MATHEMATICS AND COMPUTATIONAL SCIENCES

Vol 6(1), 2025, pp:104-115
DOI:10.30511/mcs.2025.2041761.1234
ISSN:2717-2708
http://mcs.qut.ac.ir/

Qom University of Technology

Practical algorithm for the Beltway problem with a complexity parameter introduction

Reza Nadimi [1]
Department of Computer Science, Faculty of Mathematical Sciences, University of Mazandaran, Iran

Abstract: The Beltway problem is a famous computational geometry challenge relevant to fields like bioinformatics and crystallography. Not only is the complexity class of the Beltway problem not known but there is no practical non-polynomial time algorithm to solve it. This paper models the problem as a Constraint Satisfaction Problem (CSP) and applies a backtracking approach with forward-checking to explore the solution space. Moreover, it introduces a "multiplicity rate" parameter to measure the input's complexity, observing that higher multiplicity generally increases run-time. To the best of the author's knowledge, no deterministic algorithm exists that can solve large instances of the Beltway problem. Experimental results indicate that the algorithm performs efficiently for low-multiplicity instances and acceptably under higher multiplicities.

Keywords: Beltway; Reconstruction; Constraint satisfaction; Forward checking
2010 Mathematics Subject Classification: 51-08; 65L10; 65L12; 65L20; 65L70
Receive: 23 September 2024, **Accepted:** 17 January 2025

1 Introduction

When a computational problem perches between P and NP-complete classes, it is noticeable and challenging for computer scientists worldwide. The Beltway problem and its twin Turnpike problem are situated in such a position. There is neither a polynomial-time algorithm to solve these problems and classify them as a member of P, nor a polynomial-time reduction from some NP-hard problems to classify them as an NP-hard problem. In addition to their importance as a theoretical challenge in the complexity phase, concerning their applications in bioinformatics and crystallography, designing efficient algorithms for them has been in demand for recent decades.
In contrast to the Turnpike problem, for which several non-polynomial-time algorithms exist, no deterministic fast algorithm has been developed for the Beltway problem. Although the algorithm presented in this paper works for the Turnpike problem with little adoption, we will not consider it to focus on the Beltway problem.

The Beltway problem involves reconstructing a set of points on a circle from a given multiset of pairwise distances. We call the points on the circle "generating points" and the distances "fragments". Figure 1 shows the relation of generating points and distances.

[1] nadimi@umz.ac.ir

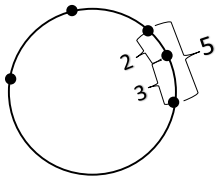

Figure 1: Generating points on a circle with pairwise distances

The Beltway problem is often mentioned alongside the Turnpike problem and there is no standalone study about it and there is no deterministic fast algorithm has been developed to solve it [1–6]. Lemke et al. [4] showed that the running time of their backtracking method for the Turnpike problem is $O(n^n logn)$ for the Beltway problem. In recent years via a series of related papers, Fomin investigated the Beltway problem in [1], [2], and [3]. Shuai H. et.al. considered this problem using a distance distribution [5].

In Section 2 we model the Beltway problem as a Constraint Satisfaction Problem (CSP), Section 3 is dedicated to the Algorithm and Section 4 presents the experimental results of the work.

2 Modeling

This section presents the Beltway problem modeled as a Constraint Satisfaction Problem (CSP). There is no CSP model for the Beltway problem already. In CSP literature, decisions are represented as variables, possible values for variables are domains, and the rules governing variable arrangements are constraints. An instance of the Beltway problem is a multiset of $m = n(n-1)$ real numbers and the goal is to find n points on a circle in which the pairwise distances of these n points are equal to the given multiset. Note that each pair of points on the circle gives us two distances (divide the circle into two fragments) therefore we have $n(n-1)$ fragments generated by n points.

The key Idea of this modeling is mapping an interval $[0, L)$ to the points on the circle. In this mapping, L is the supposed circle's circumference equal to $min + max$ (sum of minimum and maximum values in the given numbers). We can map any arbitrary point on the circle to 0 and the remaining points to $(0, L)$ concerning the distances from 0. The mapped numbers to the points are increasing clockwise.

