Genetic algorithm parameter effects analysis

Setup the environment

Import packages

[95]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
from glob import glob
```

Read in the data

[96]:
```
filenames = glob('GA_experiments*.csv')
```

[97]:
```
dfs = [pd.read_csv(f) for f in filenames]
```

Set the `'Run'` column as the index

[98]:
```
dfs[0].set_index('Run', inplace = True)
```

[99]:
```
dfs[1].set_index('Run', inplace = True)
```

[100]:
```
dfs[2].set_index('Run', inplace = True)
```

Check the best parameter for the **Population Size** experiment

[101]:
```
dfs[1]
```

[101]:

Run	GA_popexp_40cities_40pop_500gen	GA_popexp_40cities_100pop_500gen	GA_popexp_100cities_40pop_500gen	GA_popexp_100cities_100pop_500gen
1	5299.702544	5132.366461	12799.95247	10576.57790
2	5436.922478	5064.955562	12822.10277	10010.48964
3	5172.611982	5204.002724	12634.92498	10663.31816
4	5176.667460	5086.245295	11689.79719	10412.23577
5	5462.663692	5068.949469	12581.69032	10526.14193
6	5279.847403	5078.424621	13608.15721	10480.65579
7	5234.914225	4957.362999	13509.69346	10049.52110
8	5266.917313	4932.434984	12674.75449	10138.13114
9	5052.898908	5035.873683	13010.86964	10322.80276
10	5273.514545	4951.065374	13244.65037	10884.06930

[102]:
```
dfs[1].describe()
```

[102]:

	GA_popexp_40cities_40pop_500gen	GA_popexp_40cities_100pop_500gen	GA_popexp_100cities_40pop_500gen	GA_popexp_100cities_100pop_500gen
count	10.000000	10.000000	10.000000	10.000000
mean	5265.666055	5051.168117	12857.659290	10406.394349
std	121.268725	85.363679	546.641365	280.096366
min	5052.898908	4932.434984	11689.797190	10010.489640
25%	5191.229151	4976.990670	12644.882358	10184.299045
50%	5270.215929	5066.952516	12811.027620	10446.445780
75%	5294.738759	5084.290127	13186.205187	10563.968908
max	5462.663692	5204.002724	13608.157210	10884.069300

Check the best parameter for the **Scaling Method** experiment

[103]:
```
col_40_cities = [col for col in dfs[0] if '40cities' in col]
```

[104]:
```
col_100_cities = [col for col in dfs[0] if '100cities' in col]
```

[105]:
```
dfs[0].rename(
    columns = {
        'GA_popexp_40cities_100pop_500gen_rank':'40cities_rank',
        'GA_popexp_40cities_100pop_500gen_prop':'40cities_prop',
        'GA_popexp_40cities_100pop_500gen_top':'40cities_top',
        'GA_popexp_100cities_100pop_500gen_rank':'100cities_rank',
        'GA_popexp_100cities_100pop_500gen_prop':'100cities_prop',
        'GA_popexp_100cities_100pop_500gen_top':'100cities_top'
    }
)
```

[105]:

Run	40cities_rank	40cities_prop	40cities_top	100cities_rank	100cities_prop	100cities_top
1	5132.366461	5589.188767	4845.903118	10576.577895	17261.991261	11705.150689
2	5064.955562	5101.043996	4918.020601	10010.489637	16048.619001	11812.788703
3	5204.002724	5642.984188	4949.962070	10663.318162	15703.550217	12040.996245
4	5086.245295	5460.902516	5046.966826	10412.235772	15318.459021	11854.407850
5	5068.949469	5301.915747	5132.025341	10526.141931	16437.119599	12215.883633
6	5078.424621	5732.552043	5525.629894	10480.655786	16354.729561	11877.607378
7	4957.362999	5143.121445	5152.751345	10049.521097	15785.184765	11343.888688
8	4932.434984	5334.906407	4947.764284	10138.131141	16011.164830	10955.850217
9	5035.873683	5341.074541	5192.936990	10322.802760	15510.752401	12385.012612
10	4951.065374	5396.984577	5132.529711	10884.069304	15315.128699	11366.070284

[106]:
```
dfs[0].rename(
    columns = {
        'GA_popexp_40cities_100pop_500gen_rank':'40cities_rank',
        'GA_popexp_40cities_100pop_500gen_prop':'40cities_prop',
        'GA_popexp_40cities_100pop_500gen_top':'40cities_top',
        'GA_popexp_100cities_100pop_500gen_rank':'100cities_rank',
        'GA_popexp_100cities_100pop_500gen_prop':'100cities_prop',
        'GA_popexp_100cities_100pop_500gen_top':'100cities_top'
    }
).describe()
```

[106]:

	40cities_rank	40cities_prop	40cities_top	100cities_rank	100cities_prop	100cities_top
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	5051.168117	5404.467423	5084.449018	10406.394348	15974.669935	11755.765630
std	85.363679	205.921880	193.915104	280.096368	597.451901	432.149125
min	4932.434984	5101.043996	4845.903118	10010.489637	15315.128699	10955.850217
25%	4976.990670	5310.163412	4948.313731	10184.299046	15558.951855	11450.840385
50%	5066.952516	5369.029559	5089.496084	10446.445779	15898.174797	11833.598277
75%	5084.290127	5557.117204	5147.695937	10563.968904	16278.201921	12000.149028
max	5204.002724	5732.552043	5525.629894	10884.069304	17261.991261	12385.012612

[107]:
```
dfs[0].describe()[col_40_cities]
```

[107]:

	GA_popexp_40cities_100pop_500gen_rank	GA_popexp_40cities_100pop_500gen_prop	GA_popexp_40cities_100pop_500gen_top
count	10.000000	10.000000	10.000000
mean	5051.168117	5404.467423	5084.449018
std	85.363679	205.921880	193.915104
min	4932.434984	5101.043996	4845.903118
25%	4976.990670	5310.163412	4948.313731
50%	5066.952516	5369.029559	5089.496084
75%	5084.290127	5557.117204	5147.695937
max	5204.002724	5732.552043	5525.629894

By comparing the means we notice that during **40 cities** experiment, `rank` outperforms `prop`, but `top` is quite close in value to `rank`

We can perform a **t-test** to determine the mean difference between the two groups.

[108]:
```
dfs[0].describe()[col_100_cities]
```

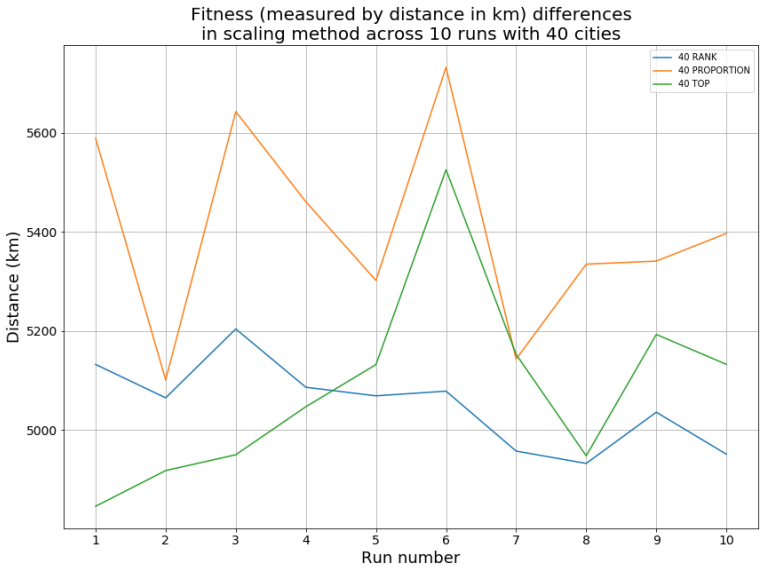
[108]:

	GA_popexp_100cities_100pop_500gen_rank	GA_popexp_100cities_100pop_500gen_prop	GA_popexp_100cities_100pop_500gen_top
count	10.000000	10.000000	10.000000
mean	10406.394348	15974.669935	11755.765630
std	280.096368	597.451901	432.149125
min	10010.489637	15315.128699	10955.850217
25%	10184.299046	15558.951855	11450.840385
50%	10446.445779	15898.174797	11833.598277
75%	10563.968904	16278.201921	12000.149028
max	10884.069304	17261.991261	12385.012612

[109]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.graph_objs as go
```

