Katholieke Universiteit Leuven

Capita Selecta Computer Science

Metaheuristics

Final Report

Authors:
Afraz Salim
Florian Vanhee

Professors:
Patrick De Causmaecker
Pieter Leyman

2018

Contents

1 Introduction

The Intermittent Travelling Salesman Problem (ITSP) (Pham and De Causmaecker, 2015) is an adaptation of the well-known Travelling Salesman Problem (TSP) where the goal is the find the shortest path visiting each city (or node of a graph) once and circling back to the first city of the tour. The ITSP is mostly inspired by industrial tasks such as drilling or texturing. It means that there is no longer a constraint on the number of time a city may be visited but instead, each node requires a number of units to be processed on the premises, which corresponds to an amount of time needed to be spent at each node. Every time unit spent processing will increase a temperature at this node until a max threshold is met. Then there are two options, either wait for the temperature to cool down or go process another node in the meantime, coming back later.

In this work, we choose to tackle the ITSP by using Evolutionary Algorithms (EAs). We also introduce a new challenge: the traffic. The focus of ITSP lies on the fact that a worker can finish his work in the shortest possible time interval. However, a worker may require some additional time due to the traffic, apart from the time which is required to move to another node at a distance d. To make our experiments, a specific dataset has been created, based on TSP's coming from TSPLIB (University).

In the next section, we will describe the general representation for the individuals of population. The section Design of Experiments will detail how we are conducting our experiments, that is specify the dataset that is going to be used, the biases introduced and some technical limitations. Afterwards, an in-depth description of the core components of the EA will be provided. The section Local Search explores two different ways to incorporate local heuristics into the algorithm to improve its performance. The temperature is a crucial point in ITSP and different types of functions can model the evolution of it over time. Therefore, in the section Temperature Profiles we investigate the impact of different temperature profiles on the performance of the algorithm using two representations. Finally, we address the traffic and how we incorporate it inside our solution.

There is no section specifically dedicated to the results and the analysis of those, but rather we choose to associate to each section (after Design of Experiments) its own results and analysis. This way, it breaks down the flow into small, digest comparisons which allows us to focus more on each feature. Another reason behind this choice is to keep the number of runs manageable to limit the time needed to run all the experiments. It means we won't be using very high dimensional tableau (Cavazzuti, 2013) and not every combination of parameters will be tested, thus we might miss some correlation effects. However, those comparisons ate still sufficient to explain each aspect of this algorithm.

2 Representation

Throughout the various features, the representation we are using to encode a potential solution as an individual always stays the same. Occasionally, some parts are deactivated

or fixed as described below, but the global idea doesn't change. Our representation is mostly inspired by the one described in (Leyman et al., 2017).

Our representation is split over several lists so it is clearer what is manipulated and it also gives the possibility to use operators independently on the different parts.

- The node list (NL) which is the list of the nodes in the order of the visits. It is important to note that a node may appear several time in that list.
- The processing time list (PTL) which for each visit specify the time spent processing at a given node.
- The split list (SL) which indicates how many time each node is visited.

The main difference between our representation and the one from the paper cited above is the way to handle the constraints set by the ITSP. Indeed, there are two type of constraints one must satisfy in order to produce a valid ITSP solution, the first is that the temperature of a node during a visit may not exceed the maximal temperature. The second is that for each city the amount of units to process is reached. In the paper, they make use of repair functions after mutation and crossover operators to maintain the solution in a valid state. In our case, we use penalty-based fitness function to penalise individuals who violate the constraints, by assigning them a lower fitness function. Here is a general scheme of a penalty-based fitness function(Eiben and Smith, 2003).

$$fitness(\bar{x}) = f(\bar{x}) + \sum_i w_i \cdot penalty(x_i)$$

where $f(\bar{x})$ corresponds to the objective function and $penalty(x_i)$ equals zero if there is no violation and is positive otherwise. This formula mostly relies on the use of weights to lower the fitness at the right level. If the weights are insignificant, invalid solutions will thrive, on the other hand if the weights are too high it might reject very good solution which only have a small number of violations which could be fixed by the mutation and crossover operators. Moreover, it is also possible that we may want the pressure we put on the solution to not violate constraints to change depending on time, sometime allowing more invalid solutions sometime focusing on turning our invalid solutions into valid ones (Eiben et al., 2000). Thus finding the right weights has a big impact on the performance of the algorithm and is a problem in itself. That is why, we choose to let the algorithm take care of that task itself by adding the weights directly inside the chromosome of every individual. This leads us to the addition of two new lists one for each type of constraints as said above.