This mapping makes it possible to determine a finite set of points on the circle as the candidates for generating points. Without losing any solution, we assume that the minimum distance on the circle is between points 0 and min (min is the length of the smallest distance). Figure 2 presents an instance of the Beltway problem with its initial circle at the beginning of the algorithm.

Now two of the n points on the circle (0 and min) are known. Any other point on the circle generates two distances from point 0; one of the distances equals the number mapped to that point and the other equals the circle's circumference minus the first distance. Therefore, the remaining generating points on the circle which we look for, are some numbers belonging to the given values. Our algorithm chooses some members of the given numbers such that the pairwise distances of the corresponding points on the circle are equal to the given multiset.

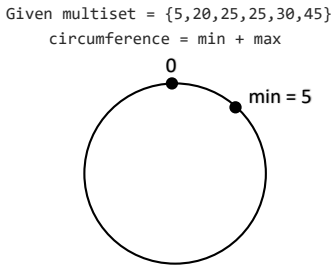

Figure 2: mapping 0 and min to two points of a circle with circumference = $min + max$.

2.1 Variables and domains

Since the goal is to select some of the members of the given instance to reconstruct the points on the circle, the straightforward way to define variables of the model is to consider a variable for each member of the instance and give it proper value e.g. 1 and 0 for selecting or not selecting of it. In our CSP model, we define the variables differently to solve the problem more efficiently. Before the definition of variables in our CSP model, it is important to note that any instance of the problem with some solution, can be considered as a set of pairs (a, b) such that, for some real numbers p and q (points on the circle) $a = q - p$ and $b = circumference - (q - p)$. In this case we say (p, q) generates (a, b).

Each member of any instance with some solution, belongs to one and only one pair. Therefore, an instance of the problem containing a non-paired member has no solution. For a given (a, b) only a finite number of candidates (p, q) exist to generate it as a pair of distances between p and q on the circle because p and q are restricted to the members of the instance. Concerning the multiplicity of the members of the instance, for a pair (a, b) with r replications of a and b in the given values, we should locate r distinct pair (p, q) on the circle, one for each (a, b) to generate it. If we have r copy of (a, b) in the instance and k distinct pairs of (p, q) generating (a, b), The number of choices equals $\binom{r}{k}$. We will use these notations for multiplicity of pair (a, b) and the set of candidates generating pair(a, b):
$m(a, b)$ = multiplicity of pairs in the given instance. $C(a, b) = \{(p, q) \mid (p, q)\ generates\ (a, b)\}$.

Now we can define the variables in our CSP model. We define a variable v_{pq} corresponding to any pair (p, q) of the instance in which (p, q) is a candidate to generate a pair (a, b) from the instance. For all variables, the domain is the same and equals $\{0, 1\}$. $v_{pq} = 1$ when both p and q are generating points on the circle and $v_{pq} = 0$ otherwise. We should choose proper values from domains such that the corresponding pairs to the variables with value 1, generate all members of the instance. In the next section, we restrict the value of variables by introducing the constraints of our model to impose the necessary and sufficient conditions of the Beltway solution to the CSP model.

2.2 Constraints

In the previous section we saw that any variable v_{pq} has domain $\{0, 1\}$ and is defined as follow:

$$v_{pq} = \begin{cases} 1 & \text{if both of } p \text{ and } q \text{ are generating points on the circle} \\ 0 & \text{otherwise} \end{cases}$$

Table 1: Important variables and parameters in the algorithm

Variable/parameter	meaning
m	the size of input
n	number of generating points ($m = n(n-1)$)
(a, b)	a pair of given values that should be generated by two points on the circle
(p, q)	a pair of points on the circle that can generate a pair of given numbers (p and q themselves belong to the given values)
min	minimum value in the input
max	maximum value in the input
$circum$	circumference of the circle (= $min + max$)
$partial\_assignment$	assignment of 0 and 1 to some of variables.
$One\_variables$	set of variables with value 1
$Zero\_variables$	set of variables with value 0
$None\_variables$	set of unassigned variables
SP	the set of selected points on the circle corresponding to $One\_variables$
CP	the set of candidate points with respect to $None\_variables$ that are not in SP
$consist(p)$	the number of members of $SP \cup CP$, which are consistent with p.
$m(a, b)$	multiplicity of (a, b) in the given instance
$one(a, b)$	number of pairs in $One\_variables$ which generate (a, b)
$C(a, b)$	set of candidates to generate (a, b)

Now we introduce the constraints of our model which are necessary and sufficient conditions for the value of variables to make a solution.

