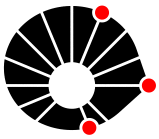
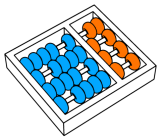




Two-stage Stochastic Traveling Salesman Problem with Evolutionary Framework

L. C. Souza *F. L. Usberti*

Relatório Técnico - IC-PFG-19-25
Projeto Final de Graduação
2019 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Two-stage Stochastic Traveling Salesman Problem with Evolutionary Framework

Lauro Cruz e Souza* Fábio Luiz Usberti†

Abstract

The Traveling Salesman Problem (TSP) is a widely studied problem in the literature since its formulation, specially because of its wide number of applications in Computer Science, mainly in routing and scheduling problems. But the Stochastic Traveling Salesman problem, the Stochastic version of the TSP, which takes into consideration future uncertainties, is less mentioned. The goal of this study is to increase the studies and implementations of the Stochastic TSP using a new framework, the Evolutionary Framework (EvFW), which obtained great results with other stochastic problems and has great applicability for large instances since it is the first of its kind to have a fully heuristic decomposition approach, not depending on Integer Linear Programming solvers. We hope to have brought new data to the literature regarding the problem and shown the usefulness of the EvFW.

1 Introduction

The problem

The Traveling Salesman Problem (TSP) is a NP-hard problem for which a lot of mathematical formulations were proposed (Miller et al., 1960 [1]; Gavish and Graves, 1978 [2]). The problem consists in finding a Hamiltonian cycle of minimum cost in a complete graph. The problem is further explained in Section 3.1.

In this project we worked with a two-stage stochastic variation of the problem (STSP). In the stochastic problem, unlike the deterministic one, we take uncertainty and future demands into consideration. The fact that this uncertanty is represented as a problem with two stages, a present scenarios with the present need, and future scenarios with uncertainties, is what maked the stochastic problem two-staged. The problem is further explained in Section 3.2. The Stochastic TSP was already studied (Maggioni el al., 2014 [3]; Bertazzi and Maggioni, 2014 [4]) and we found one study of the two-stage stochastic TSP (Adasme et al., 2016 [5]), which means the problem is not largely studied.

Our contributions

The main difference of our study from the one from 2016 is the heuristic methodology used, which is an altered Evolutionary Framework (EvFW) (Hokama et al., 2018 [6]), a two-stage framework developed for resource-allocation problems and with great results for large-scale instances, since its operations are all heuristics. A more detailed description of the framework and our alterations can be seen in sections 4 and 4.2 respectively.

In order to differentiate even more our study from the 2016 one, initially our formulation of the problem was also a little different, with each future scenario having a different subset of terminal

*lauro.souza@students.ic.unicamp.br

†Inst. de Computação, UNICAMP, 13083-852 Campinas, SP. fusberti@ic.unicamp.br

nodes to be traveled. That was specially interesting because we found no study in the literature that dealed with this variation of the Stochastic TSP. Unfortunately there were difficulties with modeling of this variation of the problem that we had no time to solve. This is expanded in section 3.2.1.

Heuristics used
Since the framework needs a solver for the deterministic problem in order to solve the stochastisc one, in our implementation we are using the Lin-Kernighan heuristic(Lin and Kernighan, 1973 [7]), which is one of the most accurate heuristics for TSP. The Lin-Kernighan is a highly complex algorithm, so we did not implement it, instead we used the the Lin-Kernighan-Helsgaun C implementation done and maintained by Keld Helsgaun[1] [8], which is one of the best implementations of the heuristics with improvementes of its own. Further details about the heuristic and its implementation can be seen in section 3.1.1.

Experiments
There aren't many instances of the Two-stage stochastic TSP available, since it is not a largely studied problem, so we had to generate our own custom Stochastic TSP instances from regular TSP instances which we got from the TSPLIB archive[2]. The specifications of the instance generator and experimental results can be seen in Section 5 and section 6 respectively.

2 Definitions and terminology

In this section we define some non-trivial terms that are essential for the comprehension of the article.

Definition 1: Stochastic Programming

Stochastic Programming is the field of research concerned with modeling resource allocation problems to deal with uncertainties concerning costs and demands of future resources. A common approach is to rely on a scenario decomposition analysis, where the uncertainty is modeled by a limited number of scenarios weighted by their occurrence probabilities. This is the approach of this project. [6]

Definition 2: Two-stage Stochastic Programming

The two-stage stochastic programming is a mathematical framework to model stochastic problems using a scenario approach. The first stage reflects decisions to be made at the present time. The second stage reflects decisions that should be made at a future time, taking into consideration a set of possible scenarios, each with a realization probability. The goal is to find the minimum cost solution which comprises the first stage cost and the second stage expected cost for all scenarios. [6]

[1] `http://akira.ruc.dk/~keld/research/LKH/`
[2] `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html`

Definition 3: NP-Hardness

A problem H is NP-Hard when every problem $L \in NP$ can be reduced in polynomial time to H. Since there is still no knowledge to whether $P = NP$ or not, saying the problem is NP-Hard also means there is no exact algorithm to solve it in polynomial time, which is the case for our problem, the Traveling Salesman.

Definition 4: Heuristic

An Heuristic is an algorithm technique to solving problems that employs practical and experimental methods which give no guarantee to the optimality or degree of optimality of the solution, but, when a good heuristic, can obtain good "enough" or even optimal results. Heuristics are used mainly for NP-Hard problems in order to obtain results in polynomial time. Our framework uses an heuristic to solve the TSP and is itself an heuristic.

Definition 5: Hamiltonian Cycle

Given a Graph $G(V, E)$, a Hamiltonian Cycle over this graph is a path that visits all nodes exactly once and returns to the source node at the end. In the TSP, since the graph G is complete, all graphs have $\frac{(|V|-1)!}{2}$ Hamiltonian Cycles, and the goal is to find the one with minimum cost.

Definition 6: Integer Linear Programming (ILP) Problem

An ILP Problem is an NP-Complete optimization problem in which some or all the variables are restricted to be integers and the objective function is linear. Its formulations are represented by an objective linear function and a series of restrictions to its variables. Solving it is finding the minimum (or maximum) global optimum of the objective function within the restricted space.

