INFO-H-413 - Learning Dynamics: Implementation Exercise #2 The Traveling Salesman Problem with Time Windows - Stochastic Local Search Algorithms

Prof. T. Stüetzle

Jacopo De Stefani

April 29, 2013

Contents

Implementation

The heuristic solver had been written using the C++ programming language, in order to be able to take adventage of the functionalities offered by the Standard Template Library (STL).
By combining an object-oriented approach with those functionalities, I built a modular application core (*HeuristicCore*) that has been used for the two exercises required for this implementation.
The core is directly linked with the reader (*InstanceReader*) which reads the files containing all the informations to run the simulation (distance matrix, time windows vector and seeds list) and the writer (*Writer*) which outputs the results of the simulation in a CSV file.
Two different command line front-ends, sharing the same underlying structure, but having a different set of parameters, have been developed to distinguish the iterative-improvement interface from the variable neighborhood descent one.
The global program structure is depicted here:

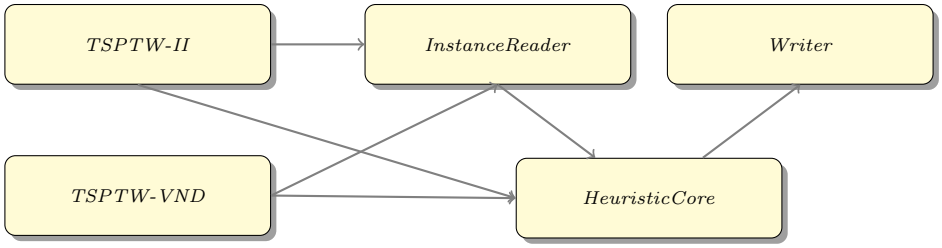


Figure 1: Simulation Code Structure

The names of the nodes corresponds to those of the implementation (.cpp) and header files for the corresponding classes. In addition to these, a class to represent the time windows (TimeWindow.h) and another one to represent the candidate solution as a user defined data type (CandidateSolution.{h,cpp}) have been implemented.

How to compile the program?

The software was developed in C++98 under Linux (Debian Wheezy), using the GNU g++ (Debian 4.7.2-5) 4.7.2 compiler and tested in this environment. The software is distributed as a compressed tar file, containing both the source code (in the `src` folder) and the scripts and instances (in the `bin` and `bin\instances` respectively). To install the program, first obtain the file TSPTW.V1.0.tar.gz.
Unzip the file by typing:

```
gunzip TSPTW.V1.0.tar.gz
```

and then unpack it by typing:

```
tar -xvf TSPTW.V1.0.tar
```

The software will be extracted in a new folder TSPTW.V1.0
Finally, by launching

```
make all
```

the Makefile will trigger the compilation of the files, producing the executables 'TSPTW-II' and 'TSPTW-VND" in the `bin` folder.
Note: The code is written in C++98. Hence, the code should be reasonable portable to other Operating Systems than Linux or Unix.

How to run the program?

Once the program has been compiled, two separate executable files, corresponding to the different kind of metaheuristic (i.e. Iterative Improvement and Variable Neighborhood Descent) can be either launched directly via the command line interface or using the bash script to launch.
The design choice to separate the two different kind of metaheuristic is made in order to limit the number of command line parameters to be given as input to the program.

Command-line execution

By launching[1]:

```
./TSPTW-II [PARAMETERS] -i [INPUT_FILE] -s [SEEDS_FILE]
```

```
./TSPTW-VND [PARAMETERS] -i [INPUT_FILE] -s [SEEDS_FILE]
```

one can display the information about the meaning of the different options that can be given as input to the program.
Additional information concerning the usage of the program can be found in the README file.
The only mandatory options are "-i, –input" and "-s,-seeds", since without these components will not be possible to execute the simulation.
The argument to the input option must contain the full path to the instance file.
The seeds file must contain a number of seeds at least equal to the number of runs required, one seed per line without any additional information.
The options controlling the same parameters are mutually exclusive, with that meaning that only one option must be selected in order to run the program. If multiple options for the same parameter are chosen, the program will not run.

