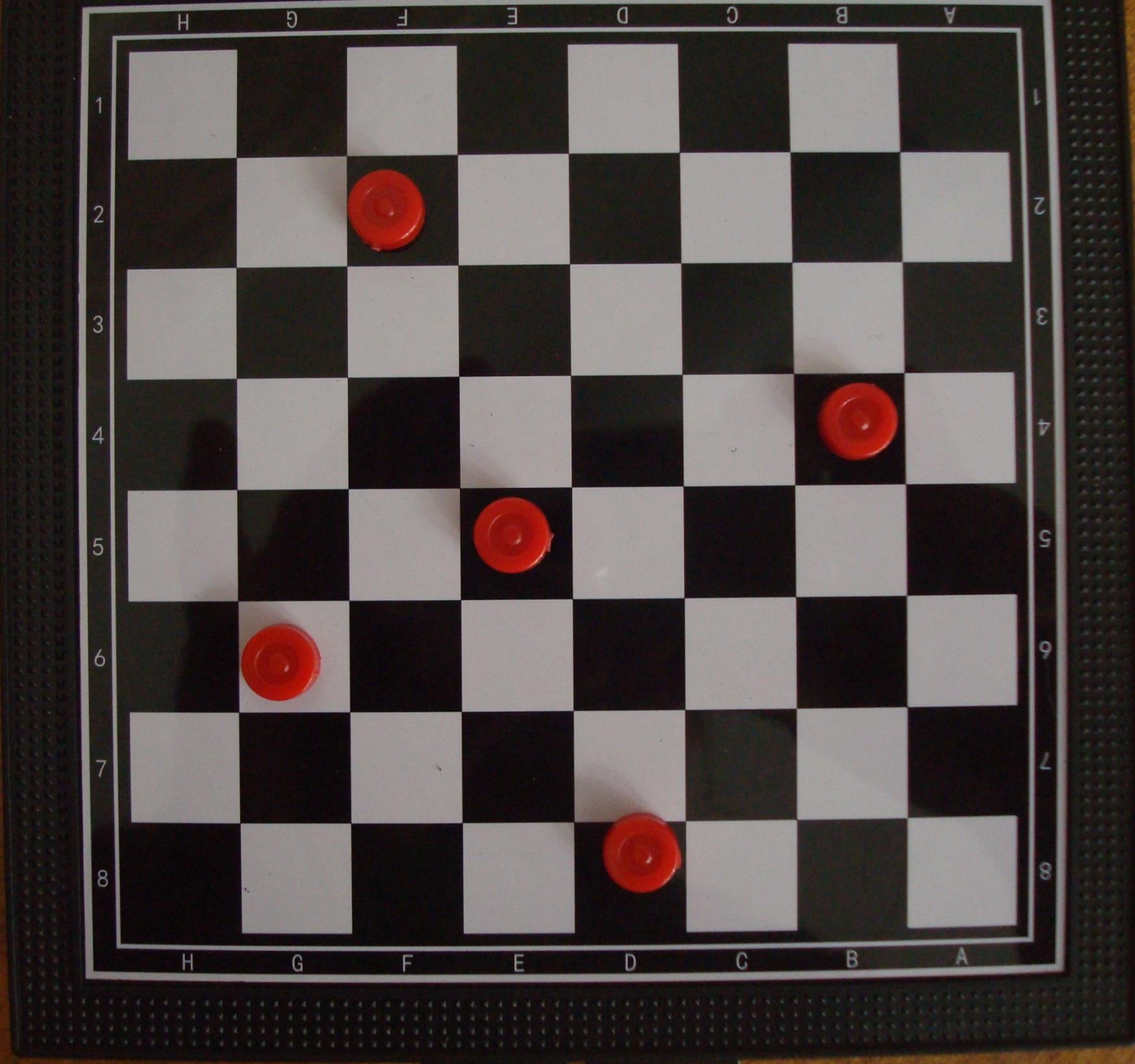
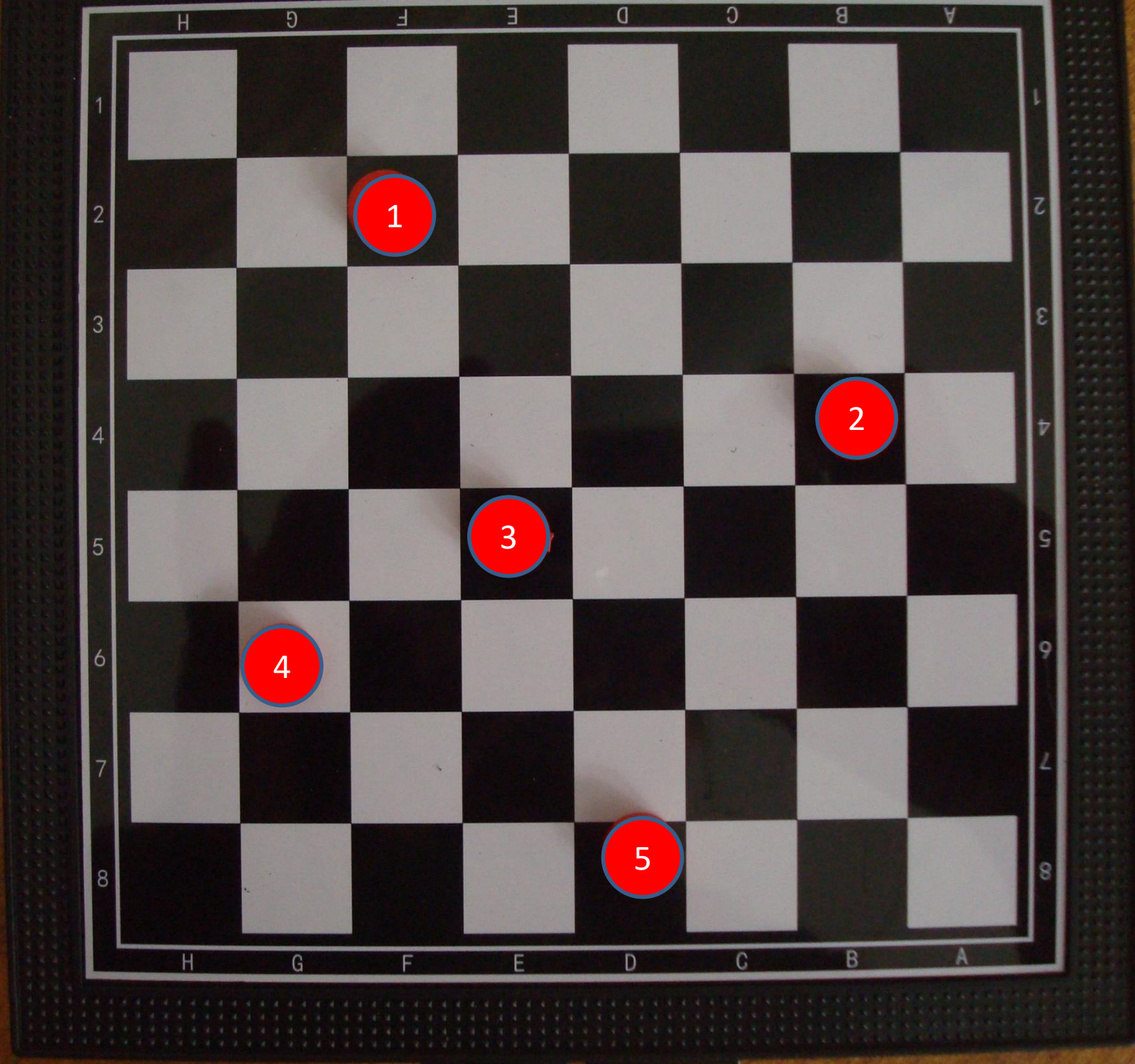
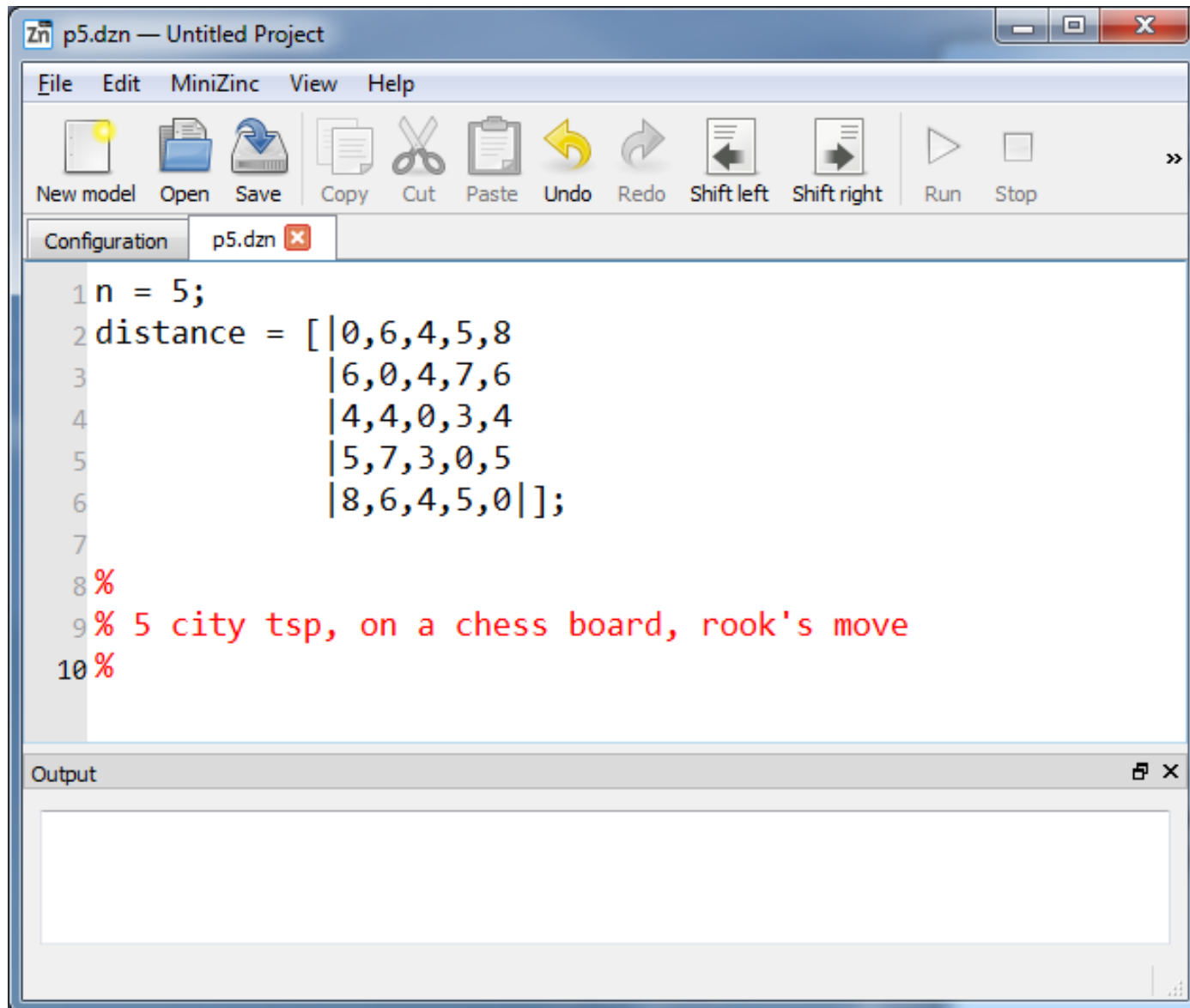