- C_1 is the list of weights associated to the requirements of units to process for each node. Each element of this list corresponds to a weight that will be applied if not enough units have been processed for the city corresponding to the index of the element in that list
- C_2 is the list of weights associated with the constraints ensuring that the current temperature at a node for a given visit doesn't exceed the maximal temperature.

Although the representation is established, it is important to define a fitness function which evaluates the quality of a solution in order to apply selection pressure in the EA. As shown above, the fitness function will consist of two parts: the objective function which evaluates the chromosome of an individual as if it were a valid solution. To do so, it mostly consists in adding the time spent travelling and the time spent processing units. Then there is the penalty that is added to the first part to make a up the final score. The weights have an important impact in the calculation of the penalty and in the case of self-adaptive weights, the algorithm may be tempted to cheat by lowering the weights instead of actually solving the constraints. To address this issue, the selection is not based on a global ranking but rather on small tournament involving 4 individuals. Then new C1 and C2 lists are created by taking for each element the highest weight between the four of them and these lists are then used in the fitness evaluation instead of the actual C1 and C2 lists of the individuals. This way, if a mutation changes a weight from a high value to a low one for someone in the population it is not likely that it will have an effect because someone else in the tournament may have an higher value for that weight negating the effect of the mutation on the fitness value of the individual.

Tournament is the technique that is used both in the parent selection phase where we are creating the mating pool and in the survivor selection when we are selection the best parents and offspring to make the next generation. In the parent selection, we are selecting the best of four individuals randomly sampled with replacement from the population until the mating pool is equal to the number of offspring. In the survivor selection, it is the worst of the four individuals randomly sampled with replacement from a new population composed of both parents and offspring that is discarded. We are repeating that until the size of this population is no larger than the maximal individuals authorised in the population.

3 Design of Experiments

To draw conclusion from the various ideas we explore with our algorithm, a number of experiments have to made. For these experiments to be relevant, they need to share the same settings as much as possible except for the parameters being investigated in the comparison. The main component of the setting that should always be the same is the dataset upon which our experiments are ran. We chose to create one so that it can fit our needs as much as possible.

3.1 The Dataset

The node location in each of our instances is coming from Tsplib, a library of TSPs which can be found (University). In order to keep the difference in the number of nodes across instances in a reasonable range and also to keep the computation time acceptable, the number of nodes in an instance is constrained between 50 and 100. As most of the problem instances in Tsplib where above that threshold, new instances where created by randomly sampling a valid number of nodes from instances with more than 100 nodes.

Now that we have the node locations, we use that information to compute the remaining parameters which are the maximal temperature and the requirements.

1. Scaling down every the coordinates of every node so that no node is located further than 100 units in the x or y-axis from the origin. This way, the variety in the disposition of nodes is kept without having different order of magnitude in the distances.

2. Using the average distance between each pair of node to compute the average requirement for a node. The mean is sampled between 10% and 40% from the average distance.

3. Computing the standard deviation as a number sampled from 20% and 80% of the average requirement.

4. The requirement for each node is sampled from a normal distribution using the mean and the standard deviation previously calculated.

5. The maximal temperature is computed as a product of the mean requirement and a difficulty variable that is randomly sampled from the list $(0.5, 1, 2)$.

This technique is used to build each of the 50 instances that compose the dataset. Each of the comparisons below are performed after running the algorithm through the whole dataset and averaging the results of each instance.

3.2 Biases

The way the dataset is constructed and the experiments are done introduces some biases that we need to be aware of. First of all, even though some instances have their node locations scaled down, There are still great disparities between the distance time needed to process units in each node. The number of nodes and the maximal temperature also play a role. Indeed, with more nodes there are more travels and with a lower maximal temperature allowed it means there will be more back and forth between nodes. Meaning that a version of the algorithm that performs well on one or two of those problems with high distance might get a better overall result even if it performs quite poorly on the others.