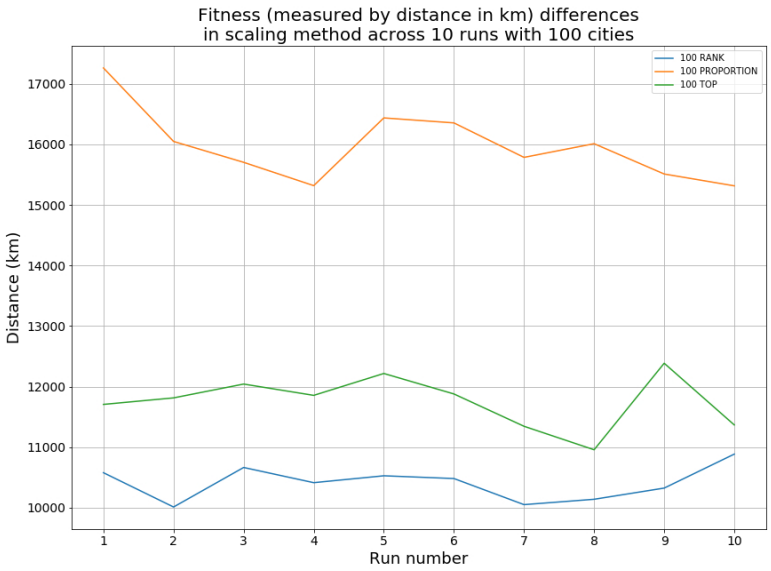
[110]:
```
names = [
    'Rank',
    'Proportional',
    'Top'
]
```

[111]:
```
ax = dfs[0][col_40_cities].rename(
    columns = {
        'GA_popexp_40cities_100pop_500gen_rank':'40 RANK',
        'GA_popexp_40cities_100pop_500gen_prop':'40 PROPORTION',
        'GA_popexp_40cities_100pop_500gen_top':'40 TOP'
    })\
.plot(
    figsize = (14,10),
    grid = True,
    xticks = range(1,11),
    title = 'Fitness (measured by distance in km) differences\nin scaling method across 10 runs with 40 cities',
    fontsize = 14
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Run number', fontsize = 18)
ax.title.set_size(20)

fig = ax.get_figure()
fig.savefig('scaling_effect_40_cities.png')
```

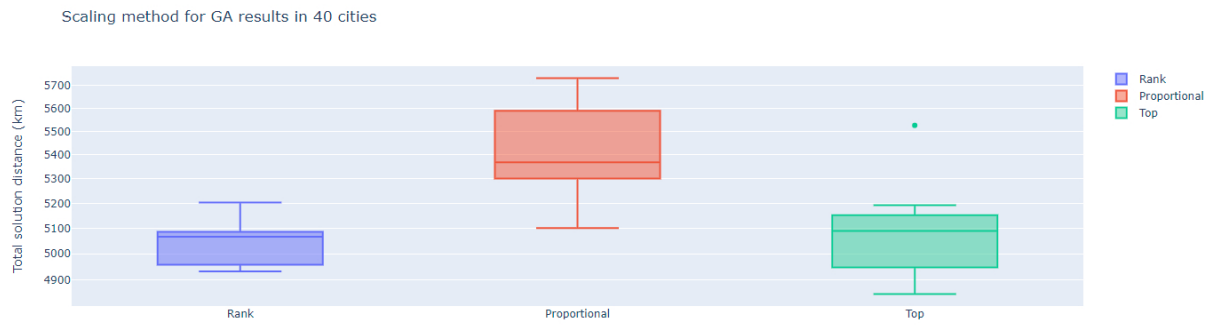


[112]:
```
ax = dfs[0][col_100_cities].rename(
    columns = {
        'GA_popexp_100cities_100pop_500gen_rank':'100 RANK',
        'GA_popexp_100cities_100pop_500gen_prop':'100 PROPORTION',
```

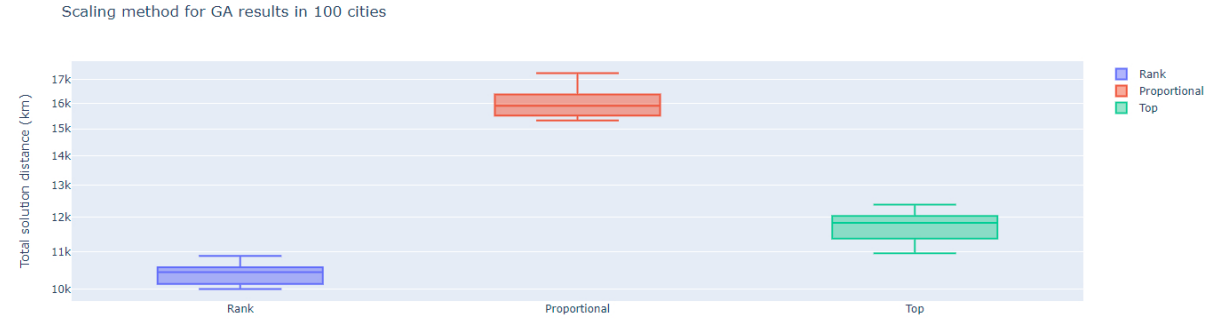
```
    })\
.plot(
    figsize = (14,10),
    grid = True,
    xticks = range(1,11),
    title = 'Fitness (measured by distance in km) differences\nin scaling method across 10 runs with 100 cities',
    fontsize = 14
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Run number', fontsize = 18)
ax.title.set_size(20)

fig = ax.get_figure()
fig.savefig('scaling_effect_100_cities.png')
```



[114]:
```
data = [go.Box(y = dfs[0][col], name = name) for col, name in zip(col_40_cities, names)]

layout = go.Layout(
    title = "Scaling method for GA results in 40 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



[116]:
```
data = [go.Box(y = dfs[0][col], name = name) for col, name in zip(col_100_cities, names)]

layout = go.Layout(
    title = "Scaling method for GA results in 100 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



By comparing the means we notice that during **100 cities** experiment, `rank` outperforms both `prop` and `top`. With a smaller **standard deviation**.

Hence a t-test is probably not necessary to determine that `rank` is the superior parameter.

A lower fitness mean can be achieved with a larger population. Hence a **100 population** genome will yield better results when compared to a **40 population** genome.

For completeness purposes the t-test is as follows:

The t-test null hypothesis states that there is no difference between the two samples.

Null Hypothesis: The true difference in means is equal to 0

Alternative Hypothesis: The true difference in means is not equal to 0

`rank` vs `prop` with **40 cities**

[117]:
```
stats.ttest_ind(dfs[0]['GA_popexp_40cities_100pop_500gen_rank'].values, dfs[0]['GA_popexp_40cities_100pop_500gen_prop'].values, equal_var = False)
```

[117]:
```
Ttest_indResult(statistic=-5.0119286868107915, pvalue=0.0003028512053794878)
```

The small p-value (0.0003028512053794878 < 0.05) indicates that we **reject the null hypothesis**, and infer that the mean difference between `rank` and `prop` are **significantly different**.

`rank` vs `top` with **40 cities**

[118]:
```
stats.ttest_ind(dfs[0]['GA_popexp_40cities_100pop_500gen_rank'].values, dfs[0]['GA_popexp_40cities_100pop_500gen_top'].values, equal_var = False)
```

[118]:
```
Ttest_indResult(statistic=-0.4967297229829585, pvalue=0.6280954349564121)
```

The large p-value (0.6280954349564121 > 0.05) indicates that we **reject the alternative hypothesis**, and infer that the mean difference between `rank` and `top` are **not significantly different**.

