In [1]:

```
import pandas as pd
import seaborn as sns
```

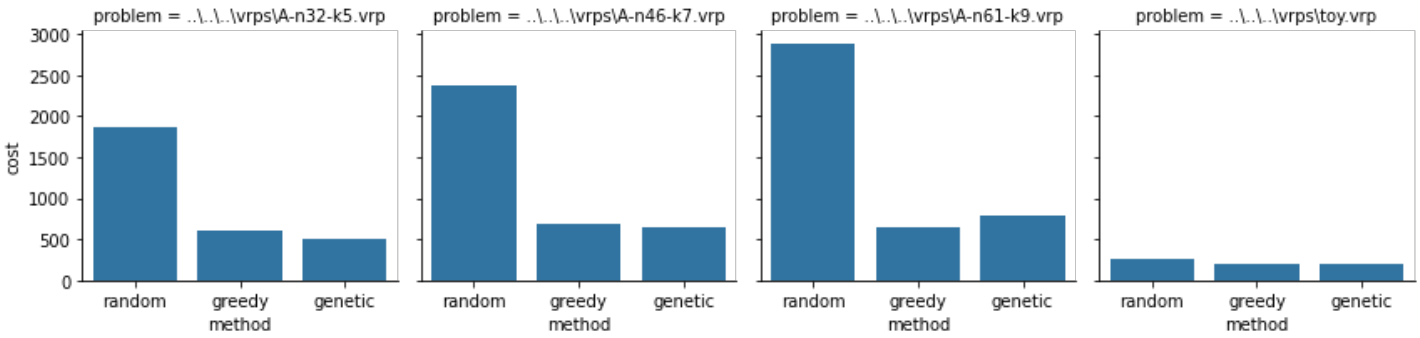
In [2]:

```
df = pd.read_csv('comparison-test.csv', sep=';')
g = sns.FacetGrid(df, col='problem')
g.map(sns.barplot, 'method', 'cost')
# for problem in df['problem'].unique():
#     sns.barplot(data=df[df['problem'] == problem], x='method', y='cost')
```

```
c:\users\kajetan\studia-mag\ai-venv\lib\site-packages\seaborn\axisgrid.py:643: UserWarnin
g: Using the barplot function without specifying `order` is likely to produce an incorrec
t plot.
  warnings.warn(warning)
```

Out[2]:

```
<seaborn.axisgrid.FacetGrid at 0x18cad676220>
```



Jak widać algorytmy zachłanny i genetyczny dają znacznie lepsze wyniki od metody losowej.

W większości przypadków algorytm genetyczny jest lepszy od zachłannego, ale nie zawsze.

Wyniki dla algorytmu zachłannego agregowałem średnią arytmetyczną, a dla losowania i algorytmu genetycznego medianą.

Dla algorytmu genetycznego użyłem parametrów:

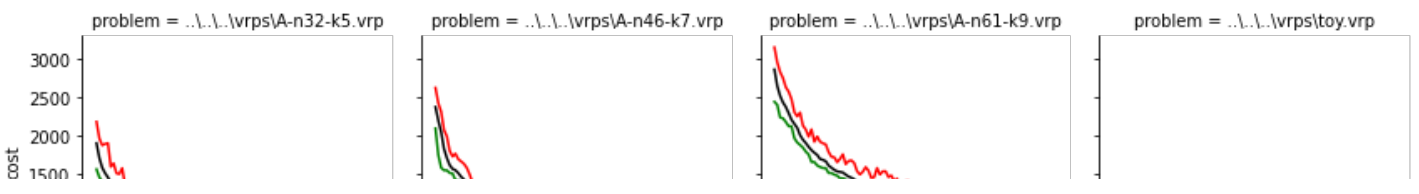
```
wielkość populacji = 100
ilość pokoleń = 100
prawdopodobieństwo krzyżowania = 0.6
prawdopodobieństwo mutacji = 0.3
wielkość próbki w turnieju = 10
```