A similar problem arises with the average processing requirement. To see how the algorithm fares with different ratios of average travel distance on average processing time we choose randomly the average processing time as said in the previous section. If one version of the algorithm is performing better on lower ratio and another on higher ratio, the latter will get a better score even if it is not necessarily justified.

Lastly, there is also a bias that is more of technical nature. Indeed, This algorithm is computationally expensive due to the general scheme of an EA with many individuals undergoing selection, recombination and mutation at each generation. Thus, running a version of the algorithm on the whole dataset may take much more than 1 hour even with

reduced numbers of individuals and generations. As a trade-off between the time spent on each run and the number of comparisons to make, the number of individuals, offspring and generations have been kept relatively low, preventing the algorithm to converge on a specific solution in most cases. Therefore, versions of the algorithm that tend to converge fast have an advantage over those that need more steps but might eventually lead to better results.

Different solutions have been used to keep the comparisons as relevant as possible, such as reducing all the node coordinates to the same range or establishing different difficulty profiles. Other solutions might consist in optimising the algorithm or allowing more computational time per run and also using a weighted average system to compute the results across the 50 instances. A weighted average could take into account things such as the average processing time per city or the number of cities to adapt the results of a specific instance.

4 Search Profiles

The representation as defined above, is not enough to fully understand how the algorithm behaves. The search profiles represent different ways to use the representation to explore the search space. As (Leyman et al., 2017) is the primary source of inspiration for our algorithm, it is logical to base our different search profiles on the ones described in the paper in which they are called 1L, 2L and 3L in reference to the number of list used in the representation.

Based on those representations and the notion of greediness, we establish three different search profiles:

- Free, where the size is fixed and the split list doesn't count and both the node list and the processing time list can evolve freely within the range of acceptable values with the use of mutation and recombination operators.
- Fixed which resembles Free except now the split list is fixed at the initialisation of the population and cannot change further. To enforce this constraint, a repair function is used after a recombination to randomly remove nodes which number of occurrences in the node list is greater than the value in the split list at the corresponding index until the constraint is no longer violated. The same reasoning applies for nodes that are not visited enough.
- Classic is the most general search profile where each list can freely evolve thanks to the operators. It is important to note that we still need the repair function to ensure that the values in the node list corresponds to the ones in the split list at any time.

Table 1 is a comparison between the three different search profiles. The structure of the results is a 4-tuple whose values are: the average fitness score of the best individual at the end of the simulation, the average number of violations for the constraint c_1, the

Free	Fixed	Classic
(3136, 7.2, 0.21, 4%)	(2623, 1.8, 0.03, 64%)	(3089, 7, 0.15, 18%)

Table 1: Comparison of the results of the various search profiles using parameters suited to their specific needs.

average amount by which the constraint C_1 is violated[1] and the success rate, which is the percentage of the runs where no constraints where violated in the best solution. The reason behind this choice is simply that they best represent the quality of a run. In several sections, other values are displayed for comparisons when they can provide further insights.

In Table 1, each search profile has been given the parameters that seem to fit it best. Fixed is ahead mostly because it has a reduced search space, with the split list being fixed. The classic profile is not far behind and in some cases where the optimal solution differs greatly from a fixed split list initialised greedily it is able to find better solutions and even the optimal one where the fixed approach will always be blocked by its restricted search space. Free lags behind, not necessarily in score but in the success rate. This is mostly because it is the worst of two world, the search space is not as restricted as in the fixed search profile to help it converge faster. On the other hand, it does not have flexibility on the size of the node list as the classic profile do. In regards of these results and to save some computation time, the free search profile is dropped from the other comparisons.

5 Mutations

Mutation is an important part of the EA as it is the operator that allows to jump to other part of the search space by randomly changing value in the chromosomes. For every list except the node list, we use the random resetting mutation which runs through each element of the list and reset its value to one randomly sampled from its domain with a certain probability. For the node list, we chose three different operators commonly used for TSP hoping they could be as effective in the context of ITSP (Otman et al., 2011)(Gremlich et al., 2005). Each operates on the entirety of the list instead of element-wise.

- Inversion operator takes two point in the chromosome and reverse the order of the elements between them (included).
- Swap also takes two point in the chromosome and swap their values.
- Adjacent swap exchanges the values of two adjacent point in the chromosome.

