report

September 16, 2018

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import os
os.environ['PATH']
```

In [2]:
```
df = pd.read_table("./reports/algorithm_def.csv", sep="," )
df.set_index('problem')
df
```

Out[2]:

	problem	description	name
0	1	Air Cargo Problem 1	air_cargo_p1
1	2	Air Cargo Problem 2	air_cargo_p2
2	3	Air Cargo Problem 3	air_cargo_p3
3	4	Air Cargo Problem 4	air_cargo_p4

In [3]:
```
df = pd.read_table("./reports/problems_def.csv", sep="," ,  keep_default_na=False)
#df.index = df['algorithm']
df
```

Out[3]:

	algorithm	type	sub
0	1	breadth_first_search	
1	2	depth_first_graph_search	
2	3	uniform_cost_search	
3	4	greedy_best_first_graph_search	h_unmet_goals
4	5	greedy_best_first_graph_search	h_pg_levelsum
5	6	greedy_best_first_graph_search	h_pg_maxlevel
6	7	greedy_best_first_graph_search	h_pg_setlevel
7	8	astar_search	h_unmet_goals
8	9	astar_search	h_pg_levelsum
9	10	astar_search	h_pg_maxlevel
10	11	astar_search	h_pg_setlevel

In [4]:
```
df = pd.read_table("./reports/report.csv", sep=" ")
cols = ['problem', 'algorithm' ]
df['prob_alg'] = df[cols].apply(lambda row: '_'.join(row.values.astype(str)), axis=1)
#df.index = df['prob_alg']
# df1= df
# df1.sort_values( ['algorithm']  )
df.head()
```

Out[4]:

	problem	algorithm	actions	expansions	goal_tests	new_nodes	plans	\
0	1	1	20	43	56	178	6	
1	1	2	20	21	22	84	20	
2	1	3	20	60	62	240	6	
3	1	4	20	7	9	29	6	
4	1	5	20	6	8	28	6	

	time	time_pypy	prob_alg
0	0.003356	0.017704	1_1
1	0.001965	0.007017	1_2
2	0.005541	0.023514	1_3
3	0.001091	0.004087	1_4
4	0.245197	0.342002	1_5

In [5]:
```python
df.pivot(index="problem", columns="algorithm", values="actions")
```

Out[5]:

algorithm	1	2	3	4	5	6	7	8	9	10	11
problem											
1	20	20	20	20	20	20	20	20	20	20	20
2	72	72	72	72	72	72	72	72	72	72	72
3	88	88	88	88	88	88	88	88	88	88	88
4	104	104	104	104	104	104	104	104	104	104	104

actions : problem + algorithm

In [6]:
```python
df.pivot(index="problem", columns="algorithm", values="expansions")
```

Out[6]:

algorithm	1	2	3	4	5	6	7	8	9	10	11
problem											
1	43	21	60	7	6	6	6	50	28	43	33
2	3343	624	5154	17	9	27	9	2467	357	2887	1037
3	14663	408	18510	25	14	21	35	7388	369	9580	3423
4	99736	25174	113339	29	17	56	107	34330	1208	62077	22606

expansions : problem + algorithm

In [7]:
```python
df.pivot(index="problem", columns="algorithm", values="goal_tests")
```

Out[7]:

algorithm	1	2	3	4	5	6	7	8	9	10	11
problem											
1	56	22	62	9	8	8	8	52	30	45	35
2	4609	625	5156	19	11	29	11	2469	359	2889	1039
3	18098	409	18512	27	16	23	37	7390	371	9582	3425
4	114953	25175	113341	31	19	58	109	34332	1210	62079	22608

goal_tests : problem + algorithm

In [8]:
```python
df.pivot(index="problem", columns="algorithm", values="new_nodes")
```

```
Out[8]: algorithm       1       2        3    4    5    6     7       8      9  \
        problem
        1             178      84      240   29   28   24    28     206    122
        2           30503    5602    46618  170   86  249    84   22522   3426
        3          129625    3364   161936  230  126  195   345   65711   3403
        4          944130  228849  1066413  280  165  580  1164  328509  12210

        algorithm      10      11
        problem
        1             180     138
        2           26594    9605
        3           86312   31596
        4          599376  224229
```

new_nodes : problem + algorithm

```
In [9]: df.pivot(index="problem", columns="algorithm", values="plans")
```

```
Out[9]: algorithm  1      2   3   4   5   6   7   8   9  10  11
        problem
        1           6     20   6   6   6   6   6   6   6   6   6
        2           9    619   9   9   9   9   9   9   9   9   9
        3          12    392  12  15  14  13  17  12  12  12  12
        4          14  24132  14  18  17  17  23  14  15  14  14
```

plans : problem + algorithm

```
In [10]: df.pivot(index="problem", columns="algorithm", values="time")
```

```
Out[10]: algorithm          1            2          3         4          5          6  \
         problem
         1           0.003356     0.001965   0.005541  0.001091   0.245197   0.182899
         2           1.100010     1.647735   2.010739  0.012014   5.241358  10.970011
         3           5.947077     0.621685   8.412868  0.022816  12.103299  14.188231
         4          54.530920  2287.498834  64.886528  0.036171  21.400484  52.189356

         algorithm           7          8            9            10           11
         problem
         1            0.304724   0.005613     0.572962      0.632937     0.663697
         2            7.823807   1.341593   133.487818    125.914516   645.784869
         3           40.906341   4.958024   214.810693   3786.037109  3413.015160
         4          189.038403  33.235617  1236.402619  37733.948257  5807.762095
```

time : problem + algorithm

```
In [11]: df.pivot(index="problem", columns="algorithm", values="time_pypy")
```

```
Out[11]: algorithm         1            2          3          4          5          6  \
         problem
```

```
1          0.017704     0.007017  0.023514  0.004087  0.342002  0.292564
2          0.290431     0.482155  0.544043  0.027981  1.324547  2.573012
3          0.745864     0.235377  1.211840  0.033488  2.634233  2.986649
4          4.349403  1043.870237  7.091650  0.043807  4.216966  9.012967

algorithm          7         8           9           10           11
problem
1           0.611727  0.024670    0.538900     0.552604     0.863610
2           2.339790  0.557923   21.842962   125.914516    99.913435
3           8.621016  1.165601   35.855637   589.084735   543.378761
4          33.798882  3.725822  189.915703  5867.480810  5807.762095
```

time_pypy : problem + algorithm

```
In [12]: # all data
         df
```

```
Out[12]:     problem  algorithm  actions  expansions  goal_tests  new_nodes  plans  \
         0         1          1       20          43          56        178      6
         1         1          2       20          21          22         84     20
         2         1          3       20          60          62        240      6
         3         1          4       20           7           9         29      6
         4         1          5       20           6           8         28      6
         5         1          6       20           6           8         24      6
         6         1          7       20           6           8         28      6
         7         1          8       20          50          52        206      6
         8         1          9       20          28          30        122      6
         9         1         10       20          43          45        180      6
         10        1         11       20          33          35        138      6
         11        2          1       72        3343        4609      30503      9
         12        2          2       72         624         625       5602    619
         13        2          3       72        5154        5156      46618      9
         14        2          4       72          17          19        170      9
         15        2          5       72           9          11         86      9
         16        2          6       72          27          29        249      9
         17        2          7       72           9          11         84      9
         18        2          8       72        2467        2469      22522      9
         19        2          9       72         357         359       3426      9
         20        2         10       72        2887        2889      26594      9
         21        2         11       72        1037        1039       9605      9
         22        3          1       88       14663       18098     129625     12
         23        3          2       88         408         409       3364    392
         24        3          3       88       18510       18512     161936     12
         25        3          4       88          25          27        230     15
         26        3          5       88          14          16        126     14
         27        3          6       88          21          23        195     13
         28        3          7       88          35          37        345     17
         29        3          8       88        7388        7390      65711     12
```

30	3	9	88	369	371	3403	12
31	3	10	88	9580	9582	86312	12
32	3	11	88	3423	3425	31596	12
33	4	1	104	99736	114953	944130	14
34	4	2	104	25174	25175	228849	24132
35	4	3	104	113339	113341	1066413	14
36	4	4	104	29	31	280	18
37	4	5	104	17	19	165	17
38	4	6	104	56	58	580	17
39	4	7	104	107	109	1164	23
40	4	8	104	34330	34332	328509	14
41	4	9	104	1208	1210	12210	15
42	4	10	104	62077	62079	599376	14
43	4	11	104	22606	22608	224229	14

	time	time_pypy	prob_alg
0	0.003356	0.017704	1_1
1	0.001965	0.007017	1_2
2	0.005541	0.023514	1_3
3	0.001091	0.004087	1_4
4	0.245197	0.342002	1_5
5	0.182899	0.292564	1_6
6	0.304724	0.611727	1_7
7	0.005613	0.024670	1_8
8	0.572962	0.538900	1_9
9	0.632937	0.552604	1_10
10	0.663697	0.863610	1_11
11	1.100010	0.290431	2_1
12	1.647735	0.482155	2_2
13	2.010739	0.544043	2_3
14	0.012014	0.027981	2_4
15	5.241358	1.324547	2_5
16	10.970011	2.573012	2_6
17	7.823807	2.339790	2_7
18	1.341593	0.557923	2_8
19	133.487818	21.842962	2_9
20	125.914516	125.914516	2_10
21	645.784869	99.913435	2_11
22	5.947077	0.745864	3_1
23	0.621685	0.235377	3_2
24	8.412868	1.211840	3_3
25	0.022816	0.033488	3_4
26	12.103299	2.634233	3_5
27	14.188231	2.986649	3_6
28	40.906341	8.621016	3_7
29	4.958024	1.165601	3_8
30	214.810693	35.855637	3_9
31	3786.037109	589.084735	3_10

