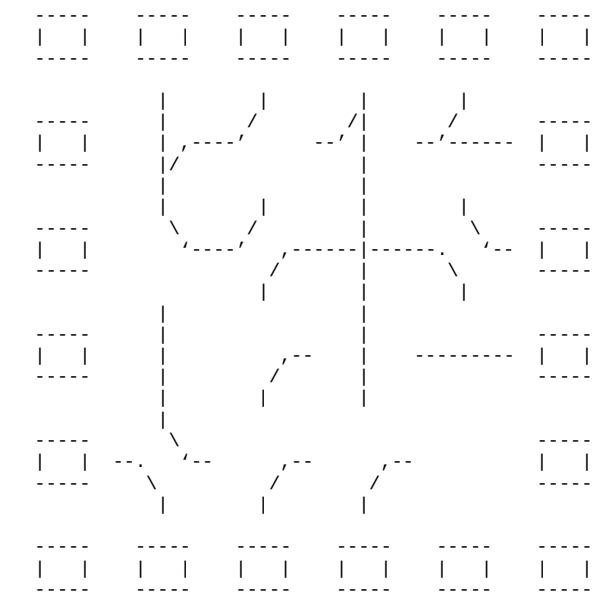
CS 188, Spring 2004
Solutions for Assignment 2

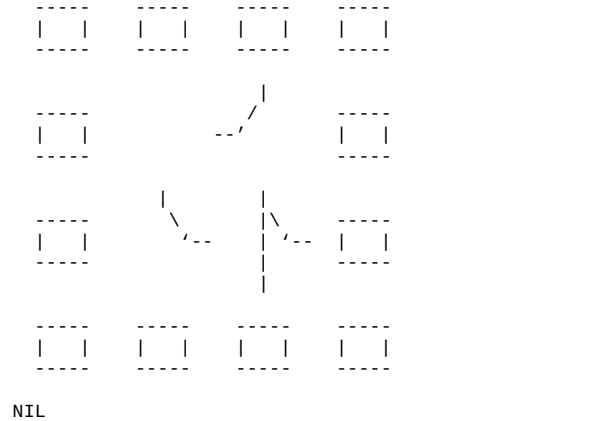
1. First, to help you become familiar with tracks and pieces, write a function (count-loose-ends track), which should return the number of loose ends in the track.

   The solution is fairly simple: just go trough the track pieces and check for each outgoing connection to an adjacent square whether that square has a matching connection. Return the number of mismatches.

```
CL-USER(335): (count-loose-ends (read-track "~cs188/code-188/search/domains/random44.track"))
20
CL-USER(336): (count-loose-ends (read-track "~cs188/code-188/search/domains/weak44.track"))
0
CL-USER(336): (count-loose-ends (read-track "~cs188/code-188/search/domains/strong44.track"))
0
CL-USER(338): (print-track (setq random44 (make-random-track-unlimited 4 4)))
```

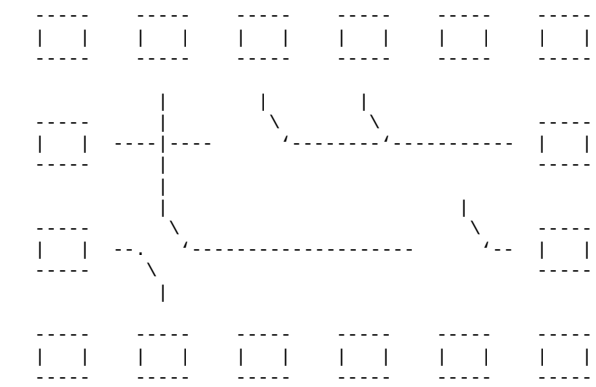


```
NIL
CL-USER(339): CL-USER(339): (count-loose-ends random44)
25
CL-USER(340): (print-track (setq random22 (make-random-track-unlimited 2 2)))
```

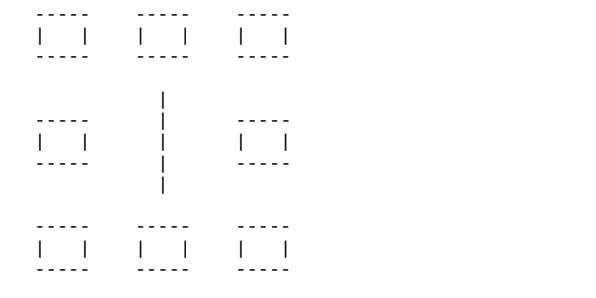


```
NIL
CL-USER(341): (count-loose-ends random22)
7
```

```
CL-USER(342): (print-track (setq random42 (make-random-track-unlimited 4 2)))
```



```
NIL
CL-USER(343): (count-loose-ends random42)
11
CL-USER(344): (print-track (setq random11 (make-random-track-unlimited 1 1)))
```



```
NIL
CL-USER(345): (count-loose-ends random11)
2
```

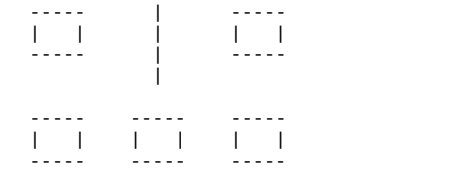
2. If a track has no loose ends, what does that imply about the numbers of lsplit and rsplit pieces?

   #lsplit-piece and #rsplit-piece have an odd number of connectors while every other piece has an even number of connectors. A necessary condition for a track to have no loose ends is that the sum of connectors of all the pieces be even; this is because each connection between pieces consumes exactly two connectors. Therefore, (#lsplit-piece + #rsplit-piece) must be even.

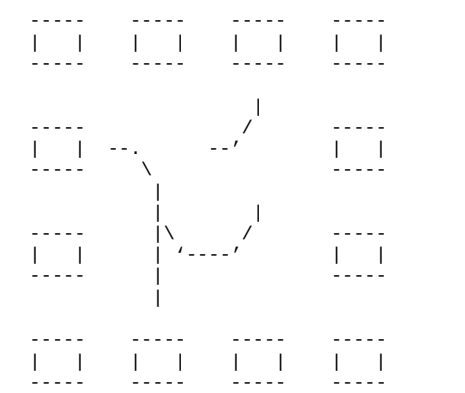
3. Now, write a function (count-wccs track), which should return the number of WCCs in the track.

   The function `count-wccs` returns the number of weakly connected components. It uses an array of marks, indicating for each edge which wcc it belongs to. It works by finding an edge (with a track connection) that has not yet been marked; this must be part of a new wcc. It then recursively explores the wcc (calling `mark-wcc`), following both links within the piece and the corresponding edge on the adjacent piece, if any.

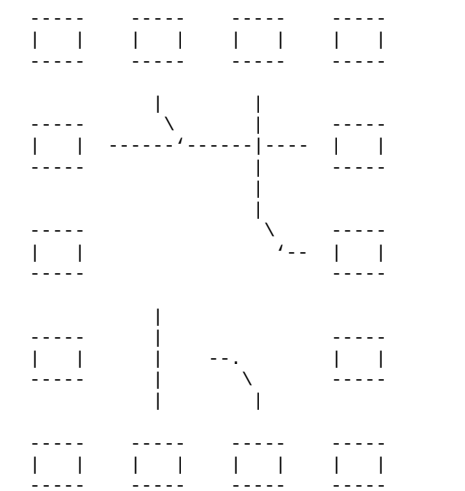
```
CL-USER(335): (count-wccs (read-track "~cs188/code-188/search/domains/random44.track"))
6
CL-USER(335): (count-wccs (read-track "~cs188/code-188/search/domains/weak44.track"))
1
CL-USER(335): (count-wccs (read-track "~cs188/code-188/search/domains/strong44.track"))
1
CL-USER(348): (print-track (setq random11 (make-random-track-unlimited 1 1)))
```



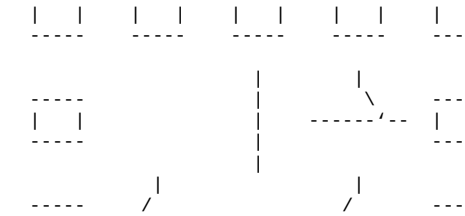
```
NIL
CL-USER(349): (count-wccs random11)
1
CL-USER(350): (print-track (setq random22 (make-random-track-unlimited 2 2)))
```

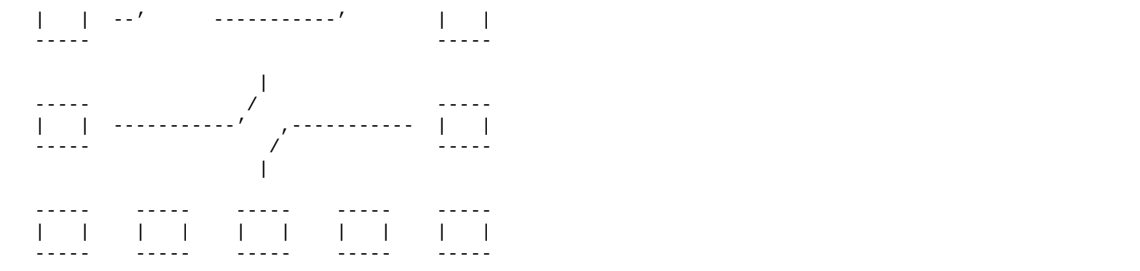


```
NIL
CL-USER(351): (count-wccs random22)
2
CL-USER(352): (print-track (setq random23 (make-random-track-unlimited 2 3)))
```

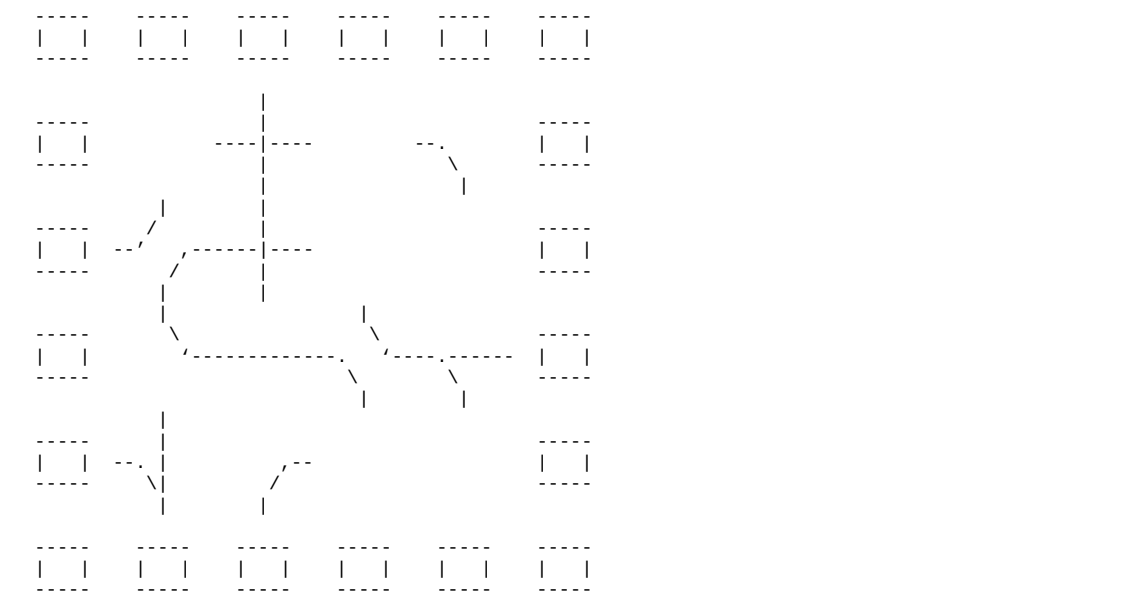


```
NIL
CL-USER(353): (count-wccs random23)
4
CL-USER(354): (print-track (setq random33 (make-random-track-unlimited 3 3)))
```