A trade-off has to be found for the mutation rate of each list to guarantee enough diversity without preventing the algorithm to converge.

[1]This number is obtained by summing the amount of the violation divided by size of the c_1 list. Therefore, It must be understand as the average amount of violation at each city.

NL	PTL	SL	C1	C2
0.4	0.02	0.01	0.1	0.1

Table 2: Table representing the mutation rate for each list in the representation.

	Inversion	Swap	Adjacent
Fixed	(3109, 0.6, 0.01, 74%)	(2623, 1.8, 0.03, 64%)	(2973, 0.7, 0.01, 70%)
Classic	(3239, 7.4, 0.15, 16%)	(3089, 7.1, 0.15, 18%)	(3051, 5.5, 0.11, 26%)

Table 3: Comparison of the effect of different mutation operators on the search profiles.

From Table 3, the swap operator seems to provide the best score at the price of constraint violations which values are above the ones obtained with the adjacent operator. The inversion operator does not provide as high results nor does it provide a better constraint management for the classic representation even though it is at the base of the 2-opt local heuristic which proves efficient as described in section 8.

6 Initialisation

Initializing the individuals in the population with some specific values might help the algorithm converge faster. However, it might lead it to converge too soon as well. That is why, we are comparing four different initialisation techniques and their impact on the overall results of the search profiles.

- Random initialises the node list and processing time list completely randomly and then generates the corresponding split list.
- Greedy SL initialises the split list of each individual so that the number of times a node is visited is equal to the node requirement divided by the max processing time possible round up.
- Greedy SL + PTL does the same as Greedy SL except it also initialises each element of the processing time list to the max except for the last visit of each node where the time is adjusted to not over process.
- Classic SL uses the same split list as Greedy SL but then applies random deviation to provide diversity in the population. The processing time list is generated at random.

	Random	Greedy SL	Greedy SL + PTL	Classic SL
Fixed	(4194, 20.1, 1.12, 0%)	(2610, 3.4, 0.07, 52%)	(2623, 1.8, 0.03, 64%)	(3419, 15.3, 0.69, 0%)
Classic	(3235, 7.4, 0.18, 20%)	(2845, 5.1, 0.11, 40%)	(2765, 3.1, 0.07, 42%)	(3089, 7, 0.15, 18%)

Table 4: Comparison of the different initialisation methods on both Classic and Fixed search profiles.

		C Prop = True				C Prop = False			
		C_2 = True		C_2 = False		C_2 = True		C_2 = False	
		$C1_{all}$ = True	$C1_{all}$ = False	$C1_{all}$ = True	$C1_{all}$ = False	$C1_{all}$ = True	$C1_{all}$ = False	$C1_{all}$ = True	$C1_{all}$ = False
Classic	Score	4155	4125	3213	3088	2106	2094	2017	1994
	Success Rate	0%	2%	20%	22%	2%	2%	2%	2%
	Avg C_1 violations	33.3	33.8	7.8	6.6	39.9	39.5	27.1	26.8
	Avg C_2 violations	0.4	0	2.2	0.2	0	0.02	0	0
	Avg amount of violations for C_1	0.73	0.73	0.17	0.16	3.44	3.44	2.61	2.6
	Avg amount of violations for C_2	0.003	0	0.001	0	0.0001	0	0.0002	0
	avg violated weights c_1	21.4	21.5	16.2	15.9	21.3	21.4	20.2	20.4
	avg violated weights c_2	14.9	0	18.4	0	1	0	11	0
Fixed	Score	3019	2943	2690	2598	2631	2558	2366	2337
	Success Rate	38%	30%	62%	64%	18%	14%	36%	36%
	Avg C_1 violations	3.24	3.18	1.56	1.58	6.4	6.94	4.9	3.9
	Avg C_2 violations	0.08	0	0.14	0	0.14	0	0.3	0
	Avg amount of violations for C_1	0.06	0.05	0.03	0.06	0.51	0.54	0.43	0.34
	Avg amount of violations for C_2	0.0006	0	0.001	0	0.003	0	0.004	0
	avg violated weights c_1	14.2	12.9	13.8	15.2	13.9	14.4	14	12.2
	avg violated weights c_2	17	0	17.4	0	18.7	0	17.4	0

Table 5: Table showing the impact of different constraint handling techniques on the results. The tuple format has been replaced by a format allowing more attributes to be displayed.