TSPTW-II

Parameter	Mutually exclusive options
Initial solution	-d,–random , -h,–heuristic
Neighborhood type	-t,–transpose , -e,–exchange , -n,–insert
Pivoting rule	-f,–first-imp , -b,–best-imp

TSPTW-VND

Parameter	Mutually exclusive options
VND algorithm	-t,–standard , -p,–piped
Neighborhood chain	-a,–TEI , -b,–TIE

Output Every experiment produces a single file:

- **(II)** $[pivoting\_rule].[neighborhood\_type].[INSTANCE\_NAME]$ with $[pivoting\_rule] \in \{first, best\}$ and $[neighborhood\_type] \in \{transpose, exchange, insert\}$
- **(VND)** $[vnd\_type].[neighborhood\_chain].[INSTANCE\_NAME]$ with $[vnd\_type] \in \{standard, piped\}$ and $[neighborhood\_type] \in \{tei, tie\}$

Examples:

```
exchange.best.n80w200.004.txt
standard.tei.n80w20.003.txt
```

The internal structure of the file is the following:

Seed	CV	CpuTime	PRPD

For each run of the algorithm, the program writes in the file, using a tabulation as separator, the used seed, the number of constraints violations, the cpu runtime and the penalised relative percentage deviation.

[1] The squared-bracket notation implies that the formal parameters to the script have to be substituted by their actual value.

Script execution

By launching:

```
./launchTSPTW-II.sh [INSTANCE_NAME] [RUNS] [SEEDS_FILE]
```

```
./launchTSPTW-VND.sh [INSTANCE_NAME] [RUNS] [SEEDS_FILE]
```

The script will:

1. Create the files required by the data processing script to write statistics.
2. Generate a seed file, named [SEEDS_FILE] containing [RUNS] randomly generated seeds.
3. Launch the corresponding TSPTW program for [RUNS] using all the possible combinations of input options.
4. Wait for the termination of all the previously launched experiment and call the R data processing script

The script assumes that all the instances are located into the instances folder, hence it is only necessary to indicate the instance name, instead of the complete path.

Output Each execution of the script will then generate the files:

- $[INSTANCE\_NAME] - CpuTime.pdf$ containing the boxplots of the runtime distribution for each algorithm.
- $[INSTANCE\_NAME] - PRPD.pdf$ containing the boxplots of the PRPD distribution for each algorithm.
-
  - **(II)** transpose.first, exchange.first, insert.first, transpose.best, exchange.best, insert.best
  - **(VND)** standard.tei, standard.tie, piped.tei, piped.tie

The internal structure of the file is the following:

Instance	**Infeasible**	**mean(PRDP)**	**mean(CpuTime)**

Each line contains the instance name, the percentage of infeasible runs, the mean PRDP and the mean runtime across [RUNS] runs.

Problem formulation

An instance of the Travelling Salesman Problem with Time Windows (TSPTW) can be expressed as a tuple $< N, E, c, t >$ where:

- $N : \{0, \cdots, n\}$ - Node set
- $E : N \times N$ - Edge set
- $c : E \mapsto \mathbb{R}$ - Edge cost function mapping a cost c to every edge $e \in E$.
- $t : N \mapsto \mathbb{R}^2$ - Time window function mapping a couple of values a_i, b_i representing respectively, the opening and closing time of the time window, such that $a_i < b_i$, to every node $i \in N$.
- A candidate solution for the problem, in this case, is represented as a permutation p of the nodes in N, where p_i represents the i^{th} solution component (node) in the sequence, such that $p_0 = 0 \forall p$.

As in [6] the formal definition of the problem is:

$$\begin{aligned} \textbf{min} \quad & f(p) = \sum_{i=0}^{n} c(e_{p_i p_{k+i}}) \\ \textbf{subject to} \quad & \Omega(p) = \sum_{i=0}^{n+1} \omega(p_i) = 0 \end{aligned} \tag{1}$$

where

$$\begin{aligned} \omega(p_i) &\begin{cases} 1 & A_{p_i} < b_{p_i} \\ 0 & \text{otherwise} \end{cases} \\ A_{p_{i+1}} &= \max(A_{p_i}, a_i) + c(e_{p_i p_{k+i}}) \end{aligned} \tag{2}$$

As one may notice in 1 and 2, a feasible solution p for the problem is a permutation where the constraints related to the time windows are met for every node i in the permutation.

Problem 1

Ant Colony Optimization

Problem statement

Implement two stochastic local search (SLS) algorithms for the traveling salesman problem with time windows (TPSTW), building on top of the perturbative local search methods from the first implementation exercise.