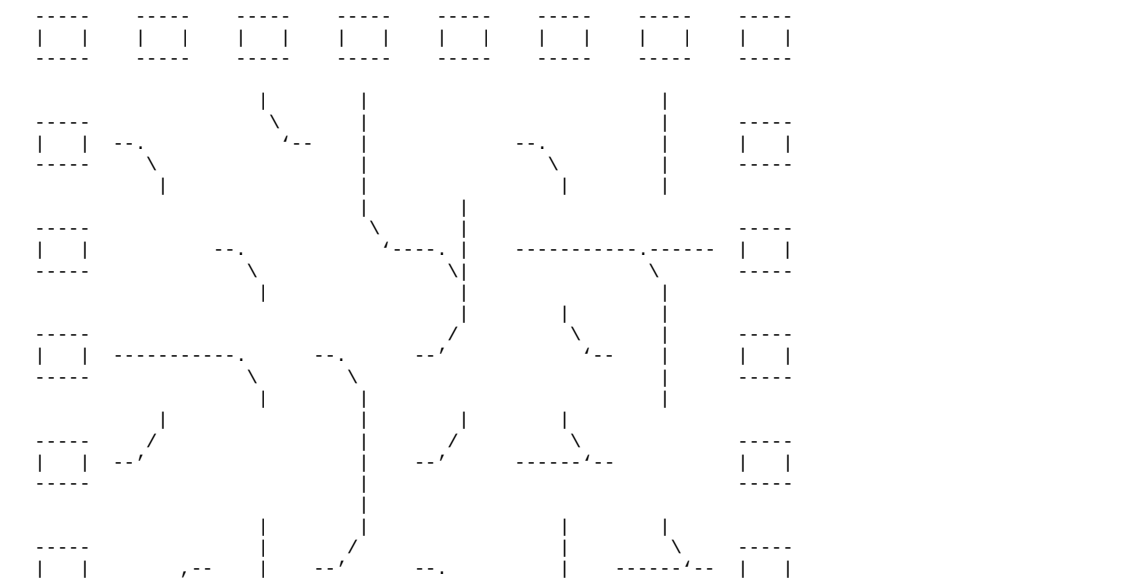


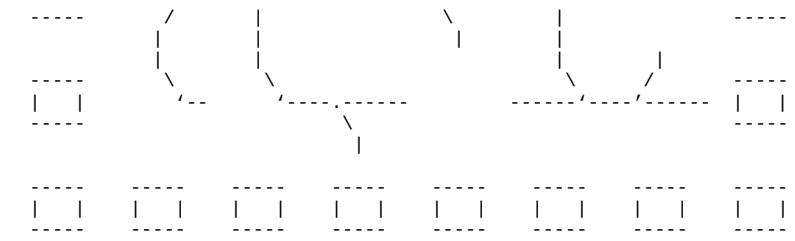


```
NIL
CL-USER(355): (count-wccs random33)
6
CL-USER(356): (print-track (setq random44 (make-random-track-unlimited 4 4)))
```



```
NIL
CL-USER(357): (count-wccs random44)
8
CL-USER(358): (print-track (setq random66 (make-random-track-unlimited 6 6)))
```



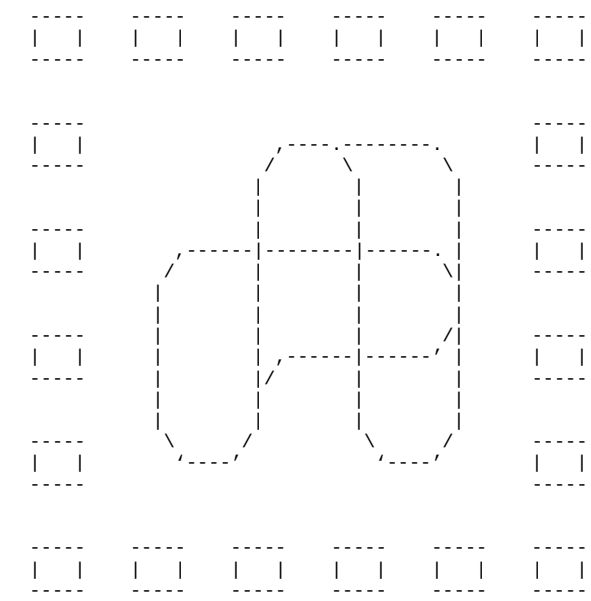


```
NIL
CL-USER(359): (count-wccs random66)
18
```

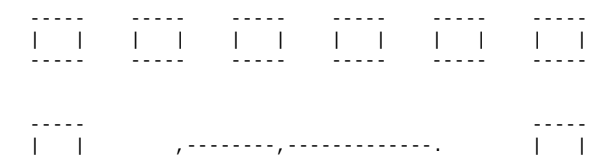
4. Now, formulate and solve the local search problem type, unlimited-track-local-problem, where the goal is to construct a track with as few loose ends and WCCs as possible. In such problems, we start with a randomly filled track and make changes to improve it, with no limitations on the pieces that can be used. You will need to define the actions or random-action method and the result, h-cost and goal-test methods. Then construct a suitable problem instance with a random initial state and apply a suitable algorithm from local-search.lisp.

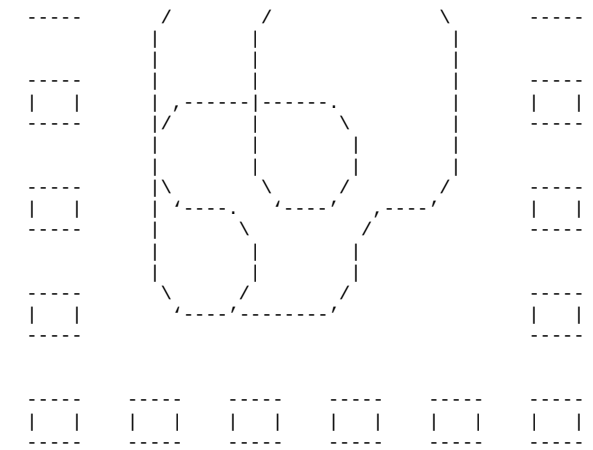
   First, the h-cost function to be minimized is defined as `(loose ends + wccs - 1)`, so that goal state have value 0, but this is just a convention. Other functions are also possible (e.g. `(2*loose ends + wccs - 1)`), but they still need to minimize both *loose ends* and *wccs*, and to have their minimum at the goal states. The neighborhood structure of the space is huge but it is never made explicit. Instead, it is defined by the `random-action` method, which generates an action that randomly selects a square and replaces the piece with a randomly chosen new piece in a random orientation. The action is executed by the `result` method, which simply applies the action (interpreted as a function) to a copy of the state.

   Here is a sample result for a 4×4 track, which took 429 steps (with simulated annealing; parameters details explained below):

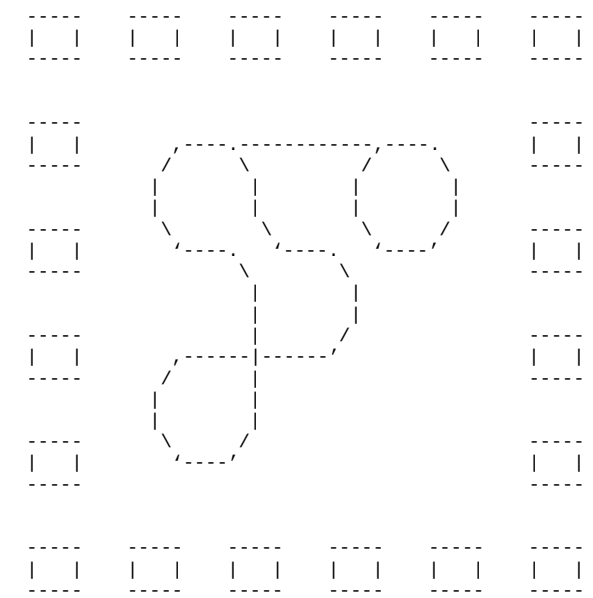


   Here is another one for a 4×4 track, which took 399 steps:

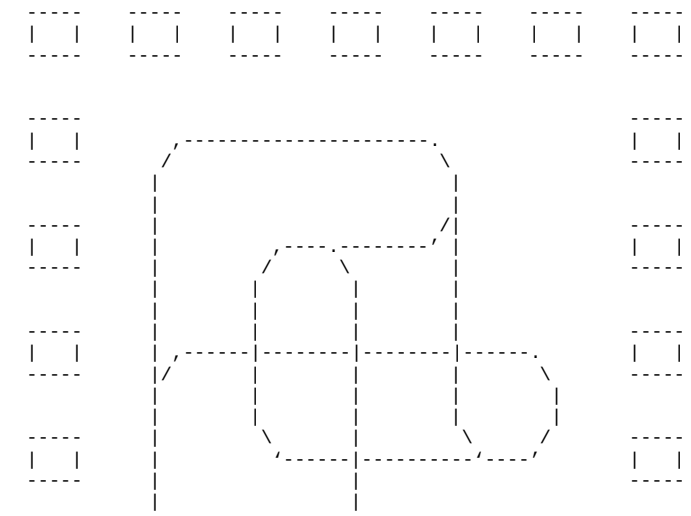


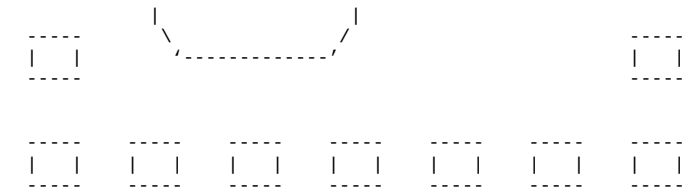


And this one (again for a 4×4 track) took 2182 steps:

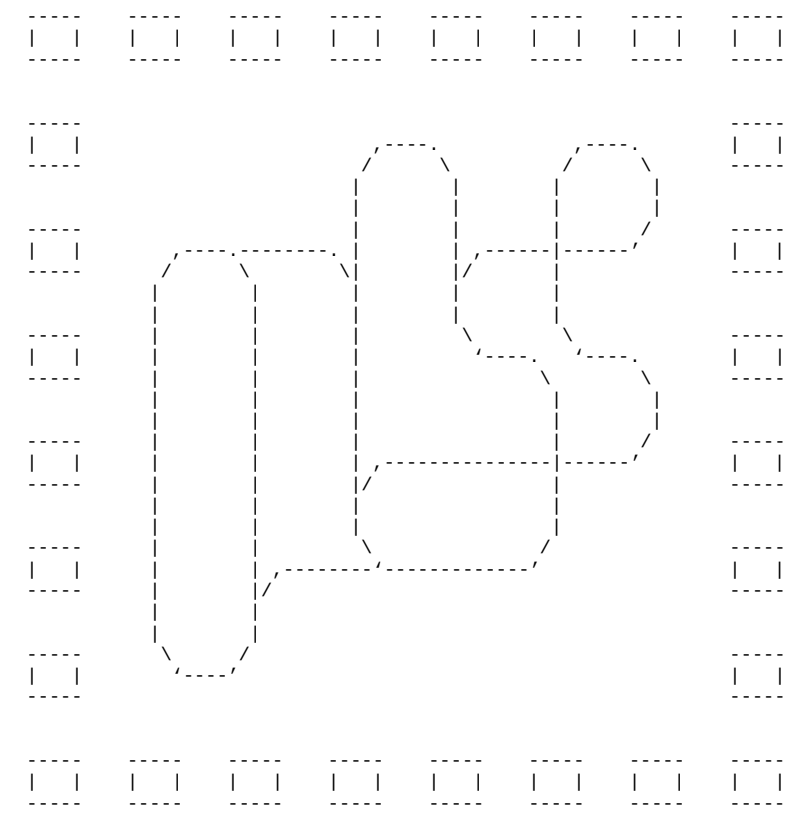


Sample result for a 5×5 track, which took 2468 steps:



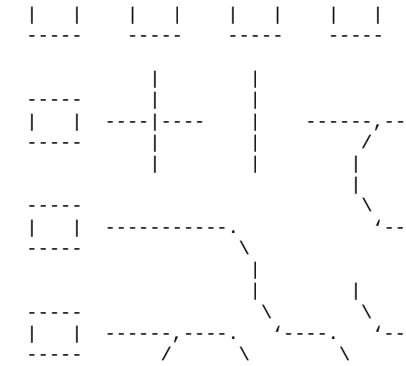


Sample result for a 6×6 track, which took 1654 steps:

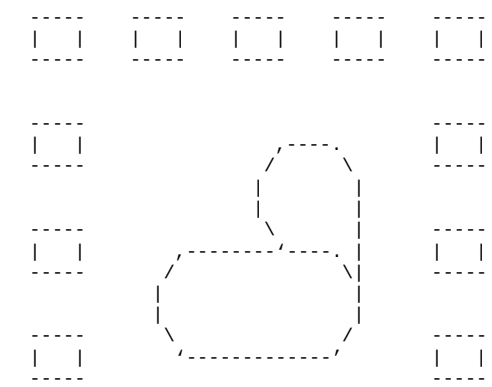


Now, let us look at some example executions for a 3×3 track follow. The `simulated-annealing-search` function has been modified to print the iteration number `t`, the h-cost `h`, as defined above, and the current temperature `d`. These are printed each 20 steps as well as at the final step after a solution has been found.

```
CL-USER(31): (print-track (setq t33 (make-random-track-unlimited 3 3)))
```

