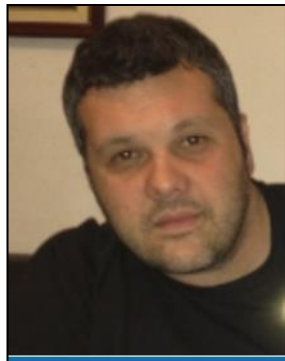
(Re)descobrindo o C++ com problemas NP-completos, lambdas, monads, IA e paralelismo

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

fabiogaluppo@acm.org



First year awarded:
2002

Number of MVP Awards:
11

Technical Expertise:
Visual C++

Technical Interests:
Visual C#, Visual F#

(RE)DESCOBRINDO O C++

http://isocpp.org/

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

O que é C++?

What is C++?

Smarter computin
Texas A&M Univer

Template meta-programming!

Class hierarchies

A hybrid language

Buffer overflows

A multi-paradigm programming language

Classes

It's C!

Too big!

Embedded systems programming language

Low level!

Generic programming

An object-oriented programming language

A random collection of features

Stroustrup - ACCU'13

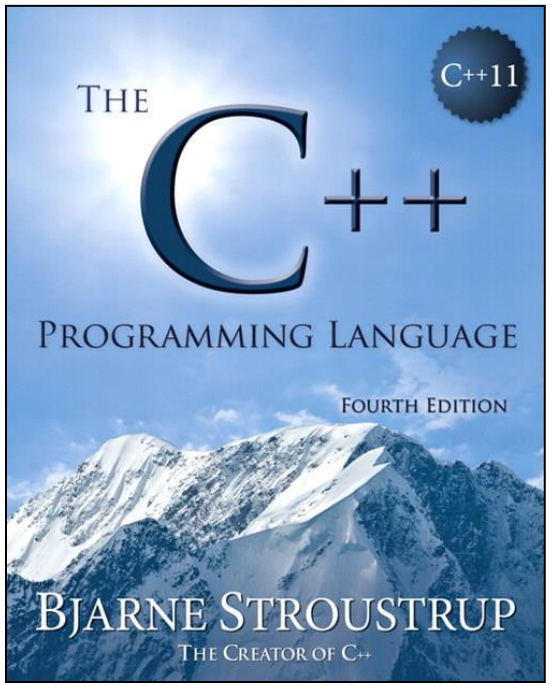
http://www.infoq.com/presentations/Cplusplus-11-Bjarne-Stroustrup

The C++ Programming Language

C++ is a general purpose programming language with a bias towards systems programming that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming.

C++ is a general purpose programming language designed to make programming more enjoyable for the serious programmer.



Domain-specific abstraction

Fortran

Cobol

General-purpose abstraction

Simula

Java

C++

C++11

C#

Direct mapping to hardware

Assembler

BCPL

C

http://www.stroustrup.com/

Linguagem + Biblioteca

C++ 17: https://github.com/cplusplus/draft

Document Number:	N4296
Date:	2014-11-19
Revises:	N4140
Reply to:	Richard Smith Google Inc cxxeditor@gmail.com

Working Draft, Standard for Programming Language C++

acesso em: 22 de Março de 2015

C++ 14: ISO/IEC 14882:2014

ISO/IEC 14882:2014 specifies requirements for implementations of the C++ programming language. The first such requirement is that they implement the language, and so this International Standard also defines C++. Other requirements and relaxations of the first requirement appear at various places within this International Standard.

C++ is a general purpose programming language based on the C programming language as described in ISO/IEC 9899:1999 *Programming languages ? C* (hereinafter referred to as the *C standard*). In addition to the facilities provided by C, C++ provides additional data types, classes, templates, exceptions, namespaces, operator overloading, function name overloading, references, free store management operators, and additional

library facilities.

https://isocpp.org/std/status

Table 3 — Keywords

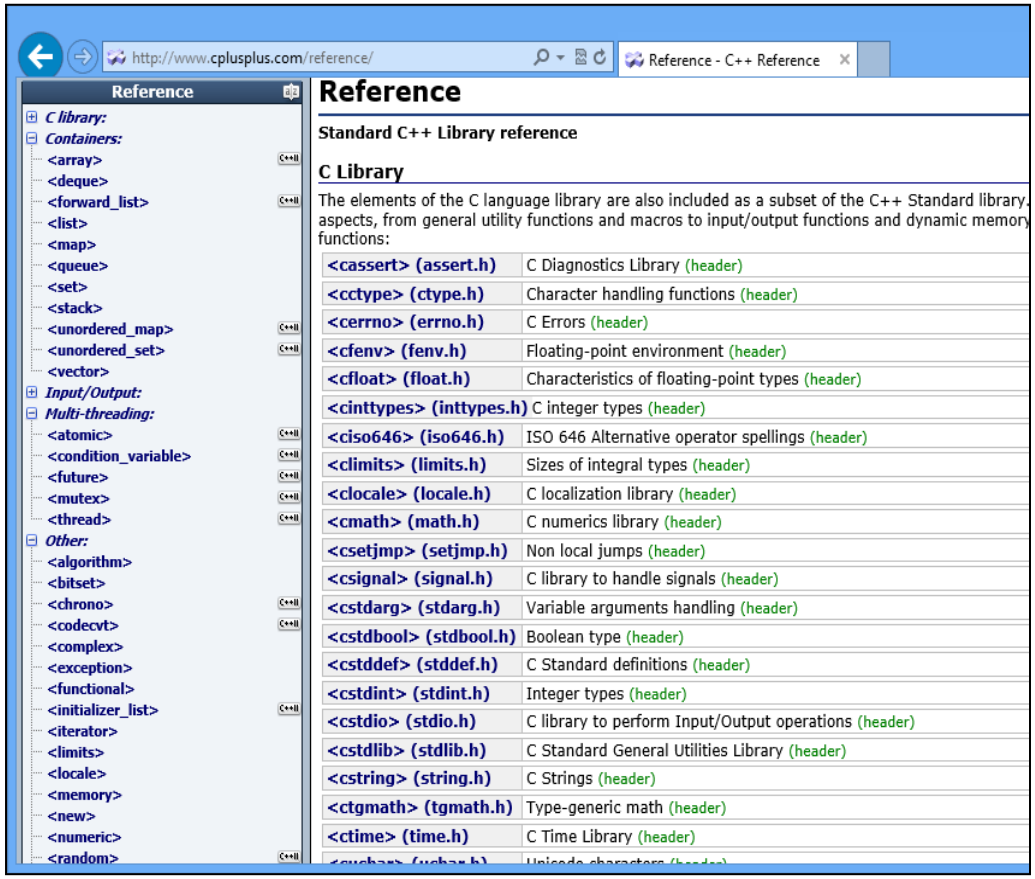
alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

2.11 Keywords [lex.key] [2. Lexical conventions]

Compiladores

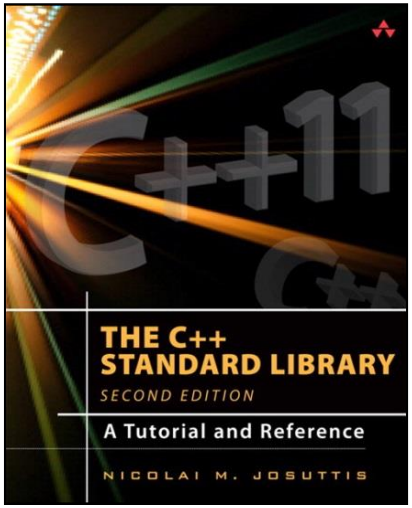
Summary of C++11 Feature Availability: http://www.aristeia.com/C++11/C++11FeatureAvailability.htm

Standard Template Library



http://www.cplusplus.com

Selected Standard Library Headers		
<algorithm>	copy(), find(), sort()	§iso.25
<array>	array	§iso.23.3.2
<chrono>	duration, time_point	§iso.20.11.2
<cmath>	sqrt(), pow()	§iso.26.8
<complex>	complex, sqrt(), pow()	§iso.26.8
<forward_list>	forward_list	§iso.23.3.4
<fstream>	fstream, ifstream, ofstream	§iso.27.9.1
<future>	future, promise	§iso.30.6
<ios>	hex,dec,scientific,fixed,defaultfloat	§iso.27.5
<iostream>	istream, ostream, cin, cout	§iso.27.4
<map>	map, multimap	§iso.23.4.4
<memory>	unique_ptr, shared_ptr, allocator	§iso.20.6
<random>	default_random_engine, normal_distribution	§iso.26.5
<regex>	regex, smatch	§iso.28.8
<string>	string, basic_string	§iso.21.3
<set>	set, multiset	§iso.23.4.6
<sstream>	istrstream, ostrstream	§iso.27.8
<stdexcept>	length_error, out_of_range, runtime_error	§iso.19.2
<thread>	thread	§iso.30.3
<unordered_map>	unordered_map, unordered_multimap	§iso.23.5.4
<utility>	move(), swap(), pair	§iso.20.1
<vector>	vector	§iso.23.3.6



Orientação a Objetos

```
//Regular type is Default Constructible, Assignable, and Equality Comparable
//InterestRate is a regular type
template <typename T>
struct InterestRate final
{
    static_assert(std::is_floating_point<T>::value, "not an floating point type");

    //explicit constructor
    explicit InterestRate(T rate)
        : Rate_(rate)
    {
    }

    //default constructor
    InterestRate()
        : InterestRate(T(0))
    {
    }

    //calculate interest rate on single period
    T operator()(T value) const
    {
        T result = value * (T(1) + Rate_);
        return result;
    }

    //equality comparer
    bool operator==(const InterestRate<T>& that) const
    {
        return Rate_ == that.Rate_;
    }

    //inequality comparer in terms of equality comparer
    bool operator!=(const InterestRate<T>& that) const
    {
        return !(*this == that);
    }

    InterestRate(const InterestRate<T>&) = default; //copy constructor
    InterestRate<T>& operator=(const InterestRate<T>&) = default; //copy assignment
    ~InterestRate() = default; //destructor

    friend std::wstring std::to_wstring(const InterestRate<T>& _Val);

private:
    T Rate_;
};
```

Programação Genérica

- Templates
- Standard Template Library
  - Containers (`vector`, `list`, `unordered_map`, …)
  - Algorithms (`sort`, `merge`, `shuffle`, `minmax`, …)
  - `Iterators`

```
amount = 5000.;
std::vector<interest_rate> periods{ p1, p2, interest_rate(0.07), interest_rate(0.045) };
for (auto& p : periods)
    amount = p(amount);
```

```
amount = 5000.;
std::for_each(periods.cbegin(), periods.cend(),
    [&amount](const interest_rate& p){ amount = p(amount); });
```

Funções e Lambdas

```
interest_rate p1(0.065);
std::function<double(double)> fp1 = p1;
amount = fp1(amount);

double rate = 0.1;
auto fp2 = [rate](double value) {
    double result = value * (double(1) + rate);
    return result;
};
```

```
amount = 5000.;
amount = std::accumulate(periods.cbegin(), periods.cend(), amount,
    [](double amount, const interest_rate& p){ return p(amount); });
```

Higher-order Function

In mathematics and computer science, a **higher-order function** (also **functional form**, **functional** or **functor**) is a function that does at least one of the following:

- takes one or more functions as an input
- outputs a function

All other functions are *first-order functions*. In mathematics higher-order functions are also known as *operators* or *functionals*. The derivative in calculus is a common example, since it maps a function to another function.

`foldl`: $(\ldots f(f\ (f\ (f\ acc\ 1)\ 2)\ 3)4\ \ldots)$

```
template<class Container, typename T, class Function>
T foldl (Function f, T init, const Container& xs)
{
    //T acc = init;
    //for (const auto& x : xs) acc = f(acc, x);
    //return acc;
    //or:
    using F = typename Container::value_type;
    return std::accumulate(xs.cbegin(), xs.cend(), init, [](T acc, const F& f){ return f(acc); });
}
```

```
    template <typename T>
    inline std::function<T(T)> interest_rate_hof(T rate)
    {
        static_assert(std::is_floating_point<T>::value, "not an floating point type");
        return [rate](T value) {
            T result = value * (T(1) + rate);
            return result;
        };
    }
```

http://en.wikipedia.org/wiki/Higher-order_function

C++ *Lambda Expressions*

$[\,]\ \ (\,)_{opt}\ \ \text{->}_{opt}\ \ \{\,\}$

[captures] (params) -> ret { statements; }

May 18th, 2011 — C++0x Lambda Functions — Herb Sutter: http://nwcpp.org/may-2011.html

```
amount = 5000.;
amount = foldl([](double amount, const interest_rate& p){ return p(amount); }, amount, periods);
```

```
foldl([](int acc, int x) { return acc + x; }, 0, xs);

int i = 10;
foldl([i](int acc, int x) { return acc + x + i; }, 0, xs);

int i = 10;
std::bind([&i](int a, int b) { i = a + b;  return i; }, 10, 20)();
//i == 30
```

Morfismo ou “Notação de” Flecha

$$\frac{f : a \to b \qquad g : b \to c}{g \circ f : a \to c}$$

In many fields of mathematics, **morphism** refers to a *structure-preserving mapping* from one mathematical structure to another. The notion of morphism recurs in much of contemporary mathematics. In set theory, morphisms are functions; in linear algebra, linear transformations; in group theory, group homomorphisms; in topology, continuous functions, and so on.