```
32   3413.015160   543.378761     3_11
33     54.530920     4.349403      4_1
34   2287.498834  1043.870237      4_2
35     64.886528     7.091650      4_3
36      0.036171     0.043807      4_4
37     21.400484     4.216966      4_5
38     52.189356     9.012967      4_6
39    189.038403    33.798882      4_7
40     33.235617     3.725822      4_8
41   1236.402619   189.915703      4_9
42  37733.948257  5867.480810     4_10
43   5807.762095  5807.762095     4_11
```

all data

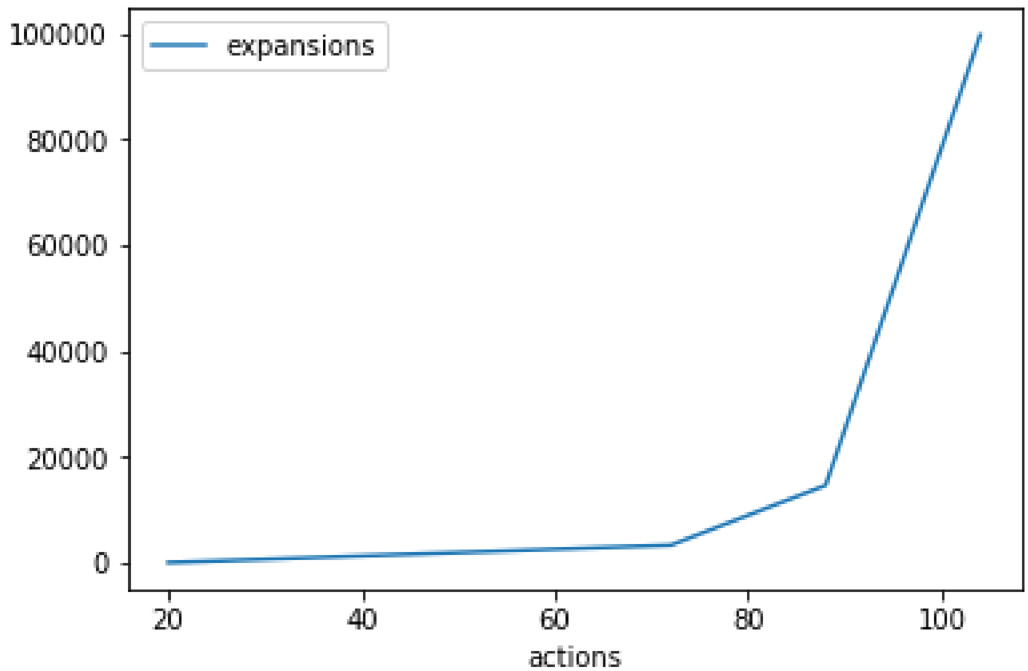
0.0.1 1.Use a table or chart to analyze the number of nodes expanded against number of actions in the domain

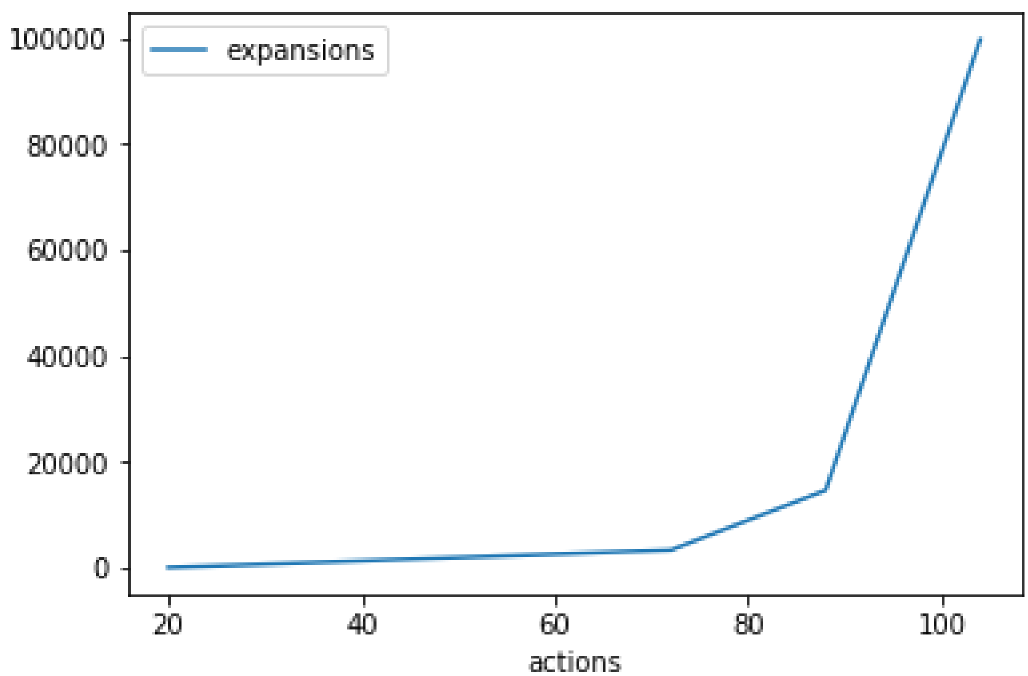
In [13]:

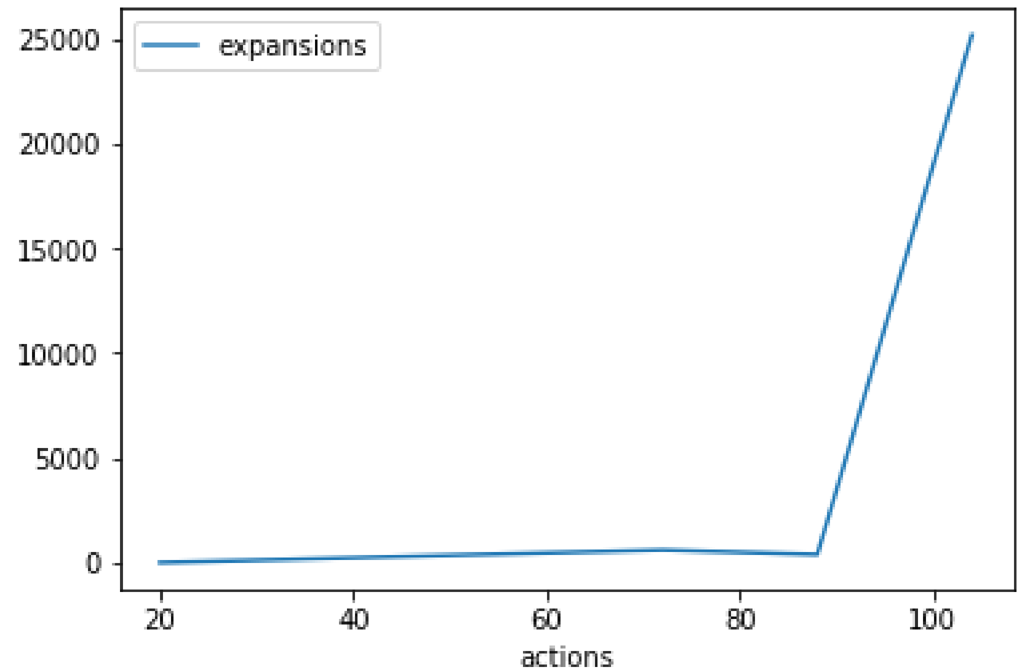
```
df_g = df.groupby('algorithm')
df_g.plot.line(x='actions', y='expansions')
```