`rank` vs `prop` with **100 cities**

[119]:
```
stats.ttest_ind(dfs[0]['GA_popexp_100cities_100pop_500gen_rank'].values, dfs[0]['GA_popexp_100cities_100pop_500gen_prop'].values, equal_var = False)
```

[119]:
```
Ttest_indResult(statistic=-26.68548731693419, pvalue=1.3842981080213172e-12)
```

The small p-value (1.3842981080213172e-12 < 0.05) indicates that we **reject the null hypothesis**, and infer that the mean difference between `rank` and `prop` are **significantly different**.

`rank` vs `top` with **100 cities**

[120]:
```
stats.ttest_ind(dfs[0]['GA_popexp_100cities_100pop_500gen_rank'].values, dfs[0]['GA_popexp_100cities_100pop_500gen_top'].values, equal_var = False)
```

[120]:
```
Ttest_indResult(statistic=-8.285888819719862, pvalue=4.5585298130038166e-07)
```

The small p-value (4.5585298130038166e-07 < 0.05) indicates that we **reject the null hypothesis**, and infer that the mean difference between `rank` and `top` are **significantly different**.

Scaling method conclusion:

For the remainder of the experiments `rank` will be used as the optimal scaling method.

Check the best parameter for the **Selection Method** experiment

[121]:
```
col_40_cities = [col for col in dfs[2] if '40cities' in col]
```

[122]:
```
col_100_cities = [col for col in dfs[2] if '100cities' in col]
```

[123]:
```
dfs[2].describe()[col_40_cities]
```

[123]:

	GA_popexp_40cities_100pop_500gen_stoUni	GA_popexp_40cities_100pop_500gen_roulette	GA_popexp_40cities_100pop_500gen_tourn
count	10.000000	10.000000	10.000000
mean	5051.168117	5096.996414	4985.931708
std	85.363679	229.717385	85.470109
min	4932.434984	4797.130361	4797.130361
25%	4976.990670	4937.603975	4952.324126
50%	5066.952516	5035.159306	4988.558137
75%	5084.290127	5312.700752	5030.601943
max	5204.002724	5435.425570	5118.171498

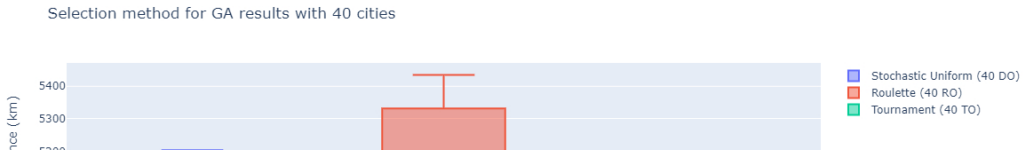
[124]:
```
dfs[2].describe()[col_100_cities]
```

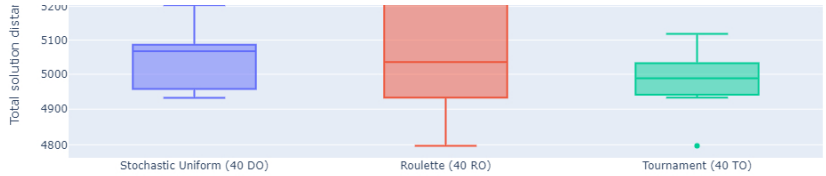
[124]:

	GA_popexp_100cities_100pop_500gen_stoUni	GA_popexp_100cities_100pop_500gen_roulette	GA_popexp_100cities_100pop_500gen_tourn
count	10.000000	10.000000	10.000000
mean	10406.394348	10362.026964	10189.489049
std	280.096368	384.787103	171.094279
min	10010.489637	9908.018409	9865.385107
25%	10184.299046	10009.224704	10084.898976
50%	10446.445779	10398.491796	10216.199671
75%	10563.968904	10591.662808	10304.574538
max	10884.069304	11074.544838	10437.128679

[125]:
```
names_40 = [
    'Stochastic Uniform\n(40 DO)',
    'Roulette\n(40 RO)',
    'Tournament\n(40 TO)'
]
```

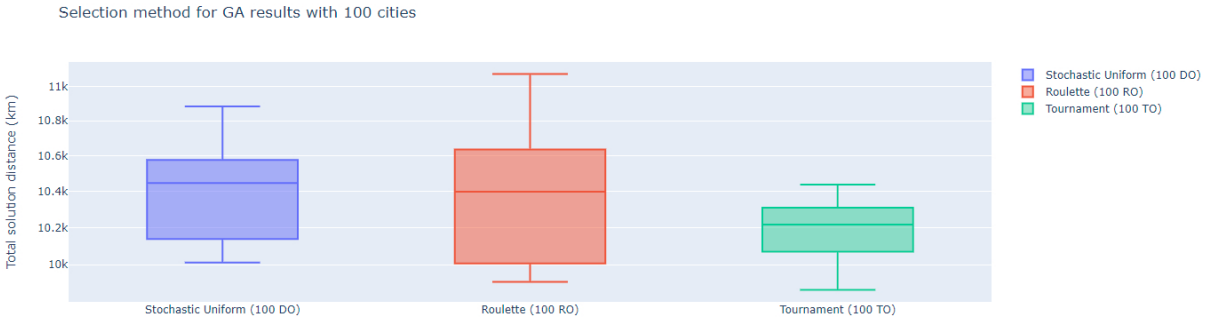
[127]:
```
data = [go.Box(y = dfs[2][col], name = name) for col, name in zip(col_40_cities, names_40)]

layout = go.Layout(
    title = "Selection method for GA results with 40 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```





[128]:
```
names_100 = [
    'Stochastic Uniform\n(100 DO)',
    'Roulette\n(100 RO)',
    'Tournament\n(100 TO)'
]
```

[132]:
```
data = [go.Box(y = dfs[2][col], name = name) for col, name in zip(col_100_cities, names_100)]

layout = go.Layout(
    title = "Selection method for GA results with 100 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



From the above plots we can observe that although roulette can display good results; it is **volatile** with **high variance** and therefore **unreliable**.

We will eliminate `roulette` as a possible parameter candidate, and perform a t-test between `stochastic uniform` and `tournament`.

t-test to determine the best selection method

Null Hypothesis: The true difference in means is equal to 0

Alternative Hypothesis: The true difference in means is not equal to 0

t-test on GA-results for 40 cities

[133]:
```
stats.ttest_ind(dfs[2]['GA_popexp_40cities_100pop_500gen_stoUni'].values, dfs[2]['GA_popexp_40cities_100pop_500gen_tourn'].values, equal_var = False)
```

[133]:
```
Ttest_indResult(statistic=1.7077771220416476, pvalue=0.10486666768945964)
```

The large p-value (0.10486666768945964 > 0.05) indicates that we **reject the alternative hypothesis**, and infer that the mean difference between `stochastic uniform` and `tournament` are **not significantly different**.

t-test on GA-results for 100 cities

[134]:
```
stats.ttest_ind(dfs[2]['GA_popexp_100cities_100pop_500gen_stoUni'].values, dfs[2]['GA_popexp_100cities_100pop_500gen_tourn'].values, equal_var = False)
```

[134]:
```
Ttest_indResult(statistic=2.0898129776508254, pvalue=0.054207215160572836)
```

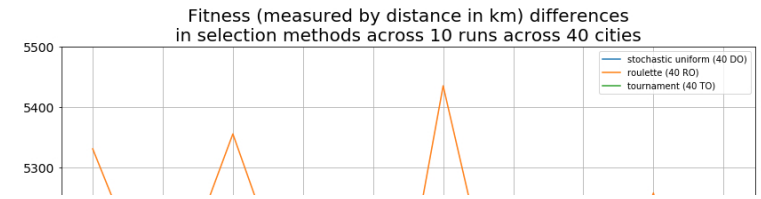
[135]:
```
stats.ttest_ind(dfs[2]['GA_popexp_100cities_100pop_500gen_roulette'].values, dfs[2]['GA_popexp_100cities_100pop_500gen_tourn'].values, equal_var = False)
```

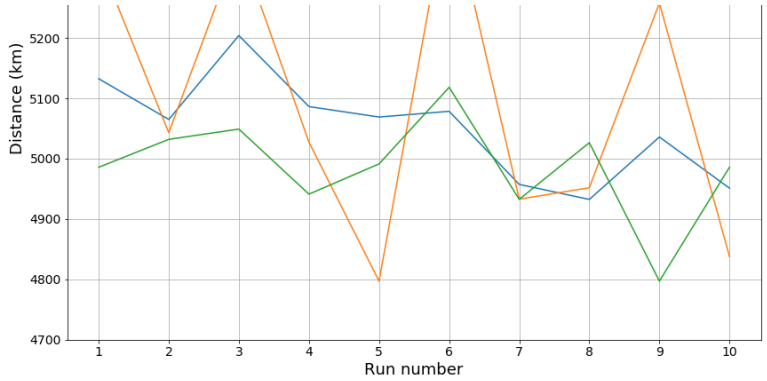
[135]:
```
Ttest_indResult(statistic=1.2956511711762007, pvalue=0.2186493701086492 4)
```

The p-value (0.054207215160572836 < 0.05) indicates that we **reject the alternative hypothesis**, and infer that the mean difference between `stochastic uniform` and `tournament` are **not significantly different** at the 95% confidence level.