Looking at Table 4, we can observe that initialisation matters and is not only an early boost. The search space being so large, it is important that the first population already contains solutions in the right part of the search space. We can also see that the fixed representation is more impacted by the initialisation technique, that is mostly due to the fact there is no operators that will randomly replace a node by a new one, they only change the order of visits, it then becomes very dependent on a good start where the classic representation by varying the split list can create and remove visits.

7 Constraints

Handling the lists C_1 and C_2 well is important to keep the constraint violations as low as possible. 3 different Boolean variables have been introduced to this effect, making a total of 8 possible combinations.

- C Prop tells the algorithm if it has to use the amount of violation of a constraint in the calculation of the penalty.
- C_1 All is a Boolean variable that when set to false will tell the algorithm to only check if the node requirement have been under-processed. If it is set to true, it will check if it has been over-processed as well.
- C_2 On enables or disables the use of the C_2 list to check if the maximal temperature has been exceeded during a visit. If it is set to false, the algorithm will always wait the minimal amount of time before starting to process in order to ensure the maximal temperature is kept below the threshold.

From Table 5, it is clear that the score is becoming better (lower) as we put less pressure on constraints validity by setting the variables to False. However, the success rate does not follow the same tendencies and a good trade-off can be found when we are enabling calculation of the penalty proportionally to the amount of violation and deactivating the

	None	2-opt	Hillclimbing crossover
Fixed	(2623, 1.8, 0.03, 64%, $8 \cdot 10^4$)	(1969, 2.6, 0.06, 48%, $4.6 \cdot 10^5$)	(2202, 8.1, 0.23, 6%, $1.1 \cdot 10^6$)
Classic	(3089, 7.1, 0.15, 18%, $8 \cdot 10^4$)	(2494, 7.6, 0.18, 22%, $4.6 \cdot 10^5$)	(3529, 18.5, 0.61, 2%, $1.1 \cdot 10^6$)

Table 6: Table showing the impact of local searches on the search profiles. The 5^{th} element of the tuple corresponds to the number of fitness evaluations calculated.

temperature constraints. The proportionality puts more pressure on the algorithm to find solutions where constraints are violated as less as possible, driving it near the feasible area. On the other hand, the wait heuristic reduces the pool of potential violations while keeping better results than when the constraints on maximal temperature are activated. the *avg violated weights* c_1 and c_2 are calculated by averaging the weights of every constraints that has been violated. The range of value is between 1 and 50 and therefore an average of the weights without any interventions of the algorithm should lie near 25. We can observe that although the weights are always lower than 25 they are not at all near zero as it would have been the case if the algorithm only tried to cheat by minimising the weights. It also appears that the average violated weights are found when the number of violations is minimal showing that the algorithm lowering the weights is not directly related to avoiding to enforce constraints and minimising the penalty.

8 Local Search

A potential area of improvement for the algorithm is the quality of the score itself. At this point, the algorithm is perfectly capable to maintain a healthy level of diversity and it should not converge too fast if a local search is being performed. On the other hand, the local heuristic could help search the very large space of solutions in an efficient way. Therefore, we introduce two local search techniques, one that has been used with success in the context of TSP and another that gives interesting results with other EAs.

- 2-opt (Croes, 1958), which consists in randomly applying the inversion operator described above to an individual and replace him with the new version if the results are better.
- Hillclimbing crossover (Jones, 1995), which is a modification of the crossover operator to incorporate local search. at each recombination, there are *MAX ATTEMPTS* to create offspring better than parents by making random 1-point crossover. If a best offspring is found we repeat the operation using the offspring as parents *MAX STEPS* and when *MAX STEPS* is reached or no better offspring is found we drop the worst of the two parents and take one randomly from the mating pool. That is called a jump and we do that *MAX JUMPS*. That gives us a 3-tuple of parameters. In the article they used (10, 3, 1000) values but to keep computation reasonable we are using (10, 3, 5).