Constraint 1: If there are r copies of a pair (a, b) in the instance, and the candidate set for generating these pairs is $C(a, b) = \{(p_1, q_1), (p_2, q_2), ..., (p_k, q_k)\}$, then any solution must satisfy:

$$\sum_{i=1}^{k} v_{p_i,q_i} = r.$$

In our model, $v_{p,q} = 1$ means p and q are selected as generating points. Now we define the set of selected points as follows:

$$S = \{s |\ s = p, q\ for\ all\ v_{p,q} = 1\}$$

Constraint 2: If the size of the instance is equal to $m = n(n-1)$, then any solution of the CSP should satisfy in the constraint $|S|$ = n.

Now the ingredients of our CSP model has completed as follows:

1- We have a variable $v_{p,q}$ for each pair (p, q) of given numbers that generates another pair (a, b).

2- The domain of all variables are the same and equals $\{0, 1\}$.

3- We have constraint 1 and constraint 2 as the set of constraints.

Any assignment of values from domains to variables satisfying constraints will be a solution for CSP and S gives us the desired solution to the Beltway problem. In next section we present our algorithm to solve CSP.

3 Algorithm

Two primitive algorithms are known to solve the constraint satisfaction problems. One of them starts with a complete assignment to the variables violating constraints and via iterations of the algorithm reduces inconsistency to reach a "consistent complete solution". Another method starts with a partial assignment to the variables maintaining consistency with constraints and uses a backtracking method to find a complete assignment to reach a "complete consistent solution". This section presents a backtracking algorithm using some filtering heuristics to solve the CSP model of the Beltway problem. In the previous section, we detected 0 and min, two points of the solution. Therefore we have $par\_assign = \{v_{0,min} = 1\}$ as our initial partial assignment.

In each iteration, the algorithm performs the following steps:
1- select a variable.
2- assign it a value.
3- filter the domains of related variables based on the partial solution.
4- run goal test to diagnose if we have solution.
5- check the partial assignment in the constraints, go ahead for promising partial solutions, and backtrack for non-promising ones.

We will explain each step in detail with related functions and definitions and then at the end of this section we gather all in the algorithm 4. At this point, we present some needed definitions and functions.

Although the solution to the Beltway problem is a set of points on the circle, the solution of our CSP model is an assignment of 0 and 1 to the defined variables. Considering this difference and correspondence between selected points S and variables with value 1 is important. In the previous section, we mentioned this correspondence by definition of the "selected points" S.

Definition 3.1 (Consistency). *Two points p and q on the circle are consistent iff (p, q) generates some pair (a, b). In other words, p and q are consistent iff two distances between these points on the circle, belong to the given instance.*

In Table 1, the main variables and parameters of the algorithm are defined. Also, in Table 2 the algorithms of four main functions are explained.

3.1 Choosing a variable and value

Choosing variables and assigning values in a proper order is very important in solving CSP models. In our algorithm, in each iteration, we choose a variable from the candidates $C(a, b)$ of a pair (a, b) that is chosen by the following formula:

$$(a, b) = argmin\{m(a, b) - one(a, b) \mid (a, b) \in input\}$$

After finding (a, b) using above formula, we choose an arbitrary variable $v_{p,q}$ that $(p, q) \in C(a, b)$.
All variables are binary and there are only two choices. We set the value 1 before 0 for all variables to make a high probability of promising.

3.2 Forward checking

Forward checking restricts the domains of unassigned variables based on the current assignment and associated constraints, ensuring consistency at each step. Since all variables are binary, filtering of the