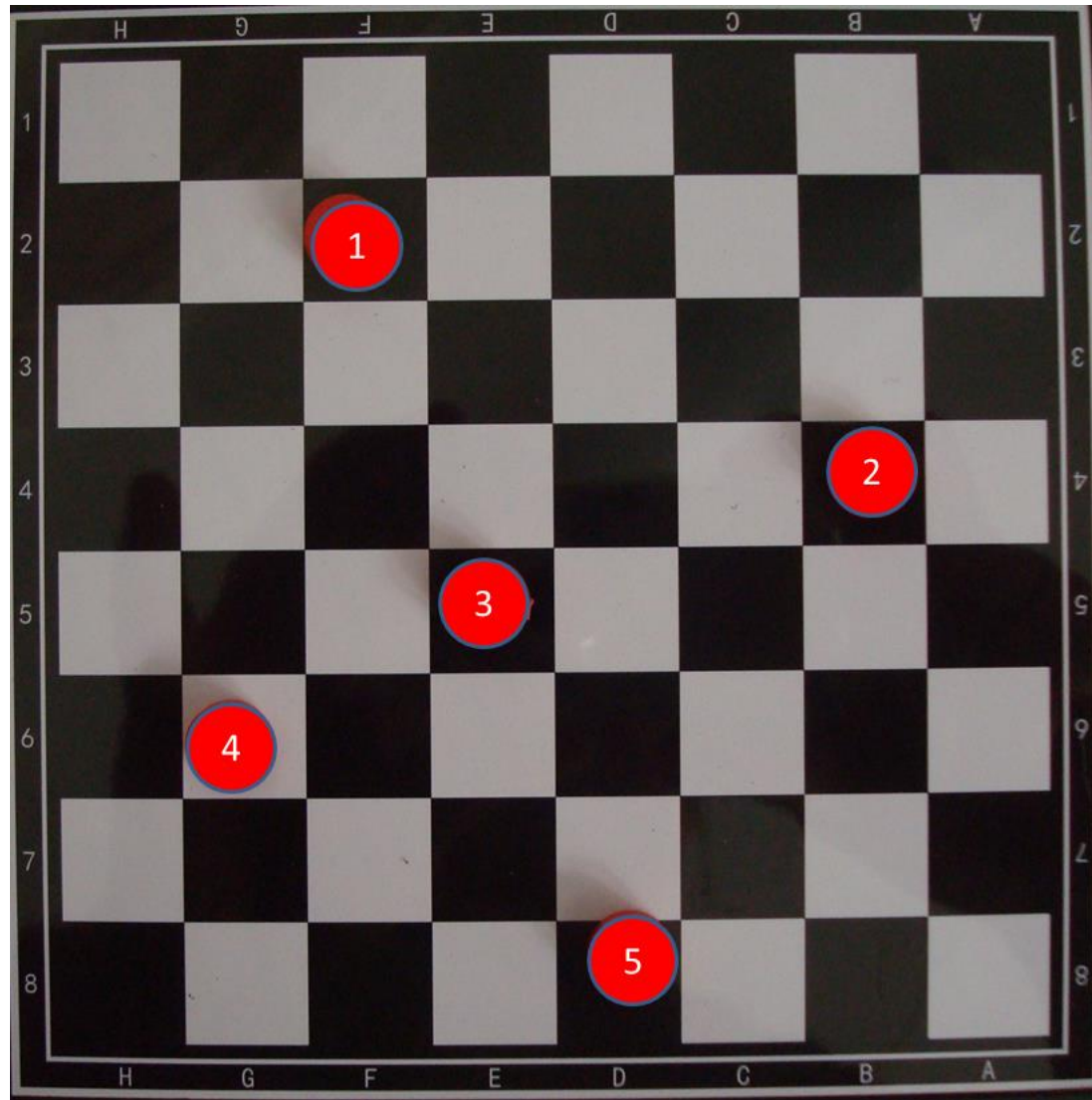
Small TSP



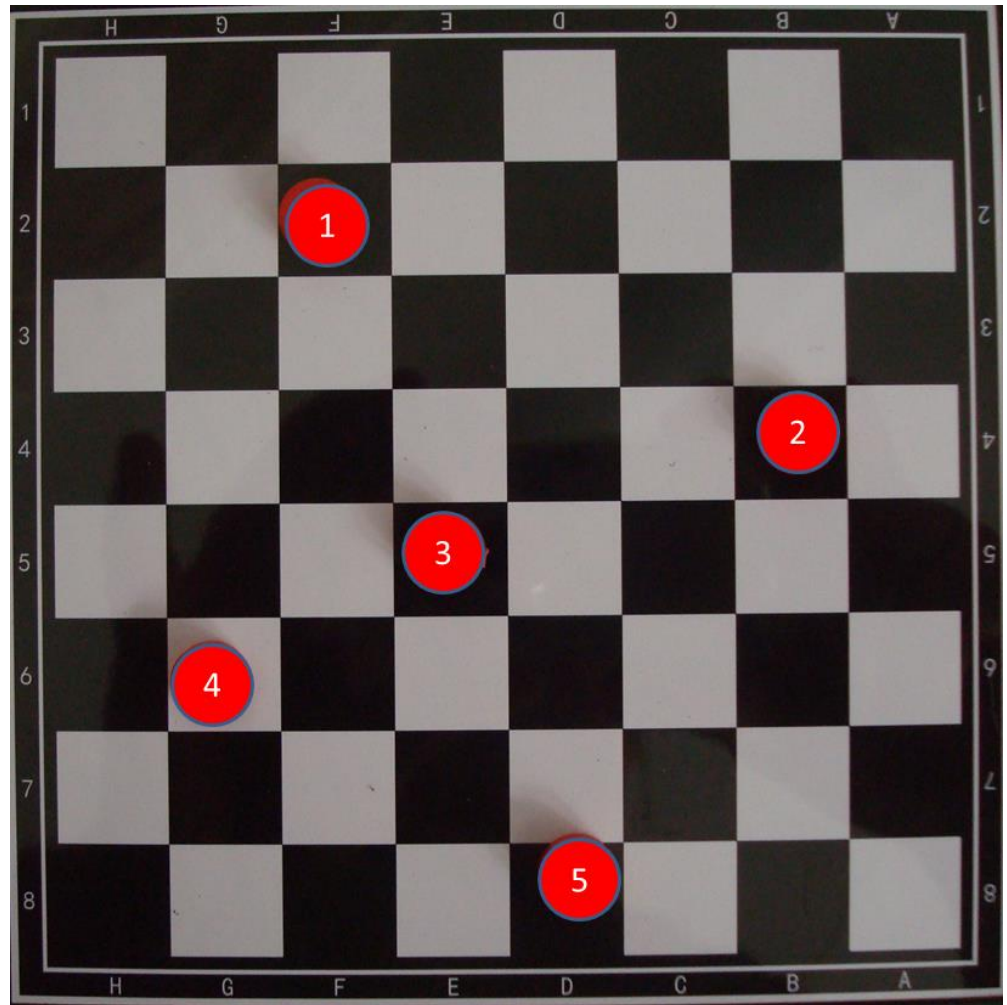






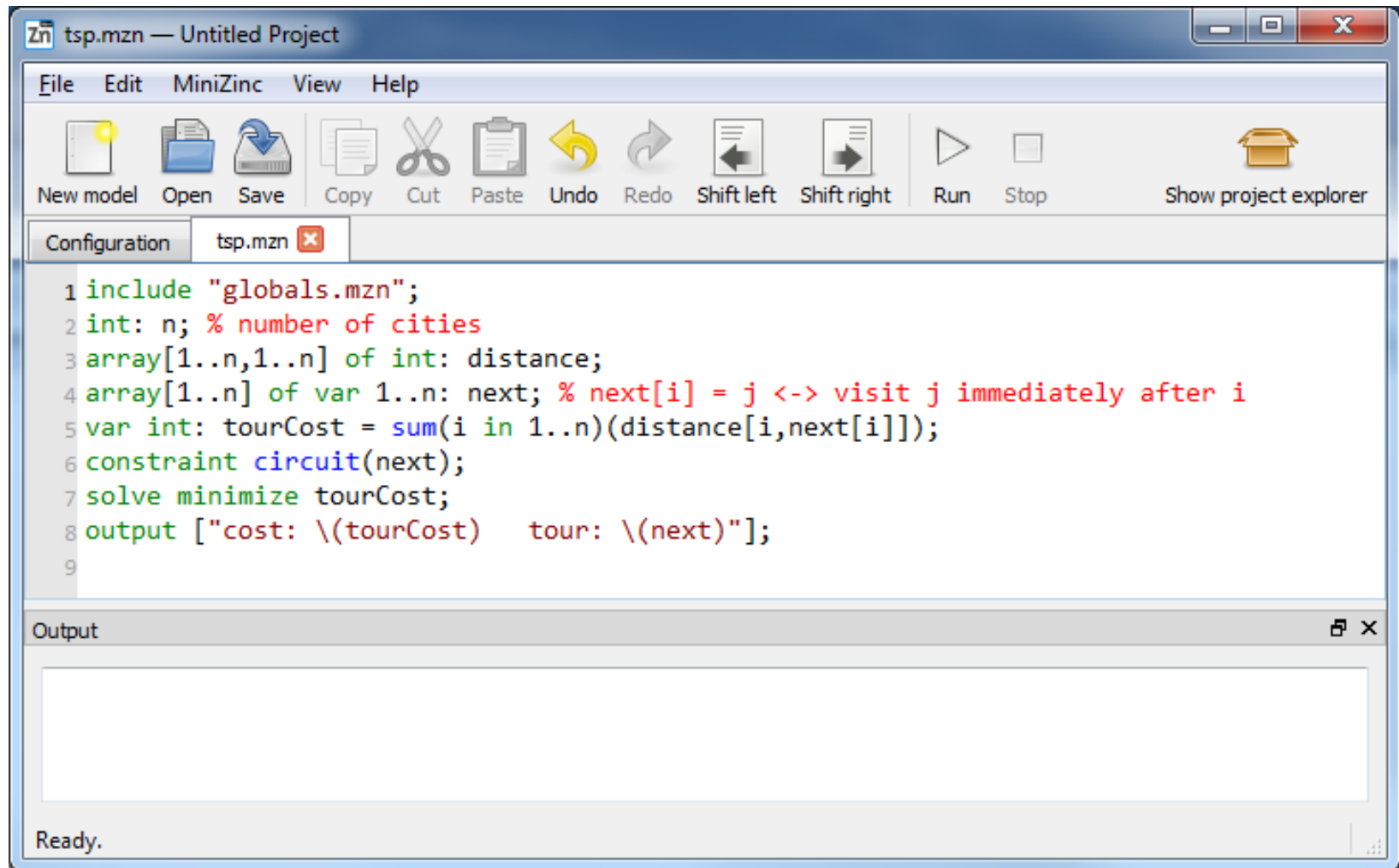


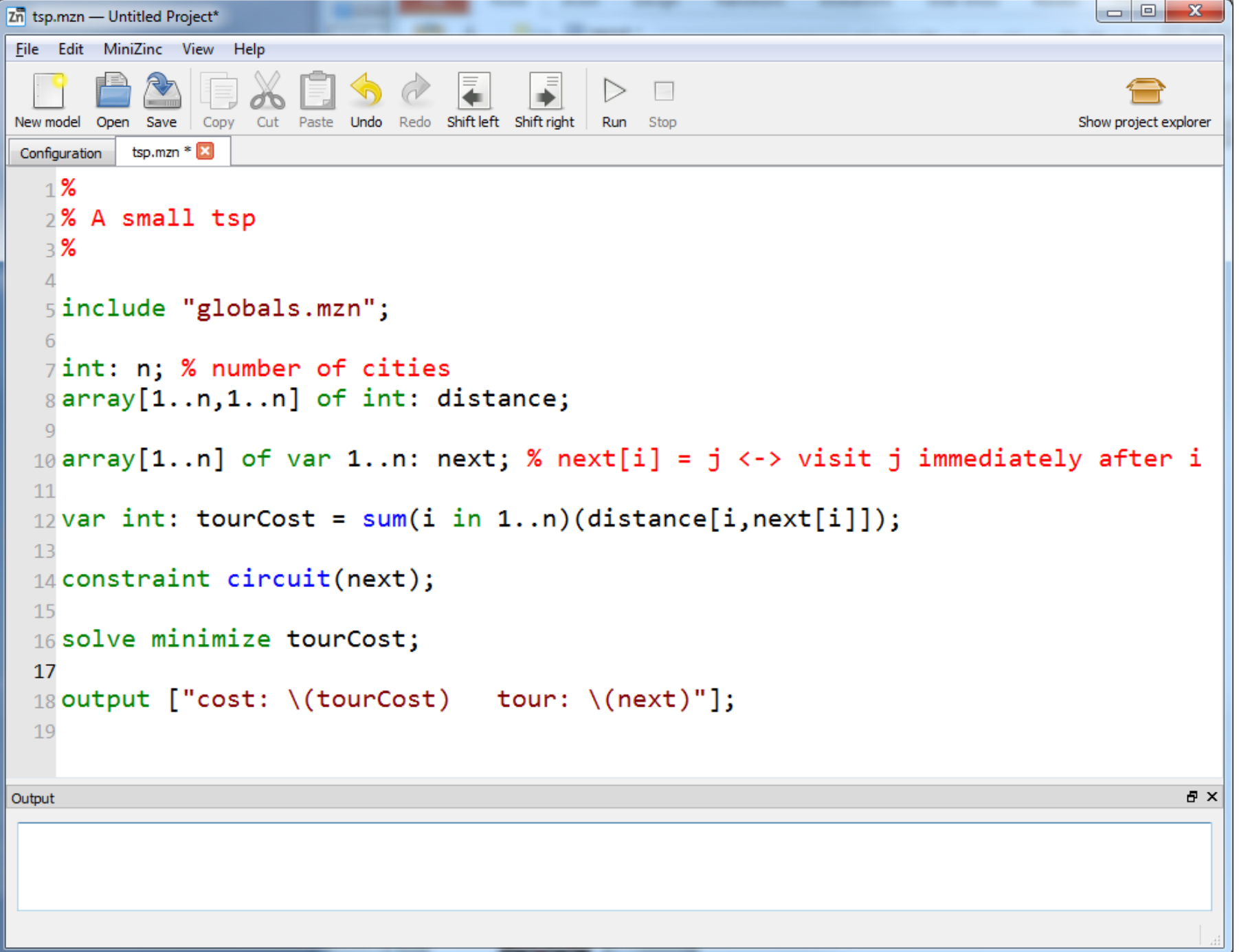
```
n = 5;  
distance = [ | 0,6,4,5,8  
              | 6,0,4,7,6  
              | 4,4,0,3,4  
              | 5,7,3,0,5  
              | 8,6,4,5,0 | ];
```

