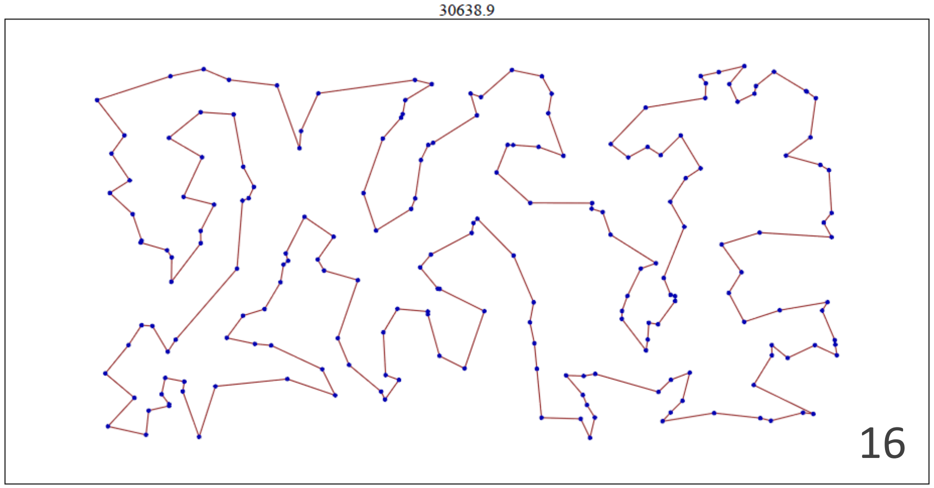
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