3 Traveling Salesman Problem (TSP)

In this project we worked with the Traveling Salesman Problem (TSP). Both its deterministic and stochastic (STSP) definitions and variations are expanded in the following subsections.

3.1 Deterministic TSP

The deterministic TSP is a NP-Hard problem that consists in finding a minimum cost Hamiltonian cycle in a complete graph with weighted edges. So, given a complete graph $G(V, E)$ and a cost function for the edges $c_e \in \mathbb{R}_{\geq 0}$, $\forall e \in E$, the problem has the following Integer Linear Programming (ILP) formulation:

$$\text{minimize} \sum_{e \in E} c_e x_e \tag{1}$$

$$\text{sujeito a} \sum_{e \in \delta(v)} x_e = 2, \ \forall v \in V \tag{2}$$

$$\sum_{e \in \delta(U)} x_e \geq 2, \forall U \subseteq V \tag{3}$$

$$x_e \in \{0, 1\}, \quad \forall e \in E \tag{4}$$

The x_e variables represent whether the edge e is part (1) or not (0) of the solution.

The model restrictions are defined as:

1. The Hamiltonian Cycle must have minimum cost, which is done by summing the cost of all the edges in the solution.
2. Since the solution is a cycle, all nodes must have two incident edges ("in" and "out").
3. It is guaranteed that cycles will be formed using only the previous restrictions, but they can be disconnected. The connectedness is guaranteed by forcing that any cut in the graph must have at least two edges.
4. Decision variables' domain restriction.

In order to create a solver for the Stochastic TSP we need a solver for the deterministic TSP. Since the problem is NP-Hard, we need a fast and accurate heuristic. For that we chose the Lin-Kernighan heuristic.

3.1.1 Lin-Kernighan

The Lin-Kernighan heuristic is one of the most accurate heuristics for the TSP. It is built as a generalization of the OPT-2 and OPT-3 heuristics, which work by switching two or three edges, respectively, to make the tour shorter. The Lin-Kernighan is adaptive and at each step decides how many paths between nodes need to be switched to find a shorter tour.

Lin-Kernighan-Helsgaun
Since the focus of the project is to solve the Stochastic TSP, we didn't implement our own version of the Lin-Kernighan heuristic, which would be a lengthy task. Instead we used the Lin-Kernighan-Helsgaun (LKH) [8] implementation of Lin-Kernighan, which is an effective implementation of the heuristic done in C with years of work and many improvements of its own. LKH was able to produce optimal solutions for all nontrivial instances with known optimality and improved the best known solution for a series of large-scale instances with unknown optima, among these a 1,904,711-city instance (World TSP) [3].

Since we did not work with instances with more than 200 nodes, the LKH is basically an exact algorithm, almost always finding the optimal solution. This made us change our approach with the Evolutionary Framework and its internal use of the BRKGA. This is better explained in section 4.2.

[3] `http://www.math.uwaterloo.ca/tsp/world/`

Changes made to the LKH code
The only problem there was with the use of the LKH was that we needed a library so we could use the heuristic inside the Framework, and the LKH is not a library, but a system by itself that takes files as input and outputs.

Initially we thought about using it as it is, which meant to write files to pass to the system and then reading the output files with the cycle found and its cost. That not only was bad programming, but also added a big overhead to the execution, making the framework much slower. We ended up creating an interface based on its "main" and "read input" functions in order to use the heuristic as a function from a library.

Since LKH has a lot of different modes, our interface is really specific for our needs, so it only solves TSP instances with the edges' costs given explicitly in the LOWER_DIAG_ROW format (Figure 1).

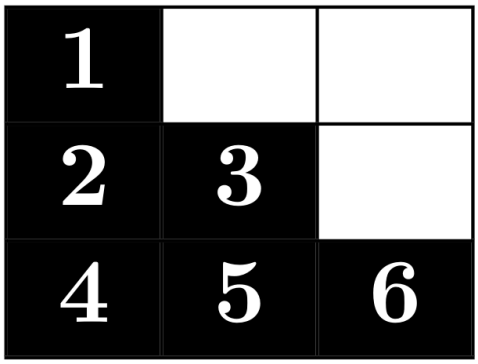


Figure 1: Adjacency Matrix for Graph with 3 nodes with the order of the weights in LOWER_DIAG_ROW. Since there is no one-node loops, the weights in the diagonal (1, 3 and 6) are always 0.

3.2 Stochastic TSP

The Stochastic TSP consists in:

1. A scenario (present) s^0 with:
   - Edges with costs $c_e^0 \in \mathbb{R}_{\geq 0}$, $\forall e \in E$.
2. A set S of future scenarios with:
   - Realization probability $p^s \in [0, 1]$, $\forall s \in S$.
   - Edges with costs $c_e^s \in \mathbb{R}_{\geq 0}$, $\forall e \in E$, $\forall s \in S$, which are inflated compared to the costs s^0 from the present scenario.
   - A set of terminal nodes $V^s \subseteq V$, $\forall s \in S$ which must be visited by the scenario s solution.

The goal is to obtain a minimum-cost Hamiltonian Cycle over V^s, $\forall s \in S$ and this is done "buying" edges from the scenario s^0, which are cheaper than the inflated ones from the future scenarios and can be used in all of them and then "buying" the rest of the necessary edges in each scenario. a ILP formulation of the problem can be seen below:

$$\text{minimize} \sum_{e \in E} c_e x_e + \sum_{e \in E} \sum_{s \in S} p^s c_e^s x_e^s \tag{1}$$

$$\text{sujeito a} \sum_{e \in \delta(v)} (x_e^0 + x_e^s) \geq 2, \qquad \forall s \in S, \forall v \in V^s \tag{2}$$

$$\sum_{e \in \delta(U)} (x_e^0 + x_e^s) \geq 2, \qquad \forall s \in S, \forall U \subseteq V^s \tag{3}$$

$$x_e^0 + x_e^s \leq 1, \qquad \forall e \in E, \forall s \in S \tag{4}$$

$$x_e^s \in \{0, 1\}, \qquad \forall e \in E, \forall s \in S_0 \tag{5}$$

The variables x_e^s, $\forall s \in S_0$ represent the edges obtained in each scenario (including the present) and $S_0 = S \cup \{s^0\}$.