In category theory, *morphism* is a broadly similar idea, but somewhat more abstract: the mathematical objects involved need not be sets, and the relationship between them may be something more general than a map.

The study of morphisms and of the structures (called objects) over which they are defined, is central to category theory. Much of the terminology of morphisms, as well as the intuition underlying them, comes from concrete categories, where the *objects* are simply *sets with some additional structure*, and *morphisms* are *structure-preserving functions*. In category theory, morphisms are sometimes also called **arrows**.

```
auto f(int a)   -> float  { return 2.f * a; }
auto g(float b) -> double { return 3.  * b; }
```

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

Composição de funções

```
template<class T, class U, class V>
inline std::function<V(T)> compose(std::function<V(U)> g, std::function<U(T)> f)
{
    return [=](T x) -> V { return g(f(x)); };
}
```

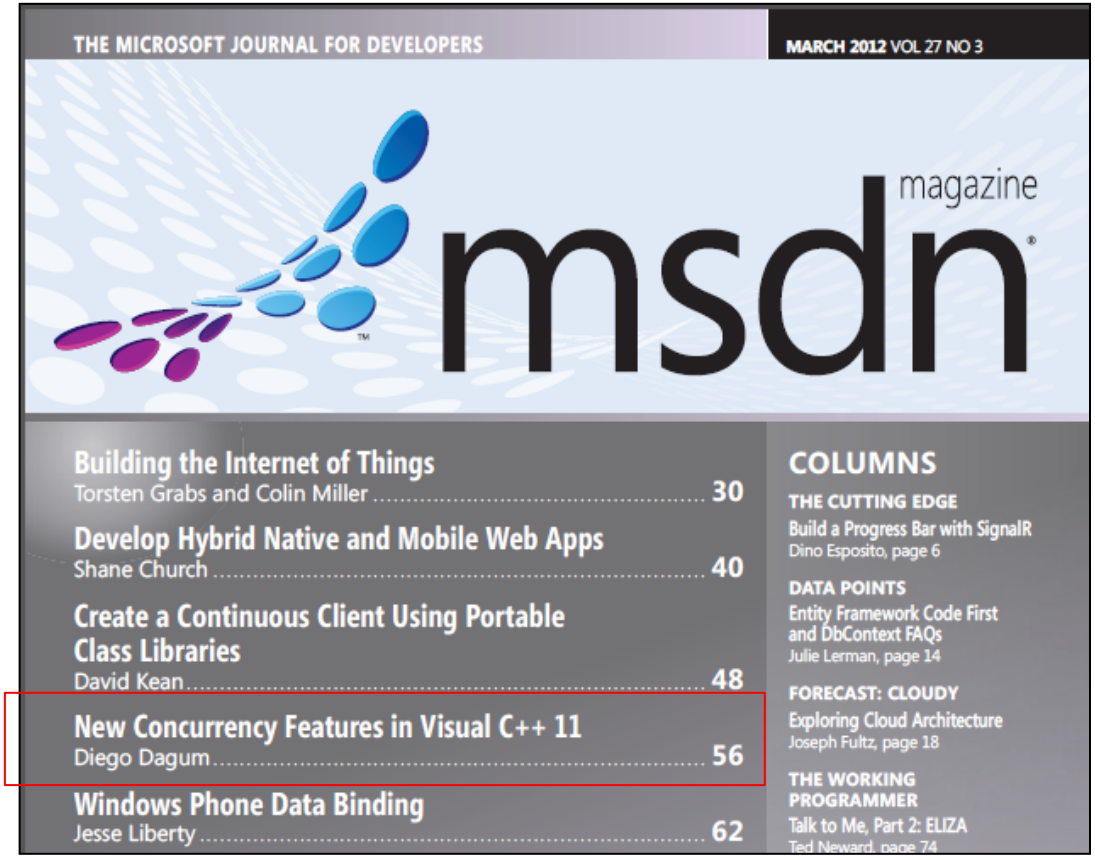
$$g \circ f : T \to V$$

```
amount = 5000.;
auto hofp1 = interest_rate_hof(0.065);
auto hofp2 = interest_rate_hof(0.1);
auto _p2_p1 = compose(hofp2, hofp1);
amount = _p2_p1(amount);
```

```
amount = 5000.;
auto _p2_p1_bnd = std::bind(hofp2, std::bind(hofp1, std::placeholders::_1));
amount = _p2_p1_bnd(amount);
```

Programação Concorrente com C++

```
#include <thread>
#include <future>
#include <atomic>
#include <mutex>
#include <condition_variable>
```



artigo online: http://msdn.microsoft.com/en-us/magazine/hh852594.aspx
código fonte: http://archive.msdn.microsoft.com/mag201203CPP

library
Multi-threading C++11

Atomic and thread support
Support for atomics and threads:

Headers

<atomic>	Atomic (header)
<thread>	Thread (header)
<mutex>	Mutex (header)
<condition_variable>	Condition variable (header)
<future>	Future (header)

This article discusses Visual C++ 11, a prerelease technology.
All related information is subject to change.

This article discusses:

- Parallel execution
- Asynchronous tasks
- Threads
- Variables and exceptions
- Synchronization
- Atomic types
- Mutexes and locks
- Condition variables

Technologies discussed:

Visual C++ 11

Code download available at:

code.msdn.microsoft.com/mag201203CPP

DIEGO DAGUM *is a software developer with more than 20 years of experience. He's currently a Visual C++ community program manager with Microsoft.*

THANKS to the following technical experts for reviewing this article:
David Cravey, Alon Fliess, Fabio Galuppo and Marc Gregoire

Programação usando futures em C++

```
#include <future>
#include <chrono>

void future_sample()
{
    const int N = 8;

    std::vector<std::future<int>> fs;
    fs.reserve(N);

    for (int i = 0; i < N; ++i)
    {
        std::future<int> f = std::async([&, i]() {
            std::this_thread::sleep_for(std::chrono::seconds(2));
            return (1 + i) * 100;
        });

        fs.push_back(std::move(f));
    }

    std::vector<int> xs(N);
    std::transform(fs.begin(), fs.end(), xs.begin(), [](std::future<int>& f) {
        return f.get();
    });

    printLn(xs);
    printLn();
}
```

Eu quero aprender C++! Por onde começo?

"C++ feels like a new language…"

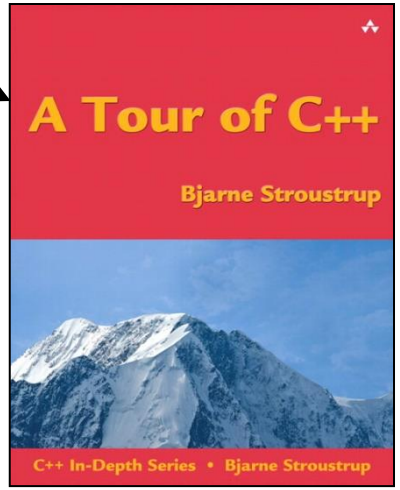
Bjarne Stroustrup

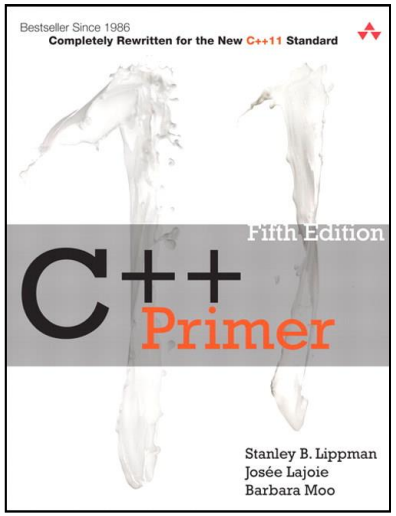
The Essence of C++: With Examples in C++84, C++98, C++11, and C++14

Abstract

- C++11 is being deployed and the shape of C++14 is becoming clear. This talk examines the foundations of C++. What is essential? What sets C++ apart from other languages? How does new and old features support (or distract from) design and programming relying on this essence.
- I focus on the abstraction mechanisms (as opposed to the mapping to the machine): Classes and templates. Fundamentally, if you understand vector, you understand C++.
- Type safety and resource safety are key design aims for a program. These aims must be met without limiting the range of applications and without imposing significant run-time or space overheads. I address issues of resource management (garbage collection is not an ideal answer and pointers should not be used as resource handles), generic programming (we must make it simpler and safer), compile-time computation (how and when?), and type safety (casts belongs in the lowest-level hardware interface). I will touch upon move semantics, exceptions, concepts, type aliases, and more. My aim is not so much to present novel features and technique, but to explore how C++'s feature set supports a new and more effective design and programming style.
- Primary audience
  - Experienced programmers with weak C++ understanding
  - Academics/Teachers/Mentors
  - Architects (?)

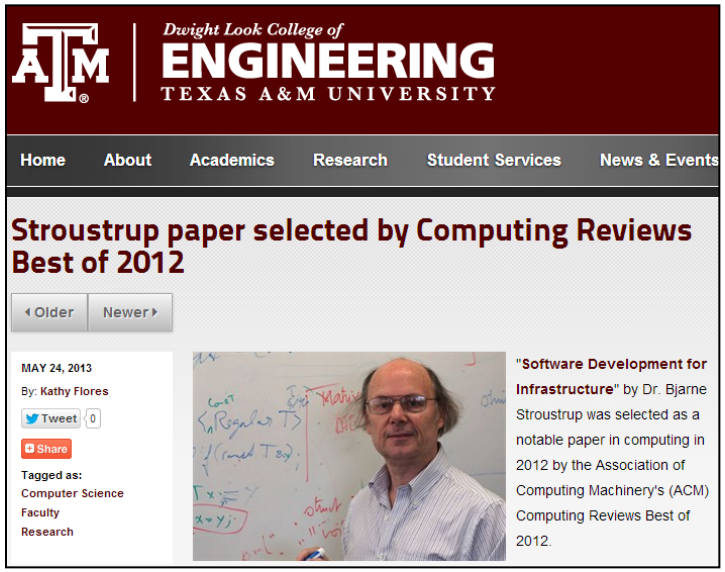
Stroustrup - Essence - Going Native'13 2

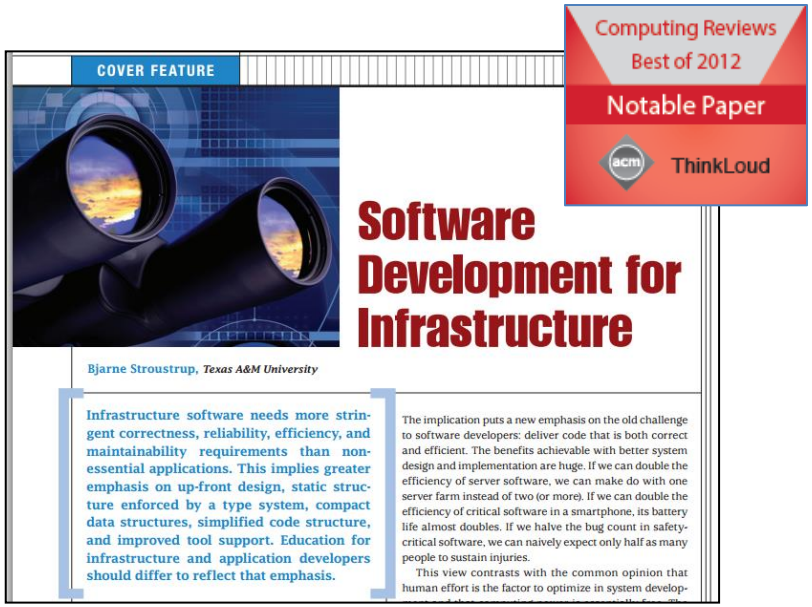




http://channel9.msdn.com/Events/GoingNative/2013/Opening-Keynote-Bjarne-Stroustrup