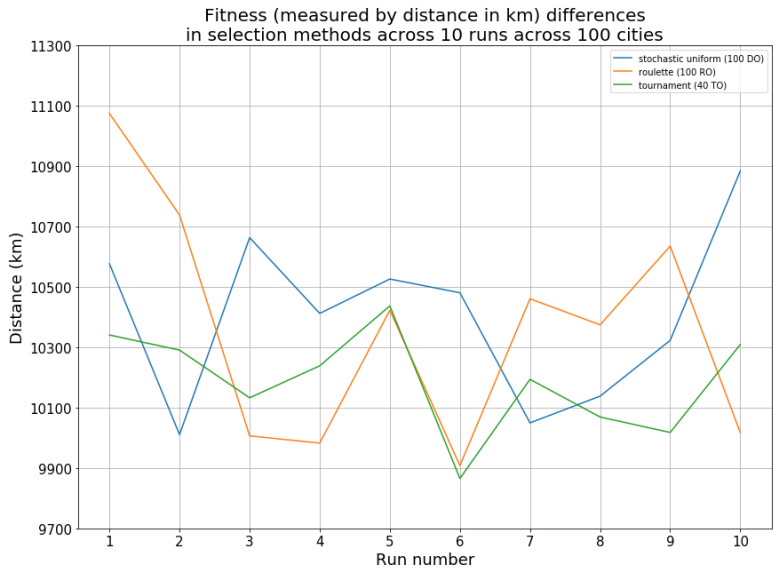
However at the 90% confidence level where the p-value (0.054207215160572836 < 0.1) indicates that we **reject the null hypothesis** and infer that the mean difference between `stochastic uniform` and `tournament` are **significantly different**.

[136]:
```
ax = dfs[2][col_40_cities].rename(columns = {
    'GA_popexp_40cities_100pop_500gen_stoUni':'stochastic uniform (40 DO)',
    'GA_popexp_40cities_100pop_500gen_roulette':'roulette (40 RO)',
    'GA_popexp_40cities_100pop_500gen_tourn':'tournament (40 TO)'
}).plot(
    figsize = (14,10),
    grid = True,
    xticks = range(1,11),
    yticks = range(4700, 5600, 100),
    title = 'Fitness (measured by distance in km) differences\nin selection methods across 10 runs across 40 cities',
    fontsize = 14
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Run number', fontsize = 18)
ax.title.set_size(20)
fig = ax.get_figure()
fig.savefig('selection_effect_40_cities.png')
```





[137]:
```
ax = dfs[2][col_100_cities].rename(columns = {
    'GA_popexp_100cities_100pop_500gen_stoUni':'stochastic uniform (100 DO)',
    'GA_popexp_100cities_100pop_500gen_roulette':'roulette (100 RO)',
    'GA_popexp_100cities_100pop_500gen_tourn':'tournament (40 TO)'
}).plot(
    figsize = (14,10),
    grid = True,
    xticks = range(1,11),
    yticks = range(9700, 11500, 200),
    title = 'Fitness (measured by distance in km) differences\nin selection methods across 10 runs across 100 cities',
    fontsize = 15
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Run number', fontsize = 18)
ax.title.set_size(20)
fig = ax.get_figure()
fig.savefig('selection_effect_100_cities.png')
```



Check the best stopping criteria - finding optimal number of generations

[138]:
```
df_gen = pd.read_csv('generations_exp.csv')
```

[139]:
```
col_40_cities = [col for col in df_gen if '40cities' in col]
```

[140]:
```
col_100_cities = [col for col in df_gen if '100cities' in col]
```

[141]:
```
df_gen[col_40_cities]
```

[141]:

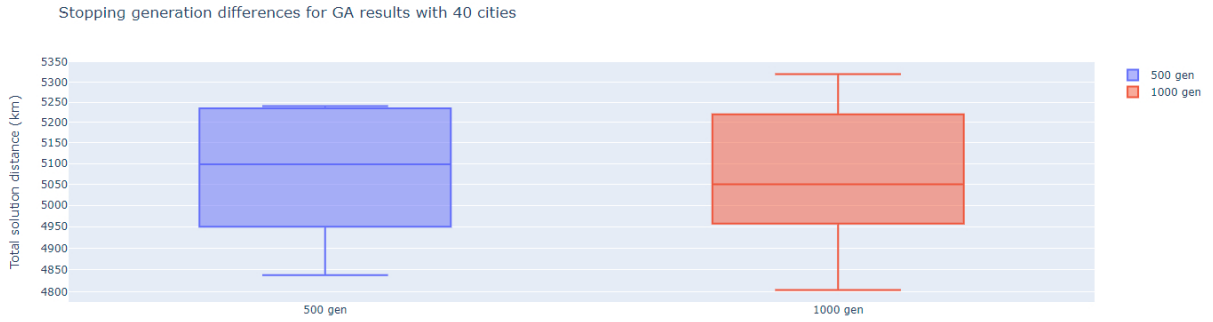
	GA_popexp_40cities_100pop_500gen	GA_popexp_40cities_100pop_1000gen
0	5240.169578	4957.362999
1	5089.331348	4845.903118
2	4902.413218	5041.221858
3	5238.614021	5320.003914
4	5234.532132	5259.943469
5	5091.847403	5110.631026
6	5228.764748	4998.335836
7	5104.878128	5058.686840
8	4949.962070	4804.147049
9	4837.870098	5219.854306

[142]:
```
df_gen[col_40_cities].describe()
```

[142]:

	GA_popexp_40cities_100pop_500gen	GA_popexp_40cities_100pop_1000gen
count	10.000000	10.000000
mean	5091.838274	5061.609041
std	150.311735	170.508980
min	4837.870098	4804.147049
25%	4984.804390	4967.606208
50%	5098.362766	5049.954349
75%	5233.090286	5192.548486
max	5240.169578	5320.003914

[143]:
```
names_40 = [
    '500 gen',
    '1000 gen'
]

data = [go.Box(y = df_gen[col], name = name) for col, name in zip(col_40_cities, names_40)]

layout = go.Layout(
    title = "Stopping generation differences for GA results with 40 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



[144]:
```
df_gen[col_100_cities]
```

[144]:

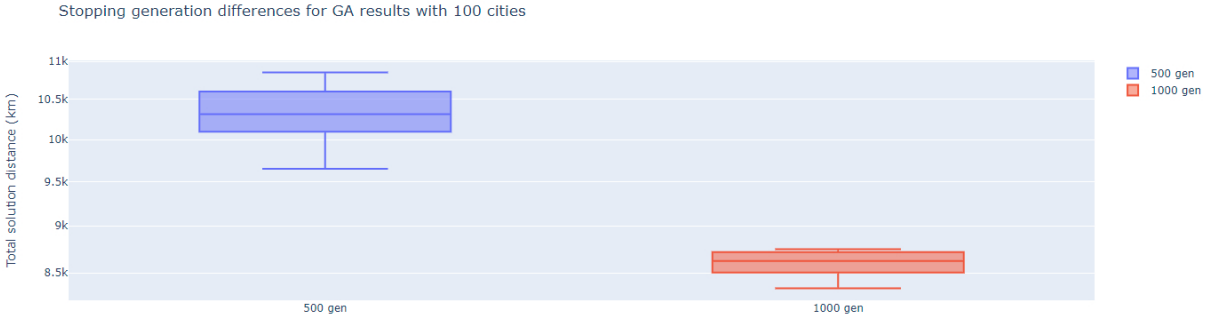
	GA_popexp_100cities_100pop_500gen	GA_popexp_100cities_100pop_1000gen
0	10852.573840	8718.094544
1	9848.470005	8590.160549
2	10127.271137	8503.289303
3	10603.784273	8491.782060
4	10096.084652	8747.273855
5	10500.031874	8503.508510
6	10571.939413	8341.006026
7	9650.167759	8734.731690
8	10620.953767	8666.636609
9	10103.562598	8655.741902

[145]:
```
df_gen[col_100_cities].describe()
```

[145]:

	GA_popexp_100cities_100pop_500gen	GA_popexp_100cities_100pop_1000gen
count	10.000000	10.000000
mean	10297.483932	8595.222505
std	387.398761	132.772685
min	9650.167759	8341.006026
25%	10097.954139	8503.344105
50%	10313.651505	8622.951226
75%	10595.823058	8705.230060
max	10852.573840	8747.273855

[146]:
```
names_100 = [
    '500 gen',
    '1000 gen'
]