Table 2: main functions of the algorithm

function	instruction
$Remove\_candidate(p)$	CP = CP - $\{p\}$ $v_{p,q} = 0$ for all q
$Select\_candidate(p)$	$CP = CP - \{p\}$ $SP = SP \cup \{p\}$ $v_{p,q} = 1$ for all $q \in SP$
$Assign\_One(p,q)$	$v_{p,q} = 1$ if $p \notin SP$ then $Select\_candidate(p)$ if $q \notin SP$ then $Select\_candidate(q)$
$Assign\_Zero(p,q)$	$v_{p,q} = 0$ $consist(p) = consist(p) - 1$ $consist(q) = consist(q) - 1$

domain of a variable is equivalent to assigning a value to that variable. Therefore, assigning a value for a variable may cause the assignment to some other variables. Sometimes these sequential assignments make solution very quickly without a huge number of backtracking. This section explains the forward checking function of our algorithm.
Observation: generating points on the circle should be consistent. Therefore, if the number of generating points is equal to n, a point must be consistent with at least $n-1$ members of $SP \cup CP$ to be an eligible candidate for the generating points:

$$\forall p \in Solution : consist(p) \geq n-1$$

We will use following filtering rules in the forward checking step:
1- Remove inconsistent members of CP with newly selected point.
2- Remove members of CP with consistency number less than $n-1$.
3- If for some $s \in SP$, $consist(s) = n-1$ then $Select\_candidate(p)$ for all points p consistent with s.
Since there are some interactions between the main functions, running any one of the filtering rules, cause the sufficient conditions to execute the other rules. Algorithm 1 illustrates the steps of forward checking.

3.3 Goal testing and promise checking

In this section, we define two test functions of the algorithm. The first function is "$is\_complete()$" and detects if an assignment is complete. A partial assignment is complete when $|SP| = n = (1 + sqrt(1 + 4m))/2$ and the pairwise distances of the members of SP equals the input.

The second test function is "$is\_promising()$" and detects not-promising partial assignments to make a backtracking in the algorithm. For this function, we consider constraints of the CSP model. A partial assignment is promising if don't violate constraints. Algorithm 2 and Algorithm 3 explain these functions in detail.

Algorithm 1 forward Checking()

```
change = true
while change do
    change = false
    for c ∈ CP do
        if (c is inconsistent with some members of SP) or (consist(c) < n − 1) then
            Remove_candidate(c)
            change = true
        end if
    end for
    for s ∈ SP do
        if consist(s) == n − 1 then
            for c ∈ CP do
                if c is consistent with p then
                    Select_candidate(c)
                    change = true
                end if
            end for
        end if
    end for
end while
```

Algorithm 2 $is\_complete()$

```
if |SP| == n  and  ∀(a, b) ∈ input :  m(a, b) = one(a, b)) then
    return true
end if
return false
```

Algorithm 3 $is\_promising()$

```
if |SP| > n then
    return false
end if
if ∃(a, b) :  (m(a, b) > one(a, b) + |C(a, b)|  ||  m(a, b) < one(a, b)) then
    return false
end if
if |SP| + |CP| < n then
    return false
end if
if ∃p ∈ SP : consist(p) < n − 1 then
    return false
end if
return true
```

Algorithm 4 Backtracking

Require: A set of distinct pairs (a, b) and their multiplicity $m(a, b)$
Ensure: SP = generating points

$state = \{$
  $partial\_assignment = \{v_{0,min} = 1\}$
  $CP = \{c |\ c \in input$ and c is consistent with 0 and $min\ \}$
  $SP = \{0, min\}$
  $None\_variables = \{v_{c,s} | c \in CP\ and\ s \in SP\}$
  $C(a, b) = \{(p, q) | (p, q)$ generates (a, b) and $p, q \in CP \cup SP\}$
  $Constraints = \{|SP| = n,\ \forall (a, b) :\ \Sigma_{(p,q)\in C(a,b)} v_{p,q} = m(a, b)\}$
  $\}$
function $recursive\_csp(state)$:
  if $is\_complete()$ **then**
    return SP
  end if
  if $not\ is\_promising()$ **then**
    return "$failure$"
  end if
  $(p, q) = choose\_variable()$
  $Assign\_One(p, q)$
  $Forward\_chacking()$
  if $is\_promising()$ **then**
    $result = recursive\_csp(state)$
    if $result\ != "failure"$ **then**
      return $result$
    end if
  end if
  $Assign\_Zero(p, q)$
  $Forward\_chacking()$
  if $is\_promising()$ **then**
    $result = recursive\_csp(state)$
    if $result\ != "failure"$ **then**
      return $result$
    end if
  end if
  return "$failure$"
end function