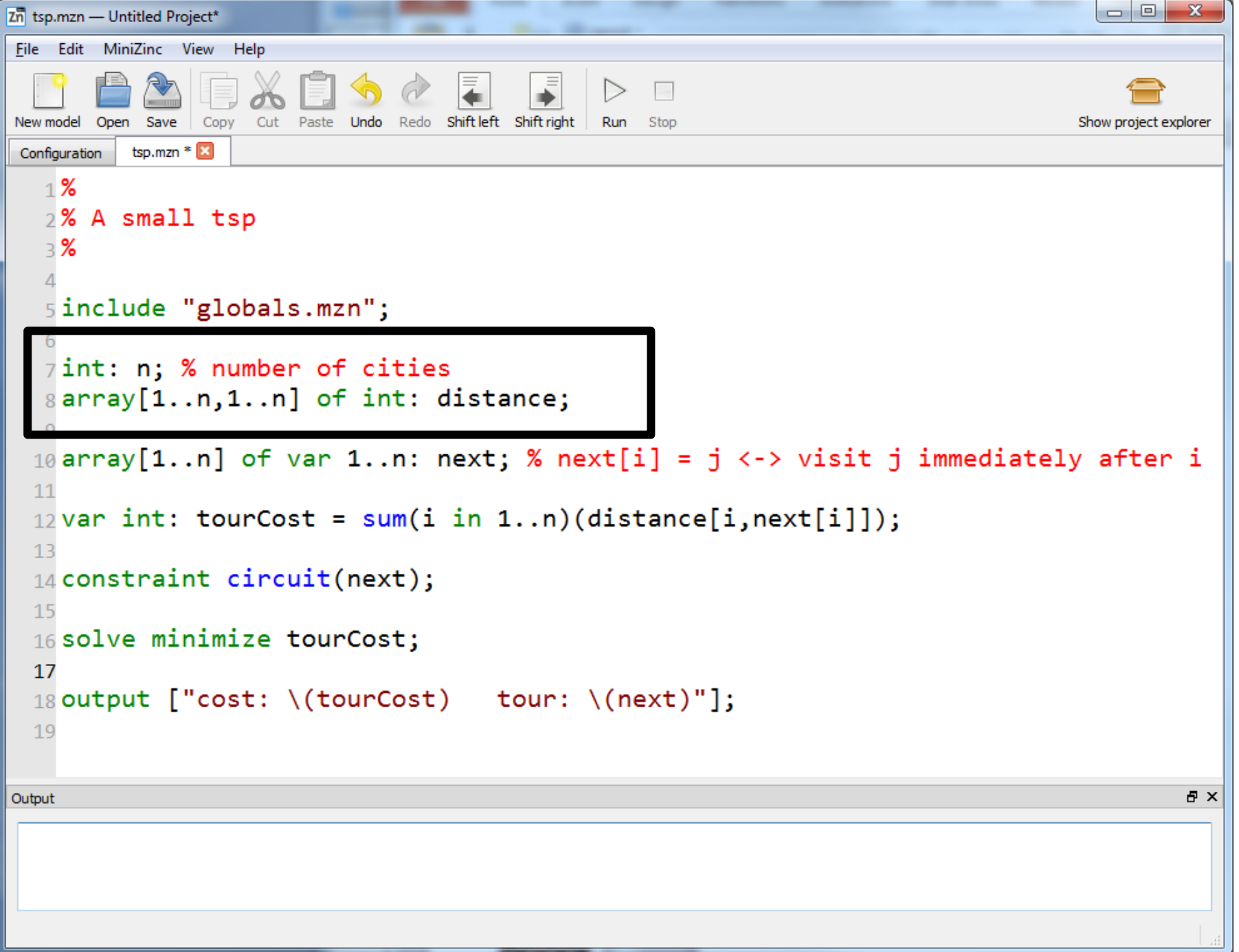


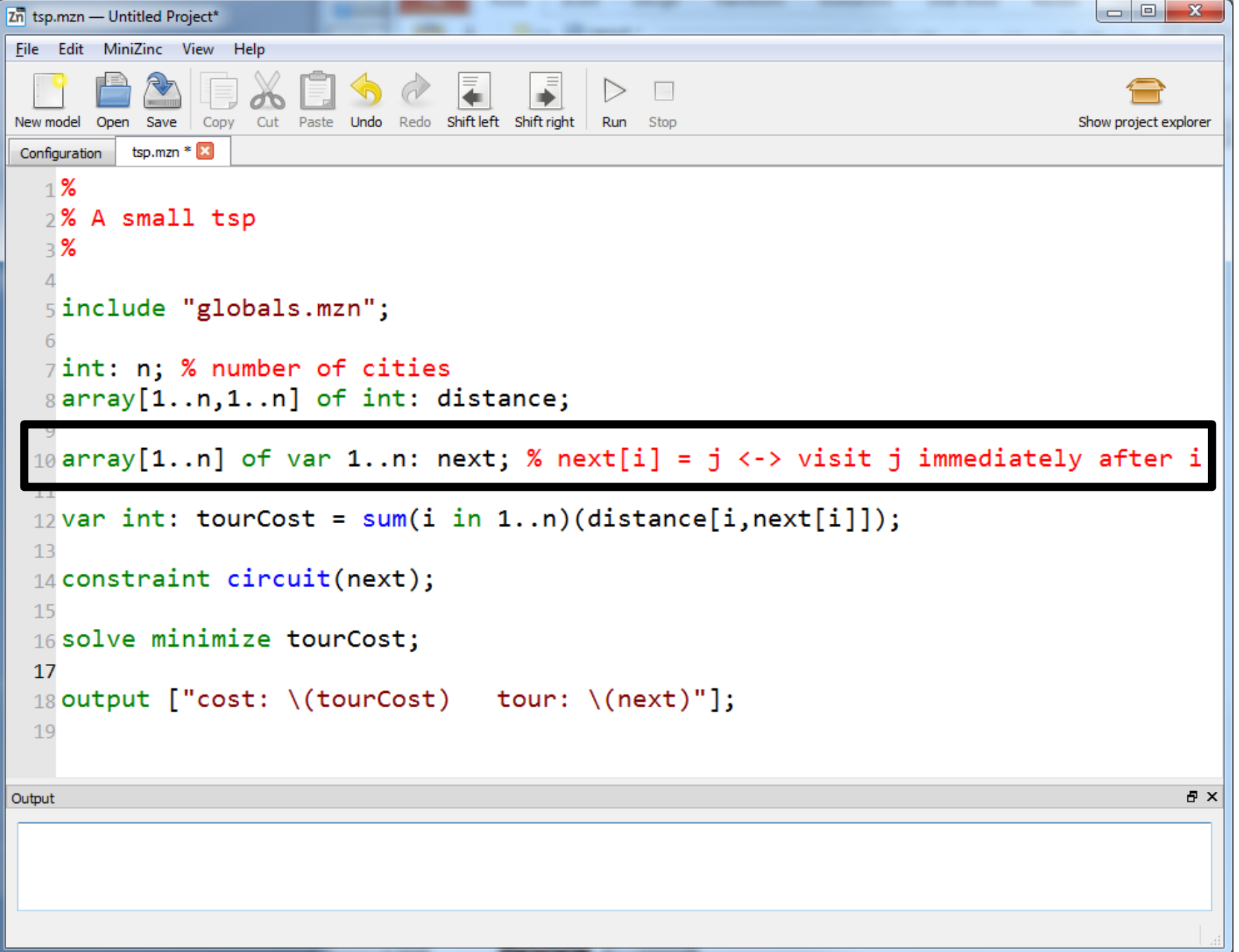
Single successor model

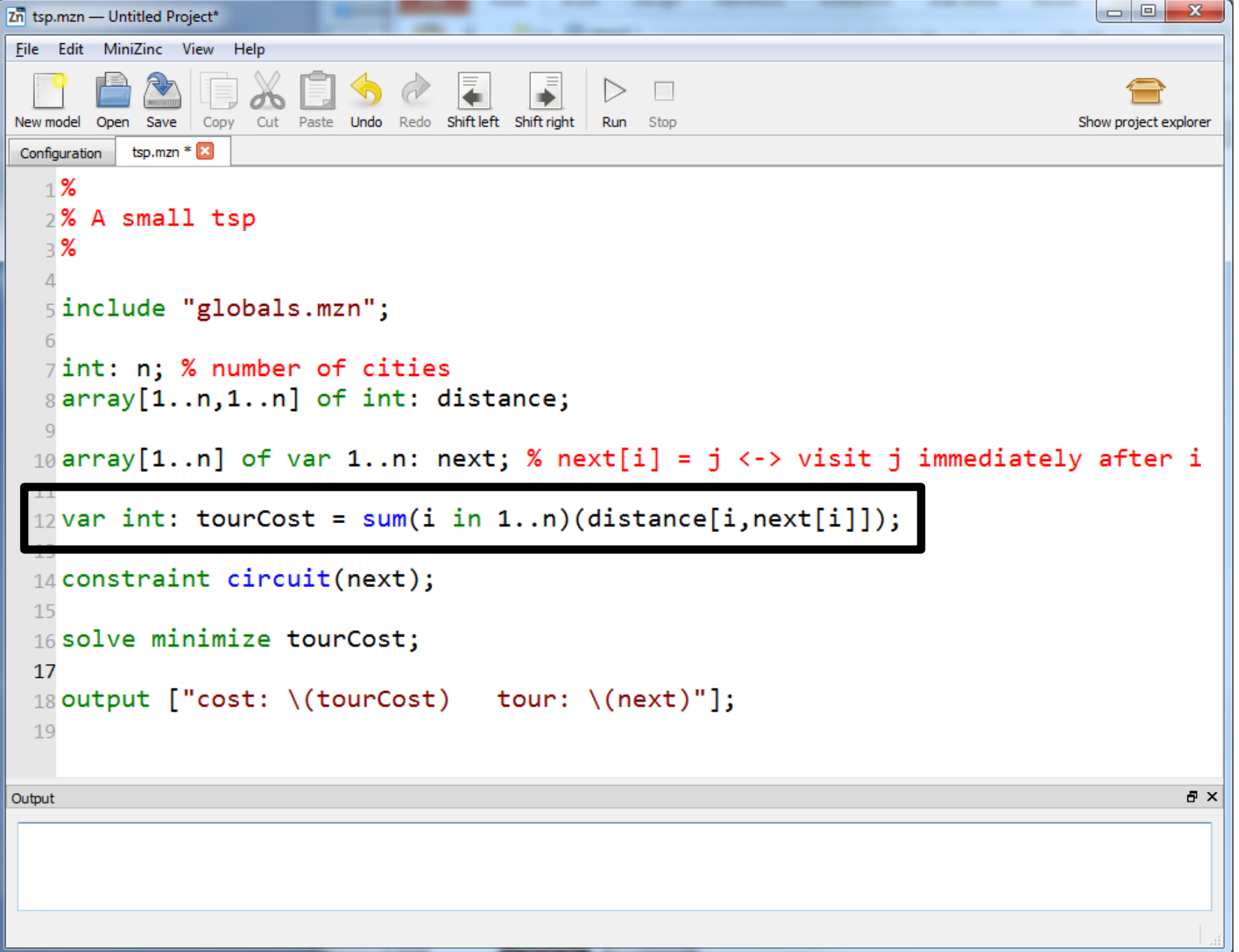
$\text{next}[i] = j$  means “visit city  $j$  immediately after city  $i$ ”

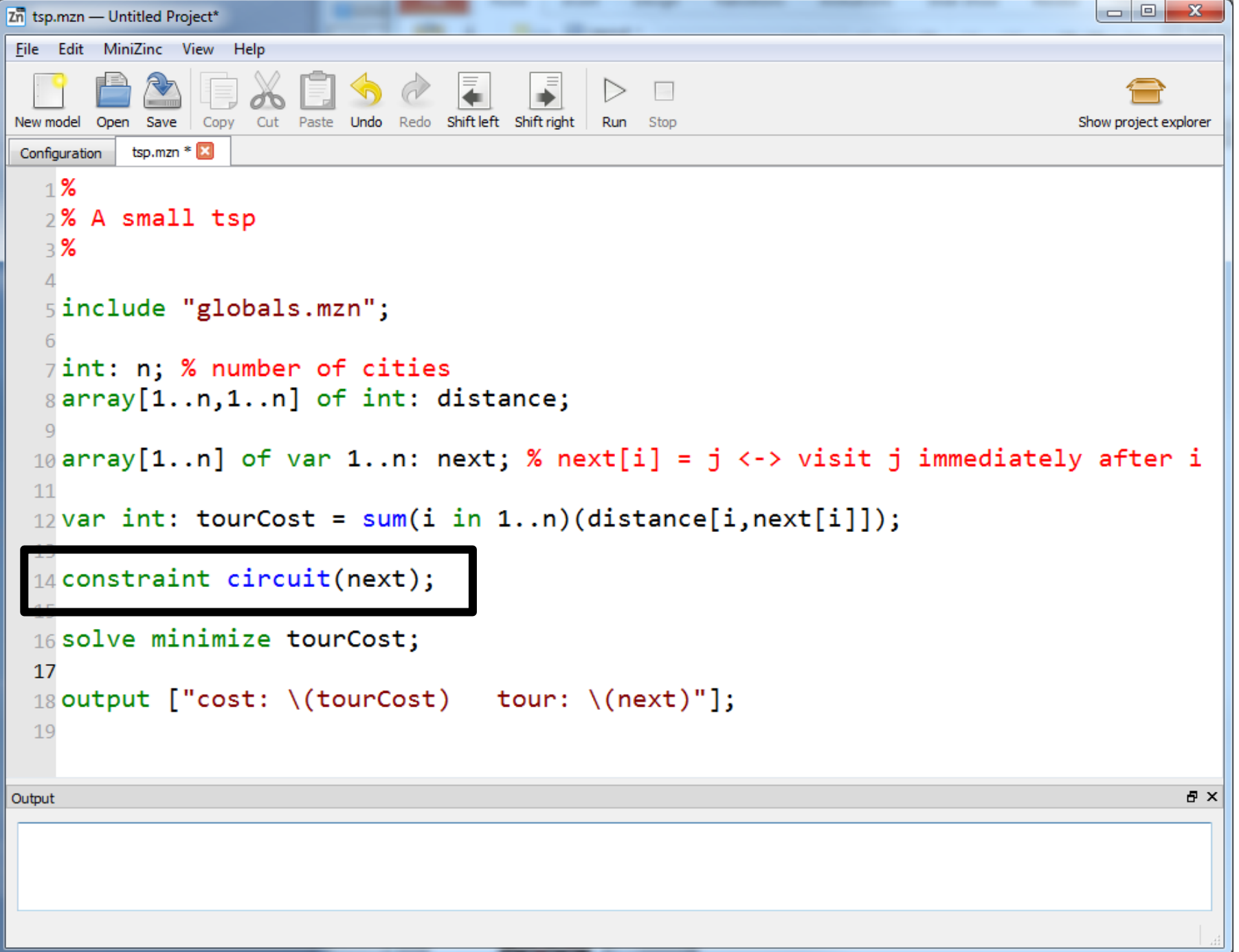












FileEditViewHistoryBookmarksToolsHelp

CiteSeerX — Solving small TSP x

citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.8945

Search

DocumentsAuthorsTablesDonateMetaCartSign upLog in

CiteSeer<sup>x</sup>

☐ Include CitationsAdvanced Search

Solving small TSPs with constraints (1997)


