Implementing efficient code without dying in the effort

Jesús Sánchez-Oro
Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclusions
Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclusions
Why I started working in routing problems?
Motivation

• Two metrics are considered to evaluate the **quality** of an algorithm:
  • **Objective function** value
  • Computing **time**
How relevant is the programmer?

• We start from a **high quality algorithm**.
• Otherwise, the programmer has nothing to do.
How relevant is the programmer?

• If the algorithm is good, but the programmer is not...
How relevant is the programmer?

• If the algorithm is good, and the programmer is reasonable …
How relevant is the programmer?

- If both the algorithm and the programmer are excellent …
How relevant is the programmer?

- If the programmer is trying *new things* ...
Select a programming language

- VEROLOG 2019 (SEVILLE)
Which is the best programming language?

• The best programming language **does not exist**
  • Otherwise, all of us will use the same language

• What are we looking for in a programming language?
  • Easy to learn
  • Performance
  • Debugging
  • External libraries
Why did I choose Java?

• It is **easy to learn** Java from scratch.
• JVM is responsible for **memory management**.
• Designed for **Object Oriented Programming**.
• A good code in Java **is not necessarily slower** than one in C/C++.
Why did I choose Java?

- **Developing time** in Java is rather smaller than in other languages.
- It has a lot of **external libraries** to help us with the code.

<table>
<thead>
<tr>
<th>May 2019</th>
<th>May 2018</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Java</td>
<td>16.005%</td>
<td>-0.38%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>C</td>
<td>14.243%</td>
<td>+0.24%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>C++</td>
<td>8.095%</td>
<td>+0.43%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>Python</td>
<td>7.830%</td>
<td>+2.64%</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>↑</td>
<td>Visual Basic .NET</td>
<td>5.193%</td>
<td>+1.07%</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>↓</td>
<td>C#</td>
<td>3.984%</td>
<td>-0.42%</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>↑</td>
<td>JavaScript</td>
<td>2.690%</td>
<td>-0.23%</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>↑</td>
<td>SQL</td>
<td>2.555%</td>
<td>+0.57%</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>↓</td>
<td>PHP</td>
<td>2.489%</td>
<td>-0.83%</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>↑</td>
<td>Assembly language</td>
<td>1.816%</td>
<td>+0.82%</td>
</tr>
</tbody>
</table>

[https://www.tiobe.com/tiobe-index/](https://www.tiobe.com/tiobe-index/)
Outline

1 Motivation
2 Code organization
3 Data structures
4 Test Problem: TSP
5 Code improvements
6 Parallelization
7 Conclusions
Code structure

• When we deal with a new problem, we first need to think about **code structure**.

• If the problem is similar to another one in which we have previously worked the **structure** will be **similar**.
Code structure

• Most of the **features** that we use for a certain problem are **repeated** for the rest of the problems.

• Is it really **necessary** to repeat the same again and again?
Code structure

- **First option**
  - *Copy and paste* the last project in which we have been working and modify the code
Code structure

• **Second option**
  • Take advantage of the **language features** in order to avoid repeating code.
Code structure

• **Proposal**: create a library which contains the basic functionality that will be required in any project.
  • Execute an algorithm over a set of instances in a folder.
  • Generate a table with the obtained results.
  • Control the computing time.
  • …
Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclusions
Data Structures

• DS define the **data organization** of our problem.

• If we choose the correct DS, we will be able to add, modify or remove data **efficiently**.

• DS are usually one of the **key parts** of our code.
Data Structures

• Most languages offer their own implementations of several data structures, so we do not usually need to implement data structures.
Data Structures

• However, if we need more **complex or specific** structures, we will need to go deeper and implement them.
• We usually believe that the programming language is the **key for developing a fast algorithm**.

• Nevertheless, the actual key is the **complexity** of the data structures considered.
Data Structures

• If we perform **many operations** over the same data structure, we would like to make it as **efficient** as possible.

• We need to focus on **reducing the complexity** of the most **common** operations.
  • Which is the cost of inserting / searching / removing an element from a data structure?
Is it really so important?

• We will consider 1000 elements.

• Test:
  • Search for a random element
Why these results?

**Arrays**

- Access to a given position in *constant* time
- Improvement in *memory storage*
Why these results?

ArrayList

- Similar to arrays in representation.
- **Overhead** to resize the data structure and offer more functionality (contains).
Why these results?

**LinkedList**

- Access to the *first and last* elements on the list.
- If we need to access $k$ element, we need to move $k$ positions starting at the first one.
Why these results?

HashSet

• Each element is identified by a unique number.
  • We need to define the mapping between element and its corresponding number (hash code).