The model restrictions are defined as:

1. The goal is to minimize the cost of all the bought edges. For that we need to minimize the sum of the edges' costs bought in the present scenario (s^0) and the sum of the edges' costs bought in each future scenario multiplied by their scenario realization probability.

2. Just like condition 2 from the deterministic TSP we need to guarantee that in all scenarios all the nodes have at least two incident edges in order to form a cycle. Since we take the edges from both the present scenario s^0 and the future scenario s into consideration, the total number of incident edges can be higher that 2 because we can end up buying an edge in the present scenario that will not be used in the future scenario.

3. Same as the restriction 3 from the deterministic TSP, but to all scenarios and using the edges from the present and future scenarios.

4. Guarantees that the same edges won't be bought in both the present (s^0) and future (s) scenarios.

5. Decision variables' domain restriction.

3.2.1 Subset of terminal nodes in future scenarios (V^s)

As described above, there is a set of nodes $V^s \subseteq V$, $\forall s \in S$ which must be visited by a Hamiltonian Cycle in all future scenarios. Our goal, initially, was to create a solver for cases where these sets can be any subset of nodes from the graph, which would make our solver more generic to other proposed solvers for this problem, like (Adasme et al., 2016 [5]), which has $V^s = V$, $\forall s \in S$, that is, all future scenarios have to visit every node in the graph.

Our main problem with this more general formulation is that even though there is a limited subset of terminal nodes, we should still be able to visit nodes not on the terminal subset. That is relevant because in graphs which edges' costs do not follow the triangle inequality there can be paths between two vertices with a lower cost than the edge that connects them. This is specially important given that when an edge is selected in the present scenario, its cost in the future scenarios becomes 0, so even though the present scenario may follow the triangle inequality, the future scenarios will not. We can see that in the example from the Figure 2, where $V^s = \{1, 2, 3\}$. If we ignored the node 0, the only possible cycle would be $\{1, 2, 3, 1\}$, with a total cost of 15, but with node 0 we could do the cycle $\{1, 0, 2, 3, 1\}$, with cost 12.

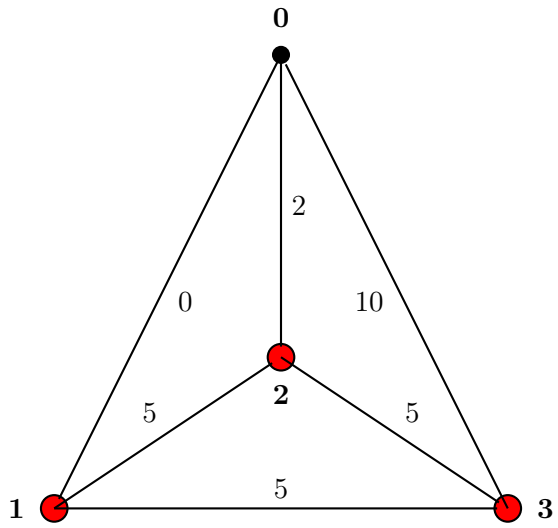
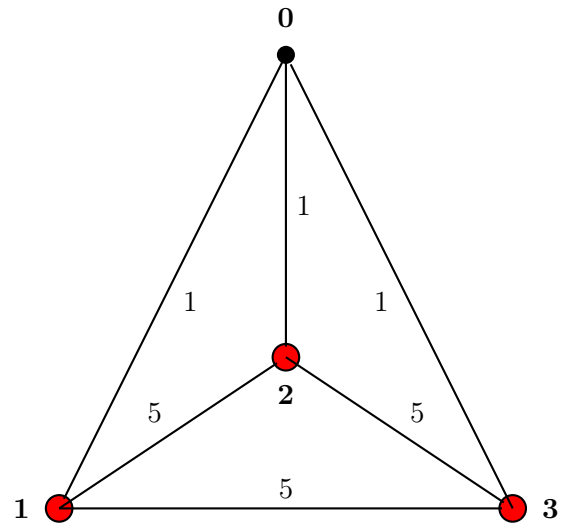
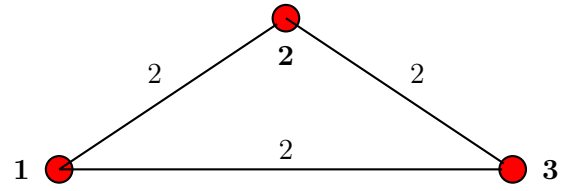


Figure 2: TSP instance with terminal node set $V^s = \{1, 2, 3\}$.

Initially we thought about doing the transitive closure of the graph and transforming the edges between terminals in the shortest path between them. The problem is that we would end up visiting nodes more than once, which is not ideal. We can see that in image 3, where if we did the transitive closure, we would have the cycle $\{1, 0, 2, 0, 3, 0, 1\}$ with cost 6.



(a) Full TSP instance with terminal node set $V^s = \{1, 2, 3\}$.



(b) TSP instance with transitive closure applied. All edges pass through node **0**.

Figure 3: Example of how the transitive closure approach is not ideal for the solving the more general version of the STSP because we end up visiting nodes more than once.

We came to the conclusion that the problem we need to solve to find our wanted cycle in this case is not the TSP, so we needed to formulate this problem and find a heuristic for it. Since there was no time to make such a change to the project we decided to limit the problem to the case of $V^s = V$, $\forall s \in S$, that way we can still work with the TSP for the future scenarios. Possible workarounds to this are commented in Section 7.

4 Evolutionary Framework

The Evolutionary Framework (EvFW) is a framework to solve large-scale general two-stage stochastic resource allocation problems. It contains a two-step main loop with each step interchanging information via feedback. The first step is a local search responsible for selecting first stage resources until a local optimum is found. The second step solves each future scenario using a *Biased Random Keys Genetic Algorithm* (BRKGA) (Gonçalves and Resende, 2011 [9]). Since the EvFW is the first of its kind to have a fully heuristic stage decomposition approach, it can more easily tackle larger instances, since it does not depend on ILP solvers.