The first thing to notice in Table 6 is the much higher number of fitness evaluations while using a local search due to the fact the local search itself makes heavy use of fitness

Increase	Linear		Quadratic		Exponential	
Decrease	Fixed	Classic	Fixed	Classic	Fixed	Classic
Linear	(2623, 1.8, 0.03, 64%)	(3089, 7.1, 0.15, 18%)	(6116, 12.3, 0.25, 12%)	(6703, 26.5, 0.9, 2%)	(8985, 19.3, 0.42, 4%)	(8685, 37.9, 1.6, 2%)
Quadratic	(2591, 1.5, 0.03, 70%)	(3064, 6.9, 0.15, 24%)	(6134, 13, 0.25, 14%)	(6680, 27.5, 0.9, 2%)	(8968, 19.5, 0.4, 8%)	(8713, 37.7, 1.5, 2%)
Exponential	(2590, 1.7, 0.003, 62%)	(3058, 6.9, 0.15, 26%)	(6157, 12.5, 0.24, 10%)	(6638, 27.1, 0.9, 2%)	(9005, 19.5, 0.42, 6%)	(8645, 36.9, 1.6, 2%)

Table 7: Comparison of the results of Fixed and Classic using different combinations of temperature profiles.

evaluation to explore the search space. This introduces a bias since with more evaluations the local searches will converge faster, but it does not mean it will find a better path in the end. We can also see that with the Hillclimbing search the number of constraint violations drastically increases. Indeed, to save some computational time from an already very heavy local search, the fitness comparisons between the offspring and the parents does not use a common weight vectors created from the maximal values of each one as said in section 2. It is then advantageous to take a solution which violates a lot of constraints but with minimal weights so that it is not harshly punished. On the overall, 2-opt provides a nice improvement compared to the version with no local search if a small loss in the success rate is tolerable.

9 Temperature Profiles

The increase in the temperature while processing a unit and the cooling down that follows are interesting aspects to considers in real-world applications where they often do not make use of linear functions to describe evolution of temperature. As we are mostly relying on the representation used in (Leyman et al., 2017), it is relevant to see if our algorithm share the same results as theirs regarding the use of different temperature profiles even considering the changes made. For this comparison the same type of functions as the ones in the paper are used: linear, quadratic and exponential.

From this table, we can draw the same conclusion has in the paper cited above, that is the greedy approach (fixed) seems to always outperforms the more general one (classic). In the case of exponential increase however, the classic search profile holds better results. With such high value, the algorithm is more incited to violate constraint in order to get better results and it seems that the classic has more flexibility to do so. One can also observe that even though the scores nor the constraint violations change much, the success rate tends to increase when a faster decrease temperature function is used. This is rather logical as it means we need less time before visiting a node again and therefore more permutation of the order of visits are satisfying the constraints.

10 Traffic

In this section we explain how to represent the concept of traffic, which is equivalent to a delay changing over time, before illustrating a few techniques that are used to optimise the algorithm. The most straightforward way to represent traffic, would be to use functions. Periodic ones such as sinusoidal function, can even be used to model its the periodicity. As discussed in paper(Evolutionary Algorithms and Dynamic Optimization Problems), traffic functions can have one of the following changes:

- A function which represents traffic can be fully replaced by a new traffic function.
- The width or the height of the curve can change while the curve remains fixed. Figure 2
- A curve can be translated to the new position with or without changing it's width or height as show in figure in 1

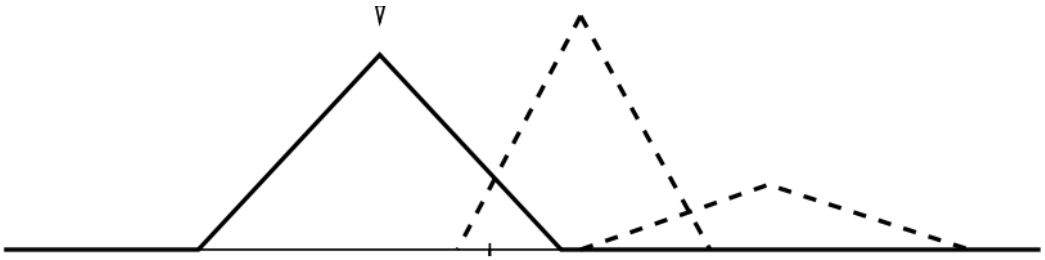

Figure 1: Moving hill

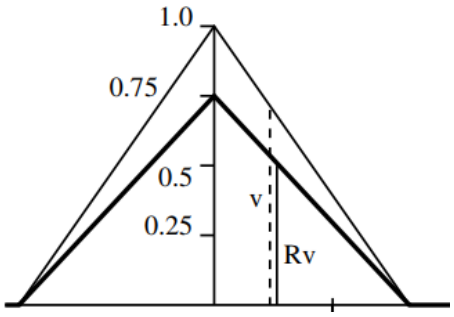

Figure 2: Area under the curve changes while the curve remains fixed.