• Check if an element is in the DS in constant time.
What data structure should I use?

• There is not a best data structure.

• It totally depends on the most common operations performed in your code.

• THINK BEFORE CODING!!!
Outline

1 Motivation
2 Code organization
3 Data structures
4 Test Problem: TSP
5 Code improvements
6 Parallelization
7 Conclusions
Test Problem: TSP

• **Input**: a set of $n$ locations and the distance between each pair of locations.

• **Objective**: find the shortest possible route that visits every city exactly once and returns to the starting point.

• Starting point is always the first node.
Test Problem: TSP

• What do we have to know about the instance?
  • Number of cities
  • Distances between cities
Test Problem: TSP

- What do we have to know about the solution?
- Which instance are we solving?
- Which is the selected route?
- Which is the total distance for the route?
Test Problem: TSP

• Two basic **movements:**
  • **Swap** between two cities
  
  ![Swap(2,5)](1 2 3 4 5 6 Swap(2,5) 1 5 3 4 2 6)

  ![Insert(2,5)](1 2 3 4 5 6 Insert(2,5) 1 3 4 5 2 6)

• **Insertion** of a city in a different position
Test Problem: TSP

• **Greedy Randomized Adaptive Search Procedure**
  • Construction phase
  • Improvement phase
Test Problem: TSP

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. while $CL \neq \emptyset$ do
6. \hspace{1em} $g_{\text{max}} \leftarrow \min_{v \in CL} g(v)$
7. \hspace{1em} $g_{\text{max}} \leftarrow \max_{v \in CL} g(v)$
8. \hspace{1em} $\mu \leftarrow g_{\text{min}} + \alpha \cdot (g_{\text{max}} - g_{\text{min}})$
9. \hspace{1em} $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
10. \hspace{1em} $v_s \leftarrow \text{Random}(RCL)$
11. \hspace{1em} $S \leftarrow S \cup \{v_s\}$
12. \hspace{1em} $CL \leftarrow CL \setminus \{v_s\}$
13. endwhile
14. $\text{return } S$
Test Problem: TSP

1. \( CL \leftarrow \{ v \in V \} \)
2. \( v_f \leftarrow \text{Random}(CL) \)
3. \( S \leftarrow \{ v_f \} \)
4. \( CL \leftarrow CL \setminus \{ v_f \} \)

5. while \( CL \neq \emptyset \) do
6. \( g_{\text{max}} \leftarrow \min_{v \in CL} g(v) \)
7. \( g_{\text{max}} \leftarrow \max_{v \in CL} g(v) \)
8. \( \mu \leftarrow g_{\text{min}} + \alpha \cdot (g_{\text{max}} - g_{\text{min}}) \)
9. \( RCL \leftarrow \{ v \in CL : g(v) \leq \mu \} \)
10. \( v_s \leftarrow \text{Random}(RCL) \)
11. \( S \leftarrow S \cup \{ v_s \} \)
12. \( CL \leftarrow CL \setminus \{ v_s \} \)
13. endwhile
14. return \( S \)

The **Candidate List** contains all the nodes but the first one, which is randomly chosen.
Test Problem: TSP

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. while $CL \neq \emptyset$ do
   6. $g_{\text{max}} \leftarrow \min_{v \in CL} g(v)$
   7. $g_{\text{max}} \leftarrow \max_{v \in CL} g(v)$
   8. $\mu \leftarrow g_{\text{min}} + \alpha \cdot (g_{\text{max}} - g_{\text{min}})$
   9. $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
10. $v_s \leftarrow \text{Random}(RCL)$
11. $S \leftarrow S \cup \{v_s\}$
12. $CL \leftarrow CL \setminus \{v_s\}$
13. endwhile
14. return $S$

The Restricted Candidate List contains all the nodes whose objective function value is better than a certain threshold.
Test Problem: TSP

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. while $CL \neq \emptyset$ do
6. $g_{max} \leftarrow \min_{v \in CL} g(v)$
7. $g_{max} \leftarrow \max_{v \in CL} g(v)$
8. $\mu \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$
9. $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
10. $v_s \leftarrow \text{Random}(RCL)$
11. $S \leftarrow S \cup \{v_s\}$
12. $CL \leftarrow CL \setminus \{v_s\}$
13. endwhile
14. return $S$

A random node from the RCL is selected as the next city of the route, updating the $CL$
Improvement phase

• We test two local search methods, one for each movement.
• First improvement approach.
• Random exploration of the neighborhood.
Test Problem: TSP

• Let’s test a **direct implementation** without any improvement.