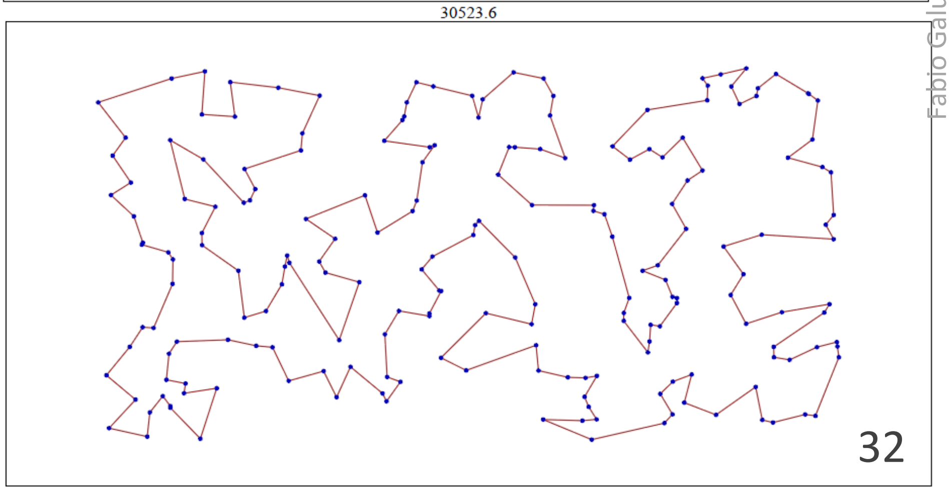
Resultados

Diversos





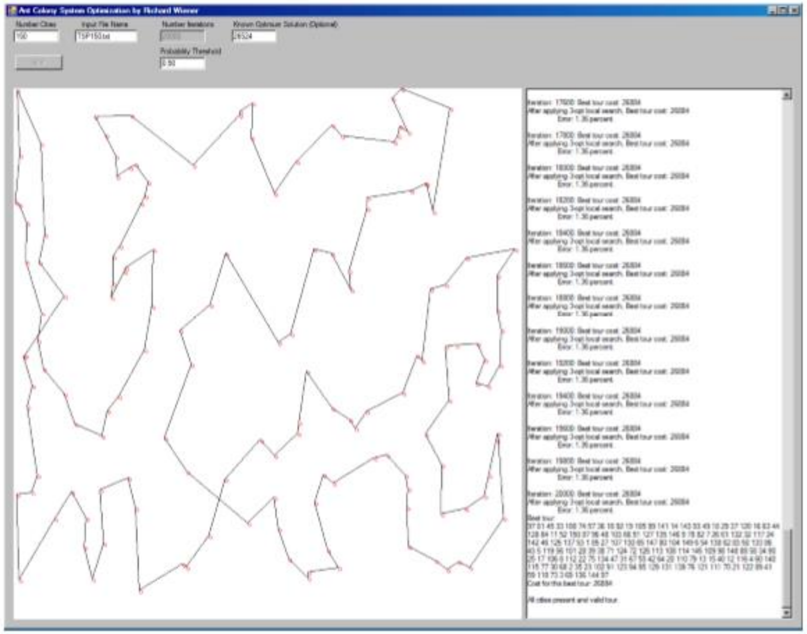




Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Observação

Ant System Colony Optimization



The Ant Colony System produces results that are superior to those obtained by this author using either simulated annealing or genetic programming. What is particularly attractive is the small number of parameters that need to be tuned, especially compared to simulated annealing.