Due to the uncertainty in the problem, it is hard to integrate a good local heuristic in our genetic algorithm. We implement some of the techniques discussed in the paper Evolutionary Algorithms and Dynamic Optimisation Problems and the results are evaluated and compared with the results of the version of the algorithm not using those techniques.

Dynamic problems cause changes in the environment, thus growing a strong need for diversity to avoid premature convergence. We use a self-adaptive mutation operator called *hyper-mutation* to preserve the diversity in the population. The self-adaptive mutation rate will be based on the average fitness of the population or on the average fitness of the last n best individuals. When the average fitness decreases then a higher mutation rate will be set and if the fitness increases then a lower mutation rate will be selected. The behaviour of such mutation is shown in figure 3

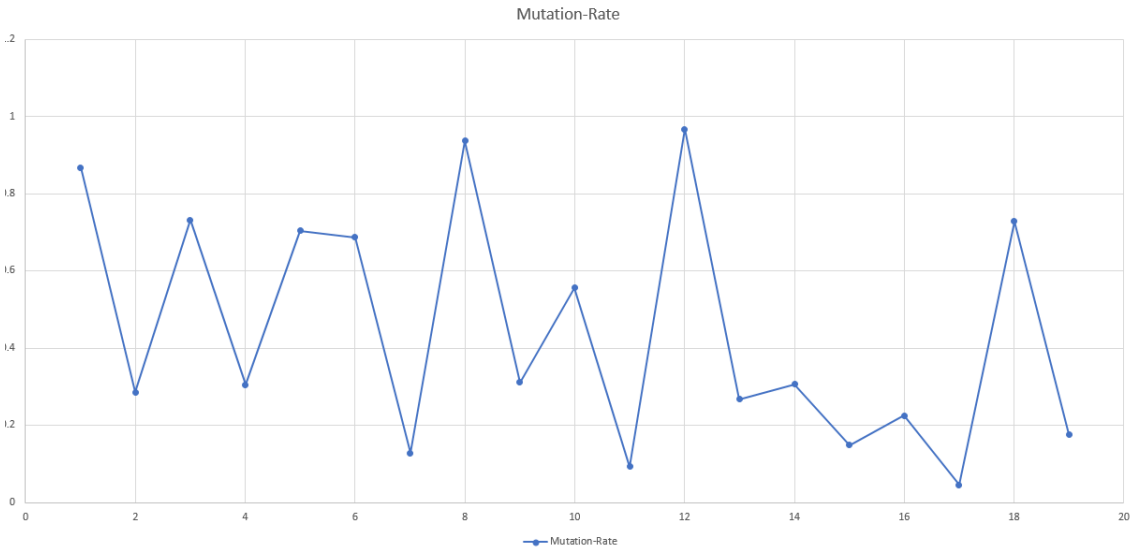

Figure 3: Evolution of the mutation rate according to the environmental changes. In our case, the average fitness of the population.

Setting a lower mutation rate creates less disturbance while on the other hand a higher mutation rate results in more random individuals.

Let us analyse the impact of hyper-mutation. First, on the different part that compose the total score to get better insights then on the overall results between a version of the algorithm which has hyper-mutation and one which do not.

The results in Table 8 are obtained running the algorithm with standard parameters and as an addition hyper-mutation is used. It shows which components of the solutions are mostly effected when we run the algorithm multiple times on the same problem instance. Three most important factors to be noticed are, delay time, penalty time and wait time. One can notice that a decrease in the delay time is associated with an increase in the penalty time. The purpose of using hyper-mutation is to take the new changes into account so that the algorithm tries to move the whole population at the same time in one direction. As we know that our goal lies on the boundary between feasible and infeasible regions and if we only change the mutation rate with respect to the delay time then it would result in more violations.