4.1 Steps of the EvFW

The main loop of the EvFW has two steps, which are the Local Search (LS) and the Second Stage Metaheuristic (SSM). The SSM uses the BRKGA to solve the future scenarios and based on

the solutions found it creates the feedback for the LS. The main loop stops after a stopping criteria is reached, which triggers a tail step and then finishes the framework.

4.1.1 Local Search (LS)

The first stage of the EvFW, the Local Search, selects first stage resources towards a local optimum and it uses feedback information from the second stage in doing that. This feedback is a chromosome with one allele key (a float value in $[0, 1]$) for each resource of the problem (in our case the edges), calculated as a weighted medium from the best chromosome found in each future scenario by the BRKGA in the second stage. This is better explained in the next section.

In the chromosome, the smaller the allele key, the higher is the chance that resource is useful for the future scenarios, so the LS tries to select each resource in increasing order, giving preference to the "best" ones. For each resource e the LS solves the future scenarios using the deterministic problem heuristic taking into consideration the set of selected resources until now including e. If the total cost increases, than the resource e is removed from the selected resources and LS continues the search.

If the Local Search is running for the first time, than the chromosome has all alleles equal to 0.5, so the order of the resource selection is irrelevant.

4.1.2 Second Stage Metaheuristic (SSM)

The second stage of the EvFW, the SSM, receives the list of selected resources in the LS and solves each future scenario using the BRKGA. All resources selected by the first step have their costs set to 0, meaning that selecting them in the future scenarios is free, since they were already selected in the present scenario.

Instead of just using the deterministic problem heuristic, the SSM uses it combined with the BRKGA, which finds a chromosome that represents the best solution found for the problem instance.

4.1.3 Biased Random Key Genetic Algorithm (BRKGA)

The BRKGA is a general search metaheuristic based on genetic algorithms, where a population of individuals evolves through the Darwinian principal of survival of the fittest.

In the population, each individual i is represented by a chromosome Q^i composed by one allele for each resource of the resource allocation problem. The alleles are random keys uniformly drawn over the interval $[0, 1]$. The BRKGA also needs a decoder, which is an algorithm that translates a chromosome into a solution. We won't go into much detail about how the BRKGA does its evolution process of the population, what is important to us here is how the decoder works.

For each resource $r \in R$ from our scenario and a chromosome Q, the allele key $Q(r)$ is used to perturb the corresponding resource cost $c(r)$. With $\alpha \in \mathbb{R}_*^+$ as a parameter that determines the perturbation intensity, the perturbed cost of $c(r)$ is:

$$c^{'}(r) = (1 - \alpha + 2\alpha Q(r)) \times c(r)$$

With the new perturbed costs we use the deterministic problem heuristic to find the solution represented by the chromosome and the cost of the solution is determined by the original costs of the selected resources (or 0 for the ones selected in the first step).

With this decoding method, for any $r \in R$, we can see that:

- If $Q(r) < 0.5$ the cost of the resource decreases.

- If $Q(r) > 0.5$ the cost of the resource increases.
- If $Q(r) = 0.5$ the cost of the resource stays the same.

With this method if, in the best chromosome of the population, the allele key r has a low value, the chromosome is "forcing" the selection of resource r in order to obtain the minimum cost solution, and vice-versa.

4.1.4 Feedback

In order to give the feedback to the Local Search a weighted medium of the chromosomes is calculated using the scenario probabilities as the weights. With Q_f and Q_s, $s \in S$ as the feedback chromosome and the best chromosome for each scenario, respectively, the function to calculate Q_f can be seen below:

$$Q_f(r) = \sum_{s \in S} Q_s(r) \times p_s, \forall r \in R$$

4.1.5 Stopping criteria

The stopping criteria for the main loop of the EvFW are two convergence criterias that need the parameters $i \in \mathbb{R}_*^+$, $b \in \mathbb{N}$ and $l \in \mathbb{N}$, that correspond to the minimum improvement ratio, the maximum number of EvFW iterations since the best solution was found, and the maximum number of EvFW iterations since the last solution improvement, respectively.

Let S^* be the best solution found so far, S' be the solution from the last iteration and S be the current solution. If $c(S^*) > (1+i) \times c(S)$ then the global improvement counter (GIC) is reset to 0, otherwise it increases by 1. Similarly, if $c(S') > (1+i) \times c(S)$ then the local improvement counter (LIC) is reset to 0, otherwise it increases by 1. If $GIC == b$ or $LIC == l$, then the stopping criteria is reached and the EvFW falls to the tail step.

In our experiments the stopping criteria values used were the default ones, which are $i = 0.001$, $b = 3$ and $l = 2$

4.2 Removal of the BRKGA

As mentioned in the Introduction, we made changes to the EvFW, and the main change was the removal of the BRKGA from the SSM. This was done because of our use of the Lin-Kernighan heuristic. As mentioned in the section 3.1.1, the LKH, for our use, can be seen as an exact algorithm. Because of that not only the BRKGA becomes useless, since we only need to run the heuristic on our scenario to immediately find the best solution, but also made the feedback useless.

Since the heuristic finds the best solution always, the best chromosome of a population is always the starting chromosome ($Q(r) = 0.5$, $\forall r \in R$), so the feedback chromosome is also $Q_f(r) = 0.5$, $\forall r \in R$. That means there is no feedback and the Local Search is always the same, always selecting the same resources, which also makes the to SSM always be the same, and with the lack of improvement in the total cost, the framework will stop because of the stopping criteria, rendering the EvFW basically useless, since it got stuck in the first local optimum it found.

In order to overcome this problem we removed the BRKGA and applied the LKH directly to the future scenarios, and based on the solution we created “fake” best chromosomes. That was done by creating a chromosome that each allele key is either 0.0 if the resource is in the solution or 1.0 if it is not. That way the chromosome tells the Local Search that if the allele key is closer to 0.0, it

is widely used in the future scenarios, and vice-versa. With this method we were able to remove a useless piece of computation (BRKGA) and maintain the flow of information between the steps of the main loop.