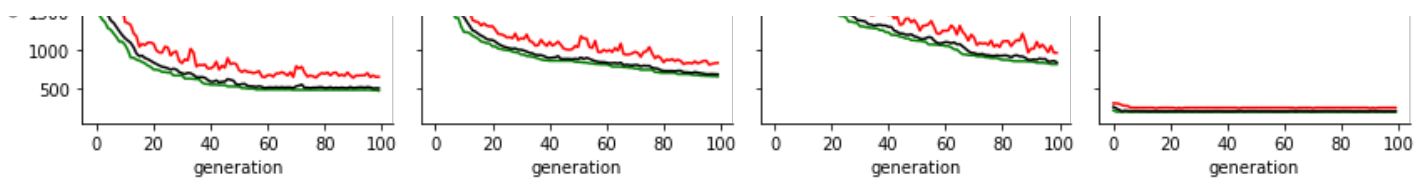
In [3]:

```
df = pd.read_csv('stats-test.csv', sep=';')
g = sns.FacetGrid(df, col='problem')
g.map(sns.lineplot, 'generation', 'best', color='green')
g.map(sns.lineplot, 'generation', 'worst', color='red')
g.map(sns.lineplot, 'generation', 'average', color='black')
g.set_ylabels('cost')
```

Out[3]:

```
<seaborn.axisgrid.FacetGrid at 0x18cad668970>
```





Czarna linia oznacza średni koszt, zielona najlepszy, a czerwona najgorszy.

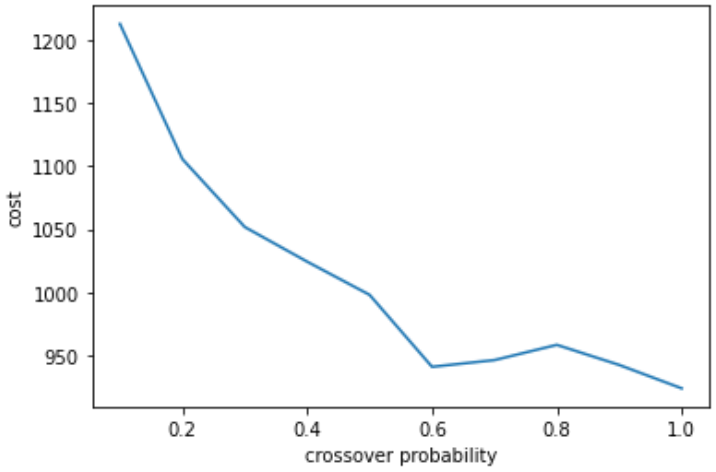
Wszystkie poniższe testy były robione na danych z pliku `A-n61-k9.vrp`

In [4]:

```
df = pd.read_csv('crossover-test.csv', sep=';')
ax = sns.lineplot(data=df, x='parameter', y='cost')
ax.set_xlabel('crossover probability')
```

Out[4]:

```
Text(0.5, 0, 'crossover probability')
```



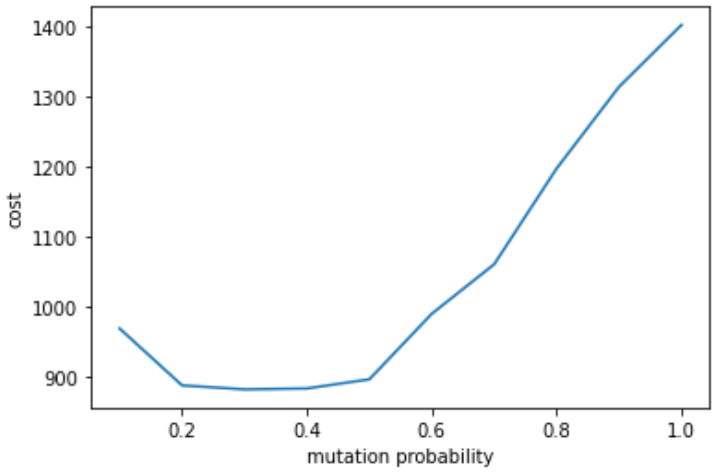
Na tych danych im więcej krzyzowania tym lepiej.

In [5]:

```
df = pd.read_csv('mutation-test.csv', sep=';')
ax = sns.lineplot(data=df, x='parameter', y='cost')
ax.set_xlabel('mutation probability')
```

Out[5]:

```
Text(0.5, 0, 'mutation probability')
```



Najlepsze wyniki zostały osiągnięte dla prawdopodobieństwa mutacji od 0.3 do 0.4.