		Iteration 1	Iteration 2	Iteration 3	Iteration 4
Without hyper-mutation	Distance Time	1649	1941	1881	1639
	Processing Time	404	362	367	363
	Wait time	90	69	74	88
	Penalty Time	0	0	0	56
	Delay Time	1263	1454	1408	1125
With hyper-mutation	Distance Time	1657	1824	1760	1766
	Processing Time	362	353	367	372
	Wait time	78	73	90	85
	Penalty Time	5	0	20	0
	Delay Time	1210	1427	1283	1163

Table 8: The effect of hyper-mutation on individual problem instances.

	Traffic	Traffic+Hyper-mutation
Fixed	(3230, 5.6, 0.52, 39%, 1103)	(3404, 7.29, 0.64, 19%,1173)
Classic	(2545, 32.43, 3.13, 3%, 632.16)	(2624, 36.36, 3.16, 2%, 641)

Table 9: A Comparison when algorithm is executed with and without hyper-mutation.

Table 9 shows the difference when the algorithm is executed without hyper-mutation and when it is done with hyper-mutation. The interpretation of the tuple is the same as usual except that we have added a new element, which is the delay time. As the complexity of the problem at hand increases with the addition of the traffic, it is logical that the quality of the solutions found decreases when compared to the ones of the previous sections. However, it is interesting to note that the classic version manage to obtain much better scores. The phenomenon behind it is probably the same that we began to observe with some temperature profiles comparisons, that is flexibility. Indeed, We can see that the number of constraints violations has exploded for the classic version if we compare them to those of the fixed one. This explosion is made possible by the fact that classic has more flexibility to cheat by adding or removing nodes. As the penalty time in response to this explosion should have grown bigger, it is likely that the algorithm began "optimising" this penalty time by cheating on the constraint weights. This results in a better score for the classic version but with a very low success rate, since this score is mostly obtained by cheating and not by actually solving the problems at hand. A last thing to notice is that hyper-mutation although interesting on paper is not able to combine well with our algorithm and leads to poorer solutions.

11 Conclusion

During this project, we used an existing design for the ITSP and made changes on the way constraints are handled, relying mostly on penalty-based fitness function. We successfully incorporated self-adaptive weights in the algorithm despite potential threats of cheating, corroborating in a somewhat more complex problem the results found in (Eiben et al., 2000).

Traffic functions have dynamic behaviour and are associated with strong uncertainty which implies that it is difficult to find an helpful local heuristic. A second difficulty is that even if we find one then it should be compatible somehow with those we are already using. To adapt to the greater need of diversity induced by a more dynamic problem, we have tried a technique called hyper-mutation. Hyper-mutation can be implemented in different ways and we used it on the basis of average best fitness of the population. This has as consequence that the algorithm will try to optimise the whole population which have not proven very effective in our case.

To conclude, the best versions of this algorithm are able to solve ITSPs with a decent success rate. However, several improvement can still be made, notably by reducing bias in the dataset, optimising the algorithm so that tests are made on a bigger population and with more generations, allowing most versions of the algorithm to converge before testing them.

References

Cavazzuti, M. (2013). *Optimization methods: from theory to design.*

Croes, G. A. (1958).

Eiben, A., Jansen, B., Michalewicz, Z., and Paechter, B. (2000). Solving csps using self-adaptive constraint weights: how to prevent eas from cheating. pages 128–134.

Eiben, A. and Smith, J. (2003). *Introduction to Evolutionary Computing.*

Gremlich, R., Hamfelt, A., de Pereda, H., and Valkovsky, V. (2005). Genetic algorithms with oracle for the traveling salesman problem. pages 27–32.

Jones, T. (1995). *Evolutionary Algorithms Fitness Landscapes and Search.*

Leyman, P., Pham, S. T., and Causmaecker, P. D. (2017). The intermittent traveling salesman problem with different temperature profiles: Greedy or not? *CoRR*, abs/1701.08517.

Otman, A., Jaafar, A., and Tajani, C. (2011). Analyse des performances d'opérateurs de mutation génétique à la résolution du problème de voyageur de commerce.

Pham, S. T. and De Causmaecker, P. (2015). The intermittent traveling salesman problem. *International Transactions in Operational Research.*

University, H. `https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`,.