C++





IEEE Computer Magazine Volume:45, Issue: 1
http://dx.doi.org/10.1109/MC.2011.353

"A light-weight abstraction programming language

Key strengths:

- software infrastructure
- resource-constrained applications"

http://www.infoq.com/presentations/Cplusplus-11-Bjarne-Stroustrup

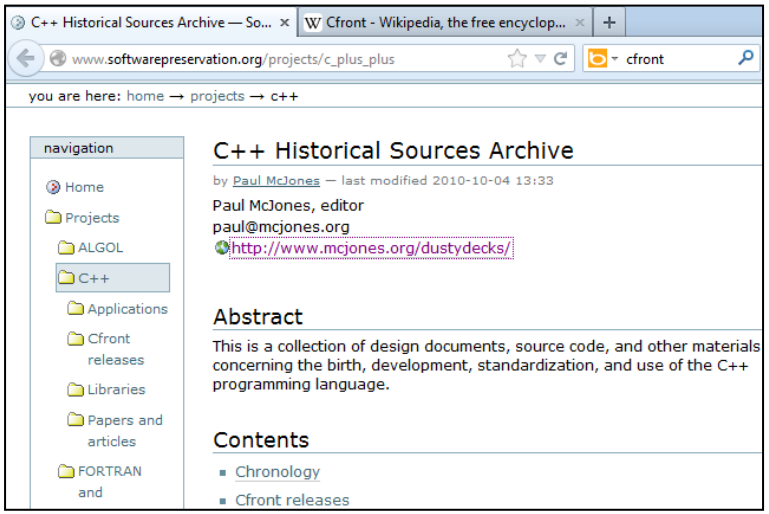
Software Infrastructure

- Core CLR - .NET Runtime
  - https://github.com/dotnet/coreclr/tree/master/src
- Evented I/O for v8 JavaScript (http://nodejs.org/)
  - https://github.com/joyent/node/tree/master/src
- HHVM - A virtual machine designed for executing programs written in Hack and PHP
  - https://github.com/facebook/hhvm/tree/master/hphp/runtime/vm
- C++ on Mars
  - http://channel9.msdn.com/events/CPP/C-PP-Con-2014/C-P-P-On-Mars-Incorporating-C-Into-Mars-Rover-Flight-Software
- C++ Conformance And Cross-Platform Mobile Development
  - http://herbsutter.com/2014/11/12/vs-clang-cross-platform-and-a-short-video/
  - http://channel9.msdn.com/events/Visual-Studio/Connect-event-2014/311
- Aplicações e casos de uso
  - http://www.stroustrup.com/applications.html

C++ e o caminho para o ISO C++

- 1979 (C com Classes)
- 1985 (Primeira versão comercial)
- 1990 (ANSI C++ Standard – baseado no “ARM”)
- 1998 (Primeira versão do padrão ISO C++)
- 2011 (Segunda versão do padrão ISO C++)
- 2014 e 2017

http://www.softwarepreservation.org/projects/c_plus_plus



http://www.softwarepreservation.org/projects/c_plus_plus/cfront/release_1.0/src/cfront/

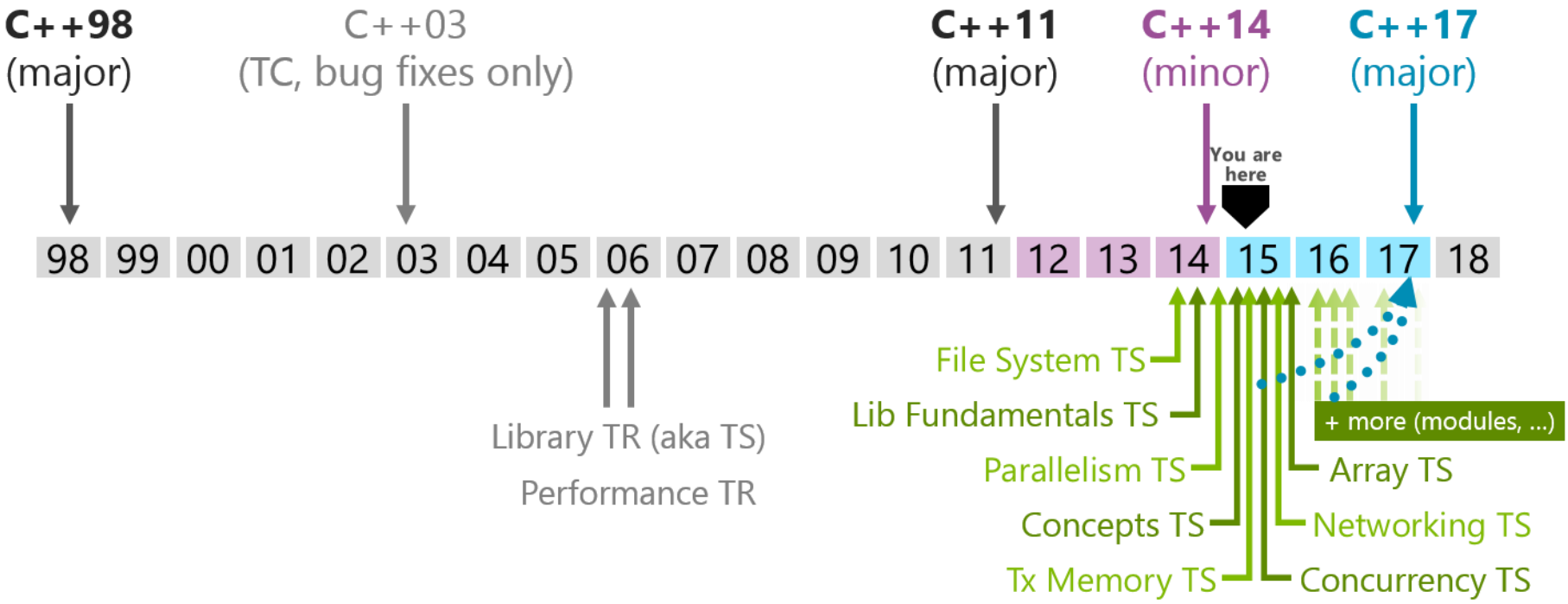
Release 1.0

Cfront 1.0, in October 1985, was the first commercial release.

Source Code

- Release 1.0, AT&T Technologies, Inc. Files timestamped February 7, 1986.

Status do C++



https://isocpp.org/std/status

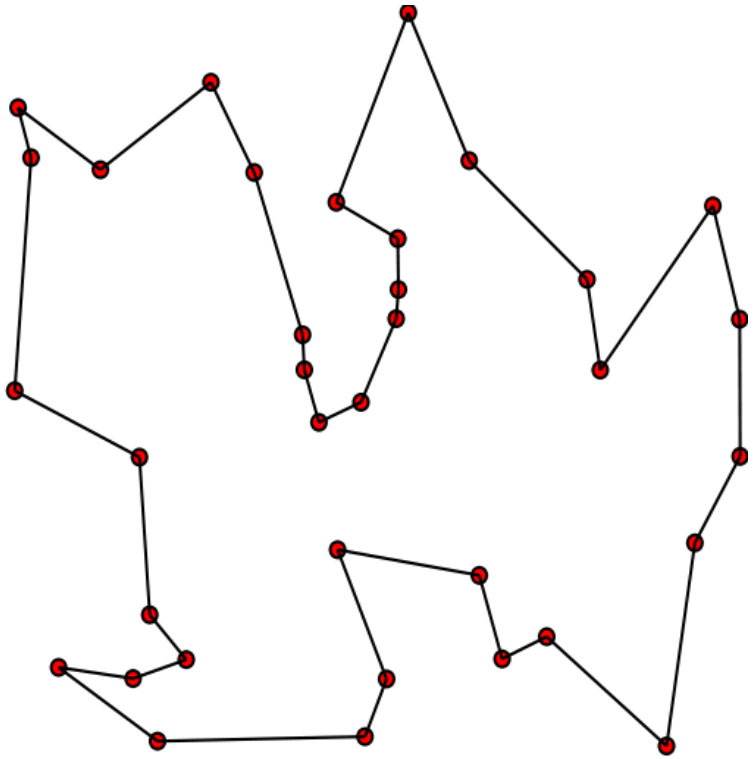
It's all about Polyglot Programming!

C++ supports systems programming. This implies that C++ code is able to effectively interoperate with software written in other languages on a system. The idea of writing all software in a single language is a fantasy. From the beginning, C++ was designed to interoperate simply and efficiently with C, assembler, and Fortran. By that, I meant that a C++, C, assembler, or Fortran function could call functions in the other languages without extra overhead or conversion of data structures passed among them.

"Nobody should call themselves a professional if they only knew one language."

...**C++**, of course; **Java**; maybe **Python** for mainline work... And if you know those, you can't help know sort of a little bit about **Ruby** and **JavaScript**, you can't help knowing **C** because that's what fills out the domain and of course **C#**. But again, **these languages create a cluster** so that **if you knew either five of the ones that I said, you would actually know the others**...

http://www.youtube.com/watch?v=NvWTnIoQZj4

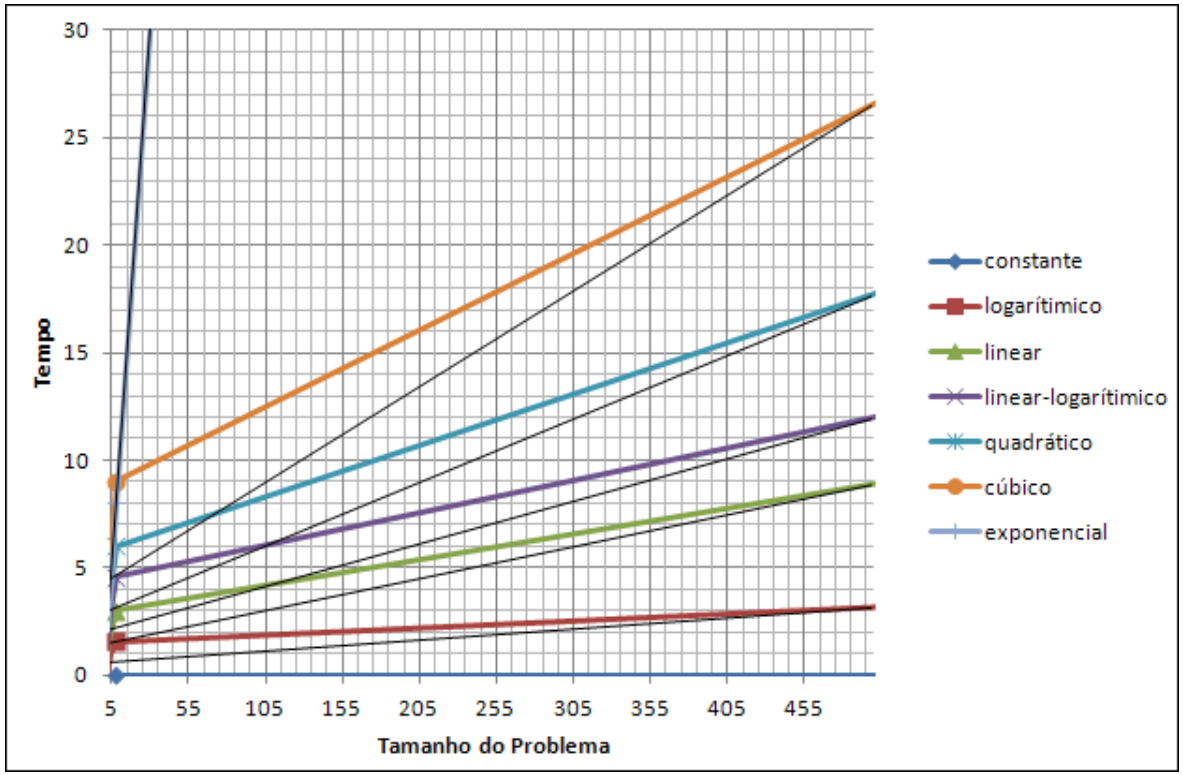


UMA VIAGEM AOS PROBLEMAS INTRATÁVEIS

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

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

Growth	n(Size) 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.)