by Yves Caseau , François Laburthe

Venue: PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING

Citations: 49 - 0 self

Save to ListAdd to CollectionCorrect ErrorsMonitor Changes

Cached



Download Links

[\[www.dcs.gla.ac.uk\]](http://www.dcs.gla.ac.uk)  
[\[www.dcs.gla.ac.uk\]](http://www.dcs.gla.ac.uk)  
[\[www.ens.fr\]](http://www.ens.fr)

SummaryCitationsActive BibliographyCo-citationClustered DocumentsVersion History

Abstract

This paper presents a set of techniques that makes constraint programming a technique of choice for solving small (up to 30 nodes) traveling salesman problems. These techniques include a propagation scheme to avoid intermediate cycles (a global constraint), a branching scheme and a redundant constraint that can be used as a bounding method. The resulting improvement is that we can solve problems twice larger than those solved previously with constraint programming tools. We evaluate the use of Lagrangean Relaxation to narrow the gap between constraint programming and other Operations Research techniques and we show that improved constraint propagation has now a place in the array of techniques that should be used to solve a traveling salesman problem.





Keyphrases

small tspssalesman problemresulting improvementbounding methodbranching schemeconstraint programming toolpropagation schemeconstraint programmingintermediate cycleimproved constraint propagationlagrangean relaxationredundant constraintglobal constraintoperation research technique

BibTeX

```
@INPROCEEDINGS{Caseau97solving-small,
  author = {Yves Caseau and François Laburthe},
  title = {Solving small TSPs with constraints},
  booktitle = {PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING},
  year = {1997},
  pages = {316--330},
  publisher = {MIT Press}
}
```

Share



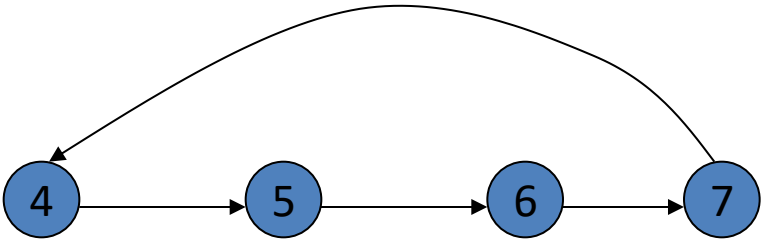
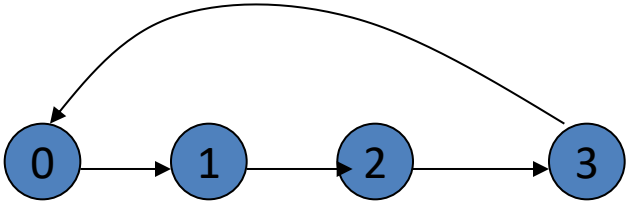
OpenURL

The single successor model

$next_i \in \{0..n-1\}$   
 $next_i = j \leftrightarrow city_j$  immediately follows  $city_i$

1	2	3	0	5	6	7	4
---	---	---	---	---	---	---	---

NOT A TOUR!



We need subtour elimination

### 3.1.3. Subtour elimination

However, propagation of the assignment constraint is not enough, since it accepts assignments that are the union of disjoint tours. In order to propagate the *nocycle* constraint to avoid subtours, we explicitly store the start, end and total number of arcs of the chain (defined by  $Next$ ) going through node  $l$ , which we respectively denote  $start_l$ ,  $end_l$ ,  $length_l$ . Upon each assignment  $Next(x) := y$  we look if subchains adjacent to  $x$  or  $y$  have already been built and call  $b$  the end of the chain starting from  $y$  and  $a$  the start of the chain ending in  $x$ ,  $length_a$  the number of arcs in the chain from  $a$  to  $x$  and  $length_b$  the number of arcs in the chain from  $y$  to  $b$ .