data = [go.Box(y = df_gen[col], name = name) for col, name in zip(col_100_cities, names_100)]

layout = go.Layout(
    title = "Stopping generation differences for GA results with 100 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



t-test to verify statistical significance in mean difference

[147]:
```
stats.ttest_ind(df_gen['GA_popexp_100cities_100pop_500gen'], df_gen['GA_popexp_100cities_100pop_1000gen'], equal_var = False)
```

[147]:
```
Ttest_indResult(statistic=13.14472358244969, pvalue=4.172165476140857e-08)
```

Small p-value (4.1721654761408576e-08 < 0.05) therefore the **500 gen group** is significantly different to the **1000 gen group**.

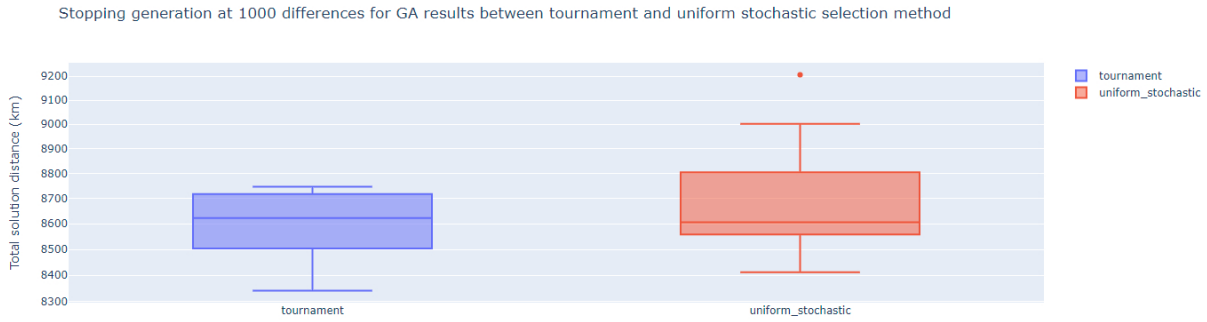
Check that `tournament` parameter GA running at 1000 generations is actually better than its `uniform stochastic` counterpart

[148]:
```
pd.concat([df_gen['GA_popexp_100cities_100pop_1000gen'], df_gen['1000gen_uni']], axis = 1).rename(
    columns = {
        'GA_popexp_100cities_100pop_1000gen': 'tournament',
        '1000gen_uni': 'uniform stochastic'
    }
).describe()
```

[148]:

	tournament	uniform stochastic
count	10.000000	10.000000
mean	8595.222505	8699.785612
std	132.772685	247.099046
min	8341.006026	8411.632206
25%	8503.344105	8565.968689
50%	8622.951226	8605.684282
75%	8705.230060	8791.319145
max	8747.273855	9206.178594

[150]:
```
data = [go.Box(y = df_gen['GA_popexp_100cities_100pop_1000gen'], name = 'tournament'),go.Box(y = df_gen['1000gen_uni'], name = 'uniform_stochastic')]

layout = go.Layout(
    title = "Stopping generation at 1000 differences for GA results\nbetween tournament and uniform stochastic selection method",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)

plotly.offline.iplot({'data':data, 'layout':layout})
```



[151]:
```
stats.ttest_ind(df_gen['GA_popexp_100cities_100pop_1000gen'].values, df_gen['1000gen_uni'].values, equal_var = False)
```

[151]:
```
Ttest_indResult(statistic=-1.1787679017620234, pvalue=0.2584203588222461)
```

`tournament` appears to be better when represented graphically; however, the difference is statistically insignificant when compared to `uniform_stochastic`.

Summary of indicators for the 200-city problem

Indicator	selection_optimized	selection_default_1000gen	selection_default
Stopping criteria	1000 Generations	1000 Generations	500 Generations
Selection method	`Rank`	`Rank`	`Rank`
Population size	100 chromosomes	40 chromosomes	40 chromosomes
Fitness scaling	`Tournament`	`Stochastic Uniform`	`Stochastic Uniform`

[152]:
```
df_op = pd.read_csv('GA_200_default vs GA_200_optimized.csv')
```

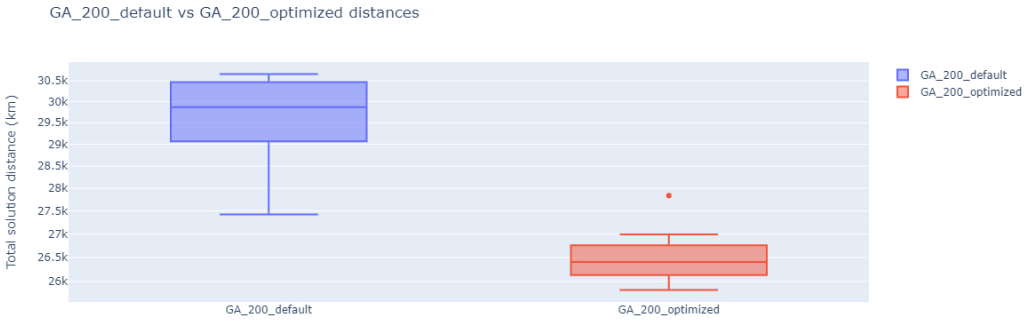
[153]:
```
df_op
```

[153]:

	Run	GA_200_default	GA_200_optimized	GA_200_optimized_1000gen	GA_200_optimized_expanded_2000gen	GA_200_optimized_expanded_3000gen	GA_200_optimized_expanded_4000gen	GA_200_optimized_expanded_5000gen	GA
0	1	30664.807641	26704.006791	19103.233993	14597.367252	12432.502034	11789.050054	11529.536059	
1	2	30658.623784	27837.441862	19636.004637	14899.368024	12440.907760	11723.386621	12028.438158	
2	3	28912.221050	26988.891539	19062.108640	14192.662771	13182.505610	11452.013924	10648.839572	
3	4	29931.756656	26238.567269	18469.010077	14246.777240	12227.990911	12107.682717	11747.216533	
4	5	29808.582204	26434.001150	19980.592906	14118.969540	12619.946150	11833.108382	11511.068578	
5	6	27422.972861	26129.205001	19698.354528	13790.362068	12025.491061	11494.586897	11075.137507	
6	7	30468.643375	26381.522599	19045.127654	13979.204818	12440.950021	12192.819071	11229.018508	
7	8	29755.483682	25823.813264	19098.760272	13894.503036	12852.614708	12021.410910	11356.253483	
8	9	30444.516159	26091.802455	18990.848543	13731.121290	12632.965402	12033.207787	11214.300122	
9	10	29065.291391	26758.861630	18616.271155	13570.449070	12876.791380	11833.557151	11639.878577	