1. Run each algorithm 25 times with different random seed on each instance. Instances will be available from http://iridia.ulb.ac.be/~stuetzle/Teaching/HO/. As termination criterion, for each instance, use the maximum computation time it takes to run a full VND (implemented in the previous exercise) on the same instance and then multiply this time by 1000 (to allow for long enough runs of the SLS algorithms).
2. Compute the following statistics for each of the two SLS algorithms and each instance:
   - Percentage of runs with constraint violations
   - Mean penalised relative percentage deviation
3. Produce boxplots of penalised relative percentage deviation.
4. Determine, using statistical tests (in this case, the Wilcoxon test), whether there is a statistically significant difference between the quality of the solutions generated by the two algorithms.
5. Measure, for each of the implemented algorithms on 5 instances, the run-time distributions to reach sufficiently high quality solutions (e.g. best-known solutions available at http://iridia.ulb.ac.be/~manuel/tsptw-instances#instances). Measure the run-time distributions across 25 repetitions using a cut-off time of 10 times the termination criterion above.
6. Produce a written report on the implementation exercise:
   - Please make sure that each implemented SLS algorithm is appropriately described and that the computational results are carefully interpreted. Justify also the choice of the parameter settings and the choice of the iterative improvement algorithm for the hybrid SLS algorithm.
   - Present the results as in the previous implementation exercise (tables, boxplots, statistical tests).
   - Present graphically the results of the analysis of the run-time distributions.
   - Interpret appropriately the results and make conclusions on the relative performance of the algorithms across all the benchmark instances studied.

Introduction

Ant Colony Optimization is an example of population-based metaheuristic (i.e a set of algorithmic concepts that can be used to define heuristic methods) inspired by the behavior of the ant species *Iridomyrmex humilis*. To be more precise, these insects are able, by means of stigmergic communication, to choose the shortest path between their nest and a food source, when given the choice ([2]).
The communication process occurs by deposing a certain quantity of pheromone in the environment that can be sensed by the other ants and that will be used by them as an heuristic (i.e an information to guide their choice) for selecting the shortest path.
Furthermore, the pheromone quantity on a certain location decreases over time because of evaporation, thus requiring a continuous deposit process to be effective.

The convergence to one of the paths will occur as a consequence of the self-reinforcing pheromone deposit mechanism. In fact the more pheromone is deposited on a path, the more ants will follow the pheromone trail on that path deposing even more pheromone.
The first application of Ant Colony Optimization method, the Ant System, has been made on the optimization version of the Travelling Salesman Problem (TSP) ([4]).
In this implemetation a population of virtual agents (an ant colony) is used to explore the search space (the virtual environment).
In the same fashion as the real insects, the ants are able to deposit virtual pheromone in the environment, to signal to the other ants the presence of promising solutions.
The general outline of the implemented algorithm is the following:

Algorithm 1 Ant Colony Optimization - Outline

```
1: InitalizePheromoneTrail
2: while !(TerminationCondition) do
3:     ConstructAntsSolutions
4:     LocalSearch (Optional)
5:     UpdatePheromoneTrails
6: end while
```

The design of the solution construction and pheromone update mechanism is the main point of the algorithm. Implementation details of the basic ACO system, the Ant System can be found in [3].

Algorithm structure

The proposed algorithm is an implementation of one of the extensions to the Ant Colony Optimization metaheuristic framework, the $\mathcal{MAX}-\mathcal{MIN}$Ant System (cf. [7]). The main differences with respect to the basic ACO approach are the following:

- Only iteration best or best-so-far ants update pheromone.
- A local search after the solution generation is used to further improve the solutions found by the ants at each iteration.
- $\forall t\ \tau_{\min} < \tau_{i,j}(t) < \tau_{\max}$ - Pheromone trails have explicit upper and lower limits
- Pheromone trails are re-initialized when stagnated.

The aforementioned design choice were made because:

- The initialzation of the pheromone trails to their upper bound favors diversification at the beginning of each trial.
- The pheromone update rule favors exploitation of (intensification on) the best solutions at each iteration of the algorithm.
- By bounding the intensity of the pheromone trails, the probability of stagnation (i.e. all the ants converging and exploiting a single sub-optimal tour) is reduced.
- If the pheromone trails values for the solution components of a certain tour s are equal to $\tau_{\max}$, the algorithm is said to be converged.

Algorithm 2 $\mathcal{MAX} - \mathcal{MIN}$ Ant System for TSPTW - Outline

procedure ACO($\alpha, \beta, \rho, \tau_0, p_b, t_{max}$) ▷ The main procedure
Require: N - Node set
Require: E - Edge set
Require: c - Edge cost function
Require: t - Time window function
    InitalizePheromoneTrail(τ_0, n_{cities})
    while *!TerminationCondition*(t_{max}) **do**
        for all Ant k **do**
            $s' \leftarrow$ *ConstructSolution*(α, β)
            if *IsImproved*(s, s') **then**
                $s \leftarrow s'$
            end if
        end for
        $s \leftarrow$*IterativeImprovementIBI*()
        UpdatePheromoneTrails(ρ, p_b)
    end while
    return s