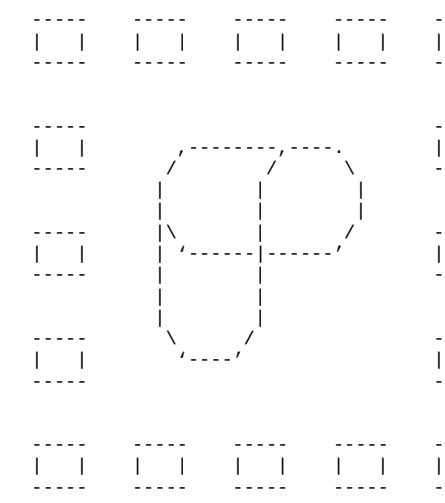


```
CL-USER(32): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

CL-USER(33):
t=20 h=11 d=0.29701495
t=40 h=8 d=0.2940596
t=60 h=4 d=0.29113367
t=80 h=4 d=0.28823686
t=100 h=4 d=0.28536886
t=120 h=3 d=0.28252938
t=140 h=3 d=0.27971816
t=160 h=3 d=0.27693492
t=180 h=2 d=0.27417937
t=200 h=1 d=0.27145123
t=220 h=1 d=0.26875025
t=240 h=1 d=0.26607615
t=260 h=1 d=0.26342866
t=280 h=1 d=0.26080748
t=300 h=1 d=0.25821242
t=320 h=1 d=0.25564313
t=340 h=1 d=0.25309947
t=360 h=1 d=0.2505811
t=369 h=0 d=0.249456
```

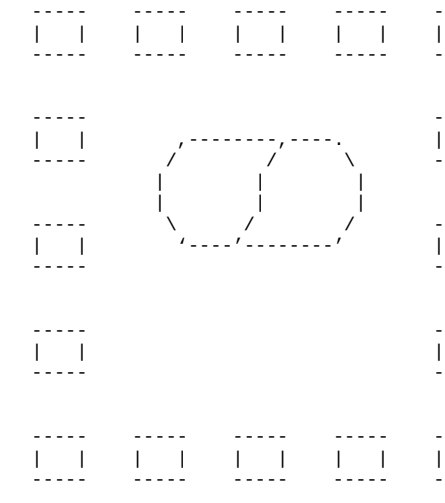



```
NIL

CL-USER(34): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=10 d=0.29701495
t=40 h=9 d=0.2940596
t=60 h=5 d=0.29113367
t=80 h=5 d=0.28823686
t=100 h=2 d=0.28536886
t=120 h=2 d=0.28252938
t=140 h=2 d=0.27971816
t=160 h=2 d=0.27693492
t=180 h=2 d=0.27417937
t=200 h=2 d=0.27145123
t=220 h=1 d=0.26875025
t=240 h=1 d=0.26607615
t=260 h=1 d=0.26342866
t=280 h=1 d=0.26080748
t=300 h=1 d=0.25821242
t=320 h=1 d=0.25564313
t=340 h=1 d=0.25309947
t=360 h=1 d=0.2505811
t=380 h=1 d=0.24808775
t=400 h=1 d=0.24561924
```

```
t=420 h=1 d=0.24317528
t=440 h=1 d=0.24075565
t=460 h=1 d=0.23836009
t=480 h=1 d=0.23598836
t=481 h=0 d=0.23587039
```

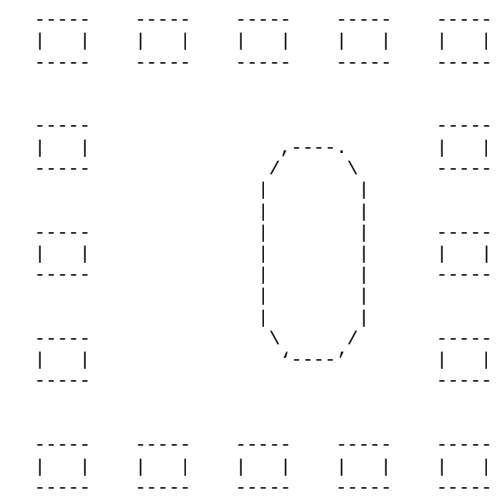


```
NIL

CL-USER(35): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=10 d=0.29701495
t=40 h=9 d=0.2940596
t=60 h=6 d=0.29113367
t=80 h=5 d=0.28823686
t=100 h=5 d=0.28536886
t=120 h=5 d=0.28252938
t=140 h=5 d=0.27971816
t=160 h=5 d=0.27693492
t=180 h=2 d=0.27417937
t=200 h=2 d=0.27145123
t=220 h=2 d=0.26875025
t=240 h=2 d=0.26607615
t=260 h=2 d=0.26342866
t=280 h=3 d=0.26080748
t=299 h=0 d=0.25834152
```



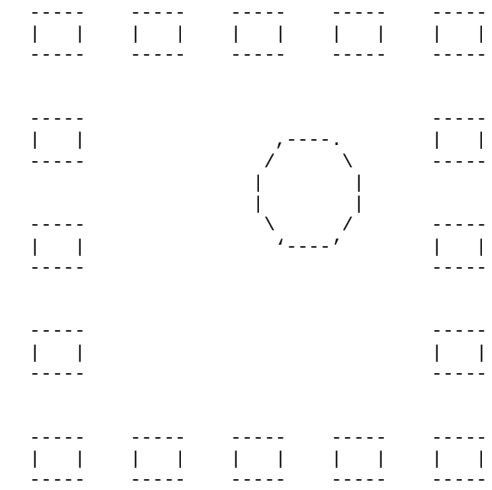
```
NIL

CL-USER(36): (print-track (simulated-annealing-search
```

```
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=12 d=0.29701495
t=40 h=12 d=0.2940596
t=60 h=8 d=0.29113367
t=80 h=8 d=0.28823686
t=100 h=5 d=0.28536886
t=120 h=5 d=0.28252938
t=140 h=3 d=0.27971816
t=160 h=2 d=0.27693492
t=180 h=3 d=0.27417937
t=200 h=3 d=0.27145123
t=220 h=2 d=0.26875025
t=240 h=2 d=0.26607615
t=260 h=2 d=0.26342866
t=280 h=2 d=0.26080748
t=300 h=2 d=0.25821242
t=320 h=2 d=0.25564313
t=340 h=2 d=0.25309947
t=360 h=1 d=0.2505811
t=380 h=1 d=0.24808775
t=400 h=1 d=0.24561924
t=420 h=1 d=0.24317528
t=440 h=1 d=0.24075565
t=460 h=1 d=0.23836009
t=480 h=1 d=0.23598836
t=500 h=1 d=0.23364024
t=520 h=1 d=0.23131548
t=540 h=1 d=0.22901386
t=551 h=0 d=0.22775774
```

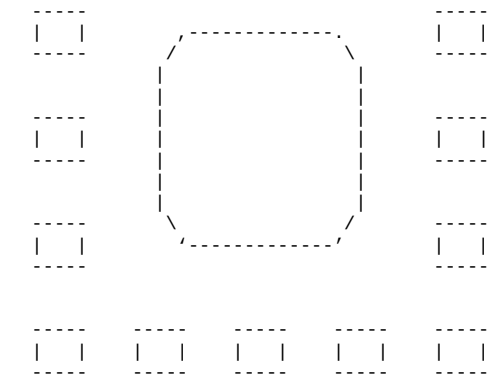


```
NIL

CL-USER(37): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=12 d=0.29701495
t=40 h=12 d=0.2940596
t=60 h=7 d=0.29113367
t=80 h=7 d=0.28823686
t=100 h=7 d=0.28536886
t=120 h=7 d=0.28252938
t=140 h=7 d=0.27971816
t=160 h=7 d=0.27693492
t=180 h=7 d=0.27417937
t=200 h=7 d=0.27145123
t=220 h=6 d=0.26875025
t=240 h=6 d=0.26607615
t=260 h=5 d=0.26342866
t=280 h=1 d=0.26080748
t=300 h=1 d=0.25821242
t=320 h=1 d=0.25564313
t=340 h=1 d=0.25309947
t=360 h=1 d=0.2505811
```

```
t=380 h=1 d=0.24808775
t=400 h=3 d=0.24561924
t=420 h=2 d=0.24317528
t=440 h=2 d=0.24075565
t=460 h=2 d=0.23836009
t=480 h=2 d=0.23598836
t=500 h=2 d=0.23364024
t=520 h=1 d=0.23131548
t=540 h=1 d=0.22901386
t=556 h=0 d=0.22718905
```

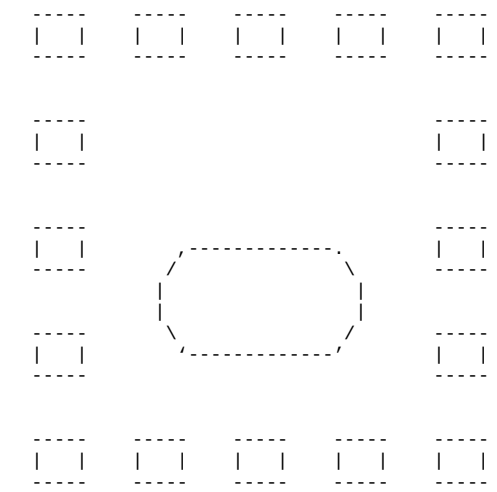


```
NIL

CL-USER(38): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=12 d=0.29701495
t=40 h=7 d=0.2940596
t=60 h=4 d=0.29113367
t=80 h=4 d=0.28823686
t=100 h=3 d=0.28536886
t=120 h=3 d=0.28252938
t=140 h=3 d=0.27971816
t=160 h=2 d=0.27693492
t=180 h=2 d=0.27417937
t=200 h=2 d=0.27145123
t=220 h=2 d=0.26875025
t=240 h=2 d=0.26607615
t=260 h=2 d=0.26342866
t=280 h=1 d=0.26080748
t=300 h=1 d=0.25821242
t=320 h=1 d=0.25564313
t=340 h=1 d=0.25309947
t=360 h=1 d=0.2505811
t=380 h=1 d=0.24808775
t=400 h=1 d=0.24561924
t=420 h=1 d=0.24317528
t=440 h=1 d=0.24075565
t=460 h=1 d=0.23836009
t=480 h=1 d=0.23598836
t=500 h=1 d=0.23364024
t=520 h=1 d=0.23131548
t=540 h=1 d=0.22901386
t=560 h=1 d=0.22673513
t=580 h=1 d=0.22447906
t=600 h=1 d=0.22224547
t=620 h=1 d=0.2200341
t=634 h=0 d=0.21849923
```



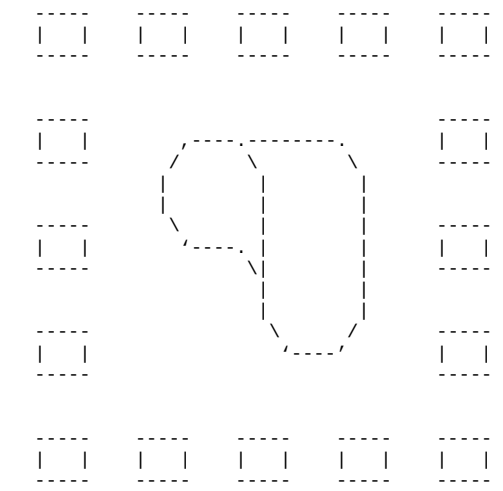
NIL

```
CL-USER(39): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=16 d=0.29701495
t=40 h=13 d=0.2940596
t=60 h=9 d=0.29113367
t=80 h=8 d=0.28823686
t=100 h=9 d=0.28536886
t=120 h=4 d=0.28252938
t=140 h=4 d=0.27971816
t=160 h=4 d=0.27693492
t=180 h=3 d=0.27417937
t=200 h=2 d=0.27145123
t=220 h=2 d=0.26875025
t=240 h=2 d=0.26607615
t=260 h=2 d=0.26342866
t=280 h=2 d=0.26080748
t=300 h=3 d=0.25821242
t=320 h=2 d=0.25564313
t=338 h=0 d=0.25335267
```

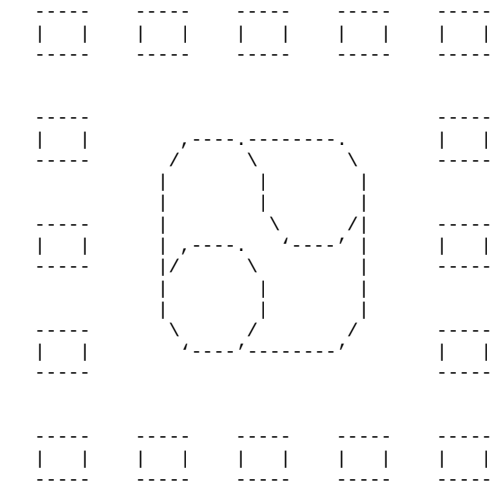


NIL

```
CL-USER(40): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=10 d=0.29701495
t=40 h=8 d=0.2940596
t=60 h=5 d=0.29113367
t=80 h=1 d=0.28823686
t=95 h=0 d=0.28608316
```