GA_200_default vs GA_200_optimized

[154]:
```
data = [go.Box(y = df_op['GA_200_default'], name = 'GA_200_default'), go.Box(y = df_op['GA_200_optimized'], name = 'GA_200_optimized')]

layout = go.Layout(
    title = "GA_200_default vs GA_200_optimized distances",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```

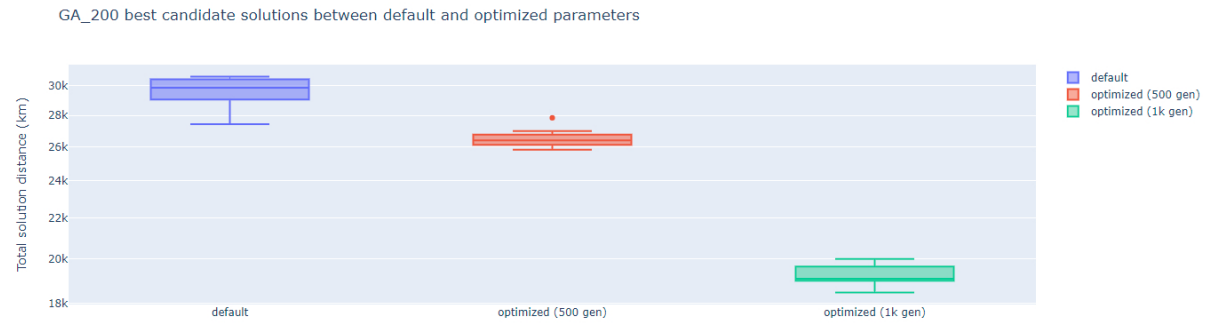


GA_200_default vs GA_200_optimized with 1000 generations

[155]:
```
stats.ttest_ind(df_op['GA_200_default'].values, df_op['GA_200_optimized'].values, equal_var = False)
```

[155]:
```
Ttest_indResult(statistic=8.608203347057026, pvalue=5.122304453348066e-07)
```

[156]:
```
data = [
    go.Box(y = df_op['GA_200_default'], name = 'default'),
    go.Box(y = df_op['GA_200_optimized'], name = 'optimized (500 gen)'),
    go.Box(y = df_op['GA_200_optimized_1000gen'], name = 'optimized (1k gen)')

]

layout = go.Layout(
    title = "GA_200 best candidate solutions between default and optimized parameters",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



[157]:
```
pd.concat([
    df_op['GA_200_default'],
    df_op['GA_200_optimized'],
    df_op['GA_200_optimized_1000gen']
], axis = 1)\
.describe()\
.rename(columns = {
    'GA_200_default':'default',
    'GA_200_optimized': 'optimized (500gen)',
    'GA_200_optimized_1000gen': 'optimized (1000gen)'
})
```

[157]:

	default	optimized (500gen)	optimized (1000gen)
count	10.000000	10.000000	10.000000
mean	29713.289880	26538.811356	19170.031241
std	1014.891732	574.399499	473.784278
min	27422.972861	25823.813264	18469.010077
25%	29237.839464	26156.545568	19004.418321
50%	29870.169430	26407.761874	19080.434456
75%	30462.611571	26745.147920	19502.811976
max	30664.807641	27837.441862	19980.592906

[158]:
```
pd.concat([
    df_op['GA_200_default'],
    df_op['GA_200_optimized'],
    df_op['GA_200_optimized_1000gen']
], axis = 1)\
.rename(columns = {
    'GA_200_default':'default',
    'GA_200_optimized': 'optimized (500gen)',
    'GA_200_optimized_1000gen': 'optimized (1000gen)'
})
```

[158]:

	default	optimized (500gen)	optimized (1000gen)
0	30664.807641	26704.006791	19103.233993
1	30658.623784	27837.441862	19636.004637
2	28912.221050	26988.891539	19062.108640
3	29931.756656	26238.567269	18469.010077
4	29808.582204	26434.001150	19980.592906
5	27422.972861	26129.205001	19698.354528
6	30468.643375	26381.522599	19045.127654
7	29755.483682	25823.813264	19098.760272
8	30444.516159	26091.802455	18990.848543
9	29065.291391	26758.861630	18616.271155

Extending the experiment - running the GA for longer to find the **global optima**

[159]:
```
gen_cols = [col for col in df_op if ('0gen' in col)&('pop200' not in col)]
```

[160]:
```
gen_cols
```

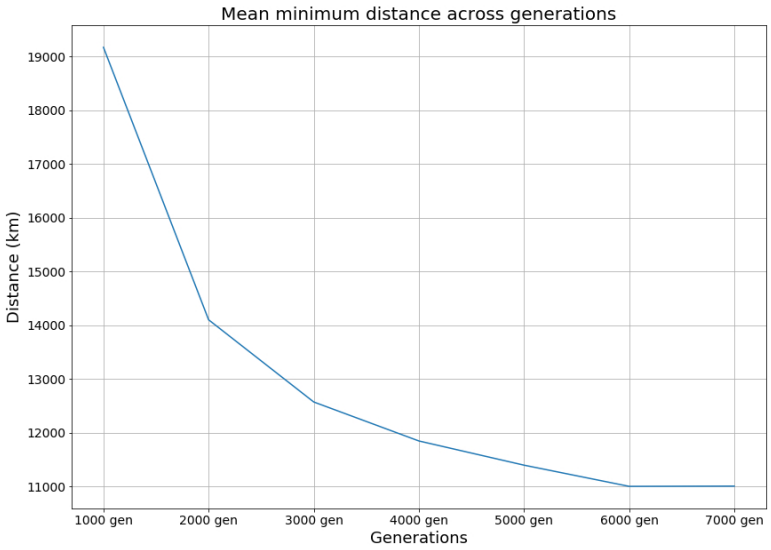
[160]:
```
['GA_200_optimized_1000gen',
 'GA_200_optimized_expanded_2000gen',
 'GA_200_optimized_expanded_3000gen',
 'GA_200_optimized_expanded_4000gen',
 'GA_200_optimized_expanded_5000gen',
 'GA_200_optimized_expanded_6000gen',
 'GA_200_optimized_expanded_7000gen']
```

[161]:
```
df_op[gen_cols].describe().rename(columns = {
 'GA_200_optimized_1000gen':'1000 gen',
 'GA_200_optimized_expanded_2000gen':'2000 gen',
 'GA_200_optimized_expanded_3000gen':'3000 gen',
 'GA_200_optimized_expanded_4000gen':'4000 gen',
 'GA_200_optimized_expanded_5000gen':'5000 gen',
 'GA_200_optimized_expanded_6000gen':'6000 gen',
 'GA_200_optimized_expanded_7000gen':'7000 gen',
}
)
```

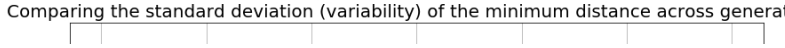
[161]:

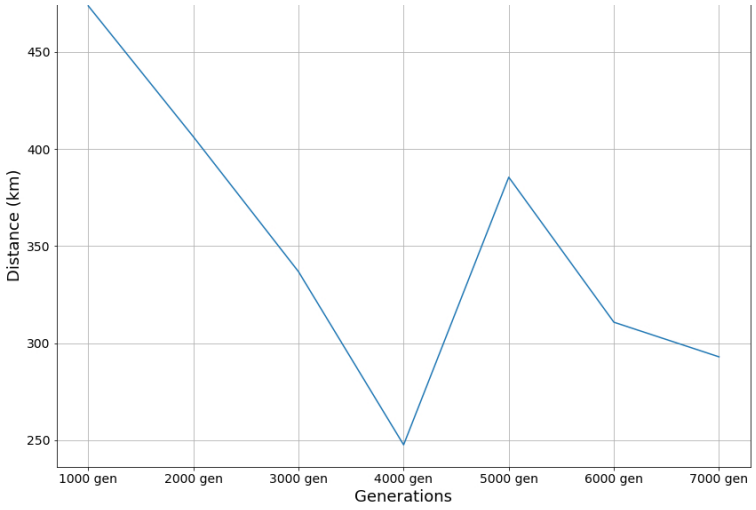
	1000 gen	2000 gen	3000 gen	4000 gen	5000 gen	6000 gen	7000 gen
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	19170.031241	14102.078511	12573.266504	11848.082351	11397.968710	11003.738128	11008.057807
std	473.784278	406.338915	336.846866	247.621630	385.504090	310.791319	292.949939
min	18469.010077	13570.449070	12025.491061	11452.013924	10648.839572	10466.290910	10636.789921
25%	19004.418321	13816.397310	12434.603466	11739.802479	11217.979719	10811.172840	10777.276315
50%	19080.434456	14049.087179	12530.448086	11833.332766	11433.661030	11027.155201	11006.559082
75%	19502.811976	14233.248623	12797.702381	12030.258568	11612.292948	11170.127274	11154.532445
max	19980.592906	14899.368024	13182.505610	12192.819071	12028.438158	11536.679147	11604.367577