5 Instance Generator

An important task we had to perform in order to experiment with our framework was generating instances for the Stochastic TSP, since we could not find much in the literature. We did that by taking TSP instances as base, which can be found extensively online (we used the ones from the TPLIB archive [4]), and applying a random generator of STSP instances.

Our generator receives two values besides the original TSP instance:

- Number of future scenarios (S).
- Maximum edge cost inflation rate (R).

Using those variables our generator assigns random probabilities to each future scenarios (with their sum equal to 1.0) and inflates all edge costs randomly at most $1 + R$ times.

5.1 Input and Output Files

Input

Since we are using TSPLIB instances, the input files are all TSPLIB files [10]. An example can be seen in Listing 1.

Listing 1: TSPLIB file for TSP instance with 4 nodes

```
NAME: test4
TYPE: TSP
DIMENSION: 4
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
EOF
```

In order to generate a good amount of instances, our generator can interpret TSP "TYPE" files with the following EDGE_WEIGHT_TYPEs:

- EXPLICIT: The weight of the edges are explicitly displayed in one of 8 different formats (EDGE_WEIGHT_FORMAT).
- EUC_2D, where each node is given euclidean 2D coordinates. In this case the generator calculates the distance between each pair of node to obtain the edges' costs.

There are a few other EDGE_WEIGHT_TYPEs (GEO, ATT etc), but since they are less used we did not take the into consideration.

[4] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

Output

The output file, and consequently the input file of our framework, has type ".stp" and was maintained from the original implementation of the framework by (Hokama et al., 2018 [6]) and is built specifically for stochastic problems. It displays the original costs of every edge, the total number of scenarios and their probabilities, the inflated edges' costs for each scenario and the subset of terminal nodes for each future scenario (which in our case is always the full set of nodes). An example STP file generated from the the TSPLIB file from Listing 1 can be seen in Listing 2.

Listing 2: STP file for STSP instance with $S = 5$ and $F = 0.5$. Since te TSP problem does not have a root, the root is set as 0.

```
SECTION Graph
Nodes 4
Edges 6
Scenarios 5
Root 0
E 2 1 666
E 3 1 281
E 3 2 649
E 4 1 395
E 4 2 1047
E 4 3 603
END

SECTION StochasticProbabilities
SP 0.36 0.03 0.26 0.0 0.35
END

SECTION StochasticWeights
SE 794 994 970 808 678
SE 324 375 349 346 386
SE 912 946 750 732 830
SE 479 411 465 472 422
SE 1055 1319 1309 1546 1534
SE 652 658 756 732 618
END

SECTION StochasticTerminals
ST 1 1 1 1 1 1
ST 2 1 1 1 1 1
ST 3 1 1 1 1 1
ST 4 1 1 1 1 1
END

EOF
```

6 Experimental Results

As mentioned before, to generate the test instances we used the archive of TSP instances from the TSPLIB archive, so all the original instances of the generated instances mentioned in this analysis can be found there.

When creating the instances we had 2 variables to tweak, which were *number of scenarios (S)*

and *maximum inflation rate (R)*. So for each original TSP instance we generated STSP instances with $S \in [5, 10, 20]$ and $R \in [0.3, 0.5, 1.0]$. That way there is 9 different STSP instances for each TSP instance, but some we were not able to run because of our limited time (mainly the bigger instances with many nodes and scenarios).

All experiments were run with a single thread on an Dell Inspiron 7560 with the following specifications:

- RAM
  - Capacity: 16 GB
  - Clock: 2133 MHz
- Caches
  - L1 Cache: 128 KiB
  - L2 Cache: 512 KiB
  - L3 Cache: 4 MiB
- CPU
  - Model: Intel Core i7-7500U
  - Frequency: 2.70 GHz
  - Width: 64 bits
  - Clock: 100 MHz

One important pattern we noticed when analyzing our experiments was that most of the solutions ignored the future resources. This means that most of the time the framework ends up finding a Hamiltonian Cycle in the present scenario and selects all the resources already in the present scenario. That way all the future scenarios are already solved, ending up with costs equal to 0. That makes sense since the terminal set is always the same and you can ignore the inflated resources.

Even though that happened in most of our instances it did not happen in all of them. In the Figure 4 we can see a heatmap showing which of our instances had a solution which actually selected future resources. As we thought would be the case, this is highly dependent on the original TSP instance, but has a higher chance of happening for smaller inflation rates (R), as we can see in *dantzig42*, *eil76*, *eil101*, *ch130* and *brg180*.

It is also important to note that, even though there are instances that select future resources, most of these select most of the resources in the present scenario and only a small number of future resources in each scenario. That shows that buying them all on the present scenario would still present a possibly close score. We believe that this would be seen less if the terminal node sets for each future scenario were different.