end procedure

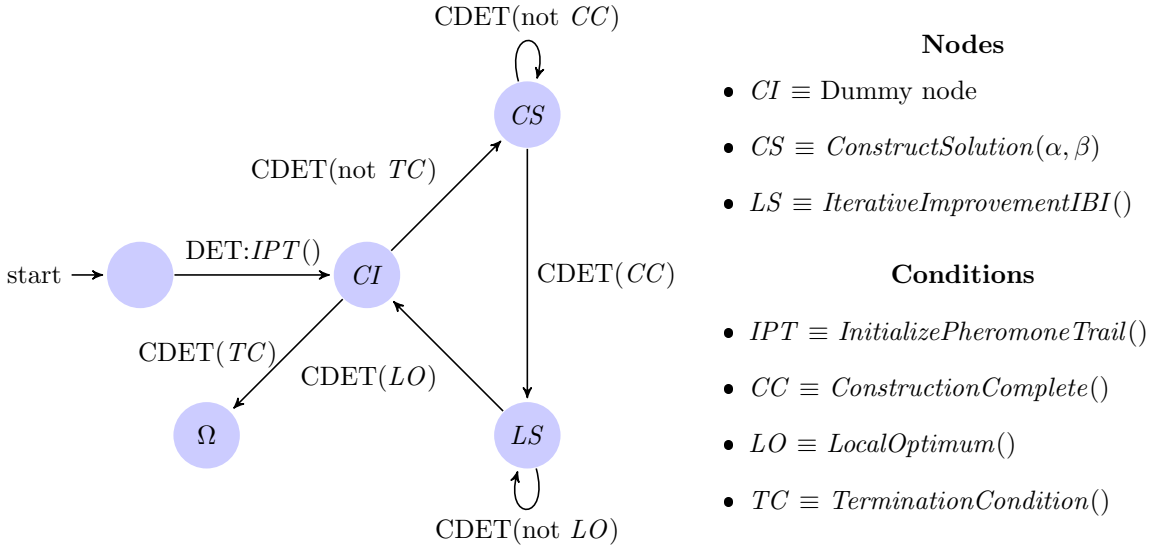


Figure 2: $\mathcal{MAX} - \mathcal{MIN}$ Ant System GLSM

In the implementation, an instance is completely defined by:

- **Cost matrix** - Encapsulating information on the node set N, edge set E, and weighting of each edge c.
- **Time window vector** - Describing the time window mapping function t for each node.

Pheromone Initialization

Algorithm 3 Pheromone Initialization

```
1: procedure InitalizePheromoneTrail(τ_0, n_cities)
2:     τ_max ←
3:     τ_min ←
4:     i ← 0
5:     j ← 0
6:     for i < n_cities do
7:         for j < i do
8:             τ_ij ← τ_0
9:             τ_ji ← τ_ij
10:            j ← j + 1
11:        end for
12:        i ← i + 1
13:    end for
14: end procedure
```

As discussed in Introduction, the ACO methods are based on stigmergic communication among the agents by means of virtual pheromone.
While the real ants can deposit pheromone anywhere in the environment, the virtual ants may only exchange informations concerning solutions components.
For this reason, every admissible edge $e_{i,j}$ of E has an associated pheromone value $\tau_{i,j}$, that have to be initialized at the beginning of the execution of the algorithm.
The inizialization value τ_0 is a parameter of the algorithm, for the $\mathcal{MAX}-\mathcal{MIN}$ Ant System $\tau_0 = \tau_{i,j}(0) = \tau_{\max}$.

Solution construction

Algorithm 4 Solution Construction

```
1: procedure ConstructSolution(α, β)                    ▷ The main procedure
2:                                                      ▷ s_i reprents the i^th component of the solution
3:     s_0 ← 0                                          ▷ Every solution starts at the depot
4:     s_1 ← RandomCitySelection()                      ▷ Random choice of the starting city
5:     i ← 1
6:     while !SolutionComplete() do
7:         s_i ← RouletteWheelSelection()               ▷ s_i stochastically chosen according to the probability distribution defined by 3
8:         i ← i + 1
9:     end while
10: end procedure
```

The solution construction process, used by every ant k in the system, consist of a probabilistic selection of solution components. Every edge $e_{i,j}$ has a selection probability $p^k_{i,j}(t)$ (also called transition probability) defined as follows:

$$p^k_{i,j}(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^\alpha \cdot [\eta_{i,j}]^\beta}{\sum_{k \in A(s_i)} [\tau_{k,j}(t)]^\alpha \cdot [\eta_{k,j}]^\beta} & j \in A(s_i) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

As one can see in 3, the transition probability is determined by a constant, locally available heuristic information $\eta_{i,j}$ and by the time varying pheromone trail $\tau_{i,j}(t)$. This probability is defined on the set $A(s_i)$ of available (i.e. not yet visited) cities while visiting solution component s_i. The value of the parameters α and β determines the relative imporance of the heuristic information and the pheromone trail, respectively.