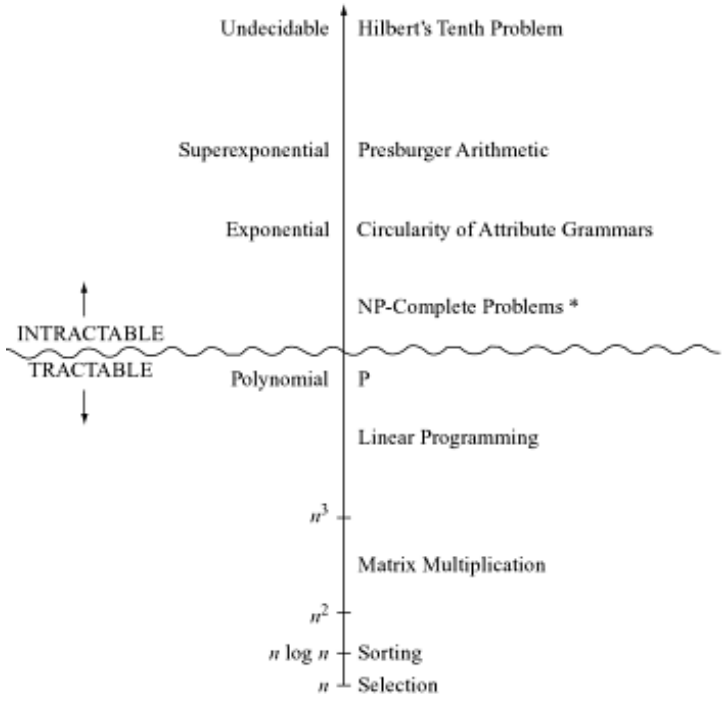
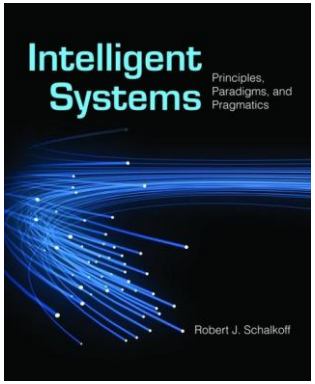


Figure 3.4 The Spectrum of Computational Complexity



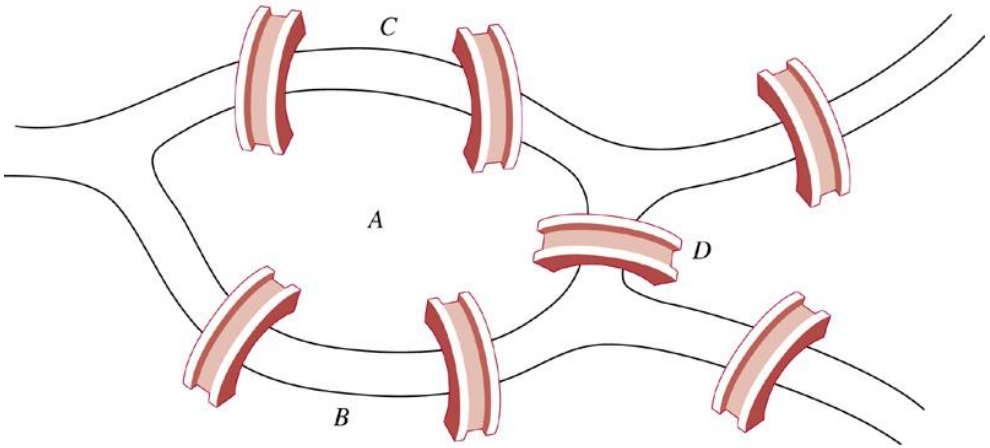
http://www.jblearning.com/catalog/9780763780173/preview/

O Problema do Caixeiro Viajante (PCV)

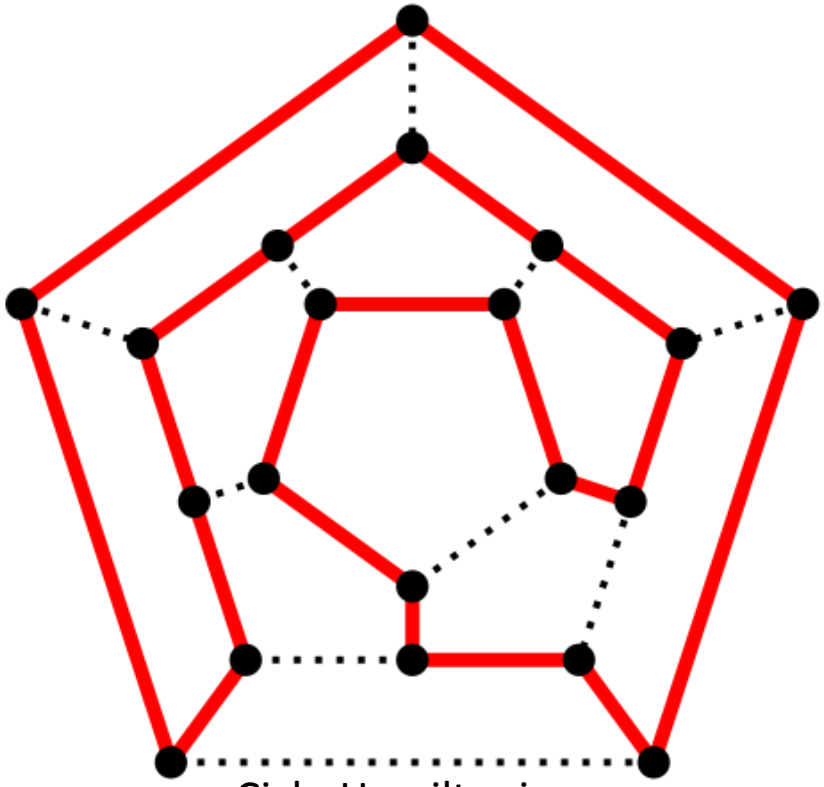
- Problema de Minimização
  - Encontrar o menor ciclo para um conjunto de cidades a serem visitadas obrigatoriamente e retornando a origem
  - Uma de suas instâncias considera a função objetivo como a distância euclidiana entre as cidades
    - Problema NP, inerentemente intratável
- Intratabilidade
  - 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:

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

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

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

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

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

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

Restrições:

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

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

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

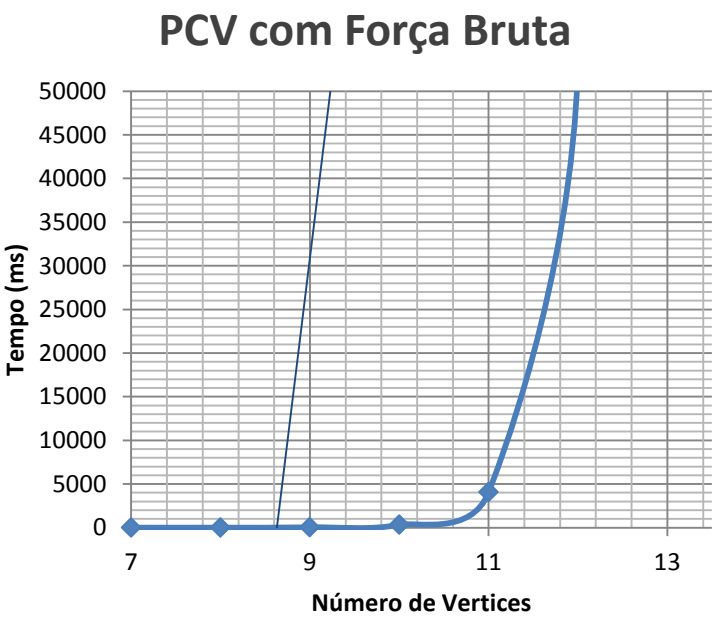
Modelagem Matemática
programação binária

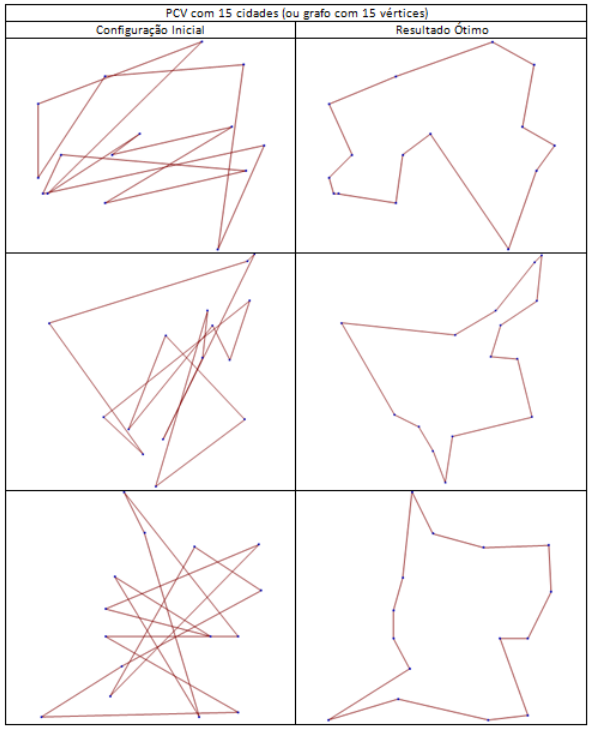
O Problema do Caixeiro Viajante (PCV)

Complexidade

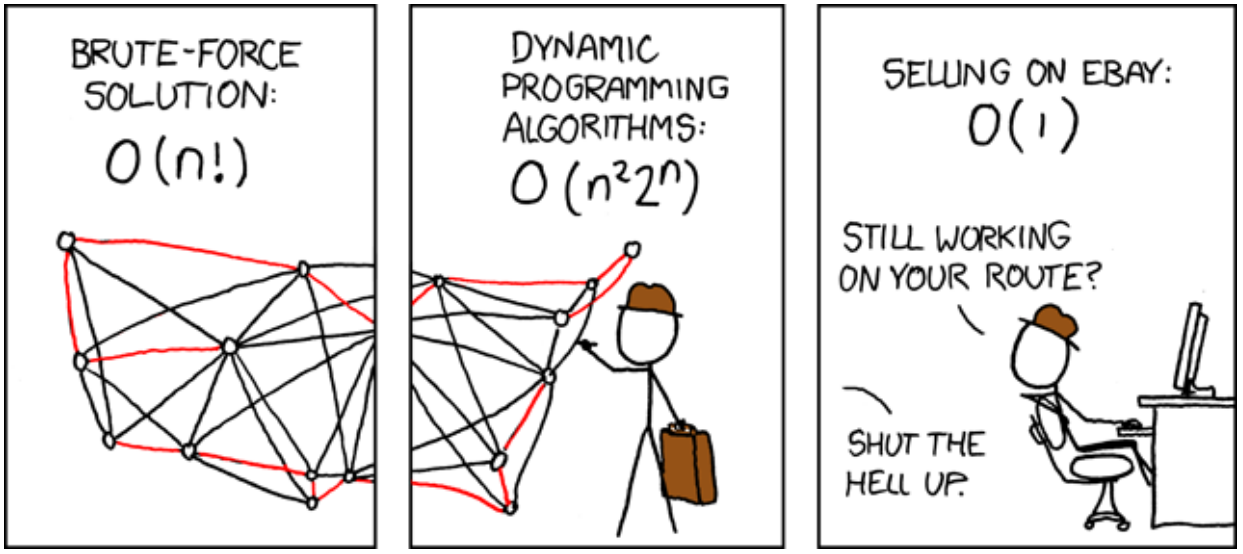
	Tempo (ms)
Número de Cidades	Força Bruta
13	743691
12	53093
11	4056
10	331
9	39
8	2
7	1

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





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



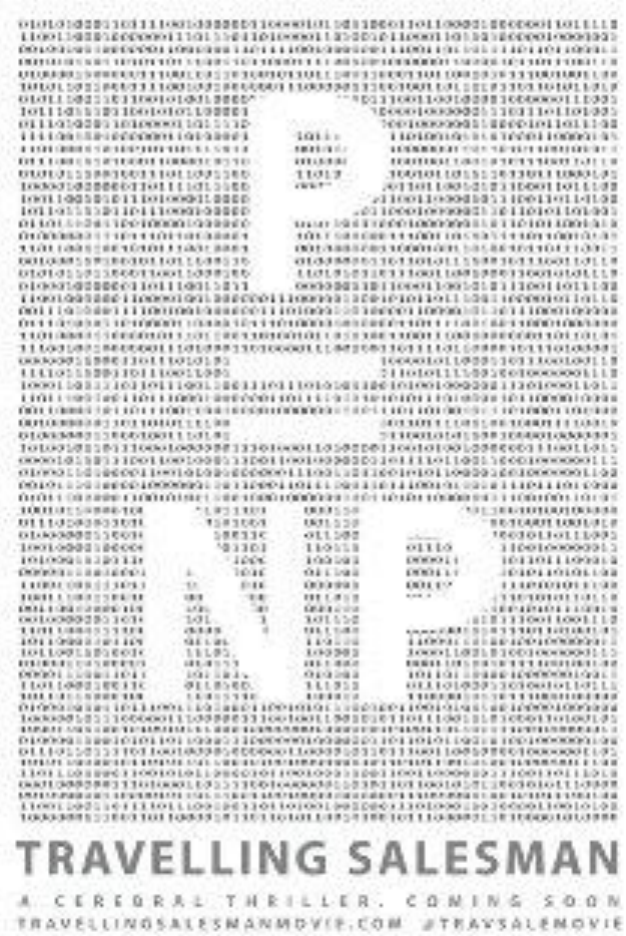
Travelling Salesman Problem XKCD
http://xkcd.com/399/