Out[13]:

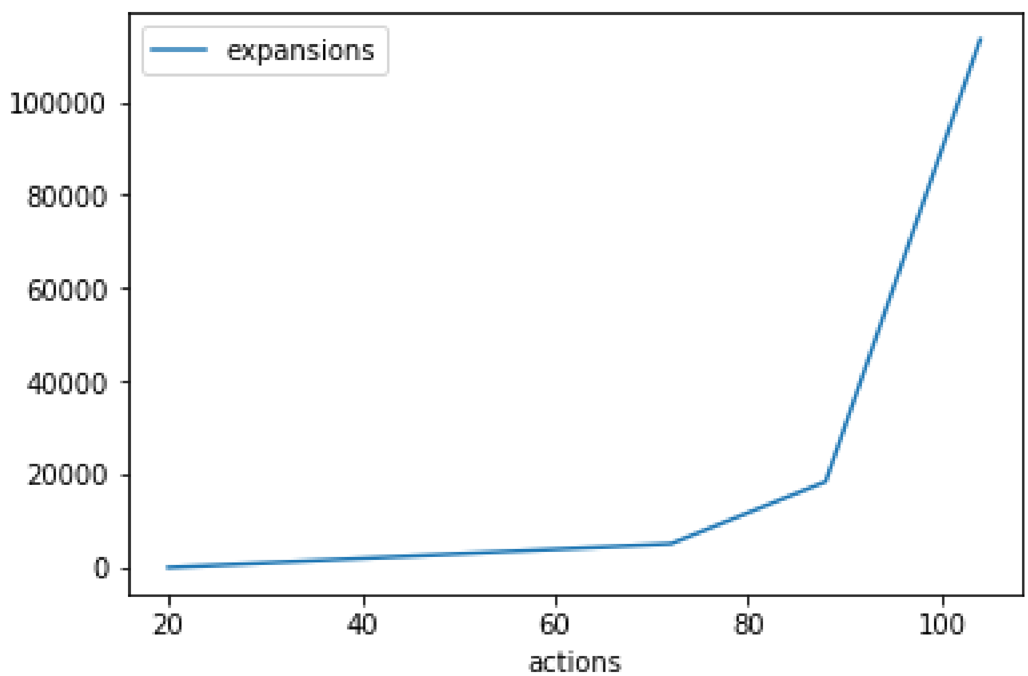
```
algorithm
1      AxesSubplot(0.125,0.125;0.775x0.755)
2      AxesSubplot(0.125,0.125;0.775x0.755)
3      AxesSubplot(0.125,0.125;0.775x0.755)
4      AxesSubplot(0.125,0.125;0.775x0.755)
5      AxesSubplot(0.125,0.125;0.775x0.755)
6      AxesSubplot(0.125,0.125;0.775x0.755)
7      AxesSubplot(0.125,0.125;0.775x0.755)
8      AxesSubplot(0.125,0.125;0.775x0.755)
9      AxesSubplot(0.125,0.125;0.775x0.755)
10     AxesSubplot(0.125,0.125;0.775x0.755)
11     AxesSubplot(0.125,0.125;0.775x0.755)
dtype: object
```


expansions
100000
80000
60000
40000
20000
0
20
40
60
80
100
actions

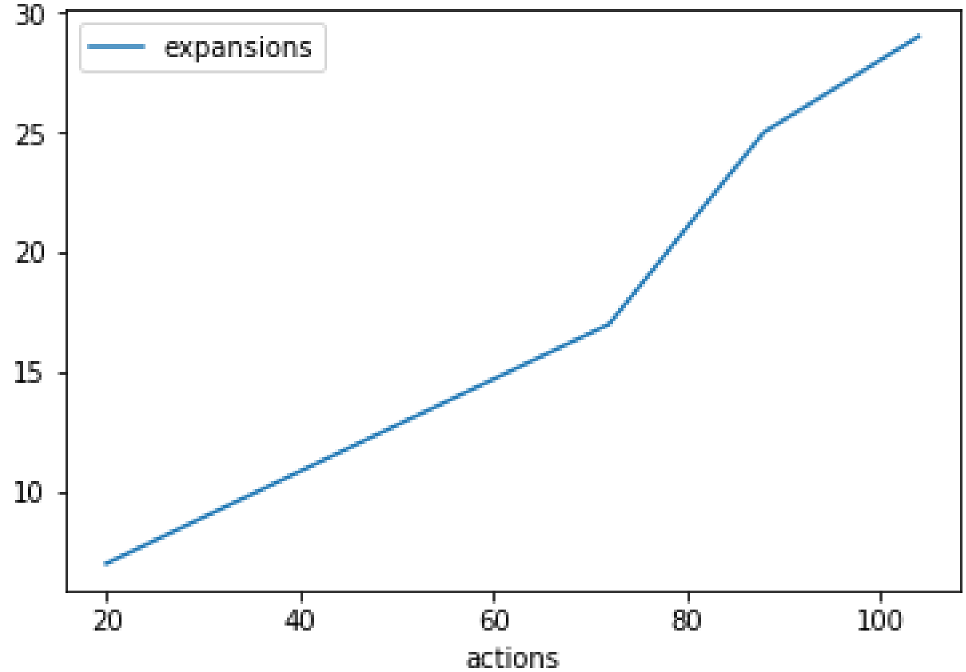

expansions
100000
80000
60000
40000
20000
0
20
40
60
80
100
actions



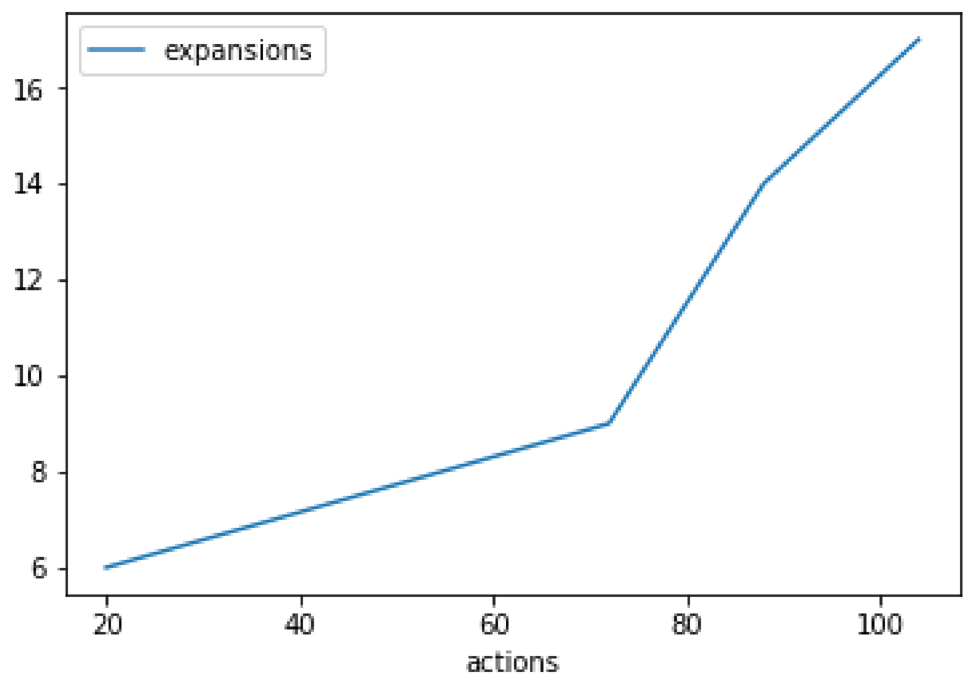
expansions
25000
20000