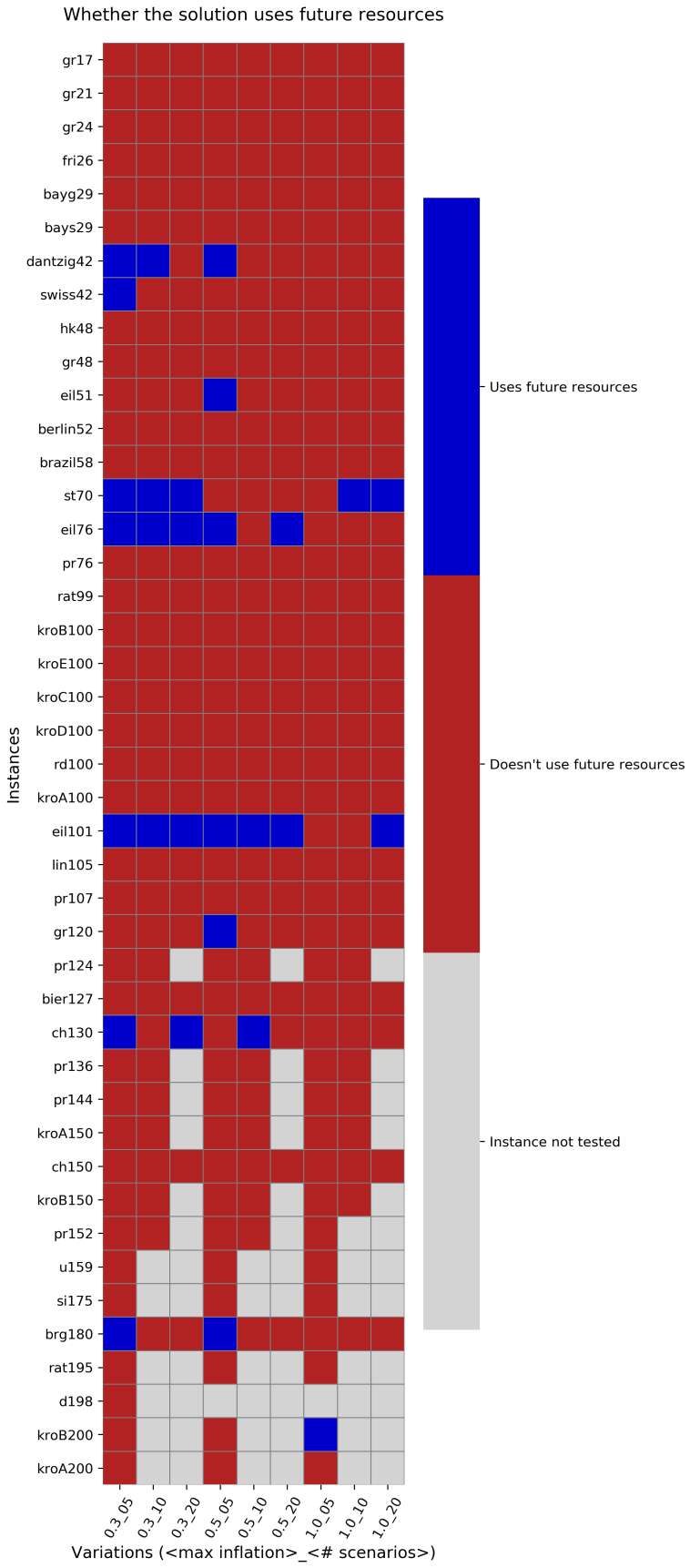
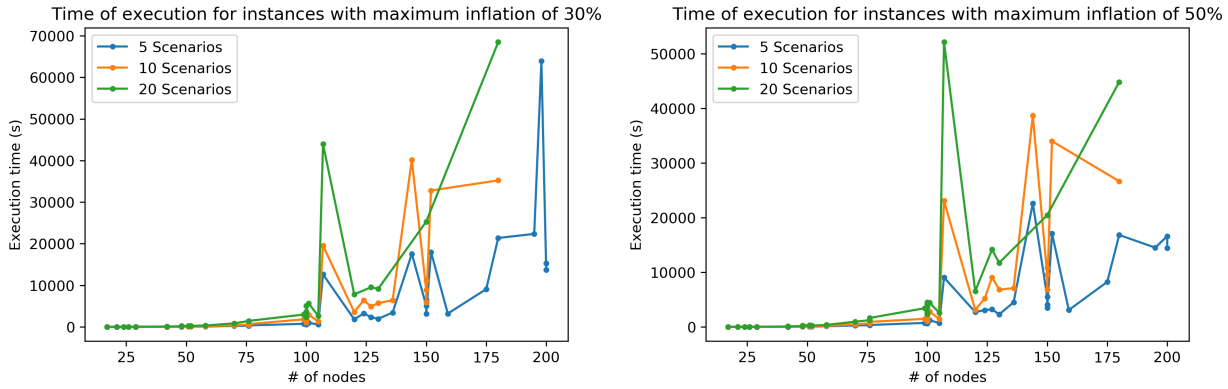


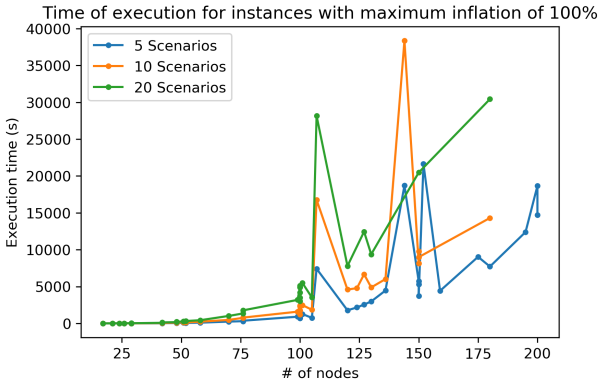
Figure 4: Heatmap of whether the solution of the instance uses any of the future resources.

In the Figure 5 we can see the plots for the execution time in seconds for all our instances. We can see that the instances tend to have a lower running time the lower the number of scenarios (S), which was expected, but their execution times also tend to lower the higher is the maximum inflation rate (R). We believe that has to do with the fact that higher inflation rates mean that the future edges are more easily discarded, as we saw in Figure 4, and the solution converges more quickly.



(a) Execution time (s) for experiments with maximum a max of 30% inflation.

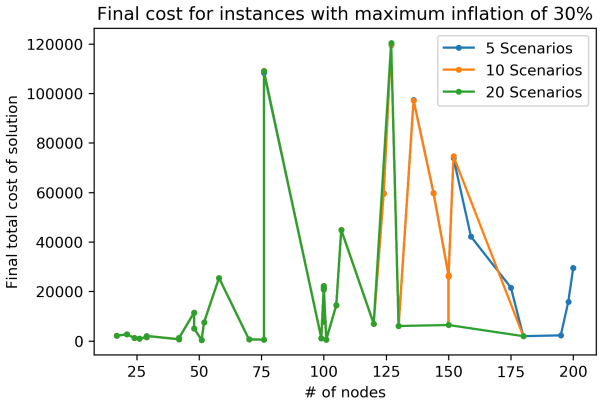
(b) Execution time (s) for experiments with maximum a max of 50% inflation.