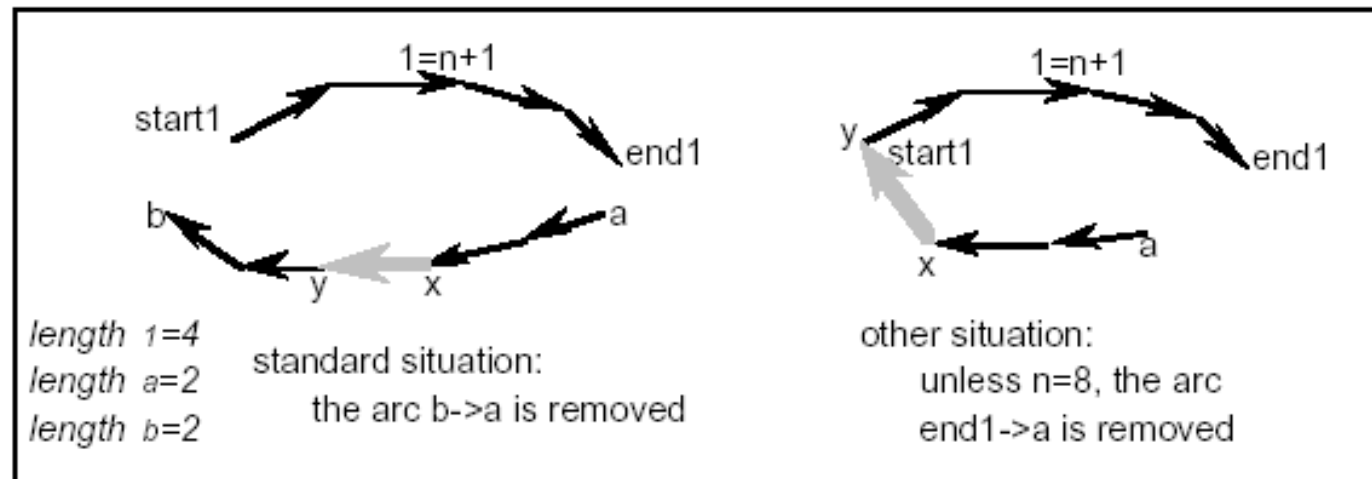
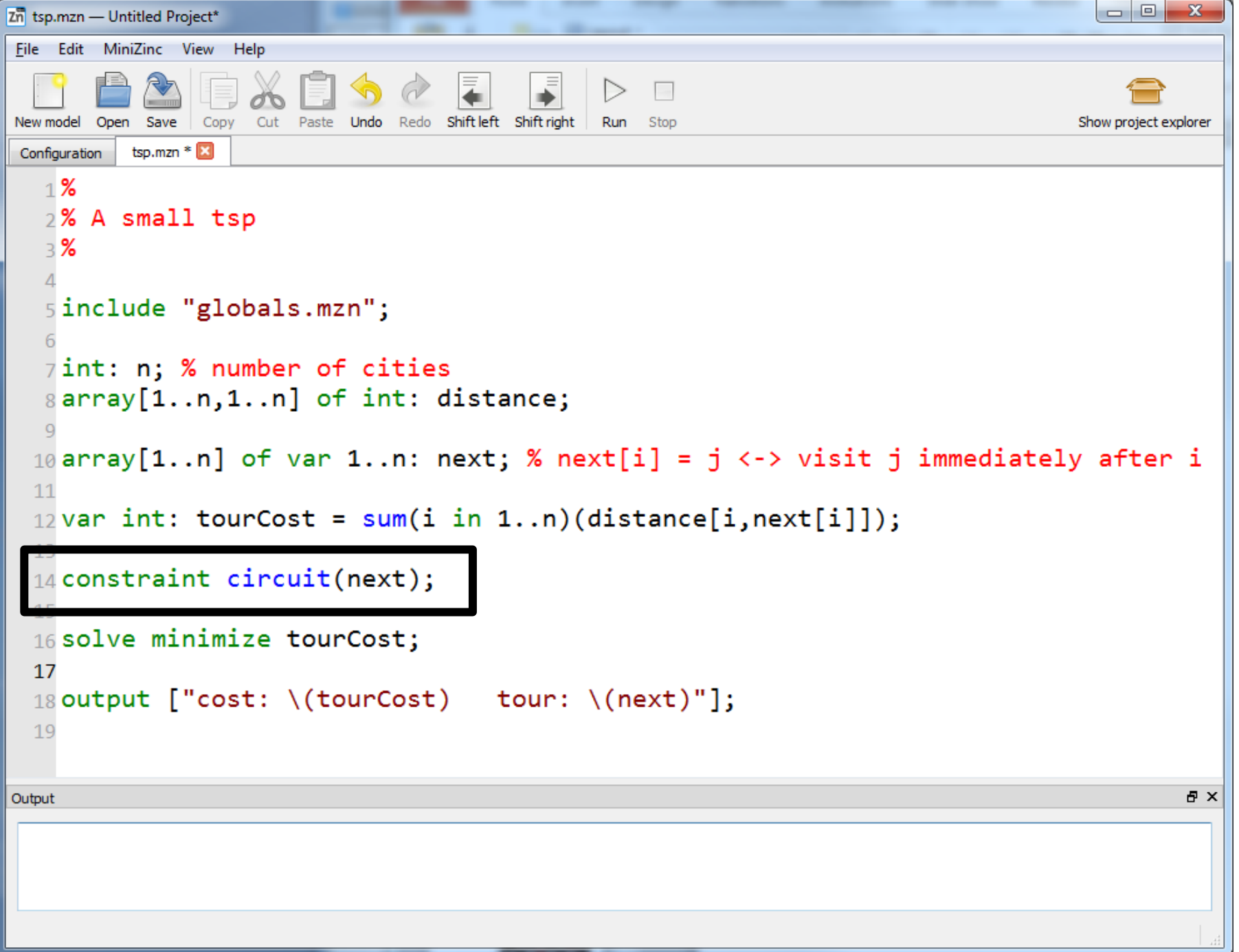
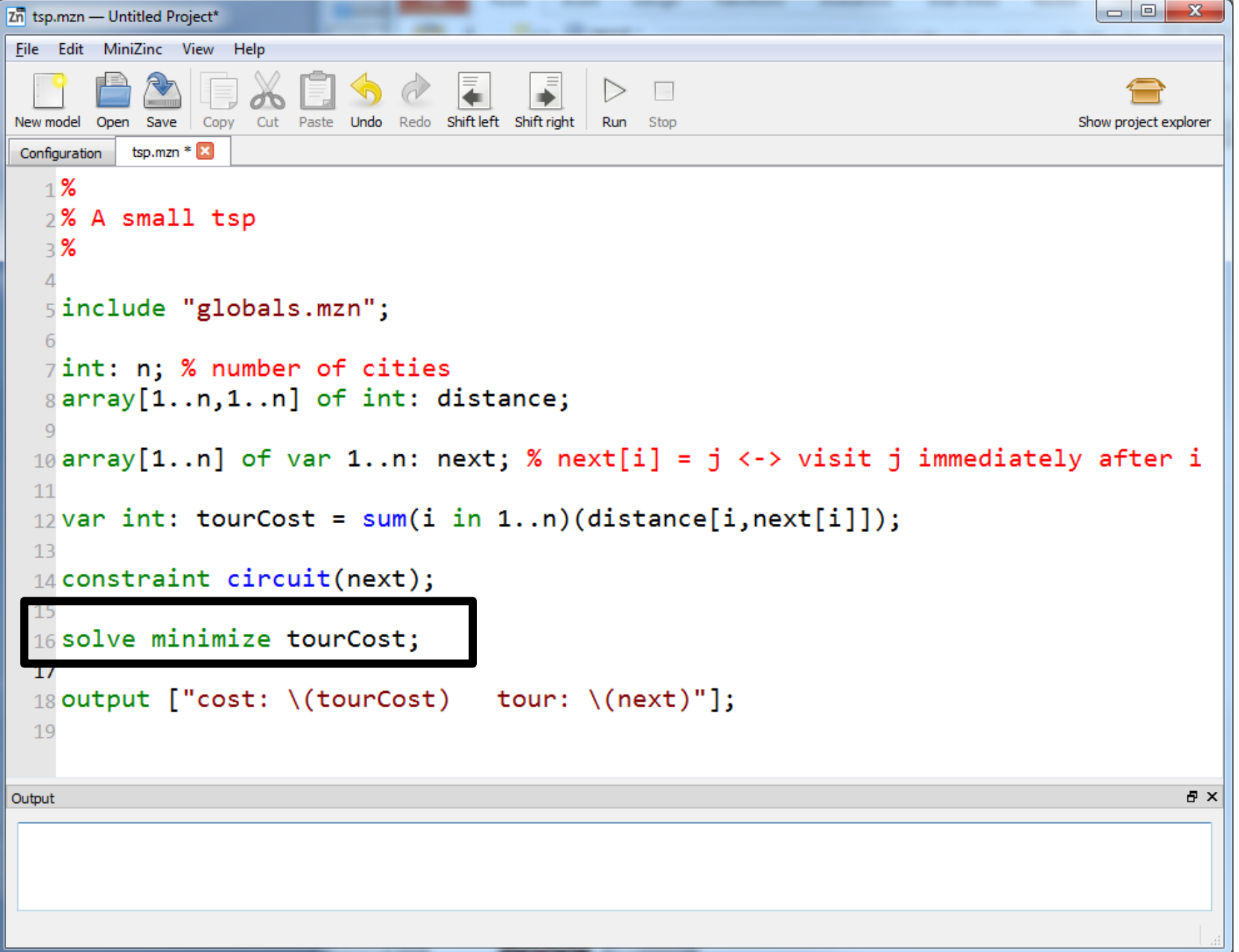
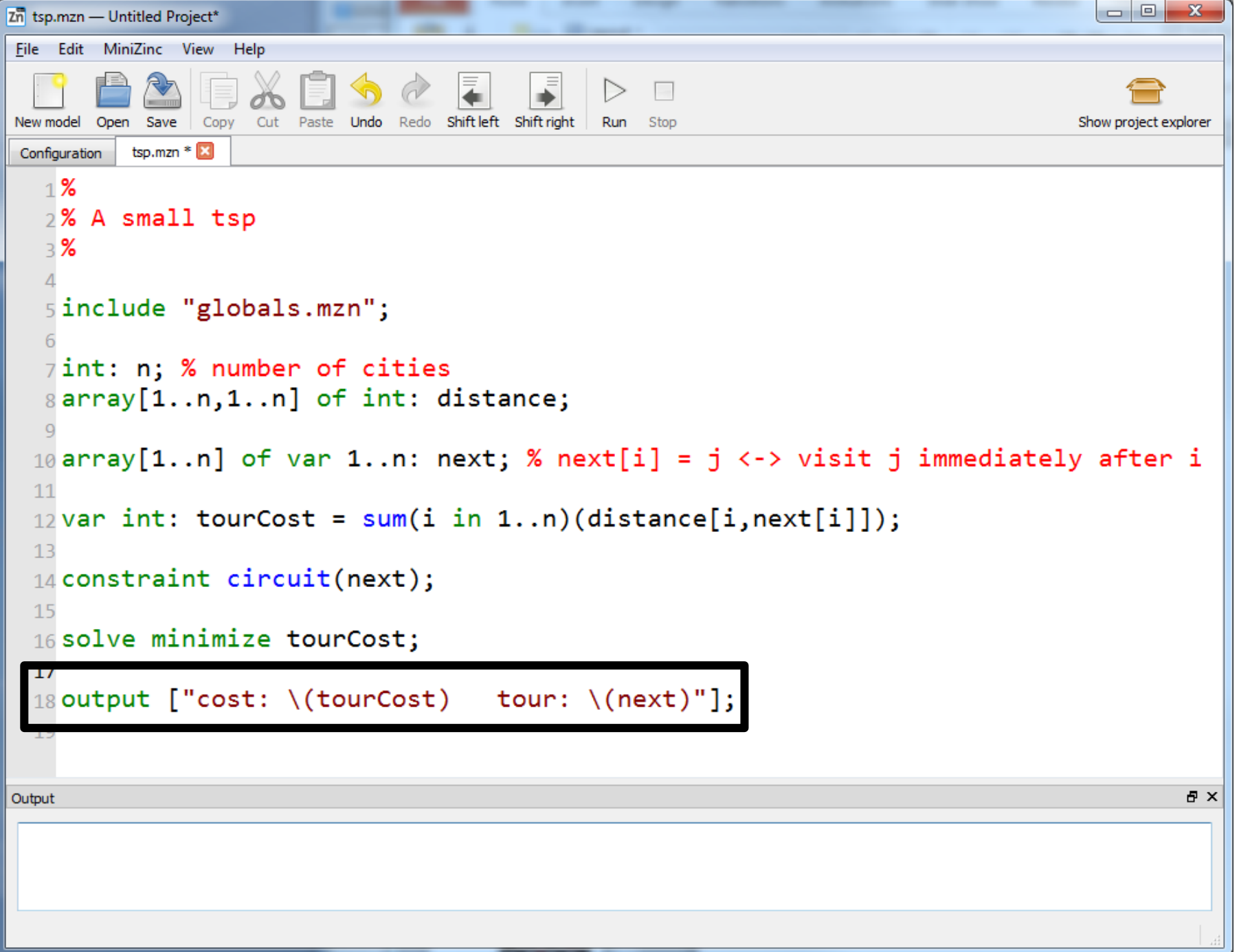


Figure 1: Propagation of the nocycle constraint

- If  $x=end_l$  and  $length_l+length_b < n-2$  we infer  $Next(b) \neq start_l$ .
- If  $y=start_l$  and  $length_l+length_a < n-2$  we infer  $Next(end_l) \neq a$ .
- Otherwise, we infer  $Next(b) \neq a$ .