15000
10000
5000
0
20
40
60
80
100
actions



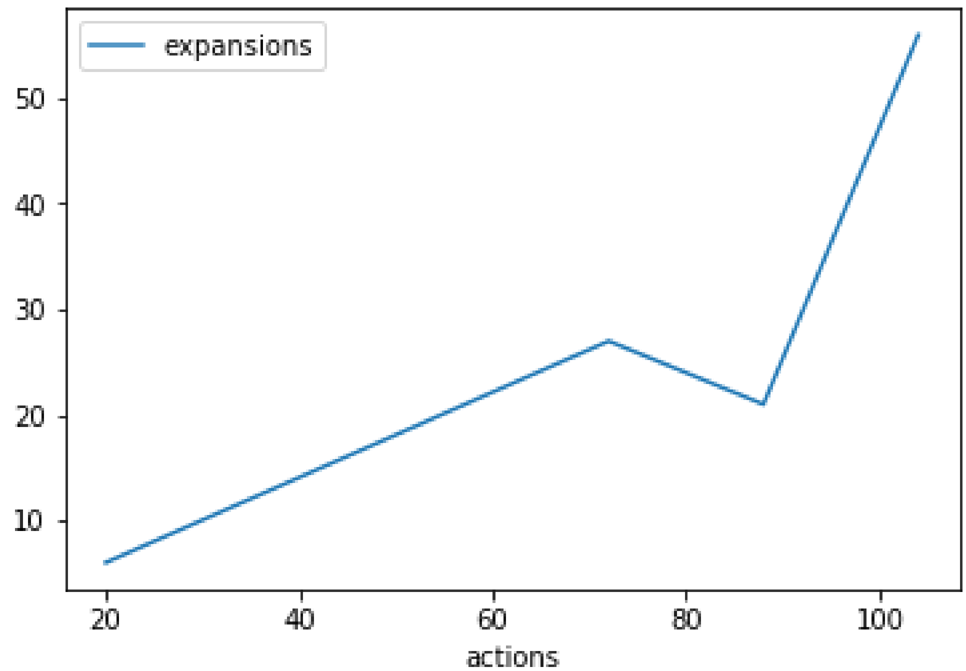
expansions
100000
80000
60000
40000
20000
0
20
40
60
80
100
actions

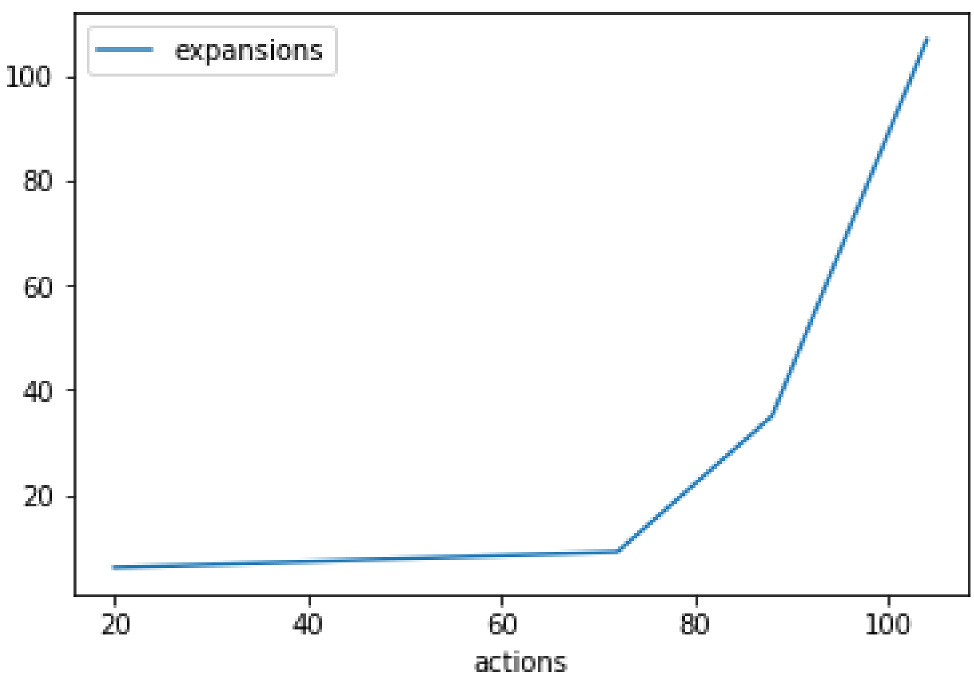


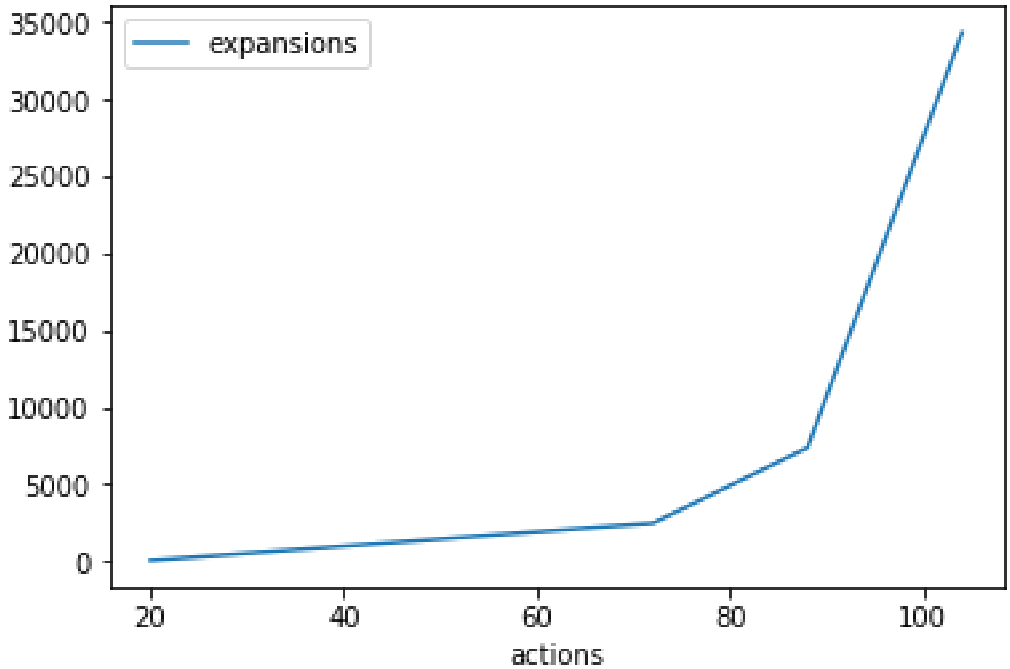
expansions
30
25
20
15
10
20
40
60
80
100
actions

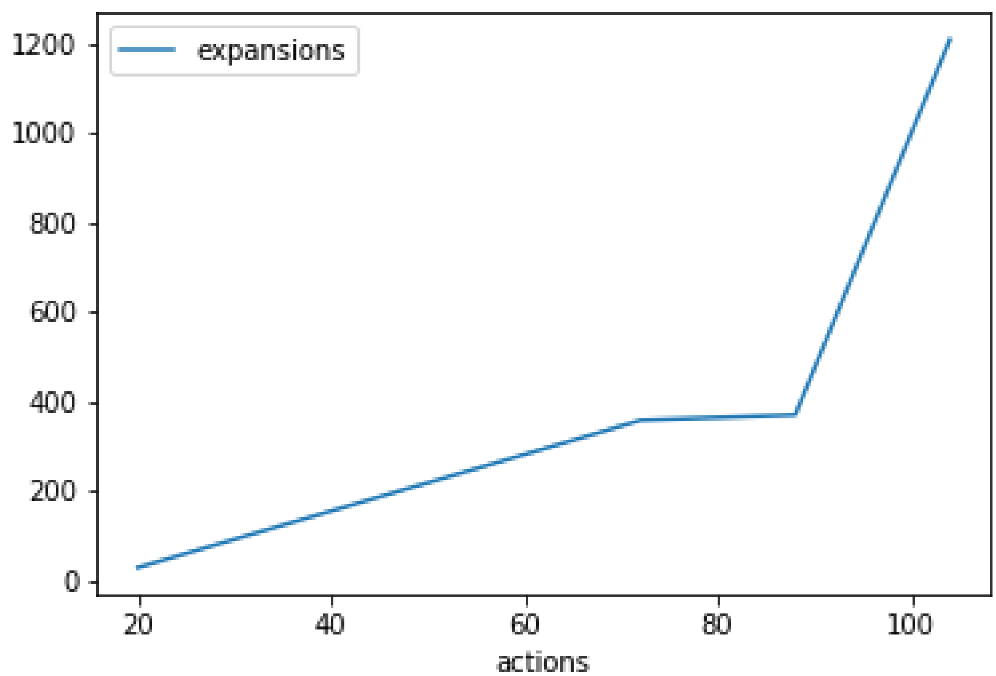


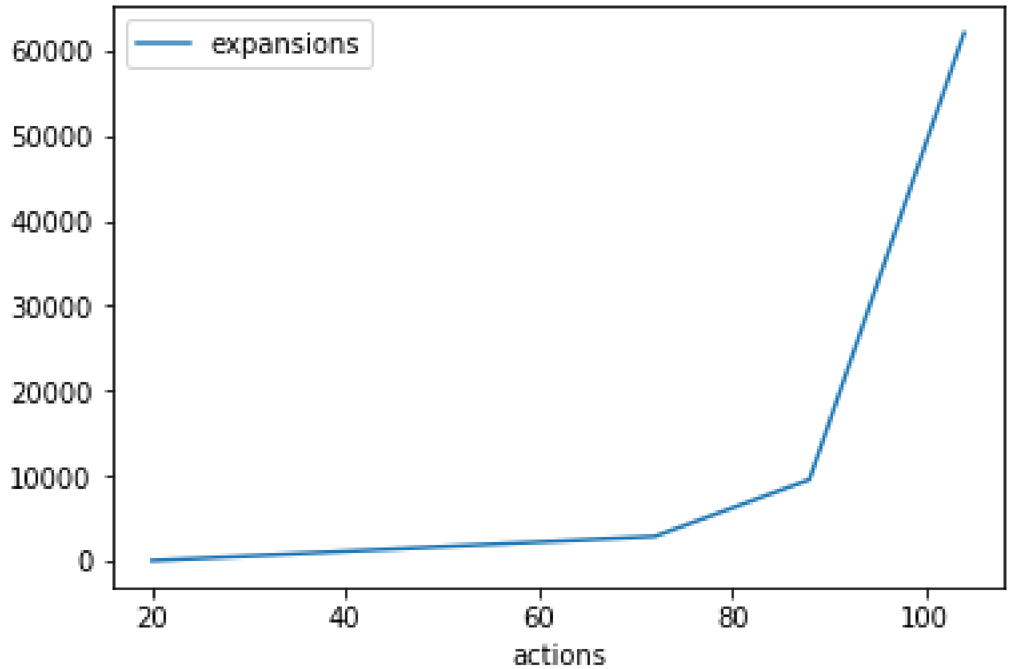
expansions
16
14
12
10
8
6
20
40
60
80
100
actions

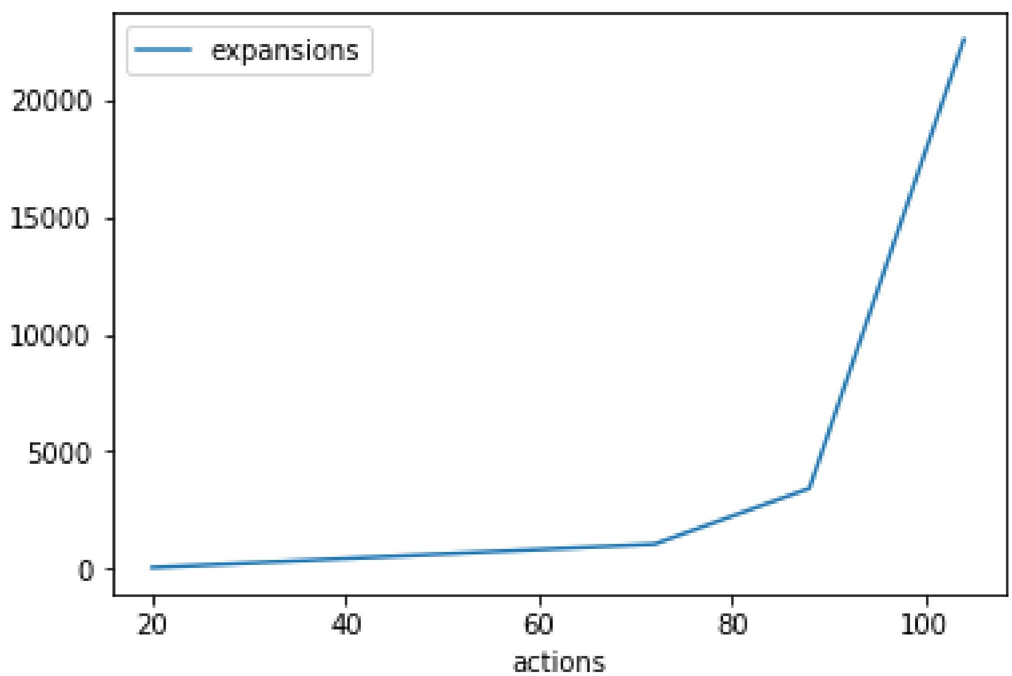

expansions
50
40
30
20
10
20
40
60
80
100
actions


expansions
100
80
60
40
20
20
40
60
80
100
actions


expansions
35000
30000
25000
20000
15000
10000
5000
0
20
40
60
80
100
actions


expansions
1200
1000
800
600
400
200
0
20
40
60
80
100
actions





Answer: there is corelation between two variables = 'number of nodes expanded' and 'actions'