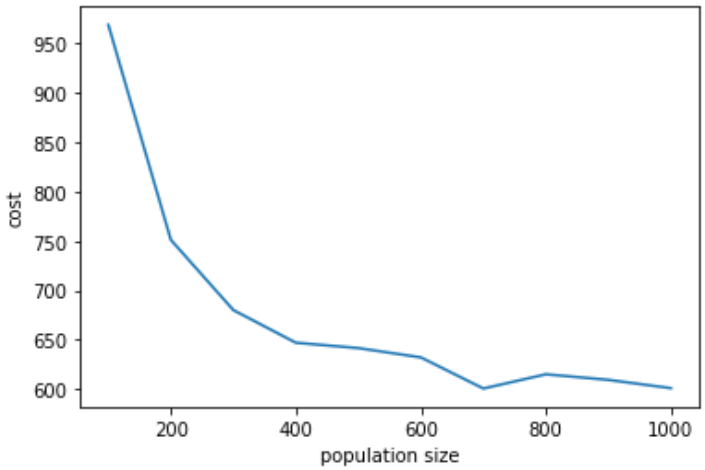
In [6]:

```
df = pd.read_csv('popsize-test.csv', sep=';')
```

```
df = pd.read_csv('popsize-test.csv', sep=';')
ax = sns.lineplot(data=df, x='parameter', y='cost')
ax.set_xlabel('population size')
```

Out[6]:

```
Text(0.5, 0, 'population size')
```



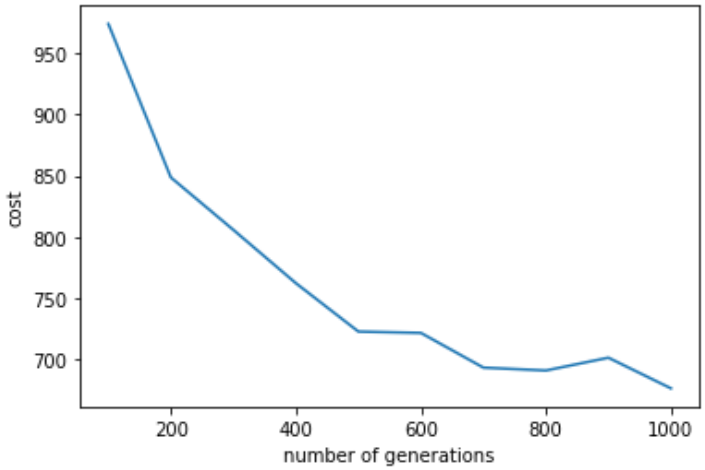
Jakość rozwiązań rośnie wraz ze zwiększaniem liczebności populacji, ale ma to koszt w postaci dłuższego działania programu.

In [7]:

```
df = pd.read_csv('generation-test.csv', sep=';')
ax = sns.lineplot(data=df, x='parameter', y='cost')
ax.set_xlabel('number of generations')
```

Out[7]:

```
Text(0.5, 0, 'number of generations')
```



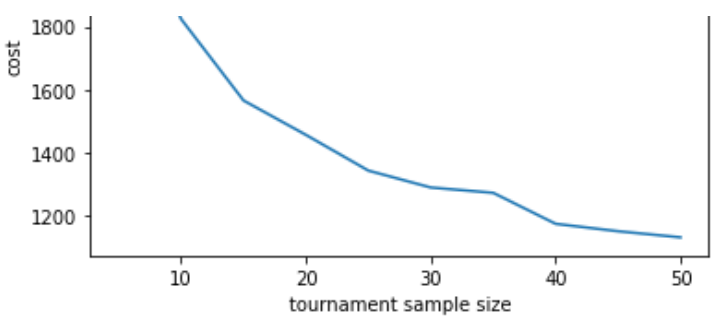
Podobnie jak w przypadku liczebności populacji, im więcej tym lepiej, ale kosztuje to czas i od pewnego momentu zwiększania liczby pokoleń pojawia się diminishing returns.

In [8]:

```
df = pd.read_csv('tournament-test.csv', sep=';')
ax = sns.lineplot(data=df, x='parameter', y='cost')
ax.set_xlabel('tournament sample size')
```

Out[8]:

```
Text(0.5, 0, 'tournament sample size')
```



Dla tych danych im większe próbki do turniejów tym lepiej. W przypadku małych próbek (szczególnie o wielkości 1) przystosowanie osobnika przestaje mieć znaczenie dla jego przetrwania i kolejne pokolenia nie są lepiej przystosowane od poprzednich.