Solution improvement

Algorithm 5 Solution improvement

```
1: procedure IsImproving(s, s')                    ▷ The main procedure
2:     if Ω(s') < Ω(s) then
3:         return true
4:     else
5:         if Ω(s') == Ω(s) ∧ f(s') < f(s) then
6:             return true
7:         end if
8:     end if
9:     return false
10: end procedure
```

A solution s' is considered improving the current best solution s if and only if:

- Either,it has a smaller number of constraints violation (i.e. $\Omega(s') < \Omega(s)$)
- Or, it has the same number of constraints violations ($\Omega(s') == \Omega(s)$) but the total tour duration is smaller ($f(s') < f(s)$).

Admissible heuristics

The heuristic component $\eta_{i,j}$ is used to guide the selection of solution compoments towards those components that are included in optimal solution.

Dorigo et al, 1996 The heuristic originally proposed by Dorigo et al, in [4], for the TSP problem is:

$$\eta_{i,j} = \frac{1}{c(e_{i,j})} \tag{4}$$

The main idea behind this heuristic is that, if the selection process tends to select, at each step, the shortest connection between the current node and the following, the built tour should be of the shortest length. This heuristic is cited for explanation purposes, even though it cannot be used for the TSPTW problem, since it will guide the exploration only towards shorter solutions, without taking into account the presence of the time windows.

Cheng and Mao, 2007 The local heuristics used in [1] are similar to that proposed by Gambardella et al. [5] in their multiple ant colony system (MACS) designed to solve the vehicle routing problem with time windows (VRPTW).

$$[\eta_{i,j}]^\beta = [g_{i,j}]^\beta \cdot [h_{i,j}]^\gamma \tag{5}$$

The two components $g_{i,j}$ and $h_{i,j}$, are designed, respectively, to avoid lateness (that is, arriving in the node where the time windows is already terminated) and waiting times (i.e. arriving in the node before the time windows open).

Lateness avoidance

$$g_{i,j} = \begin{cases} \frac{1}{1+e^{\delta \cdot (G_{i,j} - \mu)}} & G_{i,j} = b_j - t_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where

- $G_{i,j} = b_j - t_j$ - Slack corresponding to the time window j while being in node i
- t_i - Arrival time at node i
- b_i - Closing time of time window i
- $G(i) = \{k \mid G_{i,k} \geq 0\}$ - Set of feasible neighbors of node i (i.e. such that node k is reached earlier than its closing time)
- $\mu = \frac{1}{|G(i)|} \sum_{j \in G(i)} G_{i,j}$ - Average slack
- δ - Parameter to control the slope of the sigmoidal function

Waiting time avoidance

$$h_{i,j} = \begin{cases} \frac{1}{1+e^{\lambda \cdot (H_{i,j} - \upsilon)}} & H_{i,j} = t_j - a_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where

- $H_{i,j} = t_j - a_j$ - Waiting time corresponding to the time window j while being in node i
- t_i - Arrival time at node i
- a_i - Opening time of time window i
- $H(i) = \{k \mid H_{i,k} \geq 0\}$ - Set of non-waiting neighbors of node i (i.e such that node k is reached within the time window)
- $\upsilon = \frac{1}{|H(i)|} \sum_{j \in H(i)} H_{i,j}$ - Average waiting time λ - Parameter to control the slope of the sigmoidal function

Lopez-Ibanez and Blum, 2009 The approach used in [6] is instead, to perform a linear combination, based on λ_i coefficients, of the normalized values of opening and closing times of the time windows and the travelling cost from one city to another.