```
In [14]: # df_g = df.groupby('algorithm')
         # df_g.plt.plot(x='actions', y='expansions')
         # df.groupby('algorithm' )['actions','expansions'].plot(legend=True)
         # df['actions'].groupby(df['expansions']).describe()
         # df['algorithm'].groupby(df['actions']) .describe()
```

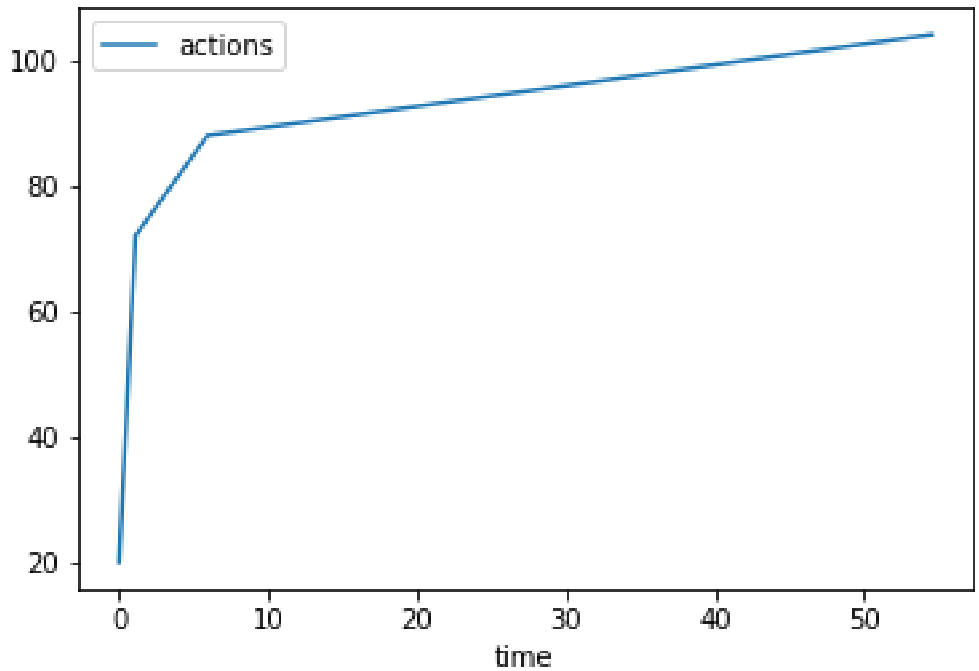
```
In [15]: # df['algorithm'].groupby(df['expansions']).plot()
         # df.groupby('problem' )['actions','expansions'].plot()
         # df.groupby('problem' )['actions','expansions'].plot()
```

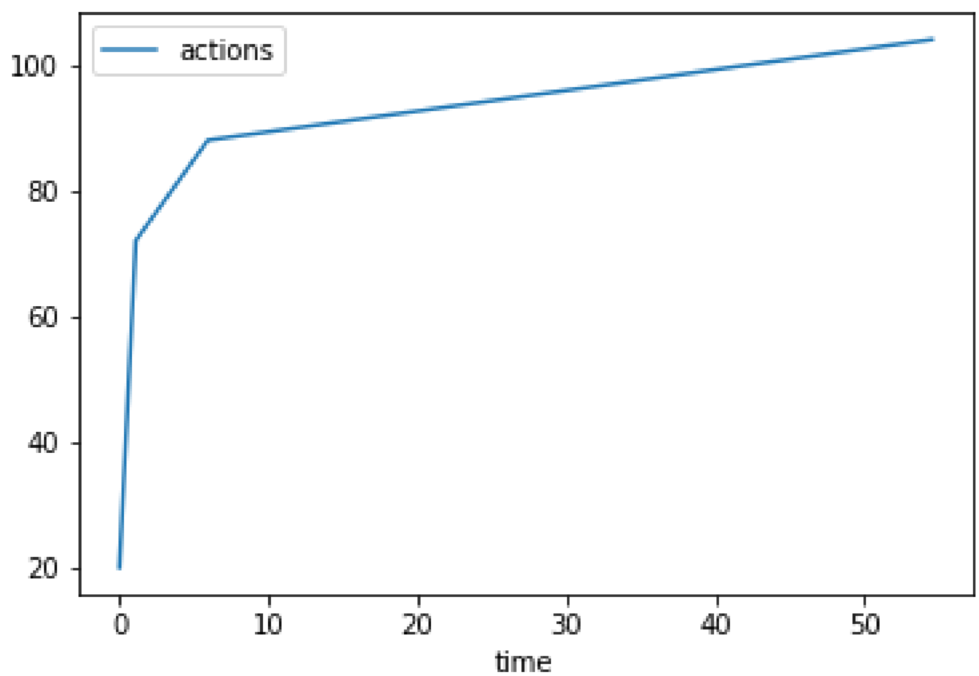
0.0.2 2.Use a table or chart to analyze the search time against the number of actions in the domain

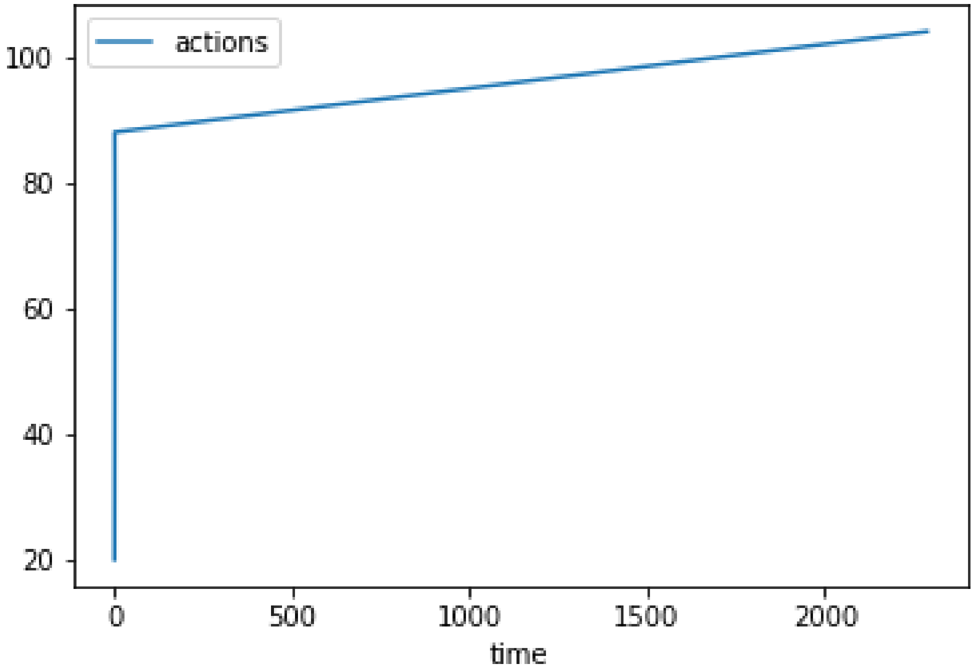
```
In [16]: # df.plot.scatter(x="time", y="actions", c='DarkBlue')
         # plt.show()

