Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi
Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering

Basılı/Printed ISSN: 1302-9304. Elektronik/Online ISSN: 2547-958X

A Progressive Search Algorithm for the Minimum Hitting Set Problem

Minimum İsabet Kümesi Problemi için Aşamalı Arama Algoritması

Hilal Arslan [1*], Onur Uğurlu[2], Vahid Khalilpour Akram [3], Deniz Türsel Eliiyi[4]

[1] Ankara Yıldırım Beyazıt University, Department of Software Engineering, Ankara, TURKEY
[2] İzmir Bakırcay University, Department of Fundamental Sciences, Izmir, TURKEY
[3] International Computer Institute, Ege University, Izmir, TURKEY
[4] İzmir Bakırcay University, Department of Industrial Engineering, Izmir, TURKEY
Sorumlu Yazar / Corresponding Author *: hilalarslanceng@gmail.com

Geliş Tarihi / Received: 05.09.2020 *Araştırma Makalesi/Research Article*
Kabul Tarihi / Accepted: 21.02.2021 *DOI:10.21205/deufmd.2021236914*
Atıf şekli/ How to cite: ARSLAN H.,UGURLU O.,AKRAM V. K., TURSEL ELIIYI D.(2021). A Progressive Search Algorithm for the Minimum Hitting Set Problem. DEUFMD 23(69), 867-874.

Abstract

Given a finite universe and a collection of the subsets of the universe, the minimum hitting set of the collection is the smallest subset of the universe that has non-empty intersection with each set in the collection. Finding the minimum hitting set is an NP-Hard problem that has many real world applications. In this study, we propose a progressive search-based approach to find the minimum hitting set of a given collection. The algorithm starts searching for the hitting sets of size 1 and increase the expected size of the minimum hitting set by a factor of *d*. After each unsuccessful search, it increases the expected size by *d* and generate the candidate sets with the expected size. After each successful search, the algorithm takes the average of last unsuccessful and successful searches and continue the searching with the new expected size. The algorithm terminates when the detected upper bound coincides with the detected lower bound. The effect of different values for *d* on the performance of the algorithm has been experimented on various data sets. Experimental results reveal that the proposed method effectively computes the minimum hitting set on real-world data and random dataset.

Keywords: *Minimum Hitting Set, NP-Hard Problems, Progressive Search*

Öz

Sonlu bir evren ve evrenin alt kümelerinin bir birleşimi verildiğinde, birleşimin minimum isabet kümesi, birleşimdeki her kümeyle boş olmayan kesişimi olan evrenin en küçük alt kümesidir. Minimum isabet kümesini bulma, birçok gerçek dünya uygulaması olan bir NP-Hard problemidir. Bu çalışmada, verilen bir birleşimin minimum isabet kümesini bulmak için aşamalı arama tabanlı bir yaklaşım öneriyoruz. Algoritma, boyutu 1 olan isabet kümelerini aramayla başlar ve minimum isabet kümesinin beklenen boyutunu *d* faktörü kadar artırır. Her başarısız aramadan sonra, algoritma beklenen boyutu *d* kadar artırır ve beklenen boyuta sahip aday kümeleri oluşturur. Her başarılı aramadan sonra, algoritma son başarısız ve başarılı aramaların ortalamasını alır ve yeni beklenen boyutla aramaya devam eder. Algılanan üst sınır, algılanan alt sınırla çakıştığı zaman algoritma sona erer. *d* faktörünün farklı değerlerinin algoritmanın performansı üzerindeki etkisi çeşitli veri kümeleri üzerinde denenmiştir. Deneysel sonuçlar, önerilen yöntemin gerçek dünya verileri ve rasgele veriler üzerindeki minimum isabet kümesini etkili bir şekilde hesapladığını ortaya koymaktadır.

Anahtar Kelimeler: *Minimum İsabet Kümesi, NP-Hard Problemler, Aşamalı Arama*

1. Introduction

The minimum hitting set (MHS) problem is one of the NP-Hard optimization problems that has a wide range of applications in different real world problems such as Boolean algebra [1], model based diagnosis [2], computational biology [3], networking [4], and data mining [5]. In addition, many other problems can be reduced to the MHS problems. For example, transversal hypergraph problem [6], set cover problem [7], and independent set problem [8] are three important combinatorics problems that can be reduced to MHS. Given a universe U of items and a finite collection $C=\{C_1, C_2, C_3, \dots, C_m\}$ of the subsets of the universe, the minimum hitting set H is the smallest subset of the universe that has non-empty intersection with each set $C_i \in C$.