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)



http://www.travellingsalesmanmovie.com/

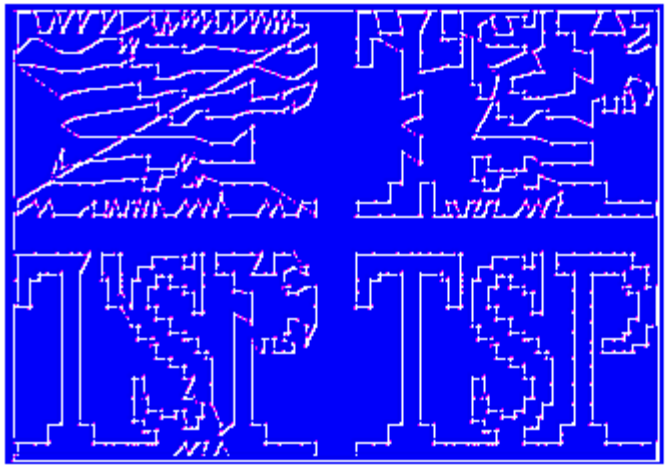
JULIAN LETHBRIDGE

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

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

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/

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

Minha Abordagem ao Problema

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

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

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

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

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

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

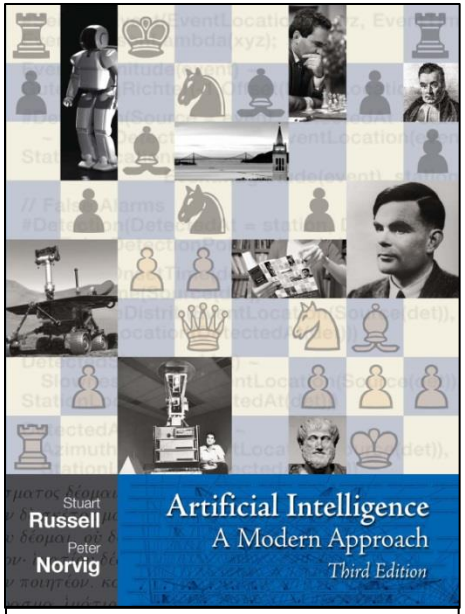
Resolução de Problemas através de Busca

Simulated Annealing

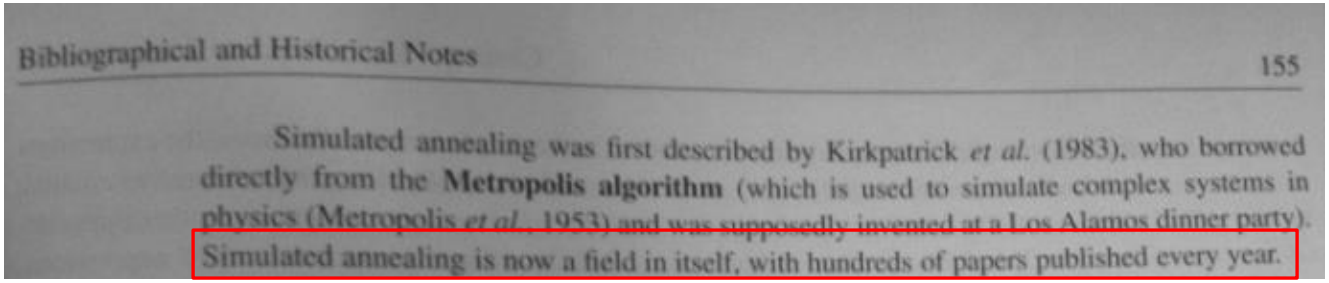
```
void simulated_annealing(const double initial_temperature,
                         const double stopping_criteria_temperature,
                         const double decreasing_factor,
                         const int monte_carlo_steps,
                         tsp_class& tsp_instance,
                         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;
    }
}
```

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

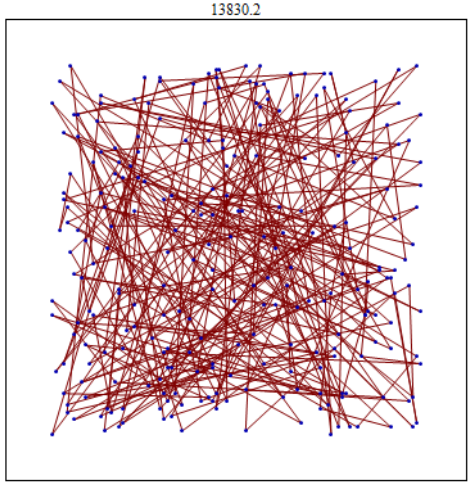
Richard Wesley Hamming

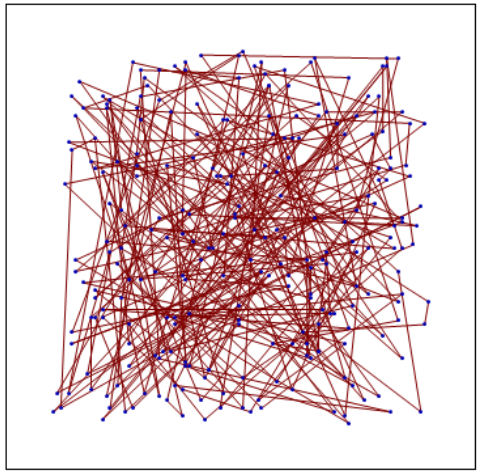
Turing Award 1968

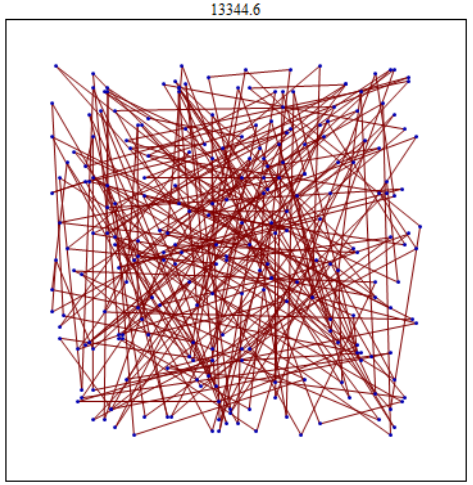
Composição de soluções

random250 (250 cidades)

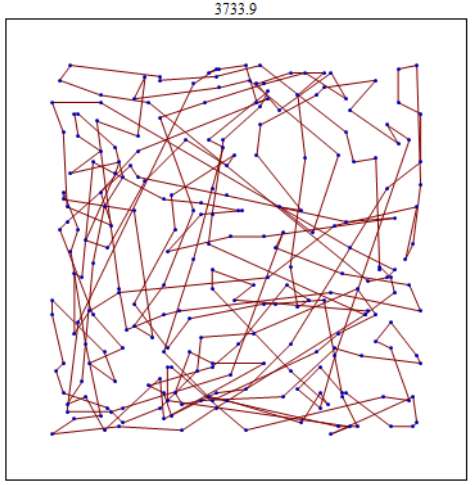
Estado inicial

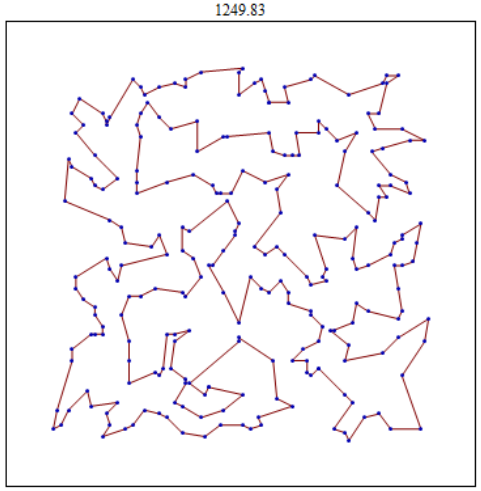


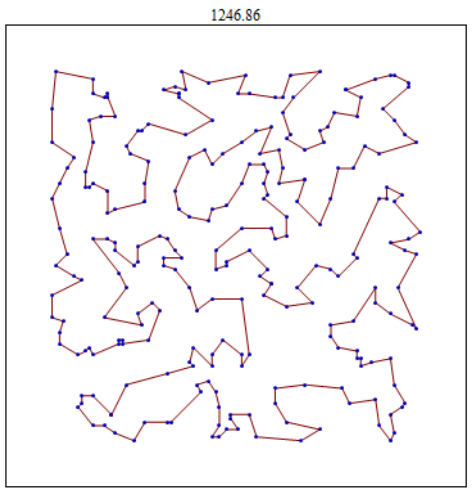




Estado final







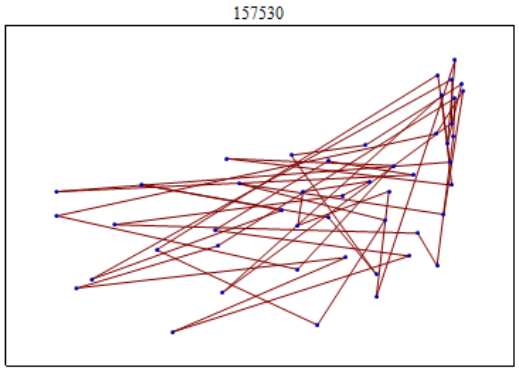
SA | 2-Opt | SA ∘ 2-Opt

In[21]:= Gamma[250]

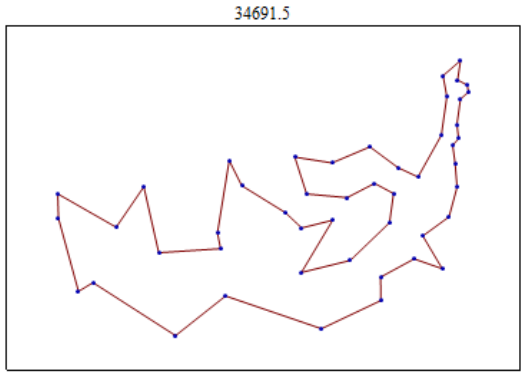
Out[21]= 12 931 425 043 636 430 929 283 258 208 097 473 883 979 374 870 695 122 666 991 769 708 451 294 990 220 444 837 955 271 661 484 112 797 803 714 029 412 757 731 706 923 726 851 235 963 291 164 121 ↓
116 285 127 686 706 108 960 757 058 932 872 164 745 044 027 331 701 460 534 715 756 358 279 742 854 120 700 162 052 715 040 308 991 732 261 609 356 024 659 302 208 995 277 746 290 344 229 596 890 ↓
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 ↓
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

Resultados

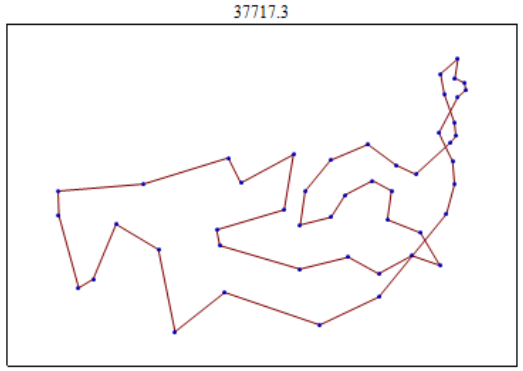
att48.tsp (48 cidades)