         df_g = df.groupby('algorithm')
         df_g.plot.line(x='time', y='actions')
```

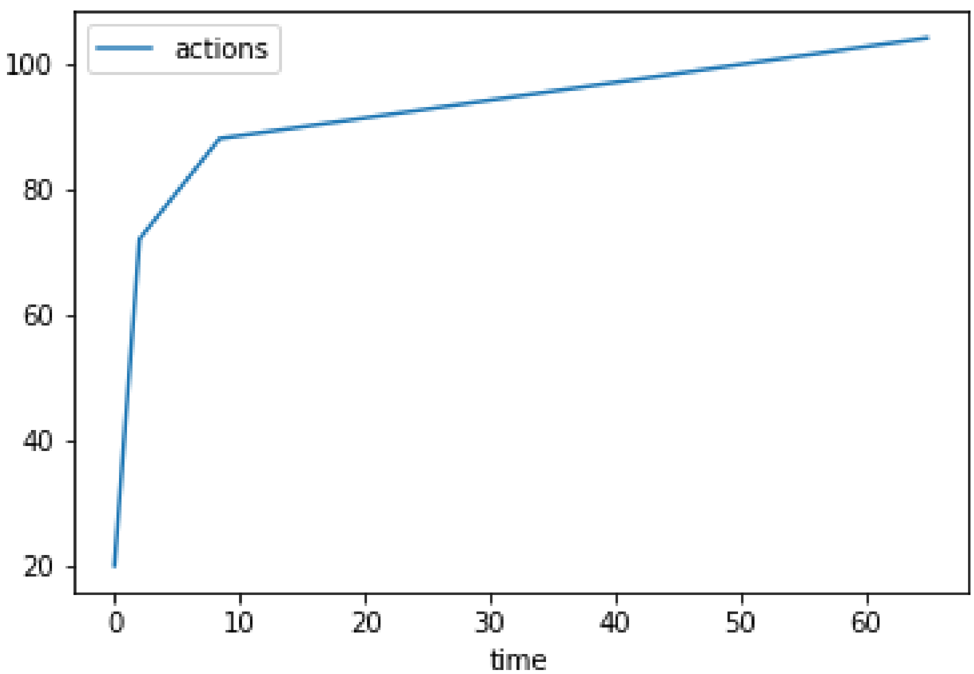
```
Out[16]: algorithm
         1      AxesSubplot(0.125,0.125;0.775x0.755)
         2      AxesSubplot(0.125,0.125;0.775x0.755)
         3      AxesSubplot(0.125,0.125;0.775x0.755)
         4      AxesSubplot(0.125,0.125;0.775x0.755)
         5      AxesSubplot(0.125,0.125;0.775x0.755)
         6      AxesSubplot(0.125,0.125;0.775x0.755)
         7      AxesSubplot(0.125,0.125;0.775x0.755)
         8      AxesSubplot(0.125,0.125;0.775x0.755)
         9      AxesSubplot(0.125,0.125;0.775x0.755)
         10     AxesSubplot(0.125,0.125;0.775x0.755)
         11     AxesSubplot(0.125,0.125;0.775x0.755)
         dtype: object
```


actions
100
80
60
40
20
0
10
20
30
40
50
time

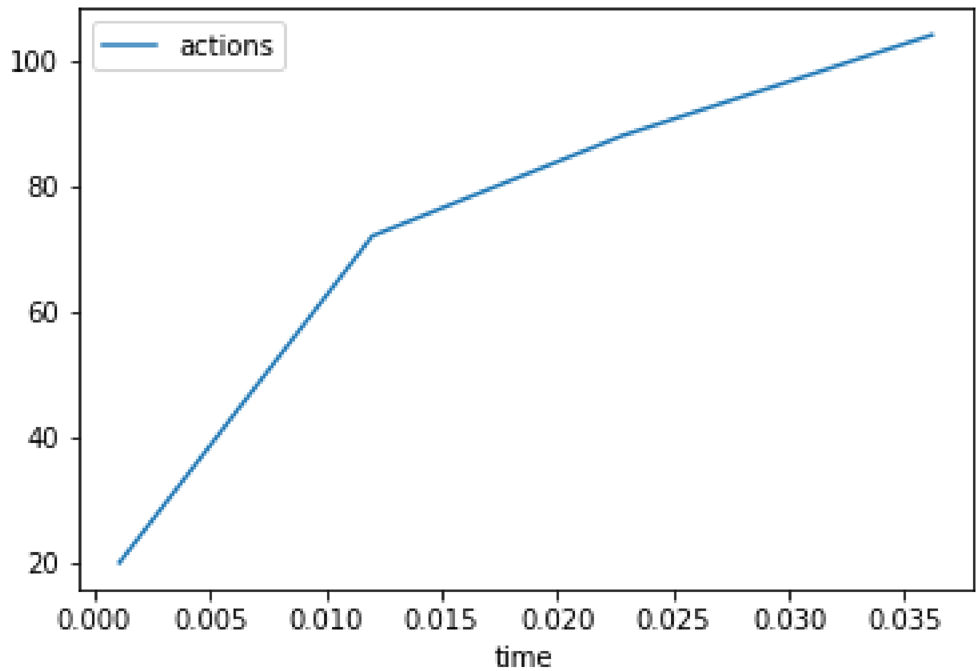

actions
100
80
60
40
20
0
10
20
30
40
50
time

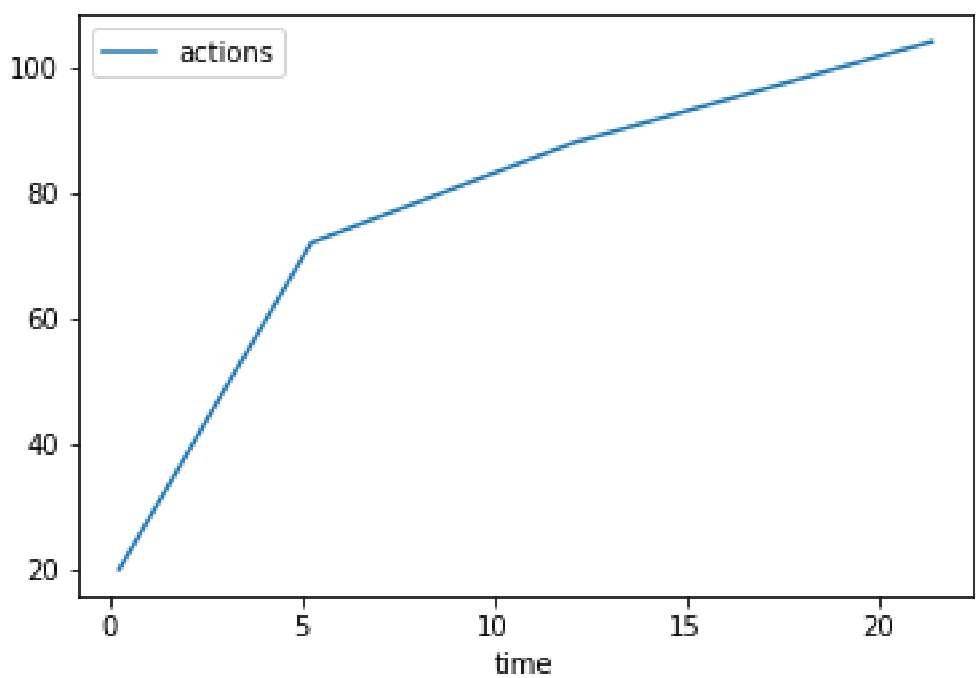


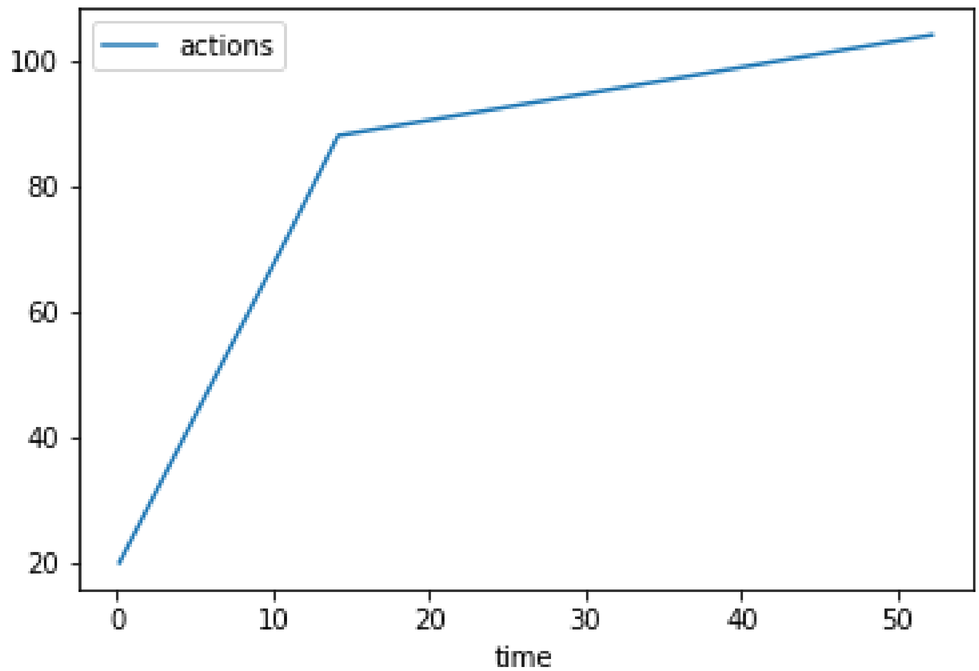
actions
100
80
60
40
20
0
500
1000
1500
2000
time

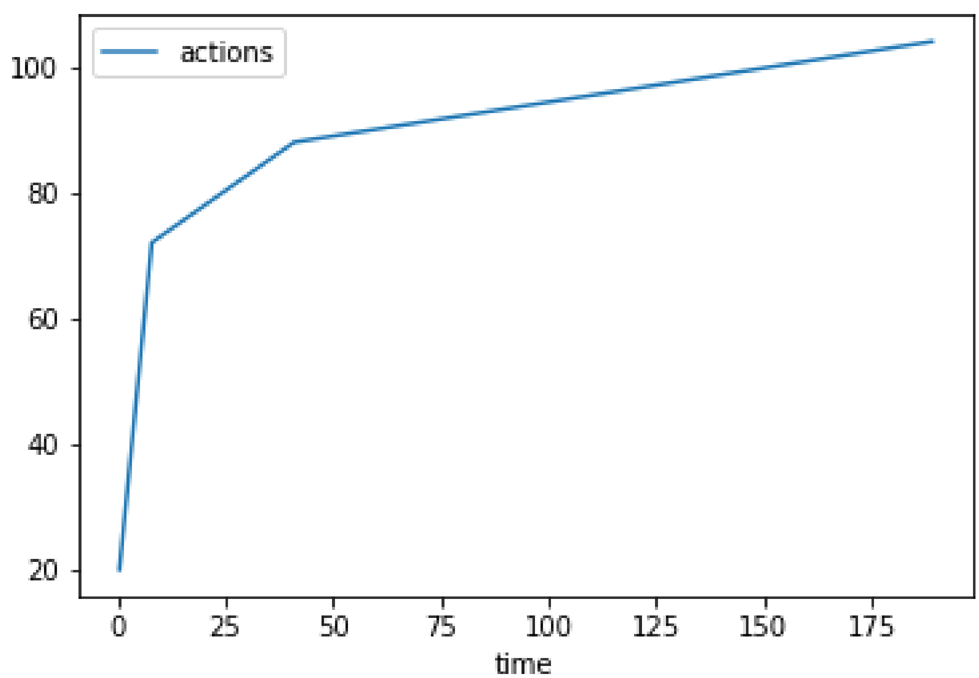


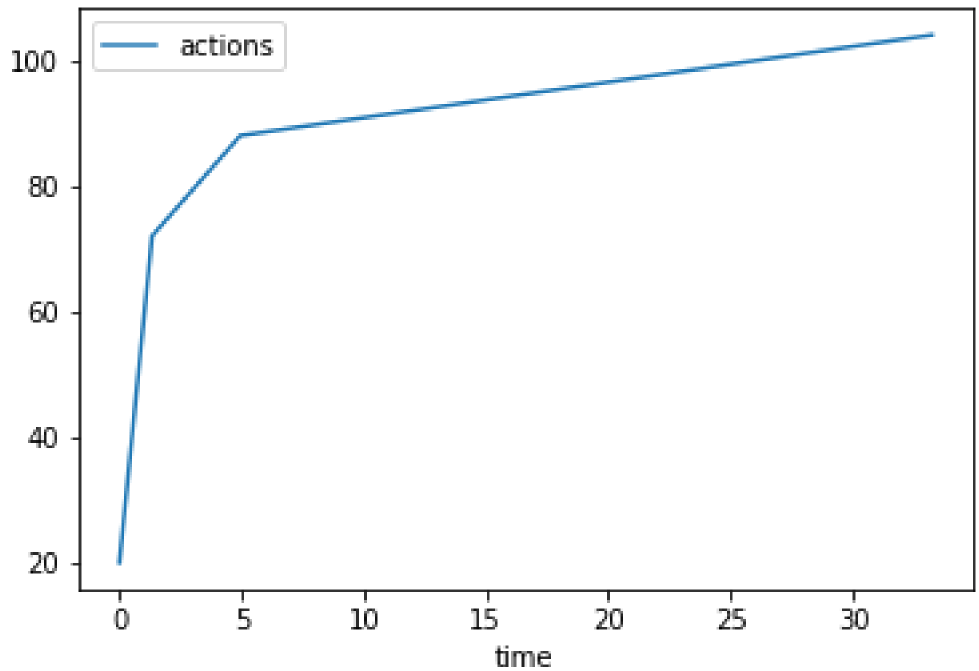
actions
100
80
60
40
20
0
10
20
30
40
50
60
time

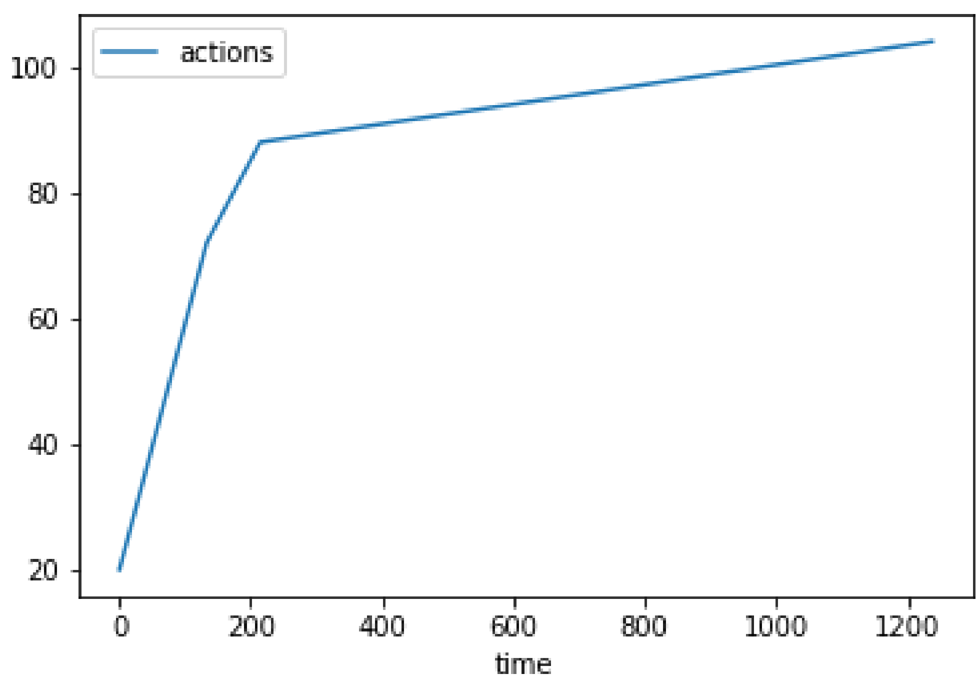

actions
100
80
60
40
20
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
time

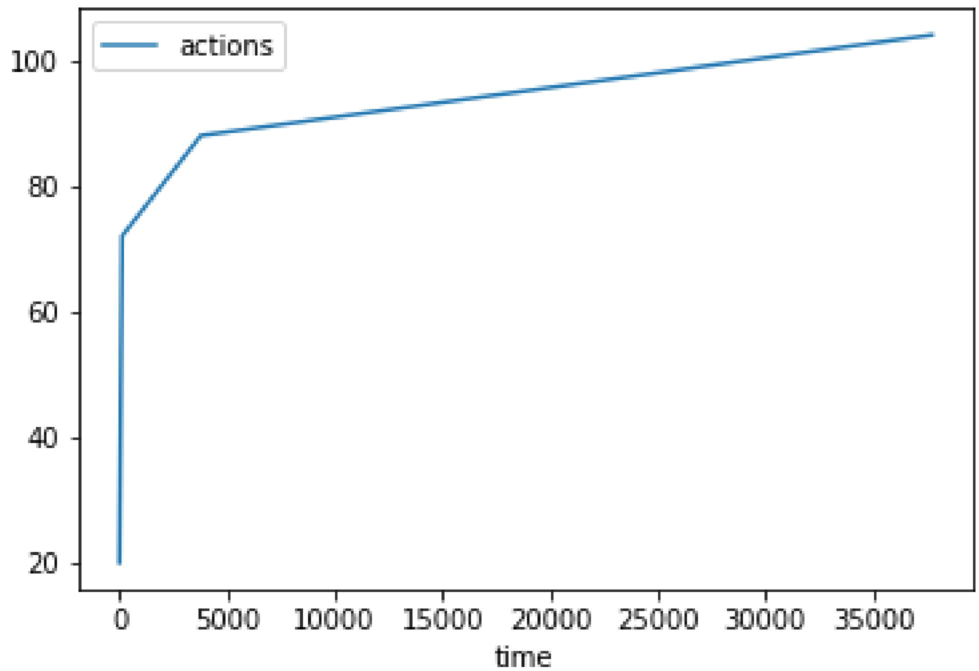

actions
100
80
60
40
20
0
5
10
15
20
time

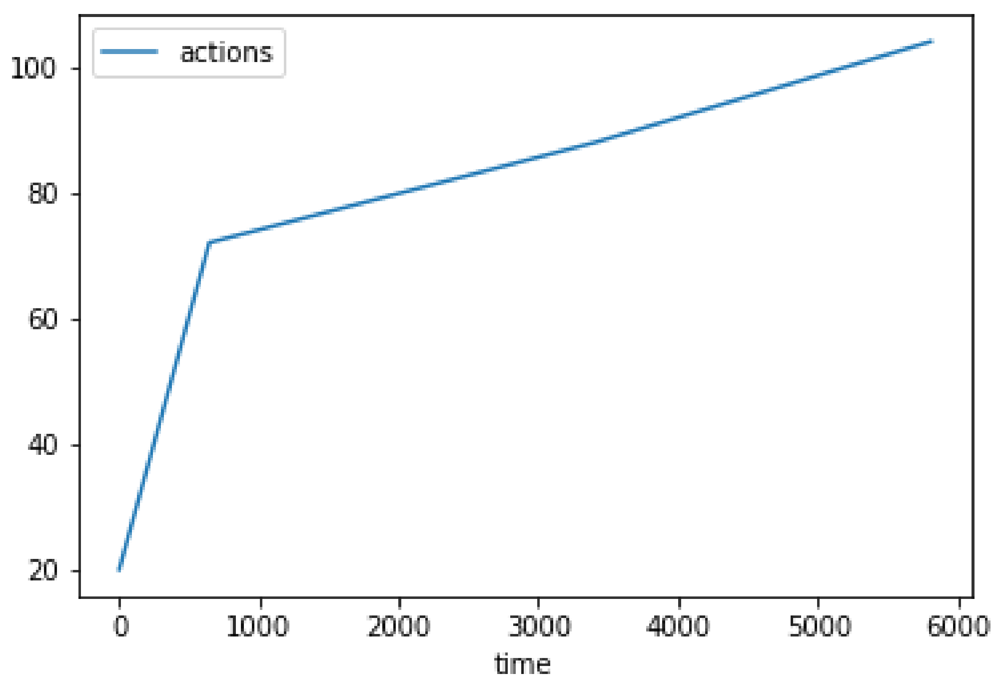

actions
100
80
60
40
20
0
10
20
30
40
50
time


actions
100
80
60
40
20
0
25
50
75
100
125
150
175
time


actions
100
80
60
40
20
0
5
10
15
20
25
30
time


actions