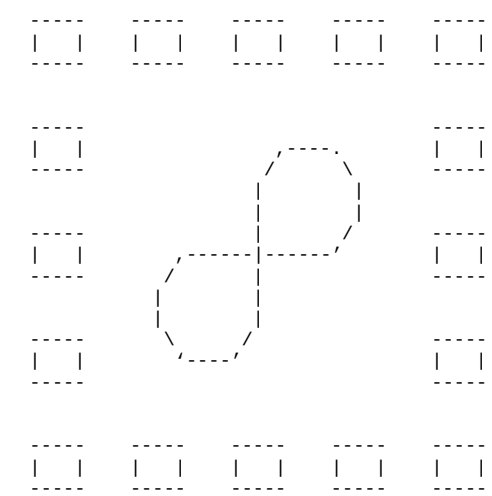


```
NIL

CL-USER(41): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=13 d=0.29701495
t=40 h=5 d=0.2940596
t=60 h=4 d=0.29113367
t=80 h=4 d=0.28823686
t=100 h=3 d=0.28536886
t=120 h=3 d=0.28252938
t=140 h=2 d=0.27971816
t=160 h=2 d=0.27693492
t=180 h=2 d=0.27417937
t=200 h=2 d=0.27145123
t=220 h=2 d=0.26875025
t=240 h=1 d=0.26607615
t=260 h=1 d=0.26342866
t=280 h=1 d=0.26080748
t=300 h=1 d=0.25821242
t=320 h=1 d=0.25564313
t=340 h=1 d=0.25309947
t=360 h=1 d=0.2505811
t=380 h=1 d=0.24808775
t=400 h=1 d=0.24561924
t=416 h=0 d=0.24366212
```



```
NIL

CL-USER(42): (print-track (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
```

```
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))

t=20 h=11 d=0.29701495
t=40 h=12 d=0.2940596
t=60 h=13 d=0.29113367
t=80 h=13 d=0.28823686
t=100 h=13 d=0.28536886
t=120 h=10 d=0.28252938
t=140 h=8 d=0.27971816
t=160 h=8 d=0.27693492
t=180 h=8 d=0.27417937
t=200 h=3 d=0.27145123
t=220 h=3 d=0.26875025
t=240 h=3 d=0.26607615
t=260 h=3 d=0.26342866
t=280 h=3 d=0.26080748
t=300 h=3 d=0.25821242
t=320 h=3 d=0.25564313
t=340 h=3 d=0.25309947
t=360 h=3 d=0.2505811
t=380 h=3 d=0.24808775
t=400 h=3 d=0.24561924
t=420 h=3 d=0.24317528
t=440 h=3 d=0.24075565
t=460 h=3 d=0.23836009
t=480 h=3 d=0.23598836
t=500 h=2 d=0.23364024
t=504 h=0 d=0.23317343
```



```
NIL
```

We can see that the h-cost is sort of monotonically decreasing (at least when looking once per 20 steps). Of course, this is not always the case. As we already mentioned above, the local search algorithm used in the examples is simulated annealing. Note that it will work only with an appropriate schedule, i.e. we need more than just executing the following code:

```
(simulated-annealing-search (make-unlimited-track-local-problem
:initial-state t33))
```

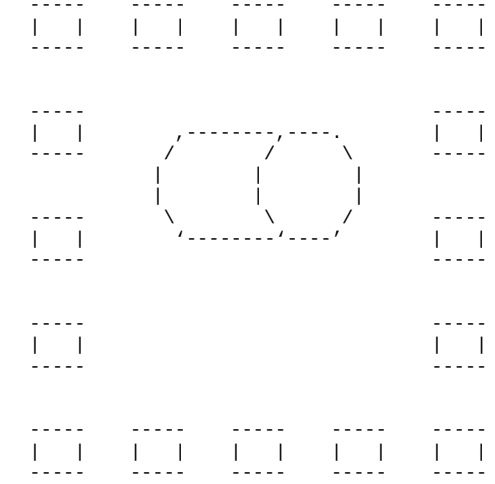
By default, the function `simulated-annealing-search` uses an exponential schedule (see the function `make-exp-schedule` in `local-search.lisp`) with parameters $k = 20$, $\lambda = 0.005$ and $limit = 100$. These are used to calculate the temperature according to the formula: $ke^{-\lambda t}$, where t is the time (the number of the present iteration). The major problem is caused by the last value, which specifies the number of iterations, as in for our purposes the search normally takes much longer than 100 steps. Experiments showed that while the exponential schedule is good to use, it needs different parameters, e.g. $k = 0.3$, $\lambda = 0.0005$ and $limit = 10000$:

```
(simulated-annealing-search (make-unlimited-track-local-problem
```

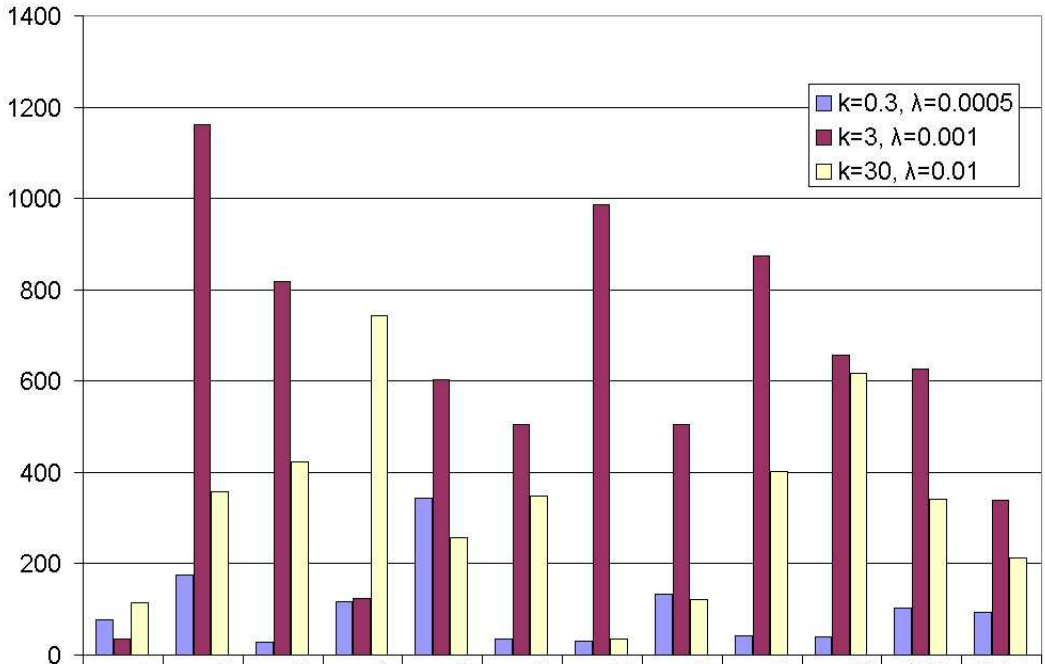
```
:initial-state t33) :schedule (make-exp-schedule :k 0.3 :lambda
0.0005 :limit 10000))
```

Why these particular values? Since we have the freedom to perform arbitrary changes to a single square, we can expect to be able to easily escape most of the local minima. So, it would be better to prefer a more conservative schedule, which starts with a low temperature that cools slowly. Looking at the formula, $ke^{-\lambda t}$, we can see that this can be achieved with a small $k = 0.3$ (low initial temperature) and also a small $\lambda = 0.0005$ (slow cooling). Another alternative is to start with a much higher k, but to alow it to go down faster, e.g. $k = 30$ and $\lambda = 0.01$. Here is a sample test:

```
CG-USER(21): (simulated-annealing-search
(make-unlimited-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 30 :lambda 0.01 :limit 10000))

t=20 h=17 d=24.561924
t=40 h=19 d=20.1096
t=60 h=21 d=16.46435
t=80 h=27 d=13.47987
t=100 h=18 d=11.036384
t=120 h=18 d=9.035827
t=140 h=12 d=7.397909
t=160 h=8 d=6.056896
t=180 h=9 d=4.9589667
t=200 h=21 d=4.0600586
t=220 h=16 d=3.3240945
t=240 h=11 d=2.721539
t=260 h=10 d=2.2282076
t=280 h=7 d=1.824302
t=300 h=11 d=1.493612
t=320 h=13 d=1.2228664
t=340 h=12 d=1.0011982
t=360 h=10 d=0.81971174
t=380 h=10 d=0.67112315
t=400 h=9 d=0.5494692
t=420 h=7 d=0.4498674
t=440 h=6 d=0.36832017
t=460 h=3 d=0.3015551
t=480 h=3 d=0.24689248
t=500 h=2 d=0.20213841
t=520 h=2 d=0.16549696
t=540 h=2 d=0.13549742
t=560 h=1 d=0.11093592
t=580 h=1 d=0.09082667
t=600 h=1 d=0.07436257
t=620 h=1 d=0.060882933
t=626 h=0 d=0.05733739
```



As you can see, as the temperature is high at the beginning, the search goes almost randomly for the first 400 steps with the h-cost going up and down randomly. Only after the temperature is already low, real progress is observed. Compare this to the previous examples, where the h-cost has been going



	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	AVG	STD
k=0.3, λ=0.0005	77	175	27	117	344	36	30	133	42	40	102.10	93.97
k=3, λ=0.001	35	1162	818	123	604	505	986	505	873	656	626.70	338.78
k=30, λ=0.01	114	358	423	744	258	348	35	122	401	616	341.90	211.83

Figure 1: **Unlimited-track problem, track 2×2:** experiments with different values for k and λ. The number of steps needed to find a perfect solution for a 2×2 track (for 10 runs) is shown. The last two bars/columns show the average and the standard deviation.

down almost monotonically. One might think that neither of these schedules is good enough and it would be better to choose something in the middle, e.g. $k = 3$ and $\lambda = 0.001$. In fact, this happens to be a worse choice in practice, as shown on Figures 1, 2, 3 and 4. The first combination of k and λ is clearly best for 2×2. The explanation should be that for such a small track bad moves are unlikely to be necessary to achieve the goal state (see Figure 2). For a 3×3 tracks though the two extremes look almost equally good. While $k = 0.3, \lambda = 0.0005$ performs slightly better than $k = 30, \lambda = 0.01$, it has a higher standard deviation: it can find a solution fairly fast as it does not loose time with a lot of initial random moves, but that way it could also miss the opportunity to move to a better state before starting the optimization.

5. Next, formulate and solve the fixed-track-local-problem, where the track must be constructed from a fixed set of pieces that exactly fills the track. (Hence, actions can rotate or swap pieces, but cannot introduce new ones.) Experiment with various different track sizes and sets of pieces. Is this problem harder or easier than the unlimited problem? Why? Which kinds of pieces seem to cause difficulties?

   The solution is very similar to the unlimited problem, except that the available actions swap two existing pieces rather than making a new one. Random rotations of the two pieces are performed during the swap.

   Below are some experiments with 6 different sets of pieces for a 3×3 track. To study the effect of going to a larger track, the same pieces are used for a 4×4 track (adding 7 more #blank-piece). Three additional configurations are tried for the 4×4 track (see sets 7, 8 and 9 below). Each set of pieces is chosen in a way that there exists an exact solution. Some sample runs for each of the sets follow, which give a visual idea of the kind of pieces used in the set.