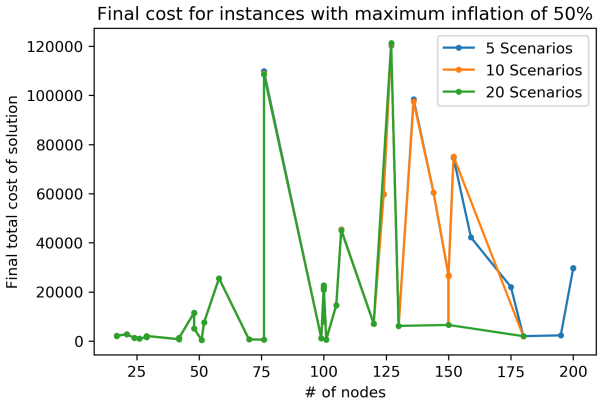
(c) Execution time (s) for experiments with maximum a max of 100% inflation.

Figure 5: Graphics for execution time (s) in all experiments.

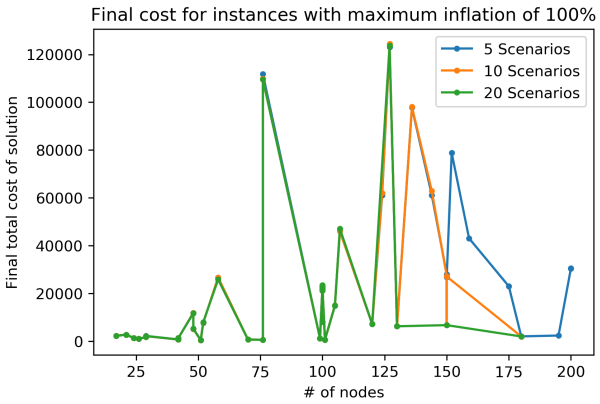
In the Figure 6 we can see the plots for the final cost found by our framework for each instance. As we can see, the costs change very little with the variation of the number of scenarios and inflation rate. This also has to do with how the future edges are easily discarded, that way the score for different variations of the same original TSP instance are the same or at least close.



(a) Final cost for experiments with maximum a max of 30% inflation.



(b) Final cost for experiments with maximum a max of 50% inflation.



(c) Final cost for experiments with maximum a max of 100% inflation.

Figure 6: Graphics for the final cost in all experiments.

In Table 1 we can see the final cost (C) and execution time (T) for all the instances we tested in the project.

	R = 0.3						R = 0.5						R = 1.0					
	S = 5		S = 10		S = 20		S = 5		S = 10		S = 20		S = 5		S = 10		S = 20	
Name	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T
gr17	2090	1.4	2088	1.9	2090	5.2	2226	1	2090	2.7	2090	4.1	2226	1	2292	2.4	2237	4.3
gr21	2709	1.7	2709	3.4	2709	6.7	2709	1.9	2709	3.6	2709	7	2709	1.9	2709	3.5	2709	8.9
gr24	1272	2.9	1272	5.9	1272	11.9	1272	3	1272	5.9	1297	12.2	1333	3.7	1337	6.5	1349	13.1
fri26	937	3.7	937	7.2	937	14.3	940	3.7	937	7.5	937	16	1002	4.5	957	7.7	957	15.3
bayg29	1641	5.9	1634	10.8	1636	25.2	1656	5.7	1656	11.1	1660	23.3	1720	6.1	1727	11.9	1735	25.2
bays29	2038	5.8	2034	12.1	2036	23.7	2073	5.3	2073	11.4	2036	23.2	2172	5.7	2071	11.2	2131	23.1
dantzig42	699	21.7	700	40	700	99.5	700	18.7	700	40.2	700	80.1	699	19.6	700	43.9	700	79.5
swiss42	1274	23.7	1273	44.1	1273	95.4	1273	23.1	1290	53.7	1279	97.2	1347	32.3	1273	58	1317	138.2
hk48	11470	36.6	11545	81.5	11474	143.5	11511	37.8	11474	76.5	11484	151.8	11689	49.1	11910	108	11741	172.5
gr48	5083	58.9	5092	100.9	5067	223.6	5066	51.6	5104	117.9	5118	228.6	5206	49.7	5148	111.8	5190	213.5
eil51	412	70.4	413	117.7	413	238.7	418	84.5	415	173.8	416	335.3	430	72.7	425	160.6	420	271.7
berlin52	7554	58.2	7526	113.5	7554	231.9	7554	60.5	7634	127.7	7554	241.9	7651	59.7	7725	139	7948	322.5
brazil58	25464	89.3	25464	174.8	25475	341.2	25510	113.4	25505	173.1	25505	360.5	26609	100.8	26640	233.8	25823	434.8
st70	654	245.3	652	404.9	654	844.8	655	258.3	663	468.2	654	949.7	676	241	678	475.7	679	1012.9
eil76	527	277.6	526	538.7	526	1413.4	527	329.4	528	643.5	531	1238.2	546	302.7	539	751.8	534	1347.4
pr76	108364	354.3	109138	670.9	108967	1445	109912	347.6	108927	914.4	108560	1634.3	111838	363.6	109804	783.3	109521	1769.8
rat99	1191	743.6	1188	1815.8	1188	2967.7	1188	725.8	1195	1479	1192	3427.3	1214	909.1	1200	1592	1197	3190.8
kroB100	22310	984.9	22222	1907.3	22202	3485.1	22328	975.1	22260	1910.6	22307	4488.6	23141	1332.1	22757	2443.4	22763	4944.6
kroE100	22221	990.7	22297	1896	22162	5029.9	22123	1081.4	22604	2273.6	22663	3874.1	22806	1020.4	23266	2351.5	23386	4873
kroC100	20809	553.2	20778	1105.2	20807	2349.8	21020	619	20816	1106.1	20877	2317	21255	719.5	21217	1233.3	21260	3479.4
kroD100	21464	705.2	21302	1722.4	21308	3472.8	21567	982.5	21544	1633	21425	4069.4	22237	1052.8	21715	1725.3	21923	5132.4
rd100	7892	658.8	7879	1332.8	7867	2498.9	7993	706.9	7873	1550	7910	2671.3	8268	735.7	8149	1709.7	8142	3041.8
kroA100	21470	735.7	21410	1786.7	21318	3684.3	21419	1016.4	21449	1727.2	21775	3625.5	21895	1039	21947	1764.4	21918	4185.1
eil101	609	984.2	610	2948.2	611	5664.4	617	1220.4	612	2801.9	610	4420.4	632	1278.8	637	2489.5	630	5504.9
lin105	14420	568.3	14456	1297.5	14506	2659.6	14699	706.5	14624	1450.4	14516	2596.7	14916	753.9	15048	1853.2	14822	3527.6
pr107	44921	12625.8	44956	19536.9	44817	43971.6	45406	9034.1	45267	23098.5	45027	52146	46153	7417.3	45961	16774.8	47086	28160.4
gr120	7005	1813.2	7015	3606.3	6976	7833.8	7127	2714.4	7023	3124.9	7027	6513.7	7172	1779.1	7161	4589.1	7217	7787.3
pr124	59612	3258.1	59504	6413.6	X	X	59712	3069.5	59789	5218	X	X	61079	2165.9	61941	4786.1	X	X
bier127	119552	2343.3	119605	4892.9	120343	9522	120344	3215.8	120821	9065.6	121292	14157.1	123104	2558.1	124409	6668.1	123645	12451.4
ch130	6090	1948	6070	5698	6062	9153.1	6143	2294.8	6127	6838.8	6142	11739.1	6271	2976	6263	4882.2	6225	9375.6
pr136	97360	3444.8	97102	6410.9	X	X	98449	4499.1	97521	7093.4	X	X	97780	4460.1	98124	6013.1	X	X
pr144	59821	17576	59676	40156.6	X	X	60437	22627.1	60455	38661.9	X	X	61147	18707.2	62860	38349.3	X	X
kroA150	26512	5106.1	26676	8878.6	X	X	26722	5499.1	26793	9543.5	X	X	28028	5720.1	27266	8169	X	X
ch150	6475	6447.9	6498	11148.5	6459	25306.1	6610	4060.2	6573	10806.8	6536	20464.1	6775	5328.1	6739	9777.1	6704	20467.4
kroB150	26224	3158.7	26104	5883.6	X	X	26544	3478.8	26392	6821.9	X	X	26720	3723.4	27049	8989.8	X	X
pr152	73887	17997.9	74624	32792	X	X	74632	17108.9	75172	34013.2	X	X	78816	21657.9	X	X	X	X
u159	42198	3198.9	X	X	X	X	42171	3090.8	X	X	X	X	43042	4419	X	X	X	X
si175	21589	9082.5	X	X	X	X	22035	8235.5	X	X	X	X	22975	9034.2	X	X	X	X
brg180	1953	21381.6	1950	35225.9	1950	68475	1979	16828.7	1950	26658.7	1960	44845.3	2000	7713.9	1970	14307	1960	30431.1
rat195	2276	22361.5	X	X	X	X	2296	14514.3	X	X	X	X	2355	12398.7	X	X	X	X
d198	15783	63903.7	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
kroB200	29543	13685.8	X	X	X	X	29638	16613.8	X	X	X	X	30550	18660.4	X	X	X	X
kroA200	29575	15333.6	X	X	X	X	29796	14447.5	X	X	X	X	30405	14717.6	X	X	X	X