$$\eta_{i-1,k} = \lambda_a \cdot \frac{a_{\max} - a_k}{a_{\max} - a_{\min}} + \lambda_b \cdot \frac{b_{\max} - b_k}{b_{\max} - b_{\min}} + \lambda_c \cdot \frac{c_{\max} - c_{i-1,k}}{c_{\max} - c_{\min}} \tag{8}$$

where

- a_i - Opening time of time window i
- $a_{\max} = \max_{j \in N(i)} a_j$ - Maximum time window opening time in the neighborhood of node i
- $a_{\min} = \min_{j \in N(i)} a_j$ - Minimum time window opening time in the neighborhood of node i
- b_i - Closing time of time window i
- $b_{\max} = \max_{j \in N(i)} b_j$ - Maximum time window closing time in the neighborhood of node i
- $b_{\min} = \min_{j \in N(i)} b_j$ - Minimum time window closing time in the neighborhood of node i
- $c_{i,j}$ - Travelling cost from node i - j
- $c_{\max} = \max_j c_{i,j}$ - Maximum travelling cost from node i
- $c_{\min} = \min_j c_{i,j}$ - Minimum travelling cost from node i
- $\lambda_a, \lambda_b, \lambda_c$ s.t. $\lambda_a + \lambda_b + \lambda_c = 1$ - Randomly selected weights

In this implementation, the heuristic information will be computed according to [6]

Pheromone trails update

Algorithm 6 Pheromone Trails Update

```
1:  procedure UpdatePheromoneTrails(ρ, p_b)                ▷ The main procedure
2:      i ← 0
3:      j ← 0
4:      for i < n_cities do
5:          for j < i do
6:              if Random() < p_b then
7:                  τ_ij ← (1 − ρ) · τ_ij + Δτ^Bi_{i,j}
8:              else
9:                  τ_ij ← (1 − ρ) · τ_ij + Δτ^Bo_{i,j}
10:             end if
11:             if τ_ij < τ_min then
12:                 τ_ij ← τ_min
13:             end if
14:             if τ_ij > τ_max then
15:                 τ_ij ← τ_max
16:             end if
17:             τ_ji ← τ_ij
18:             j ← j + 1
19:         end for
20:         i ← i + 1
21:     end for
22: end procedure
```

As discussed in Algorithm structure, the pheromone update will be made by a single ant, being either the one who found the best solution in the current iteration:

$$\Delta\tau_{i,j}^{Bi} = \begin{cases} \frac{1}{T_d^{\mathrm{Bi}}} & e_{i,j} \in T^{\mathrm{Bi}} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Or the one having found the global best (best-so-far) solution:

$$\Delta\tau_{i,j}^{Bo} = \begin{cases} \frac{1}{T_d^{\text{Bo}}} & e_{i,j} \in T^{\text{Bo}} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where

- $e_{i,j}$ - Edge connecting node i and j
- T_d^i - Complete tour duration of tour i
- T^{Bi} - Best tour of the current iteration
- T^{Bo} - Best tour overall

Provided that the best solution is known, the bounds on the pheromone values can be estimated by:

$$\hat{\tau}_{\max} = \frac{1}{\rho \cdot T_d^{\text{Go}}} \qquad \hat{\tau}_{\min} = \frac{\hat{\tau}_{\max}}{a} \tag{11}$$

Local search

Algorithm 7 Iterative Improvement - (Insert neighborhood with best improve pivoting rule)

```
1: procedure IterativeImprovementIBI(s)                    ▷ The main procedure
2:     s* ← s
3:     for i ∈ {1, ··· , |N|} do
4:         for j ∈ {1, ··· , |N|} do
5:             if i == j ∨ j == i − 1 then
6:                 continue
7:             end if
8:             s' ← InsertTourComponent(s, i, j)
9:             if IsImproved(s*, s') then
10:                s* ← s'
11:            end if
12:        end for
13:    end for
14:    return s*
15: end procedure
```

The solution construction process, used by every ant k in the system, consist of a probabilistic selection of solution components. Every edge $e_{i,j}$ has a selection probability $p_{i,j}^k(t)$ (also called transition probability) defined as follows:

$$p_{i,j}^k(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^\alpha \cdot [\eta_{i,j}]^\beta}{\sum_{k \in A(s_i)} [\tau_{k,j}(t)]^\alpha \cdot [\eta_{k,j}]^\beta} & j \in A(s_i) \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

As one can see in 3, the transition probability is determined by a constant, locally available heuristic information $\eta_{i,j}$ and by the time varying pheromone trail $\tau_{i,j}(t)$. This probability is defined on the set $A(s_i)$ of available (i.e. not yet visited) cities while visiting solution component s_i. The value of the parameters α and β determines the relative imporance of the heuristic information and the pheromone trail, respectively.

Algorithm 8 Insert neighbor solution computation

```
1: procedure InsertTourComponent(s, i, j)                ▷ The main procedure
2:                                                       ▷ s_i reprents the i^th component of the solution
3:     e ← s_i
4:     if i¡j then
5:         k ← i
6:         while k < j do
7:             s_i ← s_{i+1}
8:             k ← k + 1
9:         end while
10:    else
11:        k ← i
12:        while k > j do
13:            s_i ← s_{i−1}
14:            k ← k − 1
15:        end while
16:    end if
17:    s_j ← e
18:    return s
19: end procedure
```

Problem 2

Simulated Annealing

Problem statement

Implement two stochastic local search (SLS) algorithms for the traveling salesman problem with time windows (TPSTW), building on top of the perturbative local search methods from the first implementation exercise.