For example, let $U = \{a, b, c, d, e, f, g, h, s, t\}$ be a set of nodes in a computer network and assume that $C = \{\{bgch\}, \{aefch\}, \{gebh\}, \{abh\}, \{adcf\}\}$ are all available communication paths between nodes n and t. In this case, the minimum hitting set of C will be the set of critical nodes used in all paths between nodes n and t. For this example, the minimum hitting set of C is $H=\{a, h\}$ since the intersection of H with each set in C contains at least one element. Therefore, if nodes a and h stop working for some reason, all paths between n and t will be disconnected. As another example, suppose that a complex system consists of several components, where each part consists of several building blocks that are commonly used in different components. In case of any error in such a system, we can create a collection of components that are working incorrectly and find the hitting set of this collection to detect the broken block faster. For example, let the building blocks of the system is represented by $U = \{a, b, c, d, e, f, g, h\}$, and the set of components which are not working properly is defined by $C = \{\{c, d, g, h\}, \{a, b, g\}, \{e, f, h\}, \{e, d\}\}$. The building blocks in the minimum hitting set of C, which is $H=\{e, g\}$, are the first devices that need to be checked because these devices are used commonly in all malfunctioning parts. Figure 1 shows a graphical representation of collection $C = \{\{c, d, g, h\}, \{a, b, g\}, \{e, f, h\}, \{e, d\}\}$ and its minimum hitting set.

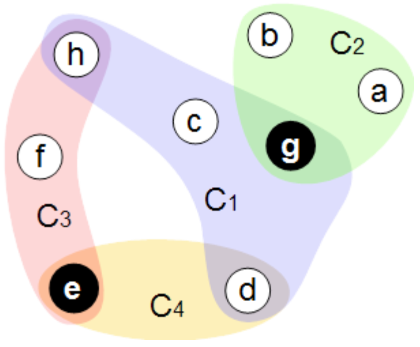

Figure 1. The MHS of $C = \{C1,C2,C3,C4\}$ is $H=\{e,g\}$

Finding or generating the minimal hitting sets has been the subject of many research studies, and different heuristic-based and approximation algorithms have been proposed for these problems. These algorithms can be grouped in two categories, which are tree-based and evolutionary algorithms. Reiter [9] introduced the first algorithm to provide an idea how to find the minimum hitting sets in 1987. However, the algorithm consumed too much time. Wotawa [10] introduced a variant of the Reiter's hitting set algorithm by reducing the graph. Pill and Quaritsch [11] used Boolean algebra to search the minimum hitting sets efficiently. Their methods greatly reduce the number of computations. Other algorithms have also been proposed for the problem [12, 13]. Although these algorithms compute the minimum hitting sets in a full accuracy, they are not suitable for computing the minimum hitting sets on a large scale. The algorithms generally create a large number of nodes and take a lot of run time to traverse all nodes, which is the main disadvantage.

Evolutionary algorithms have been proposed to compute the minimum hitting set to overcome these difficulties. Genetic algorithms and their variations [14], particle swarm optimization [15] and improved differential evaluation algorithm [16] were initially proposed to compute the minimum hitting set using a fitness function. The hybrid versions of these algorithms [17] and parallel hybrid algorithms are also proposed [18]. However, all of these algorithms have a major drawback in the sense that they may not guarantee exact minimum hitting sets.

In this paper, we propose a progressive search based algorithm for the MHS Problem. The proposed algorithm starts from the smallest candidates for MHS and checks if any hitting set with the selected size exists in the collection. After each unsuccessful search, the algorithm increases the size of the candidate hitting sets by a factor of *d* and repeats the search for the new size. If a hitting set is detected, the algorithm takes the average of the MHS size in the last unsuccessful search and the MHS size in the current search, and it repeats the search with the new size until it finds the exact hitting set. The proposed algorithm efficiently computes the minimum hitting set on large scales especially when cardinality of the hitting sets is small, consistent with real world applications.

The remaining parts of this paper have been organized as follows: Section 2 presents a formal definition of the problem and the required background. Section 3 includes the details of the proposed algorithm. Section 4 includes algorithm performance evaluation, and finally section 5 presents our conclusions.