100
80
60
40
20
0
200
400
600
800
1000
1200
time





Answer: Positive correlation between ‘time’ and ‘actions’.

0.0.3 3.Use a table or chart to analyze the length of the plans returned by each algorithm on all search problems

In [17]:
```
df.pivot(index="problem", columns="algorithm", values="plans")
```

Out[17]:

algorithm problem	1	2	3	4	5	6	7	8	9	10	11
1	6	20	6	6	6	6	6	6	6	6	6
2	9	619	9	9	9	9	9	9	9	9	9
3	12	392	12	15	14	13	17	12	12	12	12
4	14	24132	14	18	17	17	23	14	15	14	14

Answer: algorith 1 and 7 : generate longest plan length - 2=depth_first_graph_search - 7=greedy_best_first_graph_search + h_pg_setlevel.

0.0.4 4.Which algorithm or algorithms would be most appropriate for planning in a very restricted domain (i.e., one that has only a few actions) and needs to operate in real time?

In [18]:
```
df.groupby(["problem"]).min()["actions"]
```

Out[18]:
```
problem
1     20
2     72
3     88
4    104
Name: actions, dtype: int64
```

In [19]:
```
df.loc[df.problem == 1,["problem", "algorithm", "time"]].sort_values(by="time")
```

Out[19]:

	problem	algorithm	time
3	1	4	0.001091
1	1	2	0.001965
0	1	1	0.003356
2	1	3	0.005541
7	1	8	0.005613
5	1	6	0.182899
4	1	5	0.245197
6	1	7	0.304724
8	1	9	0.572962
9	1	10	0.632937
10	1	11	0.663697

Answer: Problem 1 - problem 1 = Air Cargo Problem 1 + air_cargo_p1 - algorithm 4 = greedy_best_first_graph_search + h_unmet_goals - algorithm 2 = depth_first_graph_search

In [20]:
```
df.loc[df.problem == 2,["problem", "algorithm", "time"]].sort_values(by="time")
```

Out[20]:

	problem	algorithm	time
14	2	4	0.012014
11	2	1	1.100010