[162]:
```
ax = df_op[gen_cols].describe().rename(columns = {
 'GA_200_optimized_1000gen':'1000 gen',
 'GA_200_optimized_expanded_2000gen':'2000 gen',
 'GA_200_optimized_expanded_3000gen':'3000 gen',
 'GA_200_optimized_expanded_4000gen':'4000 gen',
 'GA_200_optimized_expanded_5000gen':'5000 gen',
 'GA_200_optimized_expanded_6000gen':'6000 gen',
 'GA_200_optimized_expanded_7000gen':'7000 gen',
}
).transpose()['mean'].plot(
    figsize = (14,10),
    grid = True,
    #yticks = range(10000, 20000, 5000),
    title = 'Mean minimum distance across generations',
    fontsize = 14,
    #kind = 'bar'
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Generations', fontsize = 18)
ax.title.set_size(20)

fig = ax.get_figure()
fig.savefig('mean_minimum_distance.png')
```

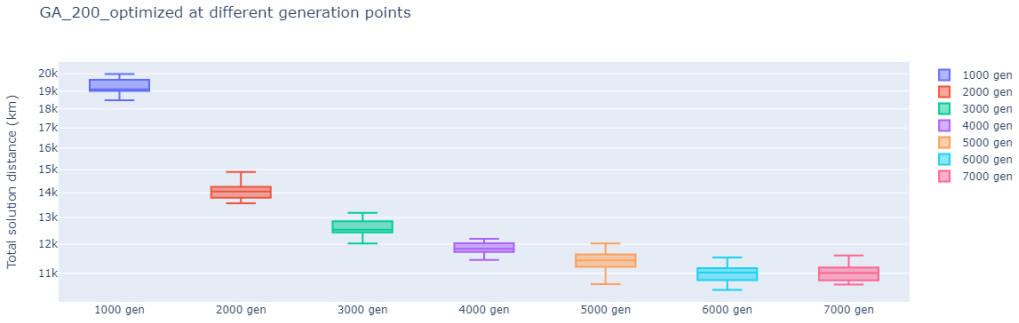


[163]:
```
ax = df_op[gen_cols].describe().rename(columns = {
 'GA_200_optimized_1000gen':'1000 gen',
 'GA_200_optimized_expanded_2000gen':'2000 gen',
 'GA_200_optimized_expanded_3000gen':'3000 gen',
 'GA_200_optimized_expanded_4000gen':'4000 gen',
 'GA_200_optimized_expanded_5000gen':'5000 gen',
 'GA_200_optimized_expanded_6000gen':'6000 gen',
 'GA_200_optimized_expanded_7000gen':'7000 gen',
}
).transpose()['std'].plot(
    figsize = (14,10),
    grid = True,
    #yticks = range(10000, 20000, 5000),
    title = 'Comparing the standard deviation (variability) of the minimum distance across generations',
    fontsize = 14,
    #kind = 'bar'
)

ax.set_ylabel('Distance (km)', fontsize = 18)
ax.set_xlabel('Generations', fontsize = 18)
ax.title.set_size(20)
fig = ax.get_figure()
fig.savefig('std_minimum_distance.png')
```





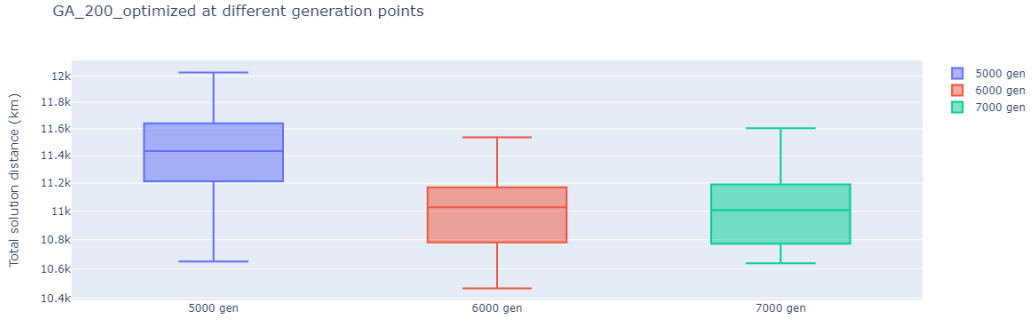
[164]:
```
gens = [
    '1000 gen',
    '2000 gen',
    '3000 gen',
    '4000 gen',
    '5000 gen',
    '6000 gen',
    '7000 gen'
]

data = [go.Box(y = df_op[col], name = gen) for col, gen in zip(gen_cols, gens)]


layout = go.Layout(
    title = "GA_200_optimized at different generation points",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



[63]:
```
gen_cols[len(gen_cols)-3:]
```

[63]:
```
['GA_200_optimized_expanded_5000gen',
 'GA_200_optimized_expanded_6000gen',
 'GA_200_optimized_expanded_7000gen']
```

[165]:
```
gens = [
    '5000 gen',
    '6000 gen',
    '7000 gen'
]

data = [go.Box(y = df_op[col], name = gen) for col, gen in zip(gen_cols[len(gen_cols)-3:], gens)]


layout = go.Layout(
    title = "GA_200_optimized at different generation points",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



Investigate if the mean fitness solution found at `5000 generation` is significantly different to those at `6000 generations`

[166]:
```
stats.ttest_ind(df_op['GA_200_optimized_expanded_5000gen'], df_op['GA_200_optimized_expanded_6000gen'], equal_var = False)
```

[166]:
```
Ttest_indResult(statistic=2.5175955535972325, pvalue=0.021987044479399687)
```

The small p-value (0.021987044479399687 < 0.05) indicated by the t-test, means we can infer that the two groups are significantly different.

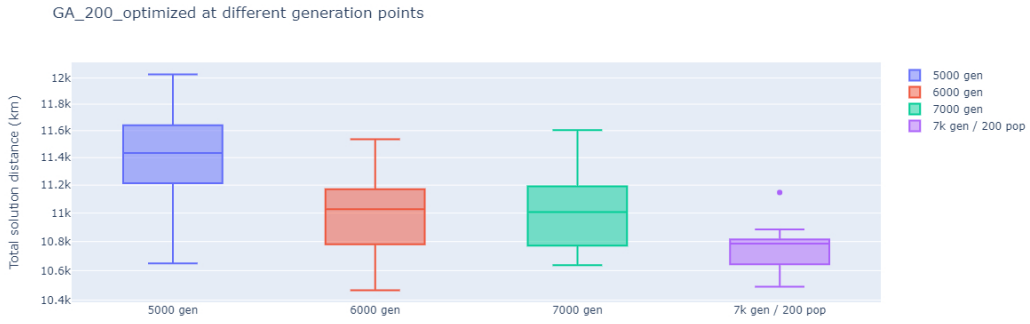
Investigate if the mean fitness solution found at `6000 generation` is significantly different to those at `7000 generations`

[167]:
```
stats.ttest_ind(df_op['GA_200_optimized_expanded_6000gen'], df_op['GA_200_optimized_expanded_7000gen'], equal_var = False)
```

[167]:
```
Ttest_indResult(statistic=-0.031983511237957084, pvalue=0.9748383996477288)
```

The large p-value (0.9748383996477288 < 0.05) indicated by the t-test, means we can infer that the two groups are **not significantly different**