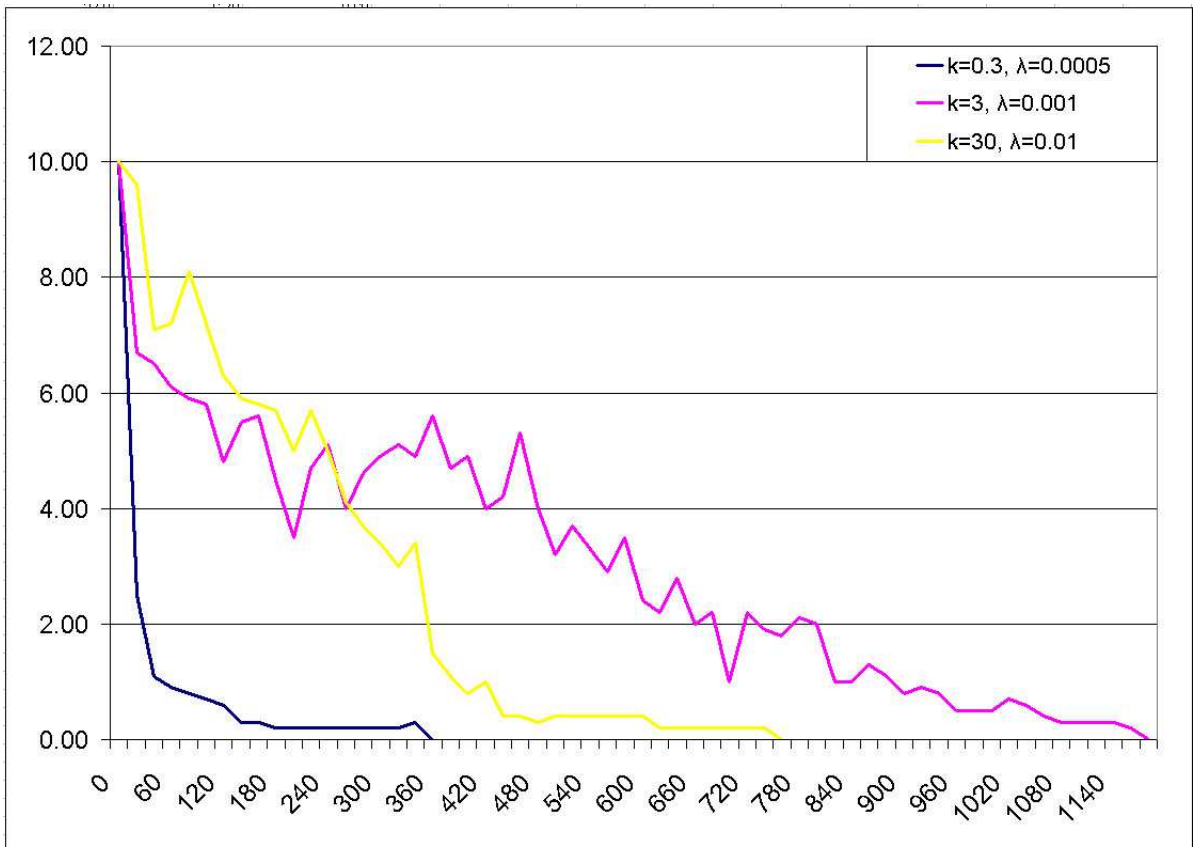
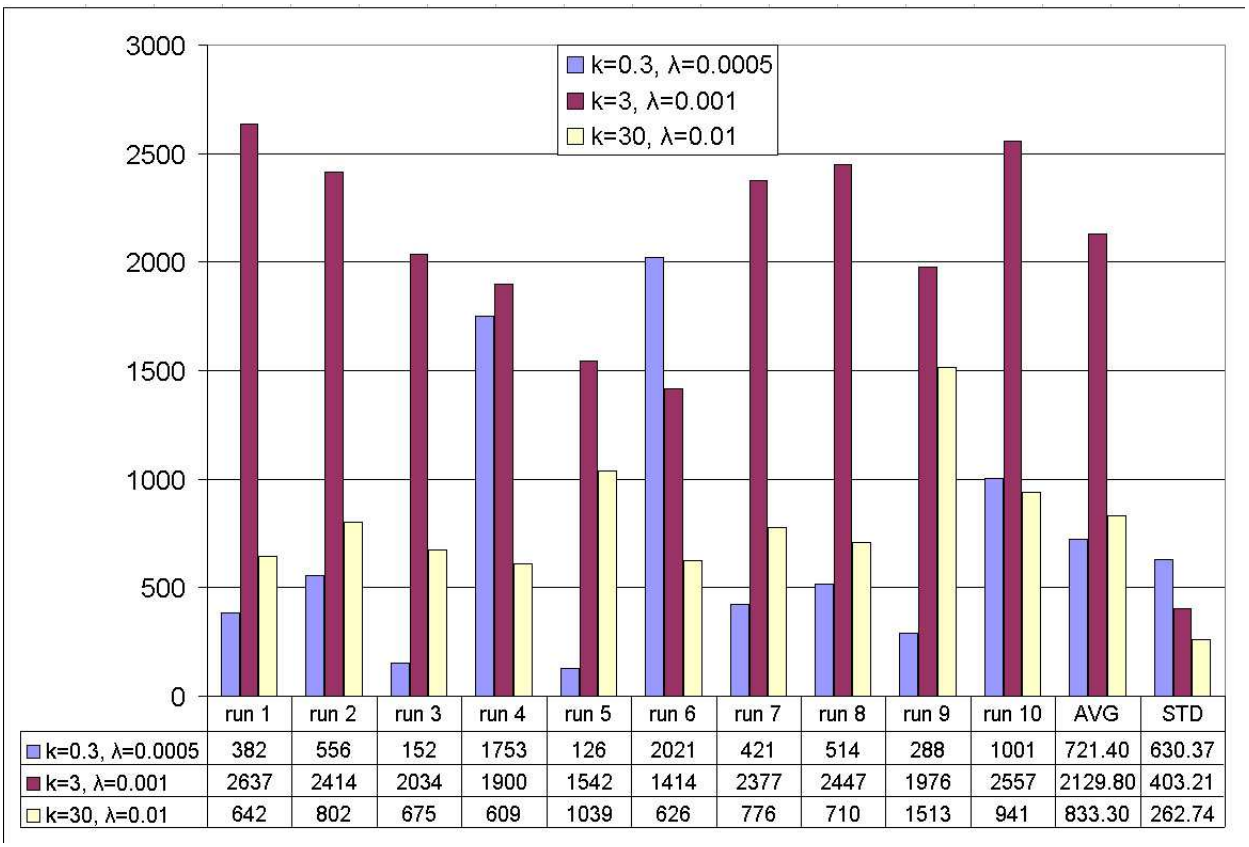


Figure 2: **Unlimited-track problem, track** 2×2**:** experiments with 3 different values for k and λ. The average value of the h-cost (over 10 runs) as a function of time is shown.



	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	AVG	STD
k=0.3, λ=0.0005	382	556	152	1753	126	2021	421	514	288	1001	721.40	630.37
k=3, λ=0.001	2637	2414	2034	1900	1542	1414	2377	2447	1976	2557	2129.80	403.21
k=30, λ=0.01	642	802	675	609	1039	626	776	710	1513	941	833.30	262.74

Figure 3: **Unlimited-track problem, track 3×3:** experiments with different values for k and λ. The number of steps needed to find a perfect solution for a 3×3 track (for 10 runs) is shown. The last two bars/columns show the average and the standard deviation.

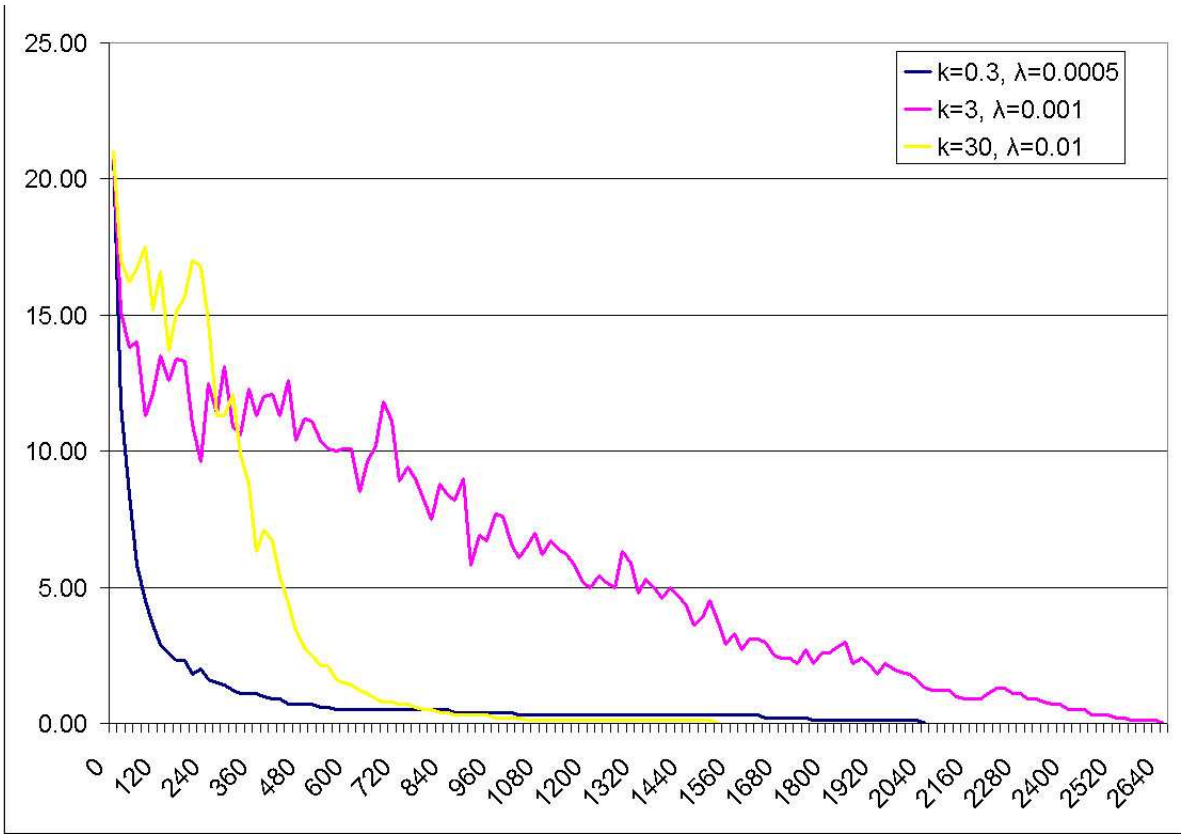
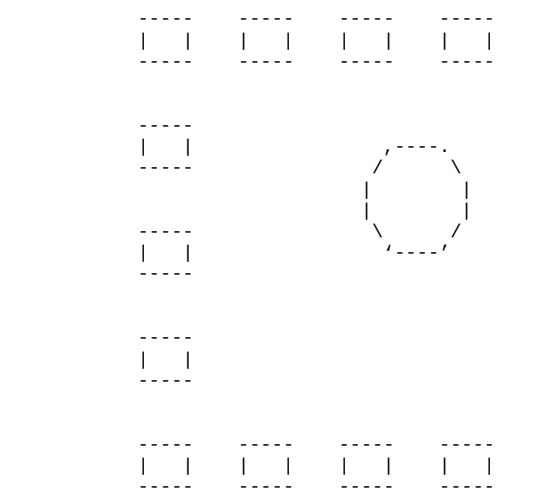


Figure 4: **Unlimited-track problem, track** 3×3**:** experiments with 3 different values for k and λ. The average value of the h-cost (over 10 runs) as a function of time is shown.

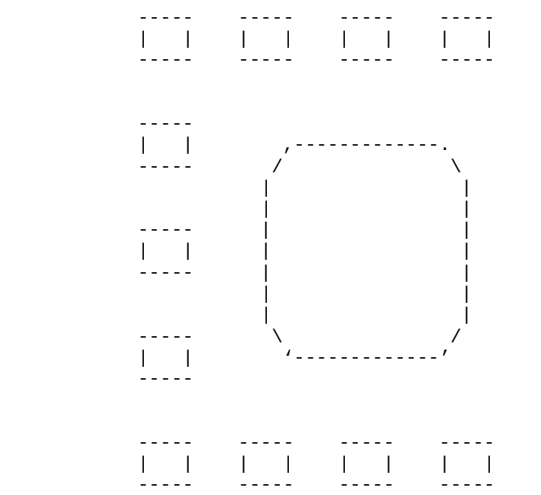
- 3×3 track
  (a) Set 1

```
CL-USER(5): (setf pieces '((curve-piece . 4) (blank-piece . 5)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



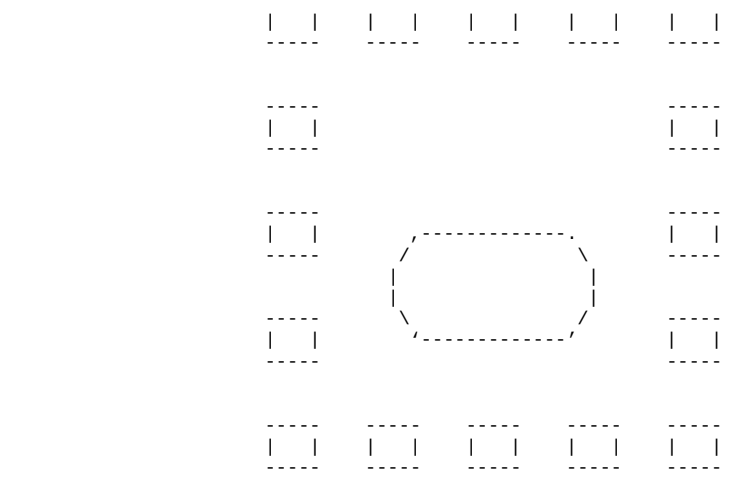
  (b) Set 2

```
CL-USER(5): (setf pieces '((curve-piece . 4)
(blank-piece . 1) (straight-piece . 4)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