1. Run each algorithm 25 times with different random seed on each instance. Instances will be available from http://iridia.ulb.ac.be/~stuetzle/Teaching/HO/. As termination criterion, for each instance, use the maximum computation time it takes to run a full VND (implemented in the previous exercise) on the same instance and then multiply this time by 1000 (to allow for long enough runs of the SLS algorithms).
2. Compute the following statistics for each of the two SLS algorithms and each instance:
   - Percentage of runs with constraint violations
   - Mean penalised relative percentage deviation
3. Produce boxplots of penalised relative percentage deviation.
4. Determine, using statistical tests (in this case, the Wilcoxon test), whether there is a statistically significant difference between the quality of the solutions generated by the two algorithms.
5. Measure, for each of the implemented algorithms on 5 instances, the run-time distributions to reach sufficiently high quality solutions (e.g. best-known solutions available at http://iridia.ulb.ac.be/~manuel/tsptw-instances#instances). Measure the run-time distributions across 25 repetitions using a cut-off time of 10 times the termination criterion above.
6. Produce a written report on the implementation exercise:
   - Please make sure that each implemented SLS algorithm is appropriately described and that the computational results are carefully interpreted. Justify also the choice of the parameter settings and the choice of the iterative improvement algorithm for the hybrid SLS algorithm.
   - Present the results as in the previous implementation exercise (tables, boxplots, statistical tests).
   - Present graphically the results of the analysis of the run-time distributions.
   - Interpret appropriately the results and make conclusions on the relative performance of the algorithms across all the benchmark instances studied.

Metric definitions

For each algorithm k, applied on instance i, using different randomly generated seeds one have to compute:

- Number of constraint violations
- Penalised relative percentage deviation (PRPD)
- Computation time (CPU time)

In order to perform a statistical analysis of the results, each algorithm k is launched 100 times on the same instance, computing the following statistics:

- Percentage of infeasible solutions (0.x has to be interpreted as x%)

- Average Penalised relative percentage deviation.
- Average Computation time (CPU time).

For each instance i, the distributions of PRPD and Cpu Times are displayed using box plots and the Wilcoxon signed rank test is performed, in order to assess the existence of a statistically significant difference among the results obtained by the different algorithms on the same instance.

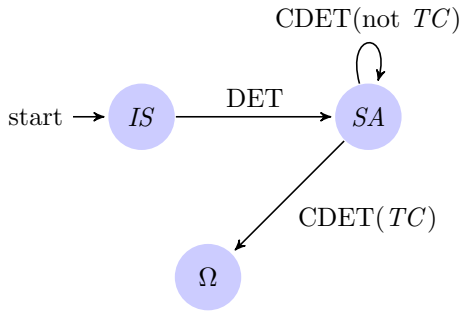
Constraint Violations In the standard formulation of the TSP problem, a solution to the problem is represented by a permutation of the different entities (solution components), that the hypothetical travelling salesman has to visit.
The best solution for the problem is the permutation that minimizes the total travelling time (distance) among the cities. The presence of time windows introduce an additional constraint on the feasibility of the solution.
In fact, each solution component has an associated time window within which it has to be visited in order to guarantee the feasiblity of the tour.
Arriving in a (city) before the opening of the corresponding time window involves a delay in the total travelling time (to wait for the time window to open) whereas the arrival after the closure of the time windows will generate a constraint violation.
Thus, a solution is feasible if and only if all the time windows constraints are met, or in other words, if there are no constraint violations.
In this case, the best solution is the feasible solution which minimizes the total travel time.

Penalised Relative Percentage Deviation The penalised relative percentage deviation (PRDP from now on) is a measure of the solution quality, with respect to the best known solution for the instance, taking into account a strong penalisation for the violation of constraints. The PRPD is computed as follows:

$$pRPD_{kri} = 100 \cdot \frac{(f_{kri} + 10^4 \cdot \Omega_{kri}) - best_i}{best_i} \tag{13}$$

Runtime The runtime is a measure of both the quality and the time complexity of the algorithm.
It is measured using the function `int clock_gettime(clockid_t clk_id, struct timespect *tp)` from the `time.h` library.
The parameter `clk_id=CLOCK_PROCESS_CPUTIME_ID`, is used to read the values from an high-resolution timer provided by the CPU for each process.
The runtime is computed (using the user defined function `ComputeRunTime`) as the difference, with a resolution of 10^-9 s, from the time obtained using `clock_gettime` at the beginning and the one obtained at the end of the simulation.