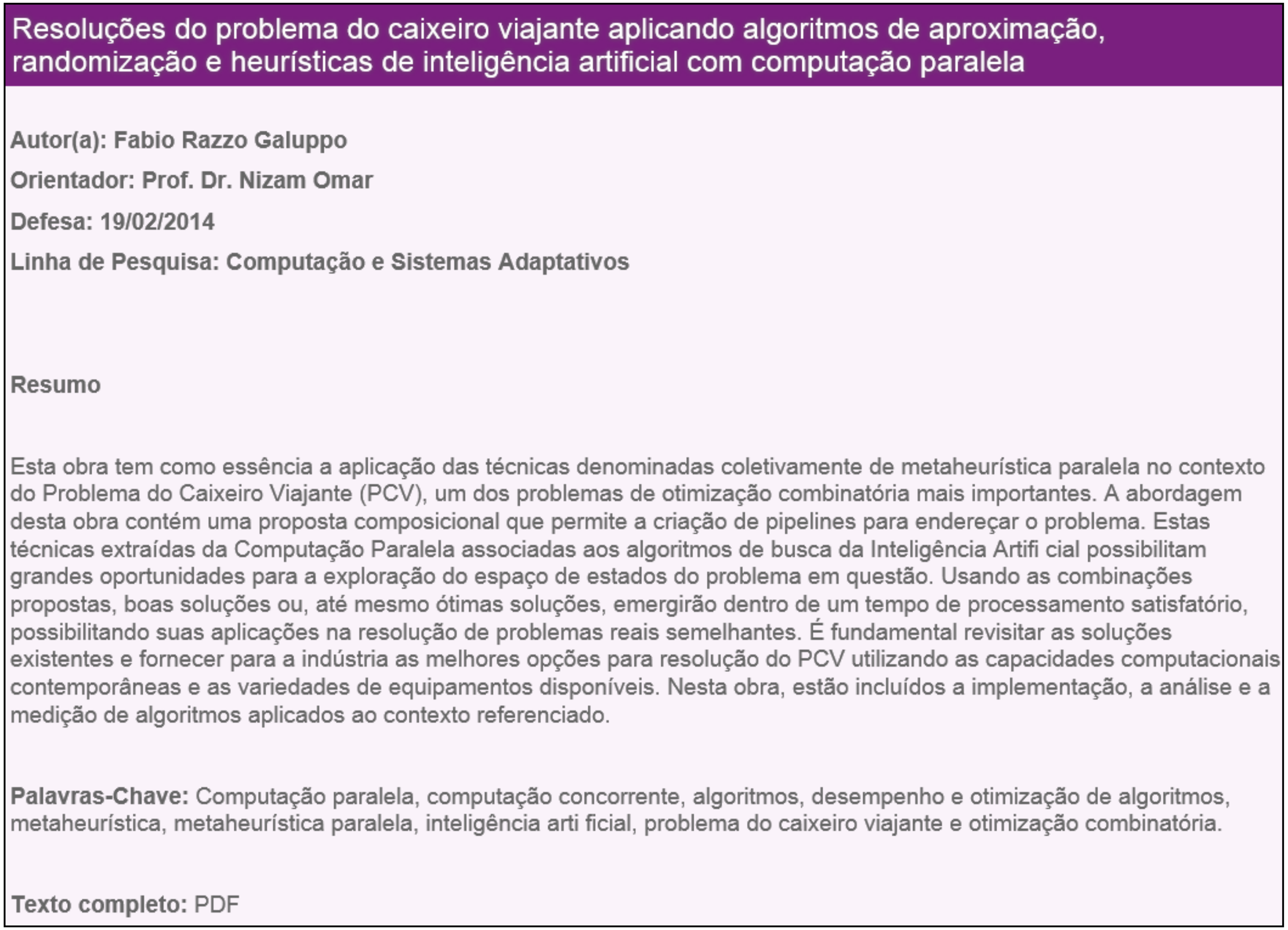
About the author

Richard Wiener is Chair of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 22 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, published by Thomson, Course Technology in April 2006, is entitled *Modern Software Development Using C#/.NET*.

http://www.jot.fm/issues/issue_2009_09/column4.pdf

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Publicação



Resoluções do problema do caixeiro viajante aplicando algoritmos de aproximação, randomização e heurísticas de inteligência artificial com computação paralela

Autor(a): Fabio Razzo Galuppo

Orientador: Prof. Dr. Nizam Omar

Defesa: 19/02/2014

Linha de Pesquisa: Computação e Sistemas Adaptativos

Resumo

Esta obra tem como essência a aplicação das técnicas denominadas coletivamente de metaheurística paralela no contexto do Problema do Caixeiro Viajante (PCV), um dos problemas de otimização combinatória mais importantes. A abordagem desta obra contém uma proposta composicional que permite a criação de pipelines para endereçar o problema. Estas técnicas extraídas da Computação Paralela associadas aos algoritmos de busca da Inteligência Artificial possibilitam grandes oportunidades para a exploração do espaço de estados do problema em questão. Usando as combinações propostas, boas soluções ou, até mesmo ótimas soluções, emergirão dentro de um tempo de processamento satisfatório, possibilitando suas aplicações na resolução de problemas reais semelhantes. É fundamental revisitar as soluções existentes e fornecer para a indústria as melhores opções para resolução do PCV utilizando as capacidades computacionais contemporâneas e as variedades de equipamentos disponíveis. Nesta obra, estão incluídos a implementação, a análise e a medição de algoritmos aplicados ao contexto referenciado.

Palavras-Chave: Computação paralela, computação concorrente, algoritmos, desempenho e otimização de algoritmos, metaheurística, metaheurística paralela, inteligência artificial, problema do caixeiro viajante e otimização combinatória.

Texto completo: PDF

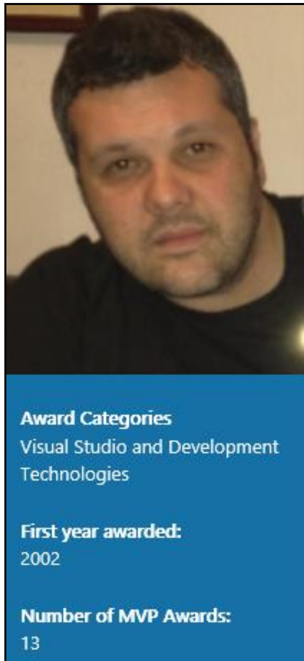
http://up.mackenzie.br/stricto-sensu/engenharia-eletrica-e-computacao/teses-e-dissertacoes-detalhada/artigo/resolucoes-do-problema-do-caixeiro-viajante-aplicando-algoritmos-de-aproximacao-randomizacao/

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Heurísticas com Paralelismo através do Problema do Caixeiro Viajante

Fabio Galuppo, M.Sc.
http://fabiogaluppo.com e http://simplycpp.com/
fabiogaluppo@acm.org
@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies
https://mvp.microsoft.com/en-us/PublicProfile/9529



Award Categories
Visual Studio and Development Technologies

First year awarded:
2002

Number of MVP Awards:
13

http://bit.ly/pcv-mack-abril-2016