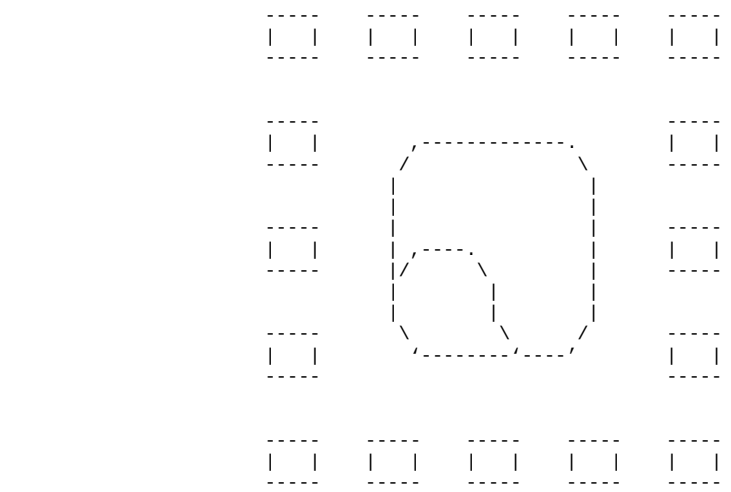
  (c) Set 3

```
CL-USER(5): (setf pieces '((curve-piece . 4)
(blank-piece . 3) (straight-piece . 2)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



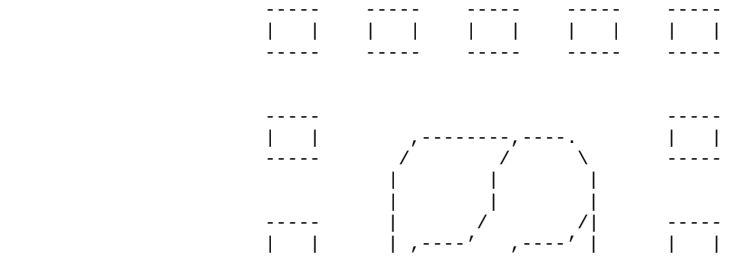
(d) Set 4

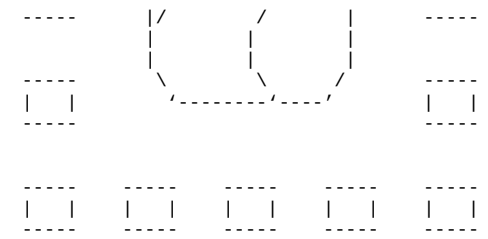
```
CL-USER(5): (setf pieces '((curve-piece . 5)
(straight-piece . 2) (rsplit-piece . 2)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



(e) Set 5

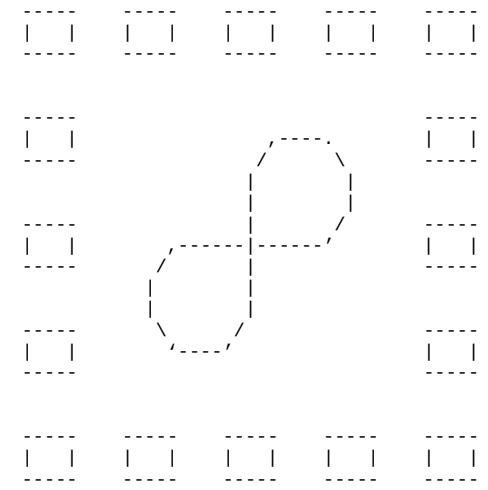
```
CL-USER(5): (setf pieces '((curve-piece . 4) (rsplit-piece . 3)
(lsplit-piece . 1) (twocurve-piece . 1)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```





(f) Set 6

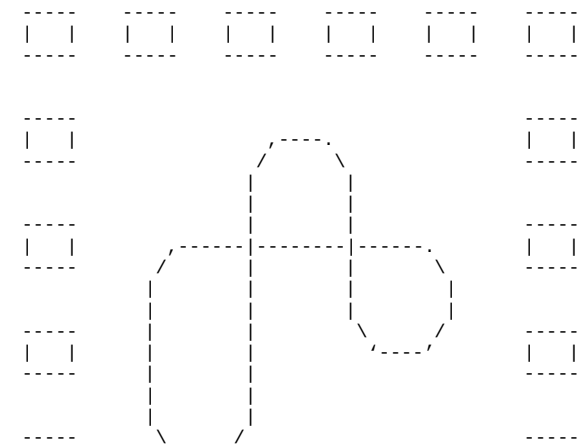
```
CL-USER(5): (setf pieces '((curve-piece . 6)
(cross-piece . 1) (blank-piece . 2)))
...
CL-USER(6): (setf t33 (make-random-track-fixed 3 3 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t33) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```

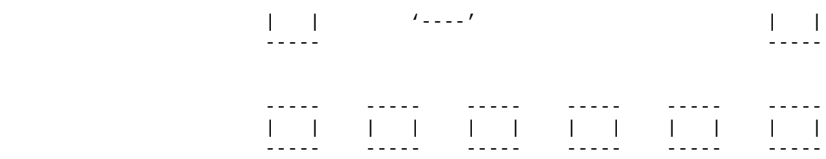


- 4×4 track The above six and the following additional ones:

(a) Set 7

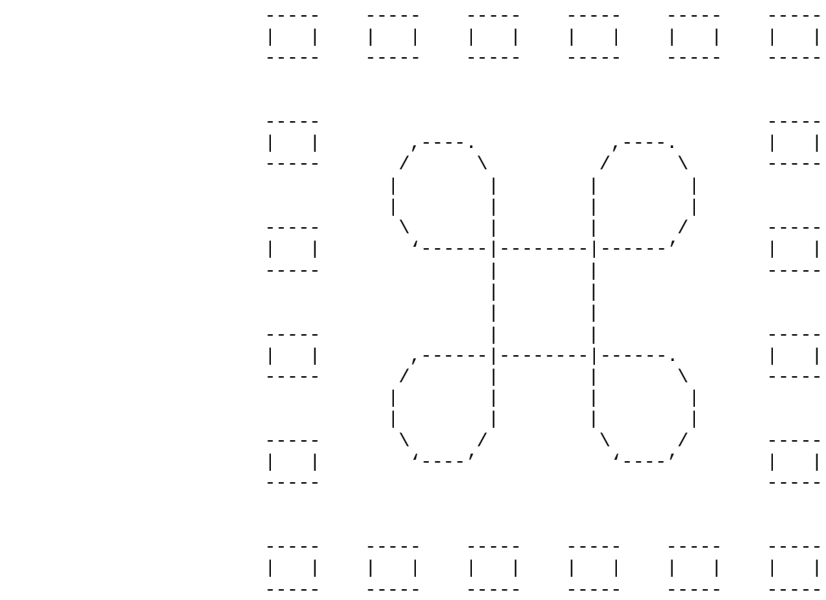
```
CL-USER(5): (setf pieces '((curve-piece . 8) (cross-piece . 2)
(straight-piece . 2) (blank-piece . 4)))
...
CL-USER(6): (setf t44(make-random-track-fixed 4 4 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t44) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```





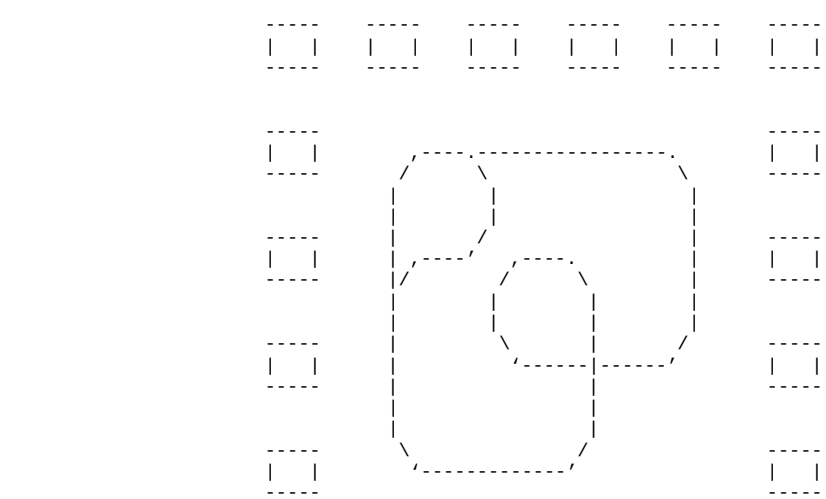
(b) Set 8

```
CL-USER(5): (setf pieces '((curve-piece . 12) (cross-piece . 4)))
...
CL-USER(6): (setf t44(make-random-track-fixed 4 4 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t44) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



(c) Set 9

```
CL-USER(5): setf pieces '((curve-piece . 7) (cross-piece . 1)
(rsplit-piece . 2) (straight-piece . 4) (blank-piece . 1)
(twocurve-piece . 1)))
...
CL-USER(6): (setf t44(make-random-track-fixed 4 4 pieces)))
...
CL-USER(7): (print-track (simulated-annealing-search
(make-fixed-track-local-problem :initial-state t44) :schedule
(make-exp-schedule :k 0.3 :lambda 0.0005 :limit 10000)))
```



	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	Average	StdDev
Set 1	1158	15	398	366	382	148	497	815	807	163	474.90	336.92
Set 2	**2000**	193	1430	829	563	**2000**	293	335	**2000**	1674	1131.70	726.59
Set 3	1775	568	820	226	171	60	**2000**	**2000**	800	200	862.00	740.21
Set 4	894	554	1403	210	412	1120	1959	486	1464	1045	954.70	523.21
Set 5	654	375	411	2455	270	280	381	319	887	227	625.90	639.01
Set 6	1165	69	880	305	835	163	287	220	494	140	455.80	356.38

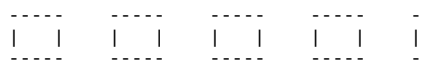
Table 1: **Fixed-track-local problem, track 3×3:** the number of steps for 10 runs to find a solution for 6 different piece sets. The last two columns show the average and the standard deviation over the 10 runs. The maximum number of iterations performed was 2,000.

	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	Average	StdDev
Set 1	3728	1064	1627	1924	1786	336	420	470	2530	1599	1548.40	1003.06
Set 2	**10000**	1329	9424	8540	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	8929.30	2572.66
Set 3	896	744	3612	7235	5335	2315	2292	**10000**	2308	7346	4208.30	2966.74
Set 4	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**	**10000**.00	0.00
Set 5	9305	**10000**	**10000**	**10000**	8533	**10000**	**10000**	**10000**	2702	**10000**	9054.00	2166.72
Set 6	3607	2453	210	803	1689	3762	3607	3133	1428	4318	2501.00	1329.75
Set 7	**10000**	**10000**	**10000**	1364	**10000**	9879	**10000**	**10000**	2333	9626	8320.20	3245.01
Set 8	566	1121	1128	878	1261	1278	971	1512	**10000**	1183	1989.80	2681.03
Set 9	3403	**10000**	3597	**10000**	9848	4456	**10000**	**10000**	3224	8994	7352.20	3034.73

Table 2: **Fixed-track-local problem, track 4×4:** the number of steps for 10 runs to find a solution for 9 different piece sets. The last two columns show the average and the standard deviation over the 10 runs. The maximum number of iterations performed was 10,000.

The results of these experiments ($k = 0.3, \lambda = 0.0005$) are shown in Tables 1, 2 and Figures 5, 6. Comparing Figures 3 and 5 (and also Figure 4 and Table 1) we see that the fixed track is harder than the unlimited one. The hardest sets for the 3×3 track are 2 and 4. Set 2 is hard as it has only one solution[1], which requires everything to be put together in a particular way. The solution is global and any move that destroys it (and thus, any move that leads to it) has a score of 5, which is more than, e.g. the score of 3, achieved for an "ellipse" (like in the solution for set 3) and two additional joined straight pieces. A similar argument holds for sets 4 and 3. In the latter case one can construct a small circle and additional two joined straight pieces.

Things get even worse for the 4×4 track, where simulated annealing (with the chosen schedule) is unable to find the exact solution for 10,000 steps (see sets 4,5,7,9 in Table 2). Interestingly, the sets 4 and 5 get much harder (set 4 never gets solved within 10,000 steps for our 10 runs). Having a fixed set of pieces severely limits the set of possible solutions and thus the probability to achieve one. In the unlimited set problem, escaping a local minimum is much easier and is often possible by introducing or removing[2] an appropriate piece. But here this requires one or more "bad" moves. Further, the larger the track, the harder to pick the right "bad" move (and even harder if a sequence of "bad" moves is needed). Here is a sample hard to escape local minimum for set 9:



[1] Strictly speaking, there are more as we can swap some elements and obtain the same solution.