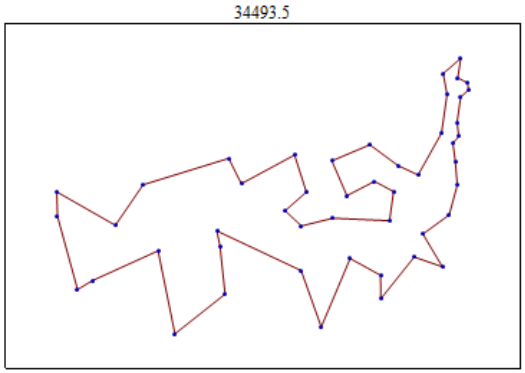
Estado Inicial



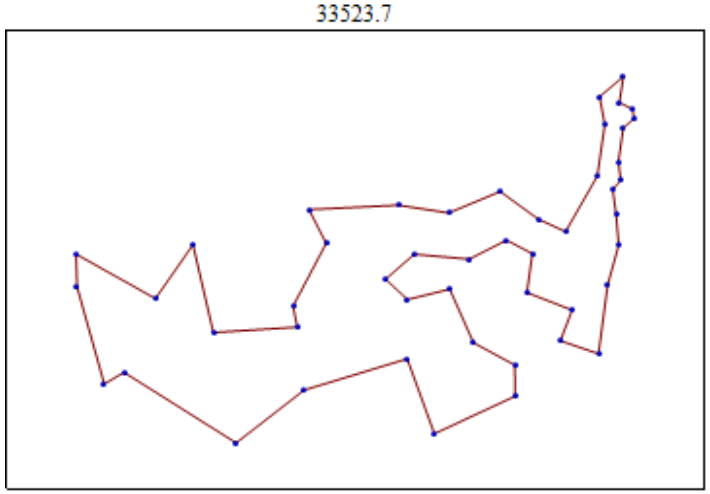
2-opt (8 tarefas)



SA (8 tarefas)



SA ∘ 2-Opt (8 tarefas)
1 geração

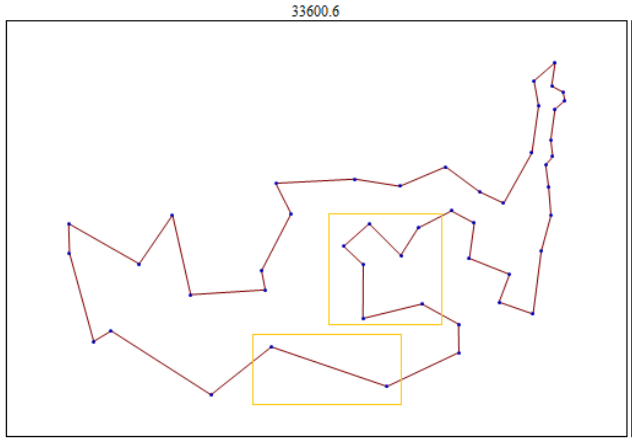


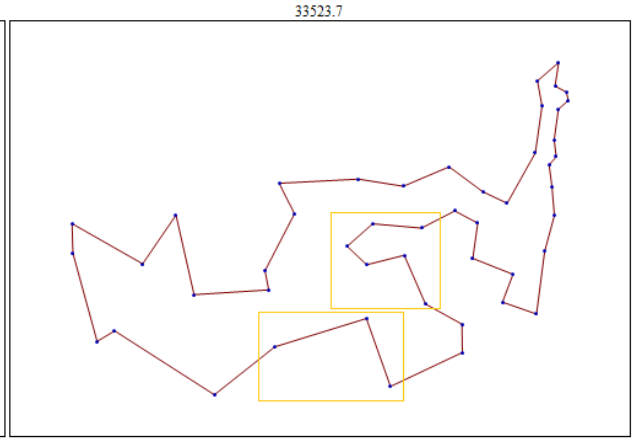
SA ∘ 2-Opt (8 tarefas)
10 gerações – resultado ótimo
encontrado na geração 7

```
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
```

Resultados

att48.tsp (48 cidades) – ajuste fino

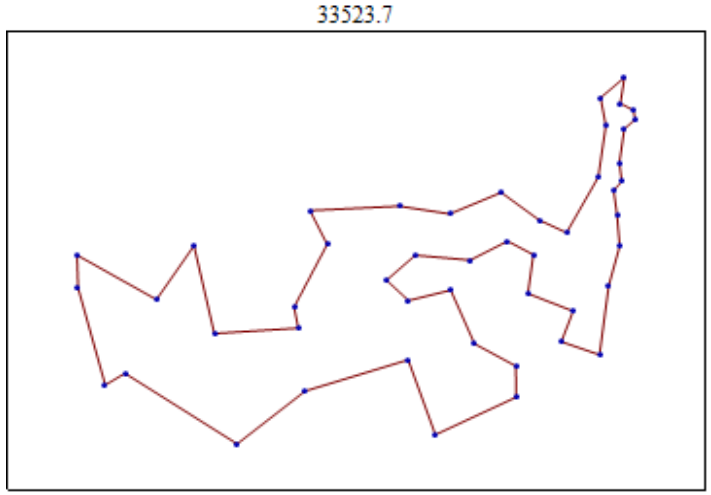





```
simulated_annealing(100.0, 0.0001, 0.999, 200, …);
simulated_annealing(1000.0, 0.00001, 0.999, 400, …);
```

```
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;

    friend auto operator==(const tsp_monad& lhs, const tsp_monad& rhs) { ... }

    friend auto operator!=(const tsp_monad& lhs, const tsp_monad& rhs) { ... }

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

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

    friend auto unit(const value_type& value) -> tsp_monad;

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

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

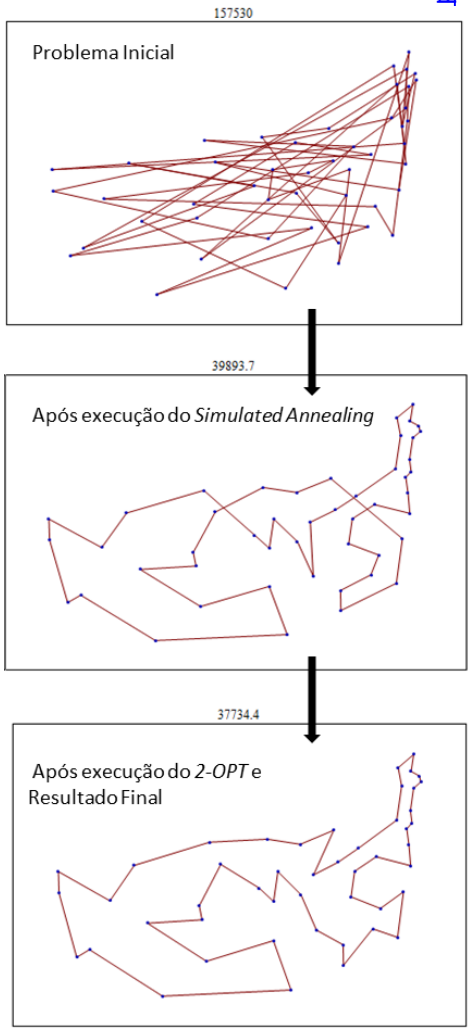
```
//Scala notation: m flatMap f flatMap g == m flatMap (x => f(x) flatMap g)
{
    //Associativity (3rd Law)
    auto m = make_TSP(read_att48_tsp());