Command Prompt

```
Y:\public_html\cpM\minizincCPM\tsp>mzn-gecode tsp.mzn p5.dzn -a -s
```

```
cost: 28    tour: [3, 5, 4, 2, 1]
```

```
-----  
cost: 26    tour: [4, 5, 2, 3, 1]
```

```
-----  
cost: 24    tour: [4, 1, 5, 3, 2]
```

```
-----  
=====
```

```
%% runtime:      0.005 (5.000 ms)
```

```
%% solvetime:    0.001 (1.000 ms)
```

```
%% solutions:    3
```

```
%% variables:    11
```

```
%% propagators:  7
```

```
%% propagations: 169
```

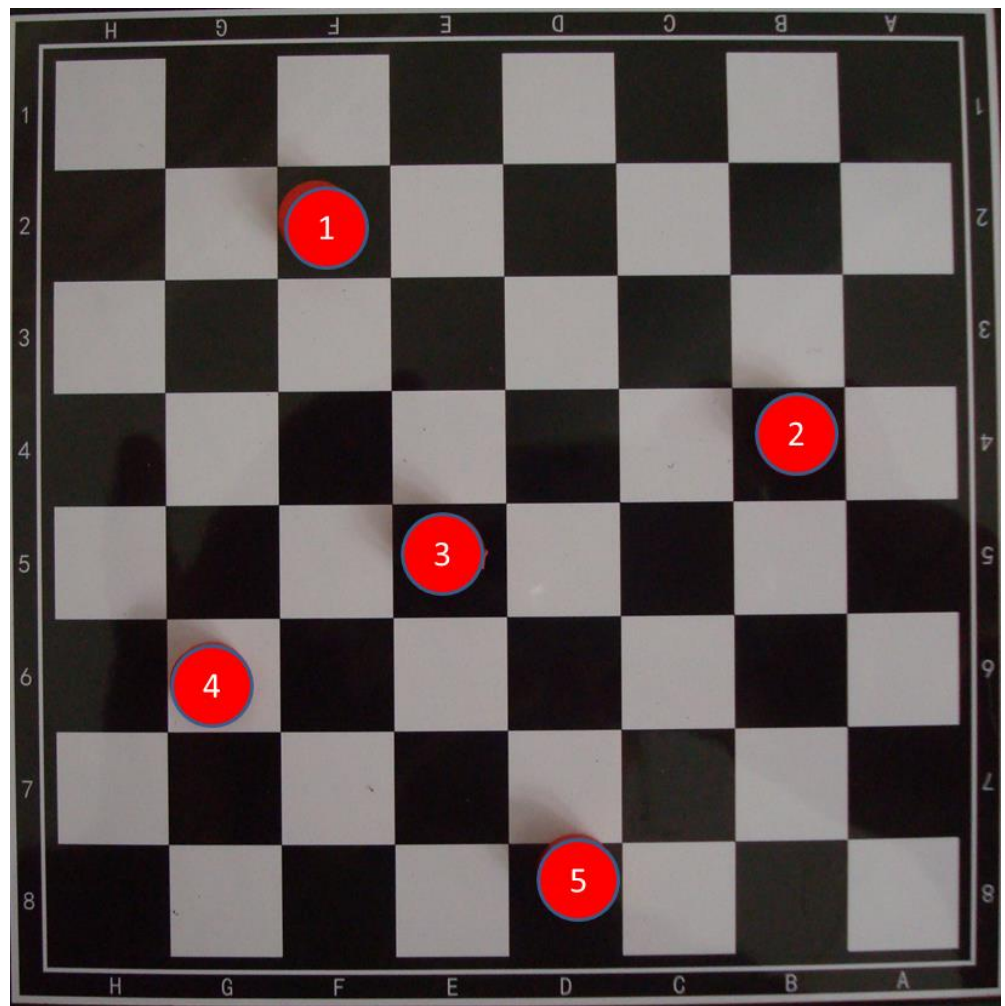
```
%% nodes:        21
```

```
%% failures:     8
```

```
%% restarts:     0
```

```
%% peak depth:   3
```

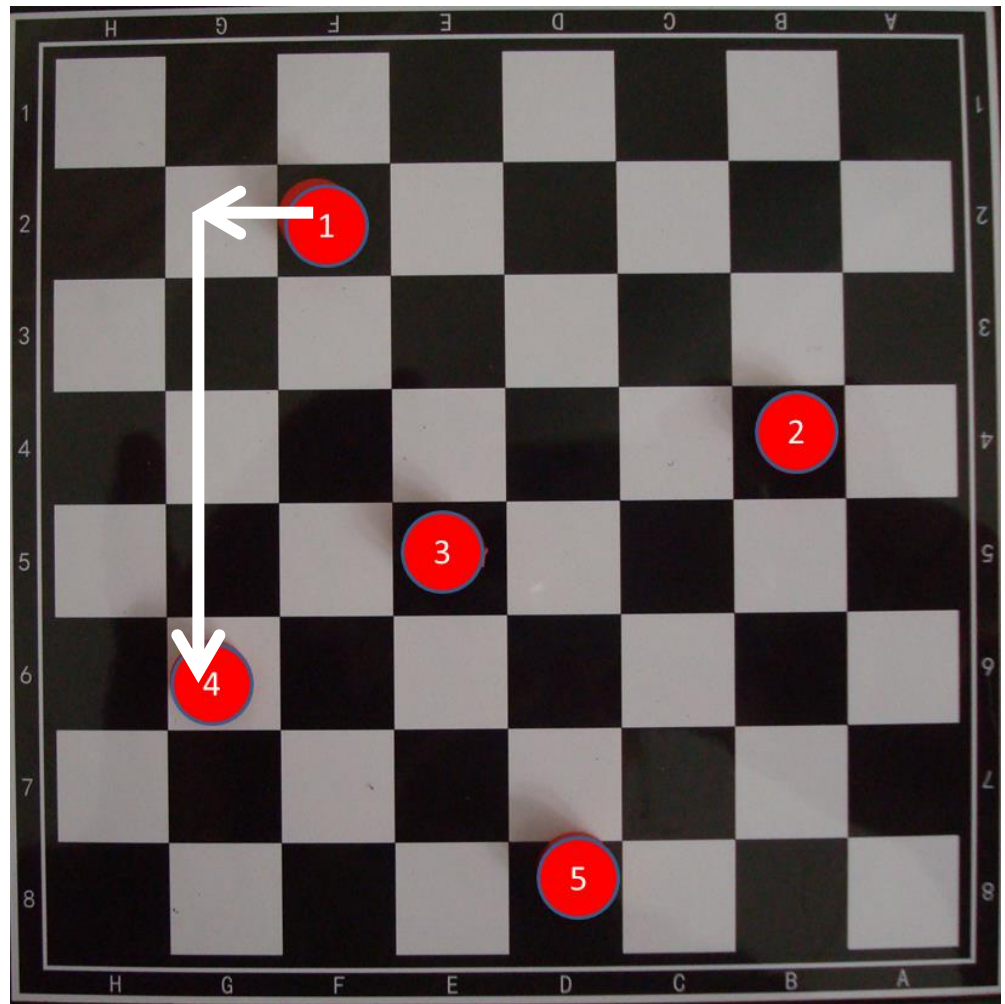
```
Y:\public_html\cpM\minizincCPM\tsp>_
```



```
CA: Command Prompt

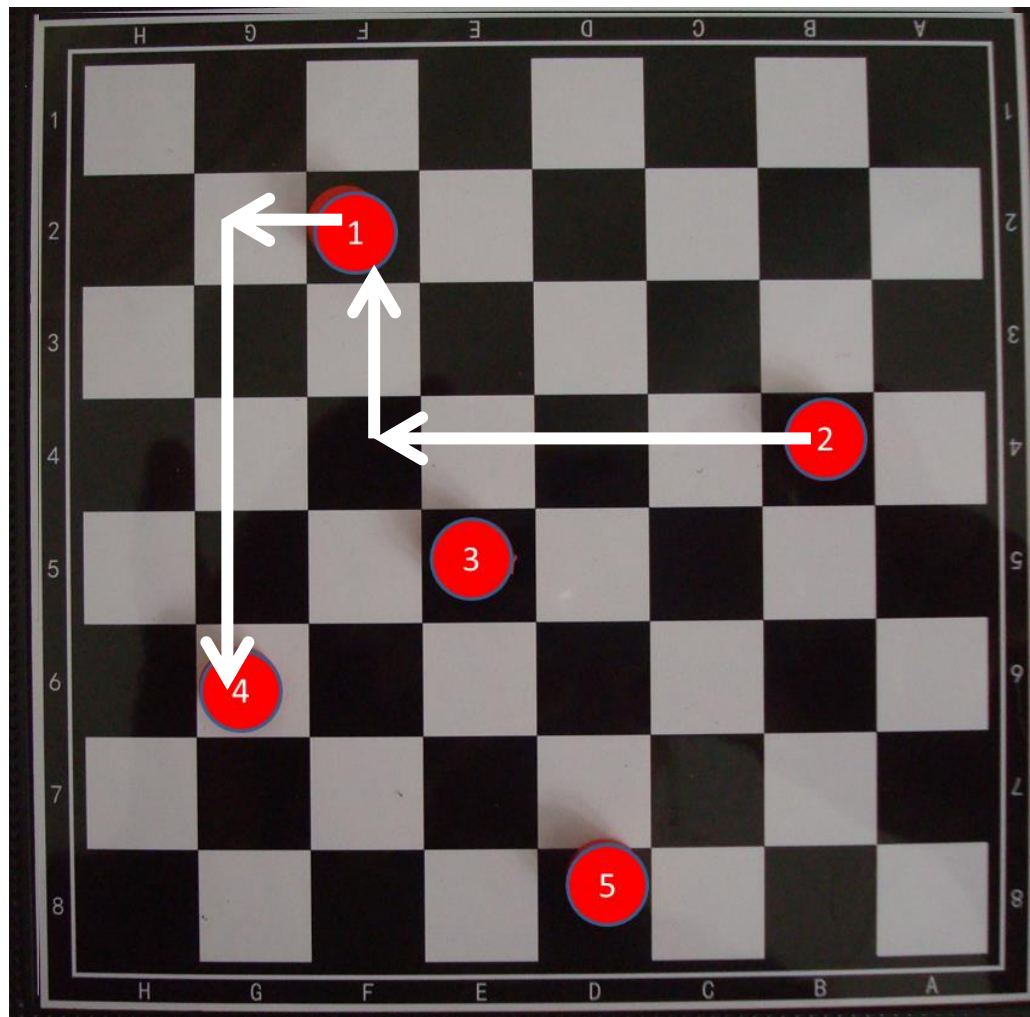
Y:\public_html\cpM\minizincCPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24   tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizincCPM\tsp>_
```