[168]:
```
gens = [
    '5000 gen',
    '6000 gen',
    '7000 gen'
]

data = [go.Box(y = df_op[col], name = gen) for col, gen in zip(gen_cols[len(gen_cols)-3:], gens)]
data.append(go.Box(y = df_op['GA_200_optimized_expanded_7000gen_pop200'], name = '7k gen / 200 pop'))

layout = go.Layout(
    title = "GA_200_optimized at different generation points",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



[169]:
```
pd.concat([
    df_op[gen_cols[len(gen_cols)-3:]].describe().rename(columns = {
 'GA_200_optimized_expanded_5000gen':'5000gen',
 'GA_200_optimized_expanded_6000gen':'6000gen',
 'GA_200_optimized_expanded_7000gen':'7000gen',
 'GA_200_optimized_expanded_7000gen_pop200':'7k gen 200 pop'
}
), df_op['GA_200_optimized_expanded_7000gen_pop200'].describe()], axis = 1).rename(columns = {
    'GA_200_optimized_expanded_7000gen_pop200':'7000gen_200pop'
})
```

[169]:

	5000gen	6000gen	7000gen	7000gen_200pop
count	10.000000	10.000000	10.000000	10.000000
mean	11397.968710	11003.738128	11008.057807	10763.517745
std	385.504090	310.791319	292.949939	178.091514
min	10648.839572	10466.290910	10636.789921	10489.853137
25%	11217.979719	10811.172840	10777.276315	10647.514328
50%	11433.661030	11027.155201	11006.559082	10785.995466
75%	11612.292948	11170.127274	11154.532445	10810.618719
max	12028.438158	11536.679147	11604.367577	11147.048973

[170]:
```
stats.ttest_ind(df_op['GA_200_optimized_expanded_6000gen'], df_op['GA_200_optimized_expanded_7000gen_pop200'], equal_var = False)
```

[170]:
```
Ttest_indResult(statistic=2.120719709795083, pvalue=0.05184333526812824)
```

[171]:
```
stats.ttest_ind(df_op['GA_200_optimized_expanded_7000gen'], df_op['GA_200_optimized_expanded_7000gen_pop200'], equal_var = False)
```

[171]:
```
Ttest_indResult(statistic=2.255610442295009, pvalue=0.03962090976613276)
```

Scaling to a larger city map

[179]:
```
df_large = pd.read_csv('GA_300.csv')
df_large.set_index('Run', inplace = True, drop = True)
```

[181]:
```
df_large
```

[181]:

Run	GA_300_default	GA_300_default_1000gen	GA_300_optimized_500gen	GA_300_optimized_1000gen
1	57712.466126	42166.633376	47019.579731	33476.114124
2	59502.182405	41878.619836	46952.196178	33126.810467
3	59663.987309	45129.621885	49860.257131	34925.855916
4	57433.034288	44132.889109	48198.898040	33539.818652
5	60901.771805	42053.839477	47347.034923	33544.004699
6	57367.792878	40353.944169	46632.148360	32927.562807
7	56972.361396	45135.737041	47916.933508	34125.095642
8	58389.314258	43745.880091	48674.729572	34796.647306
9	57866.301115	41949.059228	47509.957622	34460.066860
10	60389.313815	43528.102382	47743.934704	34147.993666

[182]:
```
df_large_sub = df_large[['GA_300_default','GA_300_optimized_500gen','GA_300_optimized_1000gen']]
df_large_sub
```

[182]:

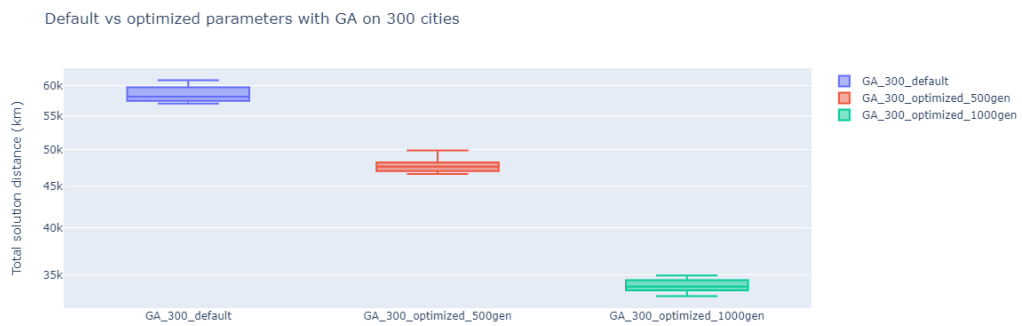
Run	GA_300_default	GA_300_optimized_500gen	GA_300_optimized_1000gen
1	57712.466126	47019.579731	33476.114124
2	59502.182405	46952.196178	33126.810467
3	59663.987309	49860.257131	34925.855916
4	57433.034288	48198.898040	33539.818652
5	60901.771805	47347.034923	33544.004699
6	57367.792878	46632.148360	32927.562807
7	56972.361396	47916.933508	34125.095642
8	58389.314258	48674.729572	34796.647306
9	57866.301115	47509.957622	34460.066860
10	60389.313815	47743.934704	34147.993666

[188]:
```
df_large_sub.rename(
    columns = {
        'GA_300_optimized_500gen':'GA_300_optimized (500 gen)',
        'GA_300_optimized_1000gen':'GA_300_optimized (1000 gen)'
    }
).describe()
```

[188]:

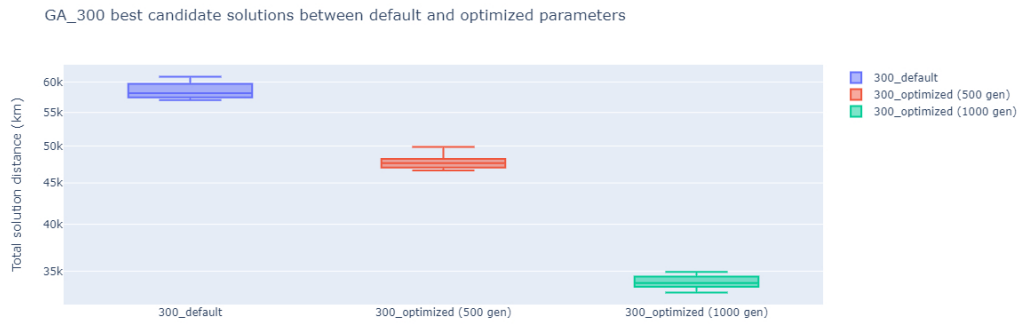
	GA_300_default	GA_300_optimized (500 gen)	GA_300_optimized (1000 gen)
count	10.000000	10.000000	10.000000
mean	58619.852540	47785.566977	33906.997014
std	1388.109343	952.462952	688.097991
min	56972.361396	46632.148360	32927.562807
25%	57502.892247	47101.443529	33492.040256
50%	58127.807687	47626.946163	33834.550170
75%	59623.536083	48128.406907	34382.048561
max	60901.771805	49860.257131	34925.855916

[185]:
```
data = [go.Box(y = df_large_sub[col], name = col) for col in df_large_sub]

layout = go.Layout(
    title = "Default vs optimized parameters with GA on 300 cities",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



[186]:
```
names = [
    '300_default',
    '300_optimized (500 gen)',
    '300_optimized (1000 gen)'
]

data = [go.Box(y = df_large_sub[col], name = name) for col, name in zip(df_large_sub, names)]

layout = go.Layout(
    title = "GA_300 best candidate solutions between default and optimized parameters",
    yaxis_title = 'Total solution distance (km)',
    yaxis = dict(
        type = 'log',
```

```
        autorange = True
    )
)
plotly.offline.iplot({'data':data, 'layout':layout})
```



To show significant differences between the group `GA_300_default_1000gen` and `GA_300_optimized_1000gen`, we can perform a **t-test**.

[75]:
```
stats.ttest_ind(df_large['GA_300_default_1000gen'], df_large['GA_300_optimized_1000gen'], equal_var = False)
```

[75]:
```
Ttest_indResult(statistic=16.79344861867695, pvalue=7.2028343598846e-10)
```

The **small p-value** (7.2028343598846e-10 < 0.05) and indicates that the two groups, `GA_300_default_1000gen` and `GA_300_optimized_1000gen`, are significantly different.