• We will try different data structures to represent a route.
• This code is **too slow!!**

• Bad performance is usually related to **repeating computations unnecessarily.**

• It is very common in the **objective function evaluation.**
Code improvements

• Is it really necessary to evaluate the **complete objective function** after performing a **single movement**?
Code improvements

• We need to study which evaluations are strictly necessary to save computing time.

• For instance, in the TSP:
  • How can I update the total distance when adding a new city?
  • How does a movement affect the objective function value?
Code improvements

- When **adding** a new city:
Code improvements

• When **adding** a new city:

![Diagram](image-url)
Code improvements

- When **swapping** two cities:
  - Swap(2,3)
Code improvements

• When **swapping** two cities:
  • Swap(2,3)
Code improvements

• When **inserting** a city in a different position:
  • Insert(3,2)
**Code improvements**

- When **inserting** a city in a different position:
  - Insert(3,2)
Code improvements

• We must analyze the **complexity** of the most common operations in the data structures.

• Complexity of adding / removing elements in:
  • ArrayList
  • LinkedList
Code improvements

- LinkedList should be the best data structure for the problem.
- However, Java implementation of LinkedList offers poor performance.
Code improvements

• Is it enough for us?
Code improvements

• What if **we implement** a new LinkedList which overcomes the disadvantages of the original one?
• MyLinkedList uses two integer arrays to represent a route:
  • $\text{prev}[v]$ indicates the city located just before $v$
  • $\text{next}[v]$ indicates the city located just after $v$

• All the operations are performed in \textbf{constant time}. 
Code improvements

prev

next

0 1 2 3 4 5

0 0 4 1 5 3

0 3 1 5 2 4
Parallelization

• Most of our computers have more than one core.
  • If not, please go now and renew your computer.
• Then, why are we still developing sequential code?
Parallelization

• In a sequential program we have a single process and a single control flow.

• In a parallel program we have two or more processes cooperating to finish a task.
  • We must ensure a correct communication and synchronization among processes.
Parallelization

• **Be careful!** Power is nothing without control.
• We should learn how to code parallel programs.
• Otherwise, it could be **slower than the sequential version**
Memory model

- We use a **shared memory model**.
- A **single memory** is shared among all the processors

![Diagram of shared memory model]

- **CPU**: Central Processing Unit
- **Cache**
- **Main memory**
How can I parallelize code?

• Using a compiler that automatically convert sequential code in parallel code.

• Advantages:
  • We do nothing.

• Disadvantages:
  • Not available for all programming languages.
  • The parallelization achieved is not the best one.
How can I parallelize code?

• Using **operating system resources**: processes, threads, semaphores, files, ...

• Advantages:
  • Available in every programming language.

• Disadvantages:
  • Ridiculously hard
How can I parallelize code?

• Using **libraries** that simplify the parallelization, like OpenMP.

• Advantages:
  • We just need to slightly modify our sequential code.

• Disadvantages:
  • Not available for every language.
How can I parallelize code?

• Using a **programming language** prepared for parallelism.

• Advantages:
  • Every modern computer language is prepared for it.

• Disadvantages:
  • We need to deeply modify our code.
Parallelization in Java

• Java is prepared for developing parallel code **easily**.

• We can use the low level tools, but Java offers a set of **high level tools** to parallelize code ignoring details.
How can I parallelize a metaheuristic?

- Parallelize independent code fragments, **without algorithm redesign**.
  - Small scientific contribution.
  - Very easy.

- **Redesign the algorithm** to make the most of available hardware.
  - Relevant scientific contribution.
  - Harder.
Java Thread Pool

Task Queue

Thread Pool

Completed Tasks
Want more parallelism?

• The second option implies **redesigning the algorithm** in a parallel way.

• Most of the metaheuristics already have a **parallel design**.
  
Parallelization objectives

• Parallelizing **does not necessarily** implies reducing computing time.

• It can be also used for exploring a **wider portion** of the **search space**.
  • We can guide the search in several directions simultaneously, instead of following a single direction.
Which language should I use?

- Look for:
  - Smooth learning curve.
  - Efficiency.
  - External libraries.
  - Documentation, support forums, …
  - Parallelizing possibilities.
  - Is it used in the heuristics community?
How should I organize the code?

• We must **waste time** deciding the structure of our code.

• If we usually work on similar problems, we should think about **developing our own library** to avoid repeating common tasks.
Which is the key to efficiency?

• We should know the **data structures** that we use.

• **Incremental evaluation** of the objective function is one of the first optimizations to consider.

• Can I use an **alternative objective function**?

• Should I try a **parallel design**?
Thanks!!

T. HANKS
Implementing efficient code without dying in the effort

Jesús Sánchez-Oro

Universidad Rey Juan Carlos