    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

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

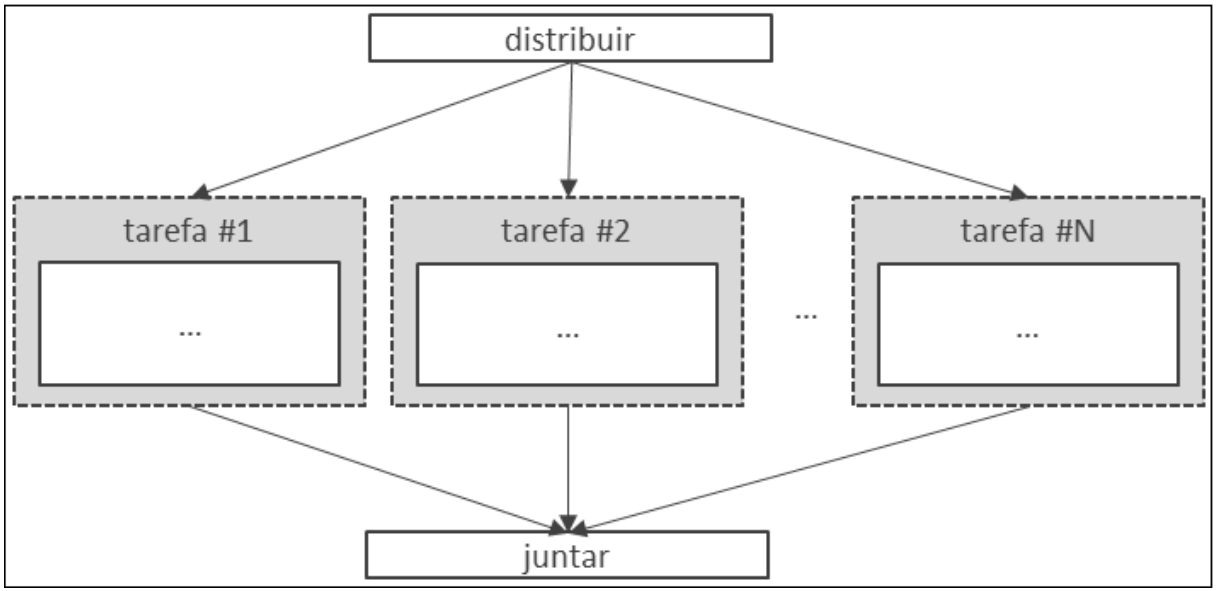
TSP -> SA -> 2-OPT -> TSP'


```
//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*



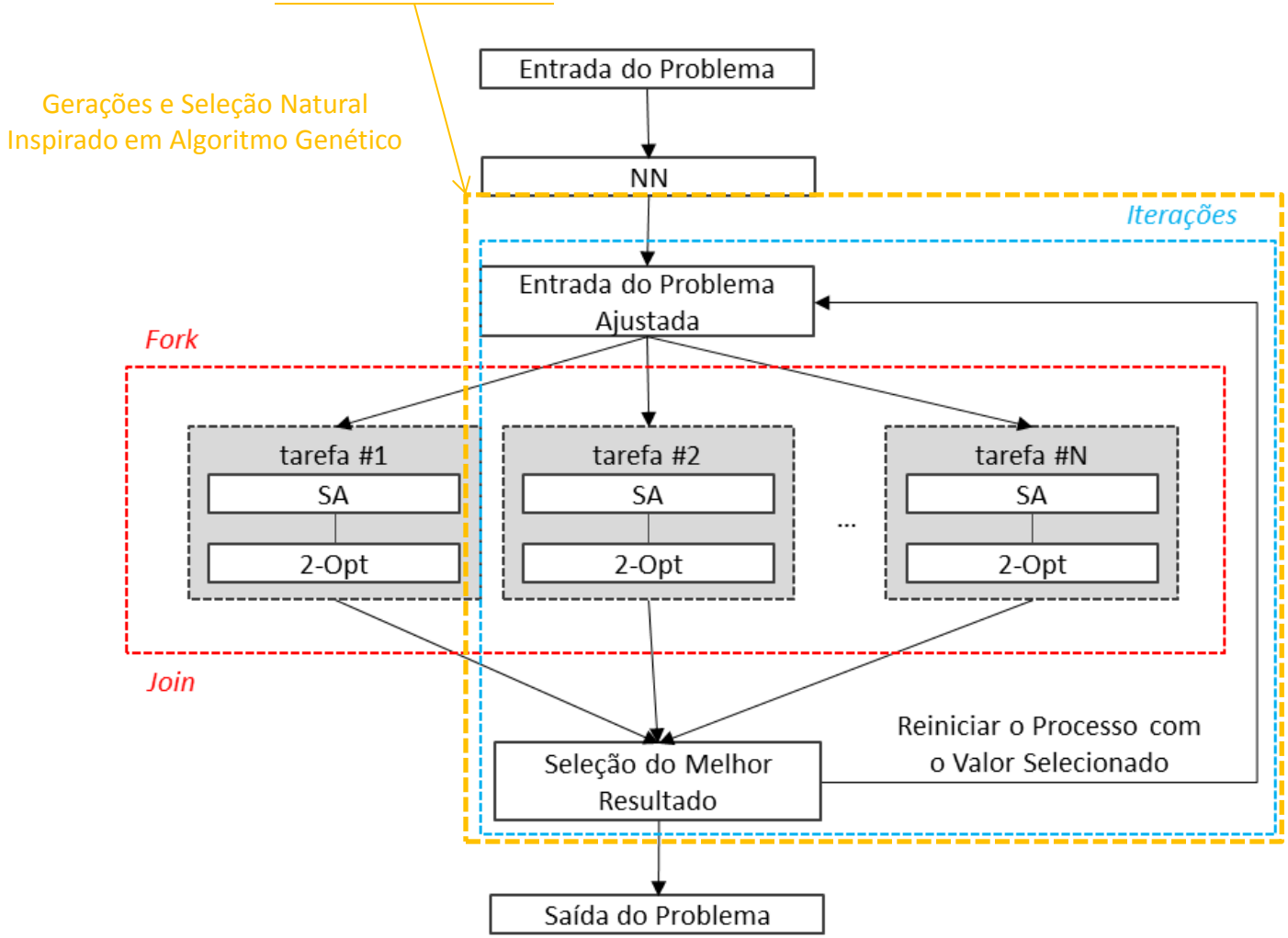
- Paralelismo e Composição

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

Exemplo de um Modelo Composicional

Modelo Composicional e Arquitetura

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



Considerações sobre o Modelo Composicional

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

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

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

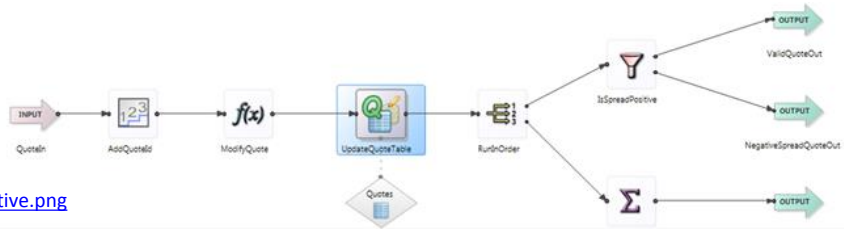
$$f_3 : TSP \rightarrow Generations(g''', ForkJoin(n''', ACO)) \rightarrow TSP^*$$

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

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

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

Empresas investem em produtos que é baseado neste conceito de composição

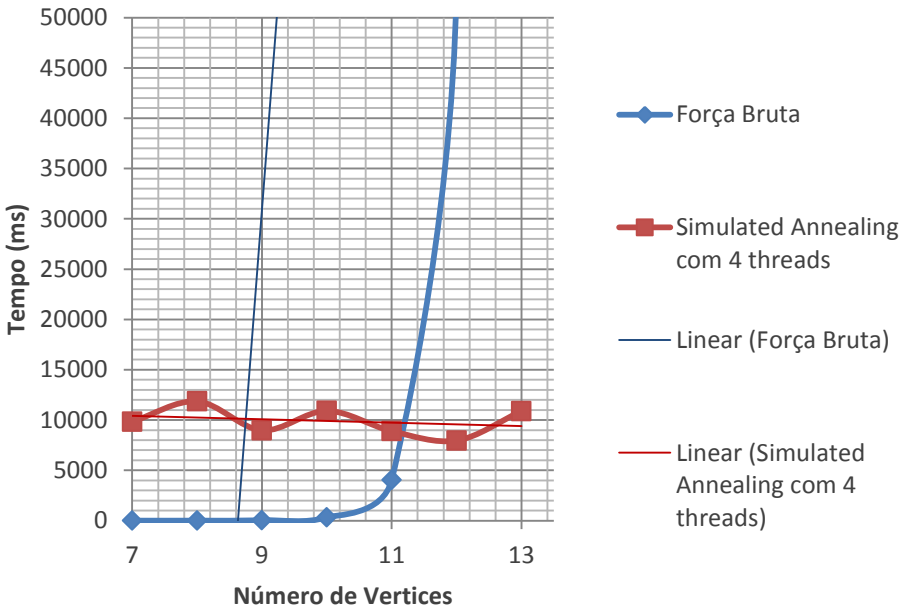


http://www.streambase.com/wp-content/gallery/streambase-screen-shots/streambase_studio_authoring_perspective.png

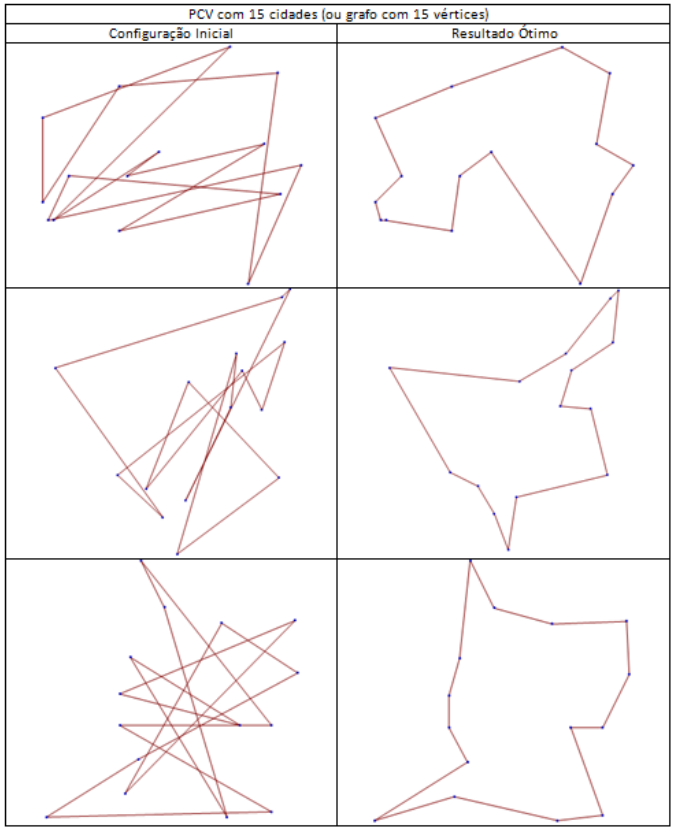
Resultados

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

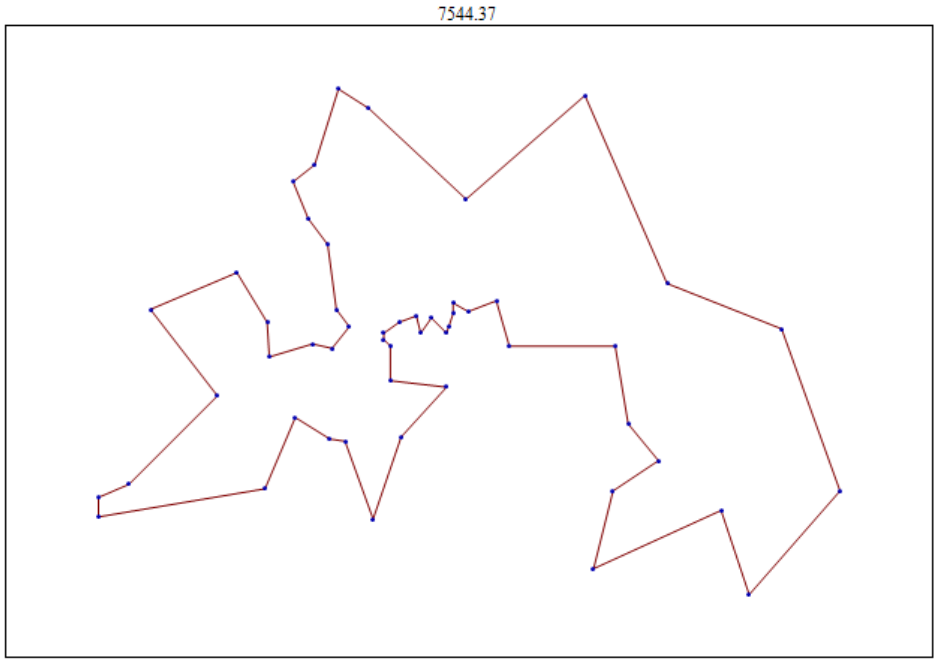


Resultados

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}}]
```

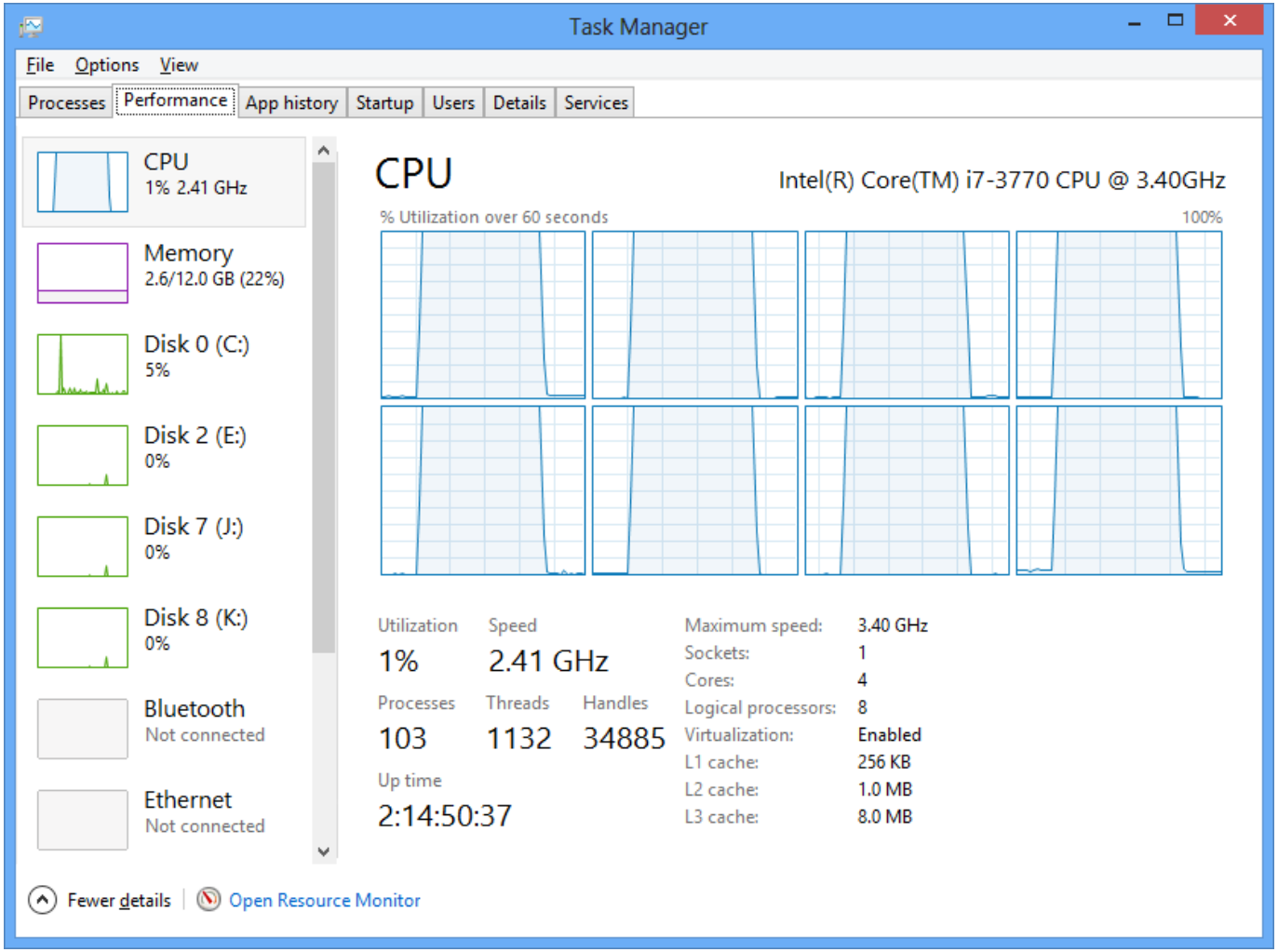
Out[4]=




```
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



berlin52.txt (312.1 KB) - BareTail

File Edit View Preferences Help

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

tspant.txt | tspga.txt | tspga_kroa100.txt | tsp_result.txt | tsp_result_pr1002.txt | result_kroB200_p3.tsp_32_2.txt | result_kroB200_p3.tsp_2_2.txt | result_kroB200_p3.tsp_4_2.txt | result_kroB200_p3.tsp_8_2.txt | result_kroB200_p3.tsp_16_2.txt | berlin52.txt | berlin52-2.txt | teste.txt | berlin52_0.txt | berlin52_1.txt | berlin52_20140906-1057.txt | berlin52_2.txt | berlin52.txt | berlin52.txt