Table 1: Final cost (C) and execution time (T) for every instance tested. Instances with an "X" were not tested.

7 Future Work

The main task for a future work would be expanding the framework for the more general case where the terminal node set for the future scenarios can be different subsets of the node set. Doing this would be interesting for two reasons:

1. That way we can see how the selection of future resources functions for different terminal sets, since even though finding a Hamiltonian Cycle in the present scenario would still solve all future scenarios, the different disposition of terminal nodes have a higher chance of making this approach too costly.

2. We have not found in the literature any study solving this general case of the Stochastic TSP, which would make a good addition to the STSP studies.

We took a quick look at the "TSP Price Collecting" problem, which could be a good candidate to solve our deterministic problem in the future scenarios.

8 Conclusion

Unfortunately we were not able to develop the more general framework like we intended to, which would probably be the first implementation in the literature, but we were still able not only

to expand the study for the Two-stage Stochastic Traveling Salesman Problem, which is not largely studied, but also to expand the study for the Evolutionary Framework, which is a very useful framework for resource allocation problems.

References

[1] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," 1960. [Online]. Available: https://dl.acm.org/citation.cfm?id=321046

[2] B. Gavish and S. C. Graves, "The travelling salesman problem and related problems," 1978. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/5363/OR-078-78.pdf

[3] F. Maggioni, G. Perboli, and R. Tadei, "The multi-path traveling salesman problem with stochastic travel costs: Building realistic instances for city logistics applications," 2014. [Online]. Available: https://www.researchgate.net/publication/268077984_The_Multi-path_Traveling_Salesman_Problem_with_Stochastic_Travel_Costs_Building_Realistic_Instances_for_City_Logistics_Applications

[4] L. Bertazzi and F. Maggioni, "Solution approaches for the stochastic capacitated traveling salesmen location problem with recourse," 2014. [Online]. Available: https://www.researchgate.net/publication/265219335_Solution_Approaches_for_the_Stochastic_Capacitated_Traveling_Salesmen_Location_Problem_with_Recourse

[5] P. Adasme, R. Andrade, J. Leung, and A. Lisser, "A two-stage stochastic programming approach for the traveling salesman problem," 2016. [Online]. Available: https://www.researchgate.net/publication/288826667_A_Two-Stage_Stochastic_Programming_Approach_for_the_Traveling_Salesman_Problem

[6] P. H. D. B. Hokama, M. C. S. Felice, E. C. Bracht, and F. L. Usberti, "Evolutionary framework for two-stage stochastic resource allocation problems," 2018.

[7] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," 1973. [Online]. Available: https://doi.org/10.1287/opre.21.2.498

[8] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," 1998. [Online]. Available: http://akira.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH_REPORT.pdf

[9] J. F. Gonçalves and M. G. C. Resende, "Biased random-key genetic algorithms for combinatorial optimization," 2011. [Online]. Available: https://doi.org/10.1007/s10732-010-9143-1

[10] G. Reinelt, "Tsplib 95," 1995. [Online]. Available: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp95.pdf