[2] I.e. putting a #blank-piece

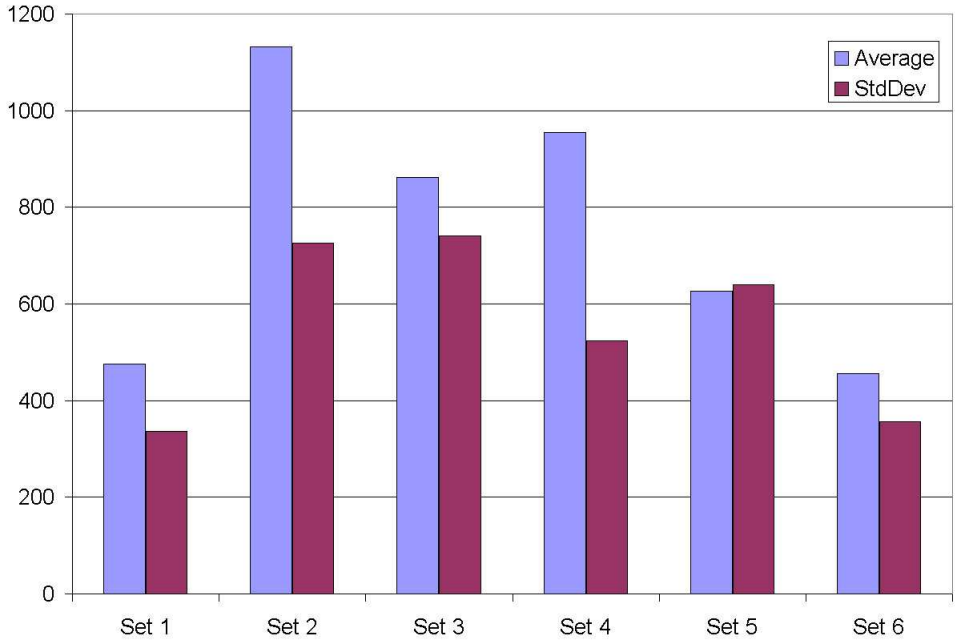
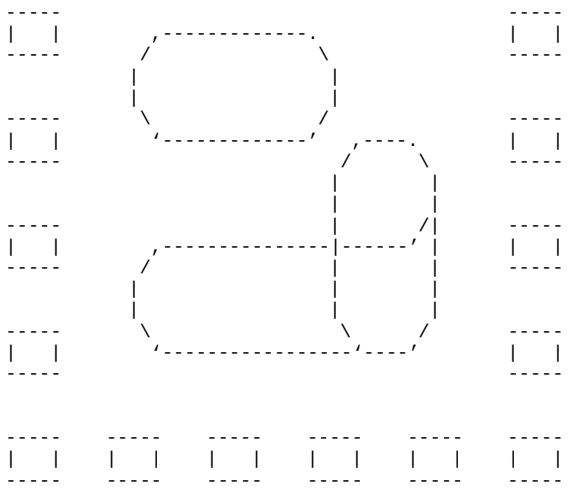


Figure 5: **Fixed-track-local problem, track** 3×3**:** the average number of steps (over 10 runs) to find a solution (and the standard deviation) for 6 different piece sets.



In general[3], in looks like the most problematic pieces are #twocurve-piece and #cross-piece, which usually require to be put in a limited number of positions in any possible solution (e.g. cannot be adjacent to a wall in an exact solution). The #straight-piece also looks problematic as it connects parts from two distant non-adjacent squares in a line, which is a limitation ()4e.g. compared with a #curve-piece.

6. Now, formulate and solve the no-loose-ends, unlimited track design problem as a CSP. (For now, ignore WCCs.) Do this by the following two methods:

[3] And in the last example.

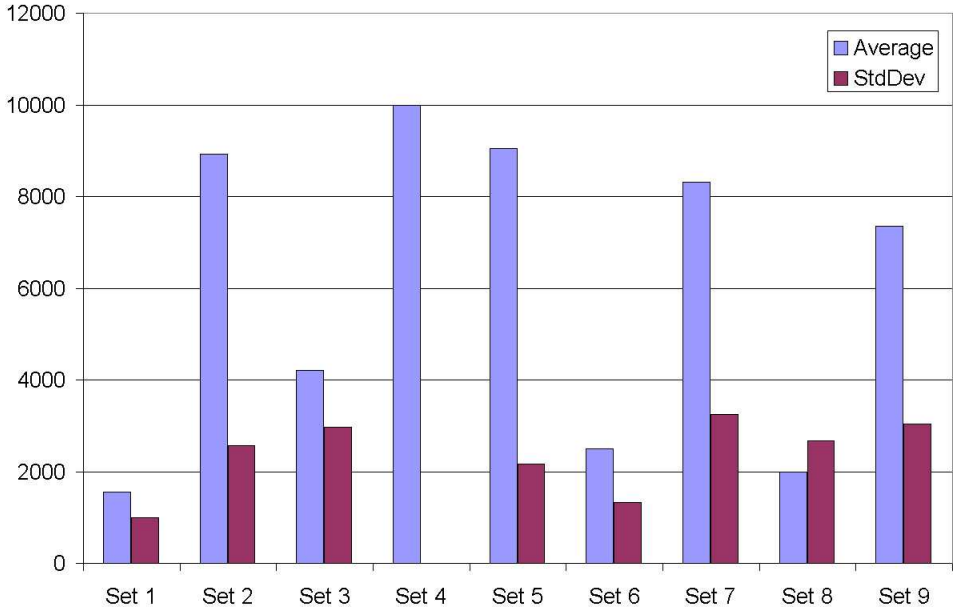


Figure 6: **Fixed-track-local problem, track** 4×4**:** the average number of steps (over 10 runs) to find a solution (and the standard deviation) for 9 different piece sets.

(a) Define a function (track-¿enumerated-csp width height), which constructs an enumerated CSP, rather like *australia-csp* but generated automatically.

There is a variable in the enumerated CSP corresponding to each tile position. The tiles on around the border of the track can only take on a single value: #barrier-piece. The remaining variables range over the set of possible track piece types (#straight-piece, #cross-piece, etc.) at each possible orientation. For every tile not on the border, there is a constraint between the tile and each of it's 4 neighbors. This constraint enumerates the possible values the two tiles can take on such that they either both have connections on the adjoining boundary or they both don't. Figure 7 shows a circle for each variable in the CSP for a 2×2 track. The shaded circles can only take on the value #barrier-piece.

```
CL-USER(6): (print-track (csp-state-to-track (recursive-backtracking
                          (make-track->enumerated-CSP 3 3)) 3 3))
```

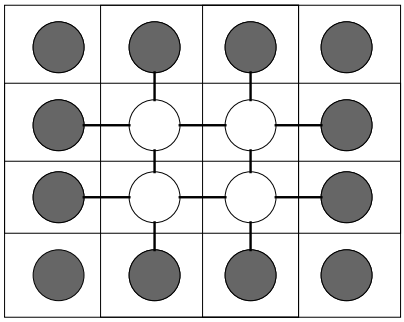
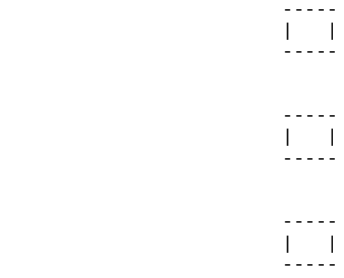
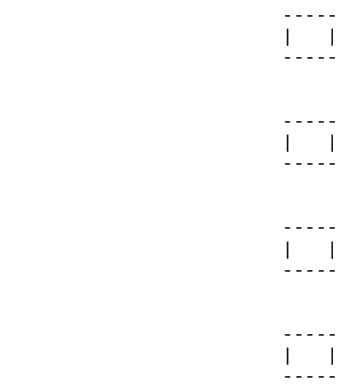


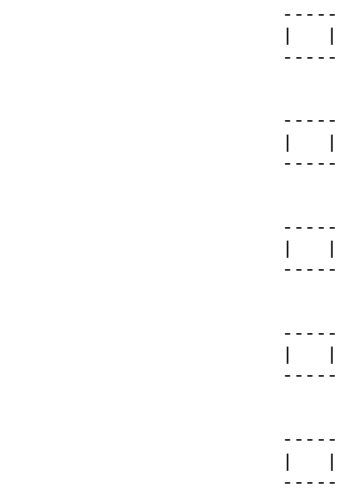
Figure 7: **Enumerated CSP formulation:** a circle for each variable in the CSP for a 2×2 track is shown. The shaded circles can only take on the value #barrier-piece.



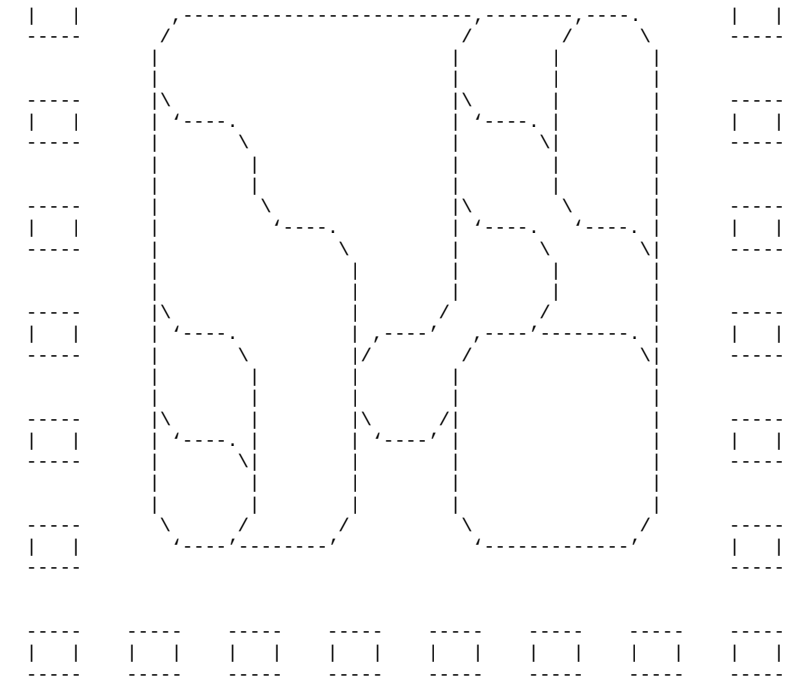
```
NIL

CL-USER(7): (print-track (csp-state-to-track (recursive-backtracking
                           (make-track->enumerated-CSP 4 4)) 4 4))
```



```
NIL

CL-USER(12): (print-track (csp-state-to-track (recursive-backtracking
                            (make-track->enumerated-CSP 5 5)) 5 5))
```



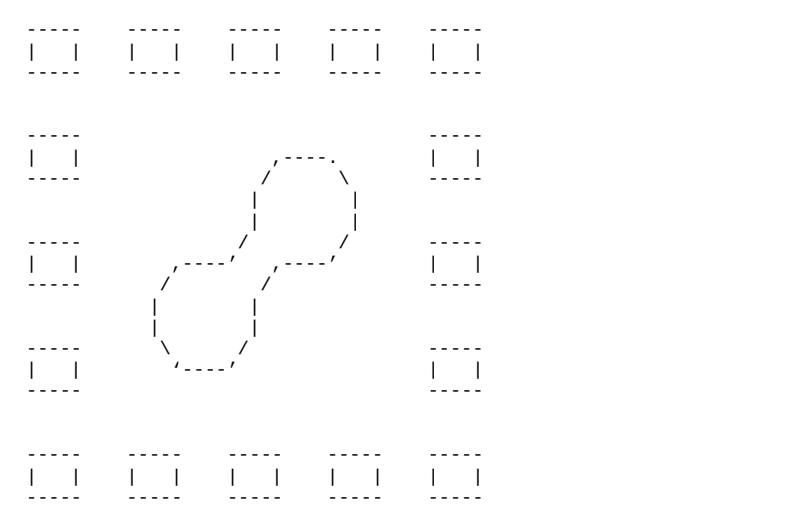
```
NIL

CL-USER(13): (print-track (csp-state-to-track (recursive-backtracking
                            (make-track->enumerated-CSP 8 8)) 8 8))
```