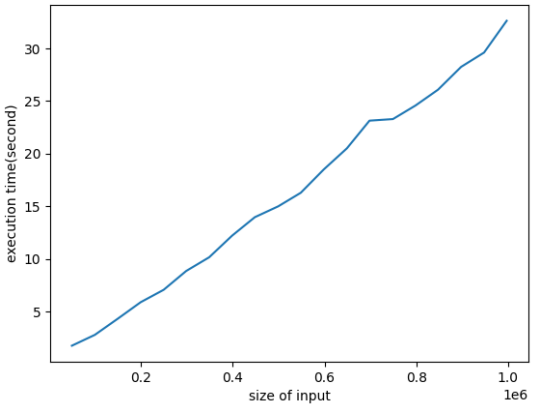

Figure 3: Execution time for instances with $mr \approxeq 0$

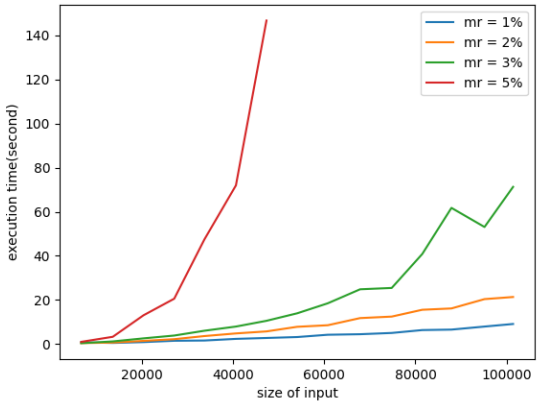

Figure 4: Execution time for instances with low multiplicity rate

4 Experiments and Results

In this section, we investigate the running time of the algorithm on several instances with different sizes and difficulties. We use the programming language Julia on a Laptop with Processor AMD Ryzen 5 2500U and 8 GB of RAM.

The algorithm's running time is strongly influenced by the multiplicity of values in the input set. Higher multiplicity generally leads to slower execution times, while lower multiplicity allows for faster processing. To measure the multiplicity and assess its effect on the running time, we define the parameter mr (multiplicity rate) for input as follows:

$$mr = \sum_{(a,b)\in input} (\frac{m(a,b)-1}{|input|})$$

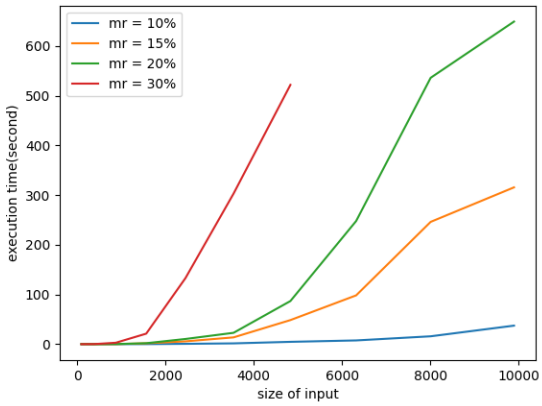

Figure 5: Execution time for instances with high multiplicity rate

This definition means that in a sample with multiplicity rate mr, the mr portion of the given values are copies of the other values. mr is a real number in $[0, 1)$. If all of the values are distinct in the input then $mr = 0$, and for these samples the algorithm runs in approximately linear time. In the figure 3 the result of experiments on data with different sizes and $mr \approxeq 0$ is presented.

Note that making random samples with exact mr is not easy and therefore we make samples with approximately mr rate of multiplicity in our experiments. This approximation is less than 0.2. The result of experiments on samples with a low rate of multiplicity is shown in figure 4 and figure 5 presents the result for samples with a high rate of multiplicity. In these figures, the x axis shows the real size of the input (not the number of generating points). Also, the random samples are generated for some n that the distances between the input sizes approximately are equal.

In addition to the running time, the number of nodes in the search tree gives a good intuition of the algorithm's behavior. The complete information of the experiments including number of generating points and nodes on the search tree are presented in the figure 6 and figure 7.

All data of running times and number of nodes in the search tree are the average of 10 experiments. In the tables NE means for related cases there is "No Experiment". Experiments show the power of the algorithm to solve the samples with a large size and small mr quickly. Also, for the samples with a high rate of multiplicity, the algorithm runs in an acceptable time for small sizes. It is important to know that the multiplicity can be greater than what we experimented with here and the algorithm runs slower than what we can make enough experiments to track the behavior.