```
18        2          8    1.341593
12        2          2    1.647735
13        2          3    2.010739
15        2          5    5.241358
17        2          7    7.823807
16        2          6   10.970011
20        2         10  125.914516
19        2          9  133.487818
21        2         11  645.784869
```

Problem 2

```
In [21]: df.loc[df.problem == 3,["problem", "algorithm", "time"]].sort_values(by="time")
```

```
Out[21]:      problem  algorithm         time
         25         3          4     0.022816
         23         3          2     0.621685
         29         3          8     4.958024
         22         3          1     5.947077
         24         3          3     8.412868
         26         3          5    12.103299
         27         3          6    14.188231
         28         3          7    40.906341
         30         3          9   214.810693
         32         3         11  3413.015160
         31         3         10  3786.037109
```

Problem 3

0.0.5 5.Which algorithm or algorithms would be most appropriate for planning in very large domains (e.g., planning delivery routes for all UPS drivers in the U.S. on a given day)

```
In [22]: df.loc[df.problem == 4,["problem", "algorithm", "time"]].sort_values(by="time")
```

```
Out[22]:      problem  algorithm          time
         36         4          4      0.036171
         37         4          5     21.400484
         40         4          8     33.235617
         38         4          6     52.189356
         33         4          1     54.530920
         35         4          3     64.886528
         39         4          7    189.038403
         41         4          9   1236.402619
         34         4          2   2287.498834
         43         4         11   5807.762095
         42         4         10  37733.948257
```

Answer: - problem 4 = Air Cargo Problem 4 + air_cargo_p4 - algorithm 4 = greedy_best_first_graph_search + h_unmet_goals - algorithm 5 = greedy_best_first_graph_search + h_pg_levelsum

0.0.6 6.Which algorithm or algorithms would be most appropriate for planning problems where it is important to find only optimal plans?

Answer: - 4 = greedy_best_first_graph_search + h_unmet_goal - 5 = greedy_best_first_graph_search + h_pg_levelsum - 8 = astar_search + h_unmet_goals

In [23]:
```python
df = pd.read_table("./reports/problems_def.csv", sep="," ,  keep_default_na=False)
df
```

Out[23]:

	algorithm	type	sub
0	1	breadth_first_search	
1	2	depth_first_graph_search	
2	3	uniform_cost_search	
3	4	greedy_best_first_graph_search	h_unmet_goals
4	5	greedy_best_first_graph_search	h_pg_levelsum
5	6	greedy_best_first_graph_search	h_pg_maxlevel
6	7	greedy_best_first_graph_search	h_pg_setlevel
7	8	astar_search	h_unmet_goals
8	9	astar_search	h_pg_levelsum
9	10	astar_search	h_pg_maxlevel
10	11	astar_search	h_pg_setlevel

-sscript used for the test

```
export path_pypy=/home/lab/software/pypy3/bin/pypy3
dir=reports
mkdir -p $dir
declare -a arr=(1 2 3 4 5 6 7 8 9 10 11)

function  pypycall {
    test=$1
    i=$2
    echo "running  $path_pypy run_search.py -p ${test}  -s $i  > ${dir}/${test}_${i}_pypy.txt"
    $path_pypy run_search.py -p ${test}  -s $i  > "${dir}/${test}_${i}_pypy.txt"

  }
function  pythoncall {
    test=$1
    i=$2
    echo "running python run_search.py -p ${test}  -s $i  > ${dir}/${test}_${i}_python.txt" ; e
    python run_search.py -p ${test}  -s $i  > "${dir}/${test}_${i}_python.txt"

  }

for i in "${arr[@]}"
do

  pypycall 1  $i
  pythoncall 1  $i
```

```
  pypycall 2  $i
  pythoncall 2  $i

  pypycall 3  $i
  pythoncall 3  $i

  pypycall 4  $i
  pythoncall 4  $i

done
```