```
[1](565, 575) : [3](575, 665) : [2](605, 625) : [33](685, 610) : [34](685, 595) : [35](700, 580) : [32](720, 635) : [31](760, 650) : [30](770, 610) : [29](795, 645) : [26](845, 655) : [25](845, 680) : [10](875, 920) : [23](945, 685) : [24](880, 660) : [27](835, 625) : [28](830, 610) : [22](975, 580) : [21](1220, 580) : [20](1250, 400) : [37](830, 485) : [36](
[1](565, 575) : [52](520, 585) : [3](575, 665) : [32](720, 635) : [31](760, 650) : [30](770, 610) : [37](830, 485) : [17](1170, 65) : [15](1530, 5) : [14](1740, 245) : [16](1465, 200) : [19](1320, 315) : [18](1215, 245) : [20](1250, 400) : [13](1605, 620) : [12](1340, 725) : [21](1220, 580) : [22](975, 580) : [23](945, 685) : [24](880, 660) : [26](845, 655) :
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [48](145, 665) : [49](345, 750) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580, 1175) : [9](650, 1130) : [11](1150, 1160) : [12](1340, 725) : [13](1605, 620) : [14](1740, 245) : [15](1530, 5) : [16](1465, 200) : [17](1170, 65) : [18](1215, 245) :
[1](565, 575) : [36](700, 500) : [38](725, 370) : [40](595, 360) : [41](560, 365) : [42](480, 415) : [39](660, 180) : [43](410, 250) : [44](25, 185) : [45](25, 230) : [46](95, 260) : [47](300, 465) : [51](420, 555) : [52](520, 585) : [50](415, 635) : [48](145, 665) : [49](345, 750) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580, 11
[1](565, 575) : [3](575, 665) : [2](605, 625) : [33](685, 610) : [34](685, 595) : [35](700, 580) : [36](700, 500) : [37](830, 485) : [22](975, 580) : [27](835, 625) : [28](830, 610) : [26](845, 655) : [25](845, 680) : [10](875, 920) : [23](945, 685) : [24](880, 660) : [20](1250, 400) : [19](
[1](565, 575) : [52](520, 585) : [3](575, 665) : [2](605, 625) : [34](685, 595) : [35](700, 580) : [33](685, 610) : [32](720, 635) : [31](760, 650) : [29](795, 645) : [28](83
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [49](345, 750) : [48](145, 665) : [47](300, 465) : [46](95, 260) : [45](25, 230) : [44](25, 185) : [43](410
[1](565, 575) : [52](520, 585) : [50](415, 635) : [48](145, 665) : [45](25, 230) : [44](25, 185) : [46](95, 260) : [47](300, 465) : [51](420, 555) : [42](480
[1](565, 575) : [52](520, 585) : [3](575, 665) : [32](720, 635) : [31](760, 650) : [29](795, 645) : [30](770, 610) : [37](830, 485) : [18](1215, 245) : [17](1170, 65) : [15](
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [48](145, 665) : [49](345, 750) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580,
[1](565, 575) : [36](700, 500) : [37](830, 485) : [22](975, 580) : [18](1215, 245) : [19](1320, 315) : [20](1250, 400) : [21](1220, 580) : [12](1340, 725) : [13](1605, 620) :
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [49](345, 750) : [48](145, 665) : [47](300, 465) : [46](95, 260) : [45](25, 230) : [44](25, 185) : [43](410
[1](565, 575) : [3](575, 665) : [4](555, 815) : [5](510, 875) : [6](475, 960) : [7](525, 1000) : [8](580, 1175) : [9](650, 1130) : [11](1150, 1160) : [12](1340, 725) : [13](1605, 620) :
[1](565, 575) : [3](575, 665) : [2](605, 625) : [34](685, 595) : [35](700, 580) : [36](700, 500) : [40](595, 360) : [41](560, 365) : [39](660, 180) : [38](725, 370) : [37](83
[1](565, 575) : [52](520, 585) : [51](420, 555) : [50](415, 635) : [49](345, 750) : [48](145, 665) : [47](300, 465) : [46](95, 260) : [45](25, 230) : [44](25, 185) : [43](410
[1](565, 575) : [2](605, 625) : [3](575, 665) : [4](555, 815) : [5](510, 875) : [9](650, 1130) : [8](580, 1175) : [7](525, 1000) : [6](475, 960) : [49](345, 750) : [48](145,
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

File Options View

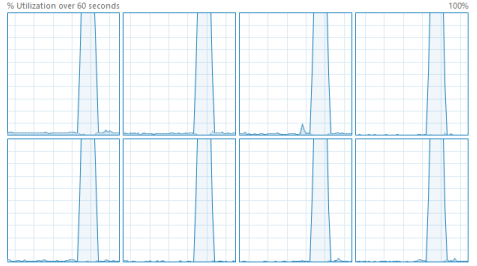
Processes | Performance | App history | Startup | Users | Details | Services

CPU 1% 3.26 GHz
Memory 4.4/12.0 GB (37%)
Disk 0 (C:) 0%
Disk 1 (E:) 0%
Ethernet S: 0 R: 0 Kbps
Ethernet S: 0 R: 0 Kbps
Bluetooth Not connected
Ethernet Not connected
Wi-Fi S: 0 R: 0 Kbps

CPU

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

% Utilization over 60 seconds 100%



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

Fewer details | Open Resource Monitor

Visual C++ 2013 (VC++ 12.0)

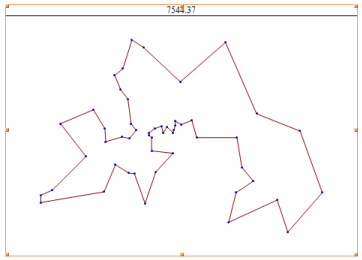
```
C:\Users\Fabio Galuppo\Documents\Visual Studio 2013\Projects\CppPlayground2\Debu
g>TSPSystem.exe -g:16 -n:16 > berlin52.txt

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>
```

Untitled-1 *

Wolfram Mathematica | HOME EDITION

Learning Center | Demonstrations | Help

```
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}}]
```



100%

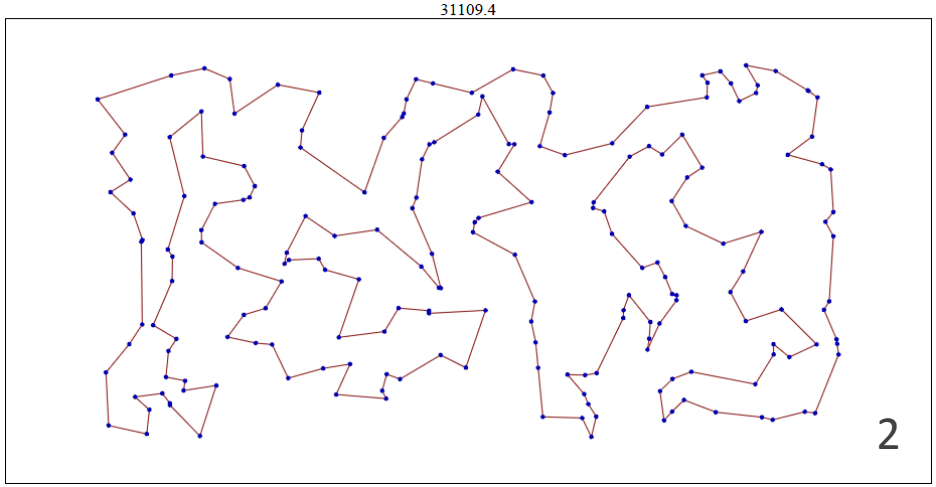
Resultados

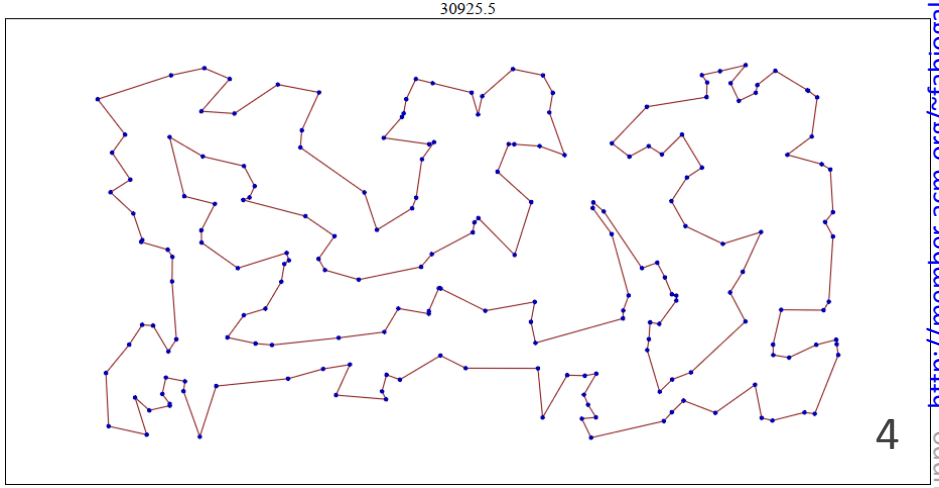
Diversos

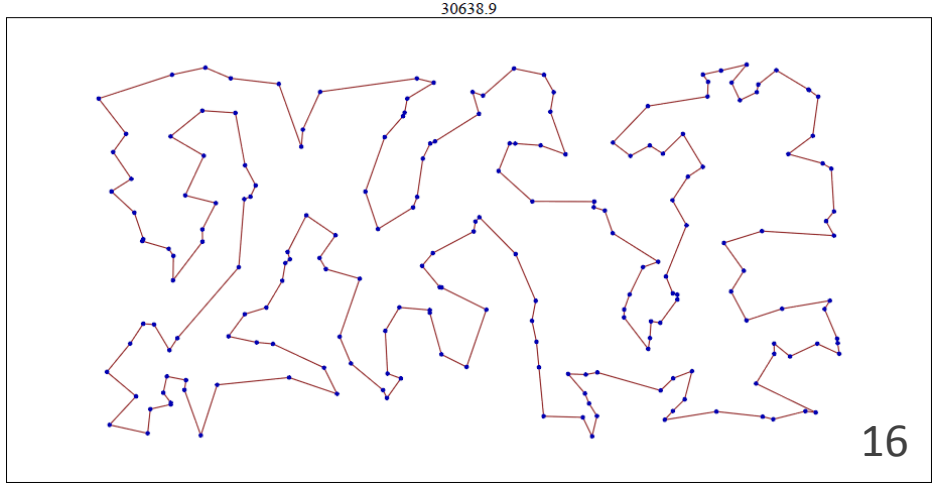
Instância TSPLIB	Número de Cidades	Distância Euclidiana (Melhor Resultado com 32 tarefas)				
		Obtido*	Literatura*	Diferença (%)	Iteração	
att48	48	33523.7	10628 (ATT) ou ~33523	0.0000	14	1
		33523.7		0.0000	9	2
		33523.7		0.0000	3	3
kroA100	100	21285.4	21282	0.0000	15	1
		21285.4		0.0000	26	2
		21285.4		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

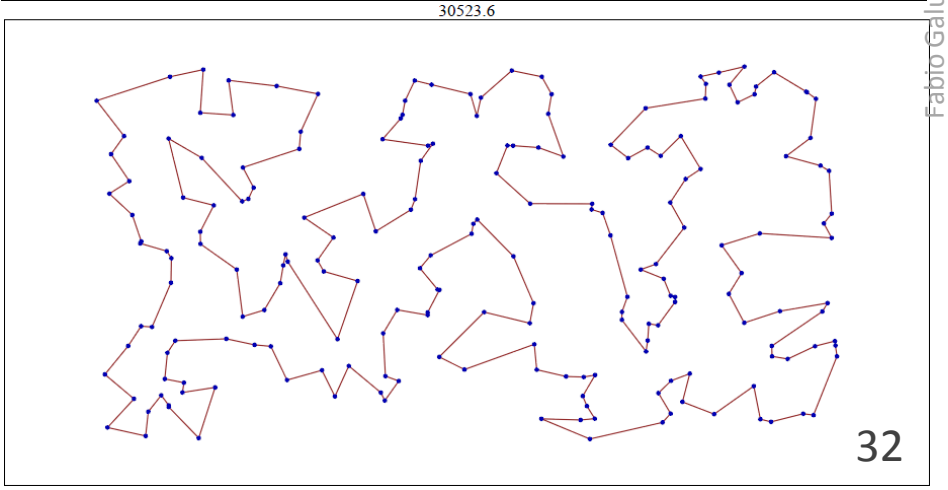
Resultados

Diversos









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 Artifi cial 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 arti ficial, problema do caixeiro viajante e otimização combinatória.

Texto completo: PDF

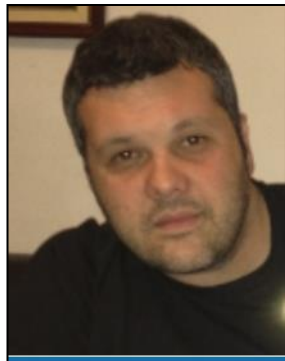
http://up.mackenzie.br/stricto-sensu/engenharia-eletrica-e-computacao/teses-e-dissertacoes-detalhada/artigo/resolucoes-do-problema-do-caixeiro-viajante-aplicando-algoritmos-de-aproximacao-randomizacao/

(Re)descobrindo o C++ com problemas NP-completos, lambdas, monads, IA e paralelismo

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

fabiogaluppo@acm.org



First year awarded:
2002

Number of MVP Awards:
11

Technical Expertise:
Visual C++

Technical Interests:
Visual C#, Visual F#