1 goes to 4



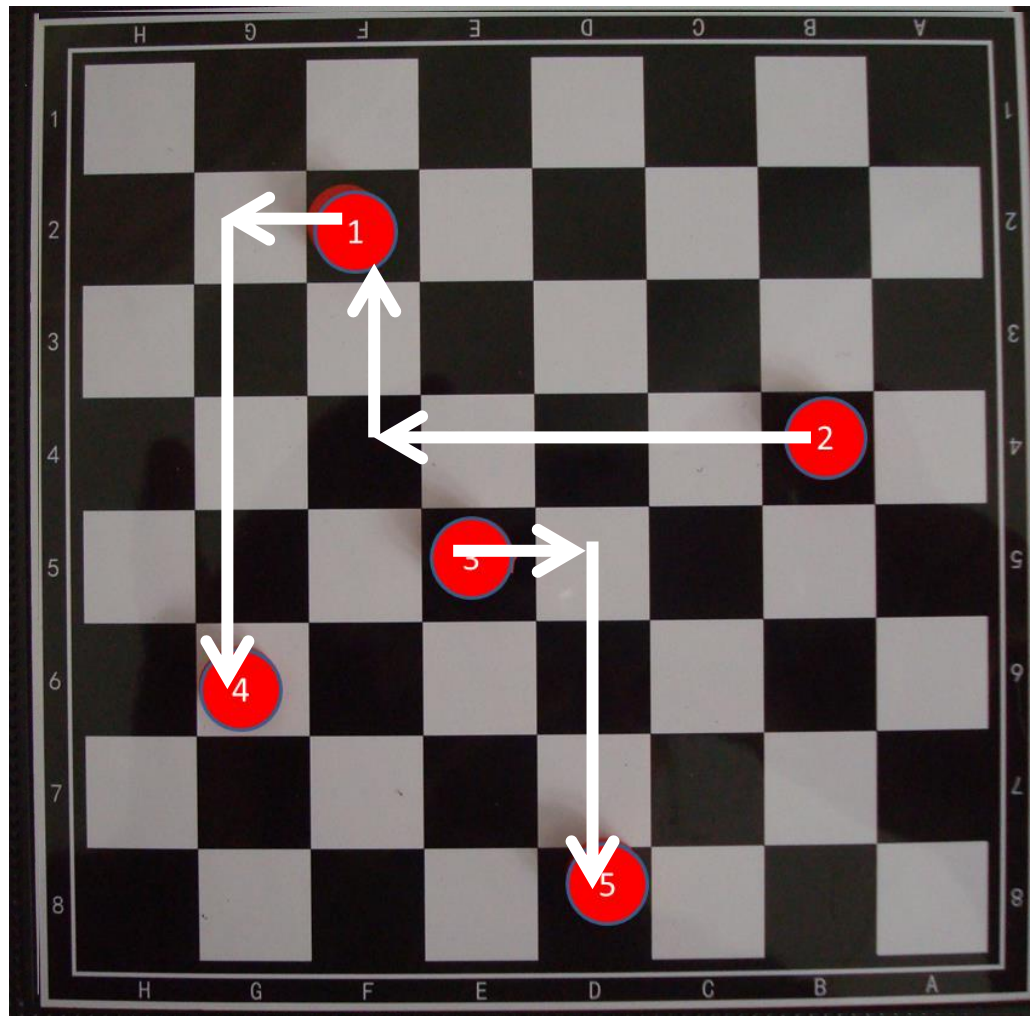
```
CA: Command Prompt
Y:\public_html\cpM\minizincCPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24   tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizincCPM\tsp>_
```

2 goes to 1



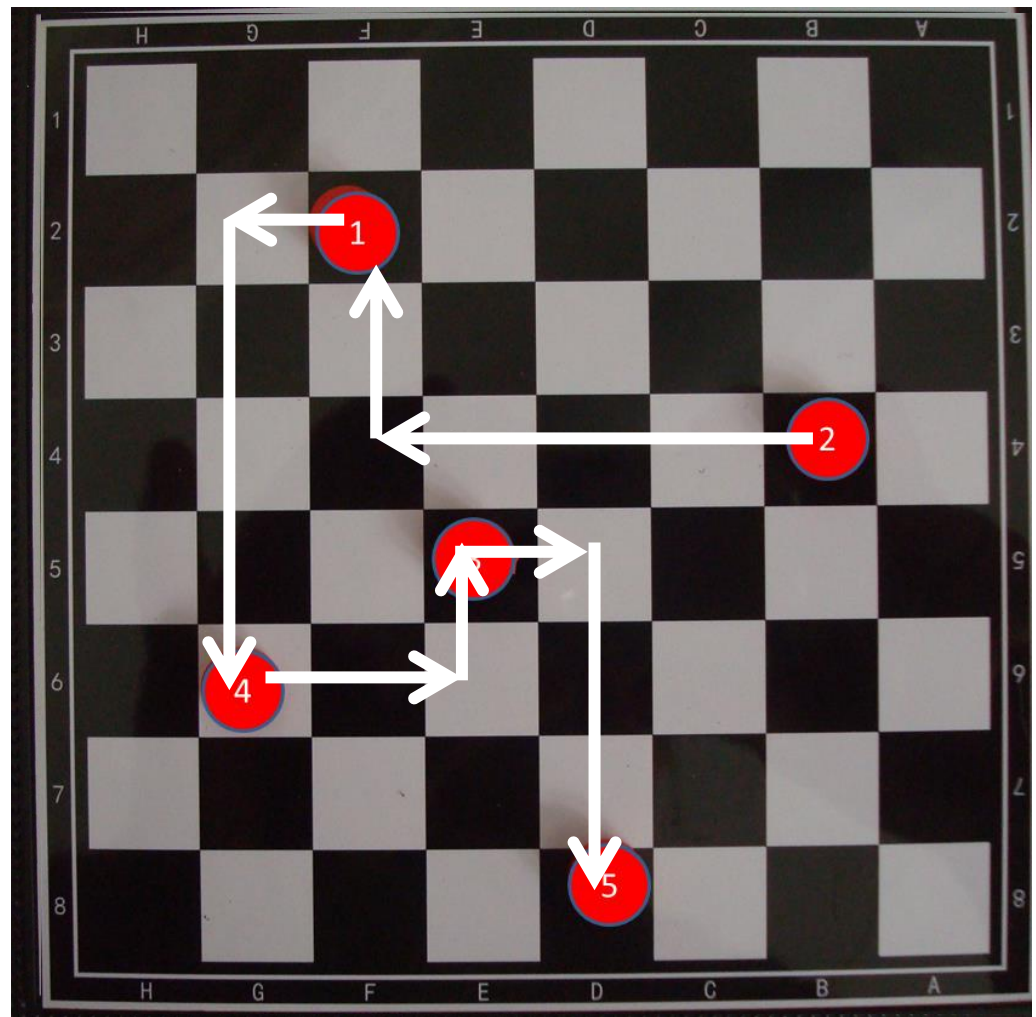
```
Y:\public_html\cpM\minizincCPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24   tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizincCPM\tsp>_
```

3 goes to 5



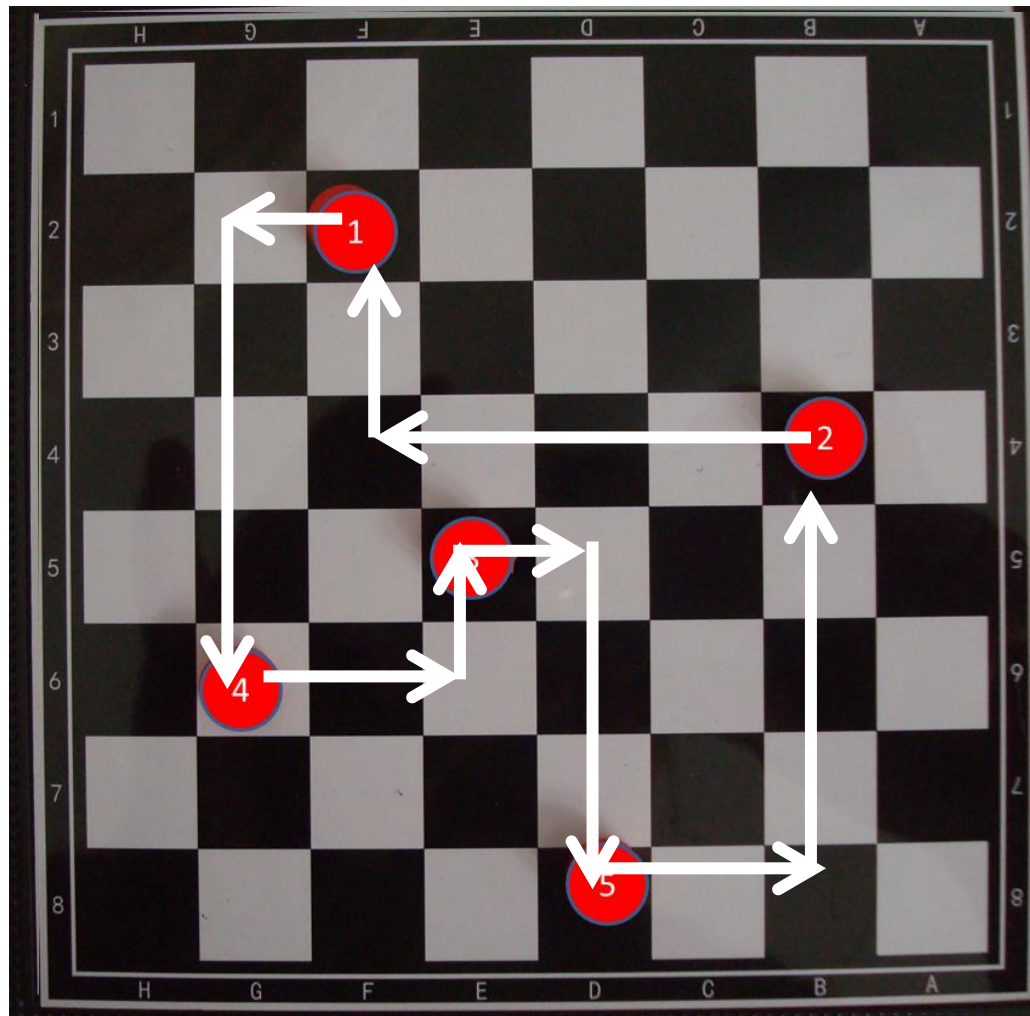
```
Y:\public_html\cpM\minizinc\CPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24   tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizinc\CPM\tsp>_
```

4 goes to 3



```
Y:\public_html\cpM\minizinc\CPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24  tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizinc\CPM\tsp>_
```

5 goes to 2



```
Y:\public_html\cpM\minizincCPM\tsp>mzn-gecode tsp.mzn p5.dzn
cost: 24   tour: [4, 1, 5, 3, 2]
=====
Y:\public_html\cpM\minizincCPM\tsp>_
```





Feic!