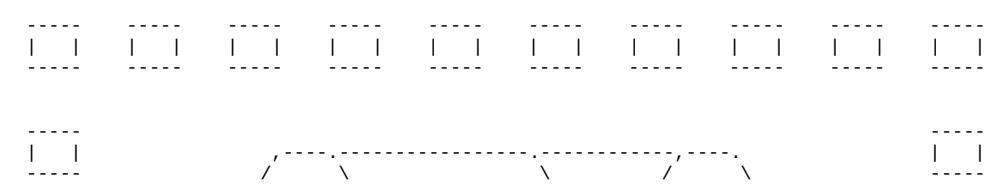
```
NIL
```

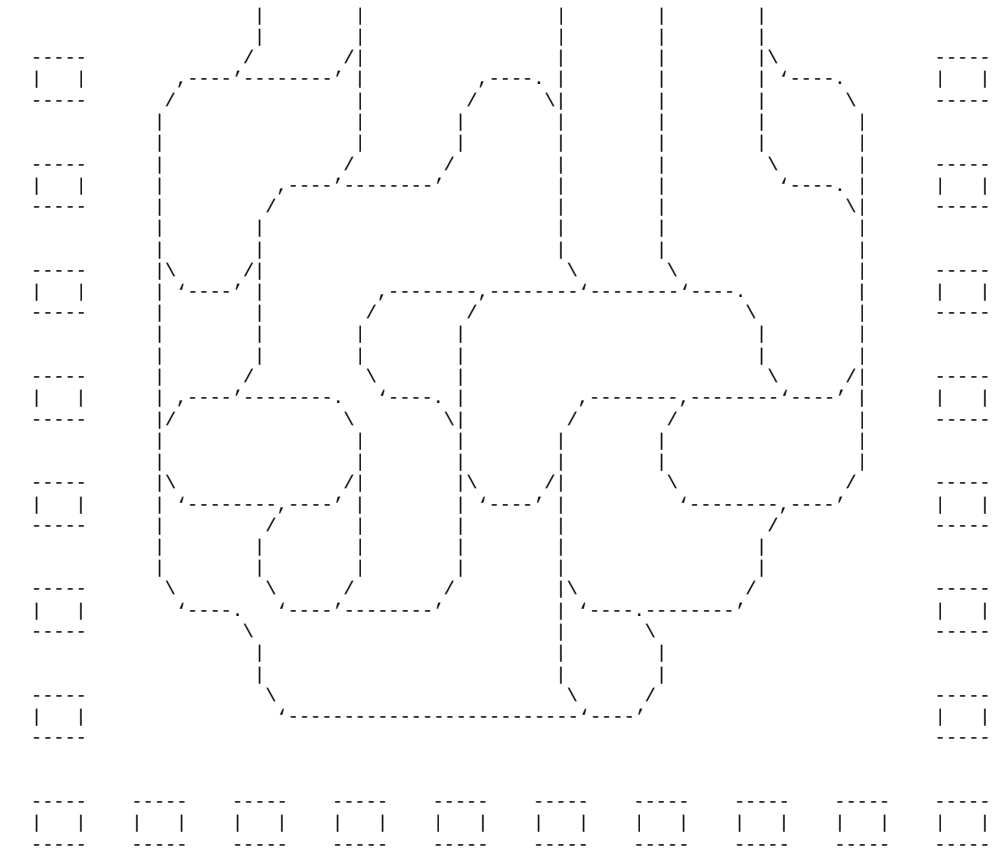
(b) Define a subtype of CSPs called a track-csp and a subtype of CSP states called track-csp-state. Define suitable methods for these, analogous to all the methods defined for enumerated CSPs, so that backtracking can be applied directly to a track-csp instance.

```
CL-USER(71): (print-track (recursive-backtracking (make-track-csp :width 3 :height 3)))
```



```
NIL
CL-USER(72): (print-track (recursive-backtracking (make-track-csp :width 8 :height 8)))
```





```
NIL
```

7. What difficulties might arise in the reduction to enumerated CSPs for the track design problem where solutions must have exactly one WCC, or where there is a fixed supply of track? How might you overcome these difficulties?

   Weak connectedness is a global constraint that involves all the squares. This creates two difficulties: 1) the number of tuples to consider in writing out the constraint is exponential in the number of variables; 2) generating and testing in order to list the tuples that satisfy the constraint is tantamount to solving the problem by exhaustive search. One solution is to replace the enumerated constraint by a functionally defined constraint that returns true iff `count-wccs` returns 1. This will have the effect of throwing out otherwise satisfactory solutions at the leaves of the search tree.

   Checking that only a fixed set of pieces is used has a similarily global character. If there are N tiles and $M \geq N$ available pieces, then for each of the N^2 pairs of tiles, we need to rule out the possibility that they both take on the value $P_1, P_2, \ldots$ etc. Since there are M possibilities, we need to rule out we end up with MN^2 additional constraints.

8. Write a predicate (strongly-connected? track), which should return t iff the track is strongly connected, and use it with any of your track design methods to make some large, strongly connected tracks.

   The idea is that if the track is strongly connected then we can pick any point we like and be able to visit from there any point in the track in either direction. We need to be a little bit careful about that as it can be the case that in a weakly connected track this could be possible if we start in one direction, but not if we go opposite at the start point. So, the idea is to pick a point on the track and try to go in both directions.

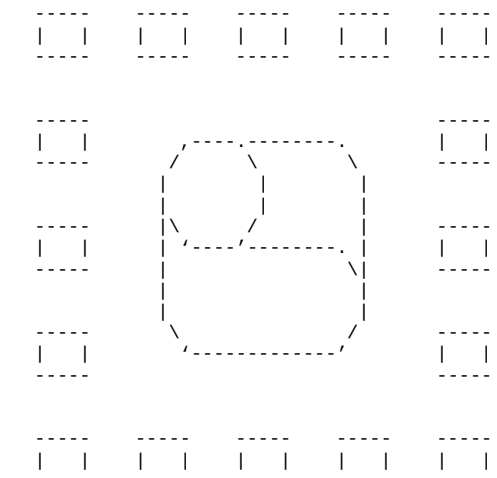
   The function `strongly-connected?` checks whether the track represents a single strongly connected component (SCC). It uses an array `mark`, where there are two elements for each combination of square and its edge: “enter” and “exit”. For each combination of (square,edge,enter/exit) the array

shows whether it has been visited/nonvisited from the starting point, going forward or both forward and backward. The values are filled in recursively with a call to `mark-scc` following the directed links within the piece and the corresponding edge on the adjacent piece, if any. The function `strongly-connected?` never calls `mark-scc` for a second time: it returns `nil` if a never visited square has been found. `nil` is also returned, if a node in the graph is visited when going forward, but not when going backward.

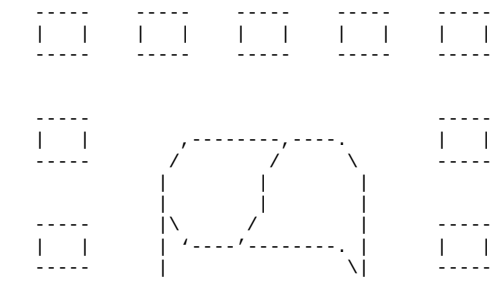
If we are interested in counting the strongly connected componets or finding the articulation points then there are efficient linear algorithms that again rely on DFS. But this is not our goal here.

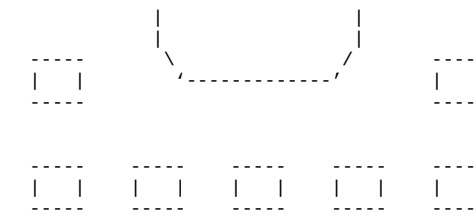
NOTE: Remember that a circle or a line represent *two* separate SCCs.

```
CL-USER(7): (strongly-connected? (read-track "~cs188/code-188/search/domains/weak44.track"))
NIL
CL-USER(8): (strongly-connected? (read-track "~cs188/code-188/search/domains/strong44.track"))
T
CL-USER(9): (print-track (setf t33-1 (make-full-track 3 3 (list (make-curve-piece :orientation 0)
                            (make-lsplit-piece :orientation 2)
                            (make-curve-piece :orientation 1)
                            (make-straight-piece :orientation 1)
                            (make-lsplit-piece :orientation 1)
                            (make-rsplit-piece :orientation 1)
                            (make-curve-piece :orientation 3)
                            (make-lsplit-piece :orientation 0)
                            (make-curve-piece :orientation 2)))))
```

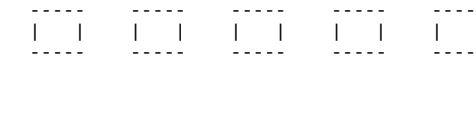


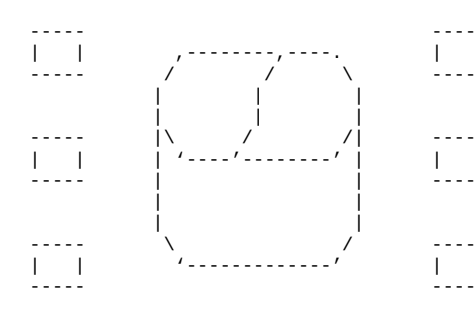
```
NIL
CL-USER(10): (strongly-connected? t33-1)
NIL
CL-USER(11): (print-track (setf t33-2 (make-full-track 3 3 (list (make-curve-piece :orientation 0)
                            (make-lsplit-piece :orientation 2)
                            (make-curve-piece :orientation 1)
                            (make-straight-piece :orientation 1)
                            (make-lsplit-piece :orientation 1)
                            (make-lsplit-piece :orientation 3)
                            (make-curve-piece :orientation 3)
                            (make-lsplit-piece :orientation 0)
                            (make-curve-piece :orientation 2)))))
```



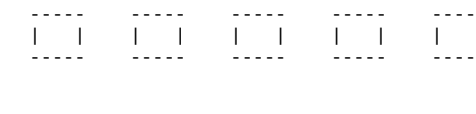


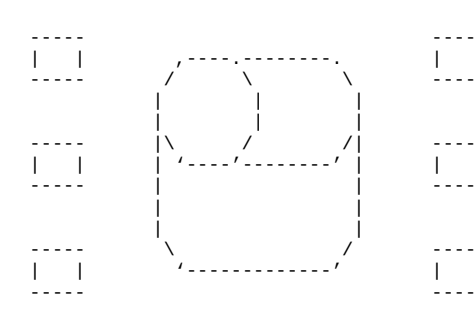

```
NIL
CL-USER(12): (strongly-connected? t33-2)
NIL
CL-USER(13): (print-track (setf t33-3 (make-full-track 3 3 (list (make-curve-piece :orientation 0)
                            (make-lsplit-piece :orientation 2)
                            (make-curve-piece :orientation 1)
                            (make-straight-piece :orientation 1)
                            (make-lsplit-piece :orientation 1)
                            (make-lsplit-piece :orientation 3)
                            (make-curve-piece :orientation 3)
                            (make-rsplit-piece :orientation 2)
                            (make-curve-piece :orientation 2)))))
```



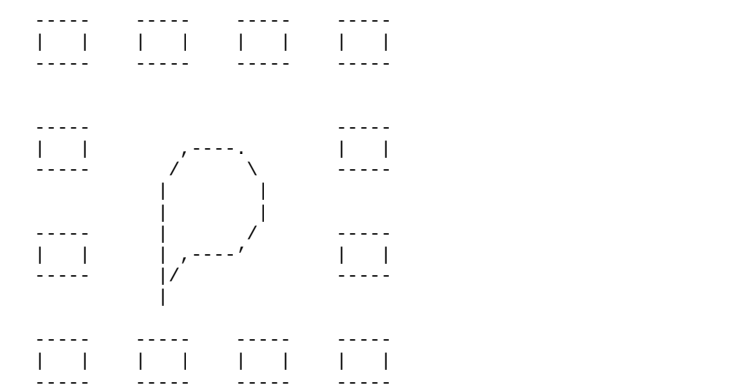


```
NIL
CL-USER(14): (strongly-connected? t33-3)
NIL
CL-USER(15): (print-track (setf t33-4 (make-full-track 3 3 (list (make-curve-piece :orientation 0)
                            (make-lsplit-piece :orientation 2)
                            (make-curve-piece :orientation 1)
                            (make-straight-piece :orientation 1)
                            (make-lsplit-piece :orientation 1)
                            (make-rsplit-piece :orientation 1)
                            (make-curve-piece :orientation 3)
                            (make-rsplit-piece :orientation 2)
                            (make-curve-piece :orientation 2)))))
```

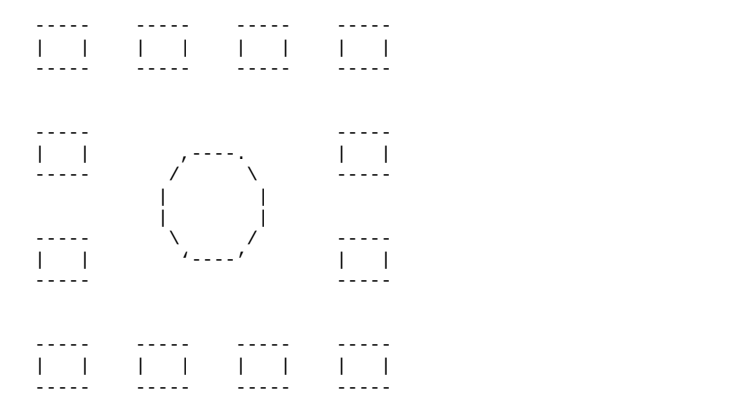




```
NIL
CL-USER(16): (strongly-connected? t33-4)
NIL

CL-USER(17): (print-track (setf t22
  (make-full-track 2 2 (list (make-rsplit-piece :orientation 0)
                             (make-curve-piece :orientation 1)
                             (make-curve-piece :orientation 3)
                             (make-curve-piece :orientation 2)))))
```



```
NIL
CL-USER(18): (strongly-connected? t22)
NIL
CL-USER(19): (strongly-connected? strong44)
T
CL-USER(20): (strongly-connected? weak44)
NIL
CL-USER(21): (print-track (setf t22-2
  (make-full-track 2 2 (list (make-curve-piece :orientation 0)
                             (make-curve-piece :orientation 1)
                             (make-curve-piece :orientation 3)
                             (make-curve-piece :orientation 2)))))
```



```
NIL
CL-USER(22): (strongly-connected? t22-2)
NIL
```