Algorithm structure



Nodes

- $IS \equiv InitialSolution()$
- $SA \equiv SimulatedAnnealingLoop$

Conditions

- $TC \equiv TerminationCondition()$

Figure 3: Simulated Annealing GLSM

Algorithm 9 Simulated Annealing TSPTW

procedure SIMULATEDANNEALING
Require: Annealing schedule
$s \leftarrow$ *InitialSolution*() ▷ Heuristic or random permutation
$T \leftarrow T_0$
while !*TerminationCondition*() **do**
$s' \leftarrow$ *ProposalMechanism()* ▷ (Often) Uniform random choice in $N(s)$
if *AcceptanceCriterion*(s, s', T) **then** ▷ (Often) Metropolis condition
$s \leftarrow s'$
end if
$T \leftarrow$ *Update*(T) ▷ $T_{i+1} = \alpha \cdot T_i$
end while
return s
end procedure

Algorithm 10 Initial solution

procedure INITIAL SOLUTION
$p \leftarrow$ *UniformRandom*$(1, \frac{|N|}{10})$ ▷ p : Number of perturbations
$s \leftarrow$ *SortCitiesUBTW*() ▷ Sort cities according upper bound of time window
while $p > 0$ **do**
$p_p \leftarrow$ *UniformRandom*$(1, |N| - 1)$ ▷ p_p : Perturbation point
$p_i \leftarrow$ *UniformRandom*$(1, p)$ ▷ p_i : Perturbation intensity
$s \leftarrow$ *Shuffle*$(1 + p_p, 1 + p_p + p_i)$ ▷ Shuffle the solution components occuping the position bounded by the indexes given as parameters
$p \leftarrow p - 1$
end while
return s
end procedure

Algorithm 11 Termination condition

procedure TERMINATION CONDITION
end procedure

Metropolis condition

$$P(s, s', T) = \begin{cases} 1 & f(s) < f(s') \\ 1 & f(s) = f(s') \wedge \Omega(s) < \Omega(s') \\ e^{\frac{f(s) - f(s')}{T}} & \text{otherwise} \end{cases} \tag{14}$$

Algorithm 12 Proposal mechanism

procedure PROPOSAL MECHANISM
  $i \leftarrow$ *UniformRandom*$(1, |N|)$
  $j \leftarrow$ *UniformRandom*$(1, |N|)$
  return *InsertTourComponent*(s, i, j)
end procedure

Algorithm 13 Acceptance Criterion

procedure ACCEPTANCE CRITERION(s, s', T)
  if *IsImproved*(s, s', t) **then**
    return 1
  else
    return $e^{\frac{f(s)-f(s')}{T}}$
  end if
end procedure

Algorithm 14 Update according to annealing schedule

procedure UPDATE(T)
  $T \leftarrow \alpha \cdot T$
end procedure

Conclusions

By combining the results from the previous analysis:

- The Iterative Improvement algorithms based on Transpose and Exchange neighborhoods do not allow to find feasible solutions hence they should not be considered for a practical application.
- On the other hand, the solution quality generated by the Iterative Improvement algorithm with the Insert neighborhood is similar to those generated by the VND algorithms, regardless of the instances.
- On this set of instances, the VND algorithms have a lower runtime than the Iterative Improvement one using Insert neighborhood, hence being similar the resulting solution quality, they should be preferred to the Iterative Improvement ones.
- The algorithm that showed the best performances in terms of solution quality and runtime is the Standard Variable Neighborhood Descent with Transpose-Insert-Exchange neighborhood chain.
- The usage of average statistics as metrics to measure the quality of the algorithms is strongly biased by the presence of outliers (penalisation, in this case).

References

[1] Chi-Bin Cheng and Chun-Pin Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46(9):1225–1235, 2007.

[2] J-L Deneubourg, Serge Aron, Simon Goss, and Jacques Marie Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2):159–168, 1990.

[3] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Artificial ants as a computational intelligence technique. *IEEE computational intelligence magazine*, pages 265–276, 2006.

[4] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[5] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New ideas in optimization*. Citeseer, 1999.

[6] Manuel López-Ibáñez and Christian Blum. Beam-aco for the travelling salesman problem with time windows. *Computers & operations research*, 37(9):1570–1583, 2010.

[7] Thomas Stützle and Holger H Hoos. Max–min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.