5 Conclusion

In this paper, a CSP model and a backtracking algorithm are presented for the Beltway problem. Although the effect of a multiplicity of values is a known factor in the running time of the reconstruction problems [4], the parameter mr that is initiated in this paper is the first parameter considering the multiplicity rate explicitly in the related literature.

Experimental results demonstrate that the algorithm performs efficiently when the multiplicity rate is low and maintains acceptable running times even at higher multiplicity rates, particularly given the lack

Input size	Number of points	mr = 1%		mr = 2%		mr = 3%		mr = 5%	
		time	nodes	time	nodes	time	nodes	time	nodes
6642	82	0.8	2	0.45	2	0.42	3	1.02	2.0
13572	117	0.53	2	0.7	2	1.22	3	3.31	2.2
20306	143	0.91	2	1.46	2	2.6	2	13.05	3.4
27060	165	1.51	2	2.23	3	3.88	3	20.58	3.5
33672	184	1.62	2	3.64	3	6.05	3	47.4	4.9
40602	202	2.37	2	4.84	3	7.95	3	71.93	5.5
47306	218	2.78	2	5.75	3	10.56	3	146.65	7.6
54056	233	3.2	2	7.84	3	13.96	3	NE	NE
60762	247	4.24	2	8.55	3	18.49	3	NE	NE
67860	261	4.48	2	11.79	3	24.84	4	NE	NE
74802	274	5.05	2	12.47	3	25.45	3	NE	NE
81510	286	6.36	2	15.56	3	40.85	4	NE	NE
87912	297	6.57	2	16.16	3	61.74	5	NE	NE
95172	309	7.95	2	20.39	3	53.08	5	NE	NE
101442	319	9.12	2	21.35	3	71.26	6	NE	NE

Figure 6: Number of nodes on the search tree and execution time of the algorithm for easy instances

Input size	Number of points	mr = 10%		mr = 15%		mr = 20%		mr =30 %	
		time	nodes	time	nodes	time	nodes	time	nodes
90	10	0.13	3	0.19	4	0.12	4	0.02	5
380	20	0.02	2	0.03	3	0.05	8	0.21	7
870	30	0.05	3	0.13	7	0.27	16	2.91	17
1560	40	0.16	4	1.79	18	1.98	26	21.22	36
2450	50	0.74	6	5.7	28	10.51	48	133	278
3540	60	1.63	13	13.71	45	22.89	62	303	317
4830	70	4.74	13	48.52	64	86.87	100	522	354
6320	80	7.57	17	98.25	83	247.88	139	NE	NE
8010	90	15.9	22	246.2	125	536.22	169	NE	NE
9900	100	37.35	30	315.66	144	649.33	204	NE	NE

Figure 7: Number of nodes on the search tree and execution time of the algorithm for hard instances

of efficient algorithms for the Beltway problem.

References

[1] Fomin E., A simple approach to the reconstruction of a set of points from the multiset of $\binom{n}{2}$ pairwise distances in n^2 steps for the sequencing problem: I. theory,J. Comput. Biol., vol 23(9), pp. 769–775, 2016.

[2] Fomin E., A simple approach to the reconstruction of a set of points from the multiset of $\binom{n}{2}$ pairwise distances in n^2 steps for the sequencing problem: II. algorithm, J. Comput. Biol.,vol 23(12), pp. 934–942, 2016.

[3] Fomin E., A simple approach to the reconstruction of a set of points from the multiset of $\binom{n}{2}$ pairwise distances in n^2 steps for the sequencing problem: III. noise inputs for the beltway case, J. Comput. Biol., vol 26 (1), pp. 68–75, 2019.

[4] Lemke, P., Skiena, S.S., Smith, W.D. Reconstructing Sets From Interpoint Distances. Discrete and Computational Geometry. vol 25, 2003. Springer, Berlin, Heidelberg.

[5] Shuai Huang and Ivan Dokmanic, Reconstructing Point Sets from Distance Distributions, IEEE Transactions on Signal Processing, vol. 69(1), pp. 1811–-1827, 2021.

[6] Skiena, S. S., Smith, W. D. and Lemke P., Reconstructing sets from interpoint distances, presented in Proceedings of the 6th SoCG, New York, NY, USA,, pp. 332–339, 1990.