2. Problem Formulation

Let $|U|$ denote the number of items in universe U, i.e. n, and $m=|C|$ denote the number of sets in C. Given a universe U of the items x_i where $i=\{1,...,n\}$ and the set C_j where $j=\{1,...,m\}$, the integer linear programming model of the minimum hitting set problem can be formulated as follows [19]:

$$min \sum_{i=1}^{n} x_i$$

such that

$$\sum_{i \in C_j} x_i \geq 1, \qquad \text{for all } C_j \in C$$

$$x_i \in \{0,1\}, \qquad i = \{1,\dots,n\}$$

In the given formulation, the constraint ensures that every set C_j has at least an item x_i of the hitting set, and the objective function minimizes the size of the hitting set. Note that the notation $i \in C_j$, means that the set C_j includes item x_i.

3. Proposed Method

The proposed algorithm performs a progressive search to find the minimum hitting set of a given collection. Initially, the algorithm tries to find a hitting set of size 1. If there is no such hitting set, the algorithm increases the expected hitting set size by a factor of *d* and search for the hitting sets of size *d*. If a hitting set of size *d* is not detected then we definitely have no hitting set with size less than *d.* In this way we may avoid checking all search space and find the solution more faster. If a hitting set of size *d* is detected, the algorithm reduces the expected hitting set to the average of the last successful and the previous unsuccessful search. On the other hand, if the search for the hitting set of size *d* is unsuccessful, the algorithm increases the expected size by *d* again, and repeats the search. For example, for *d*=4, the algorithm searches for the hitting set of sizes 1, 4, 8, 12 and so on. If all searches up to size 8 is unsuccessful, and the procedure has found a hitting set of size 12, then the size of the minimum hitting set is a value between 8 and 12. In this case, a good strategy is to search for the average of the upper and lower bound size, which is (8+12)/2=10. If the sum of the lower bound and upper bound is odd we take the floor of the average. If we find a hitting set of size 10, we should check the size 9 to ensure that no hitting set of smaller size exists. However, if we cannot find a hitting set of size 10, then we should also check hitting sets of size 11. If no hitting set of size 11 is detected then the minimum hitting set size is exactly 12.

The steps of the proposed algorithm have been presented in Algorithm 1. The proposed *ProgressiveSearch* procedure accepts a progress factor *d*, the set collection *C*, the universe *U* and also an output variable *H* for holding the detected minimum hitting set. In this procedure, *k* is the size of the candidate hitting sets in the last unsuccessful search (lower bound), *p* is the size in the current search, *q* is the size in the last successful search (upper bound), *i* is the index of the next hitting set of size *p*, and *H* is the index of selected items from the universe for the candidate hitting set.

The algorithm calls the *GenerateNextSubset* procedure to get the next hitting set candidate of size *p*. This function generates a new set of size *p* based on the *i* value. If it can generate a candidate hitting set, the function returns a nonnegative

Algorithm 1: Minimum Hitting Set

```
1: procedure  ProgressiveSearch(d, C, U, H):
2:   k ← 1,  p ← 1, i ← -1,  q ← |U|, H ← [ ]
3:   do
4:      do i ← GenerateNextSubset(p, |U|, I, H)
5:      while i ≥ 0 and Intersect(H, C, U) < |C|
6:      if i <0 then
7:          k ←  p+1, H ← [ ]
8:          if d > 1 then
9:             if p+d <q  then p ← p+d
10:            else p ← (p+q)/2
11:         else p ← k
12:     else  if q > p then q ← p-1, p ← (k+q)/2
13:     else h ← |H|
14:  while h = 0
15:  return h

16: procedure  GenerateNextSubset(r,n,I, H[]):
17:  if i =-1 then
18:     for  i ← 0 to d do  H[i] ← i
19:     i ← r-1
20:  else if H[0] < n-r  then
21:     H[i] ← H[i] + 1
22:     while i < r-1 do
23:         H[i+1] = H[i] + 1
24:         i ← i+1
25:  else i ← -1
26:  while i > 0 and H[i] = n-r+i do  i ← I-1
27:  return i

28: procedure Intersect(H, C,U):
29:    e ← 0
30:    for each S ∈ C do
31:       if S ∩ U[H] ≠ ∅ then e ← e+1
32:     return e
```

value. Otherwise, it returns -1 to indicate that all candidate sets of size *p* have been generated. After generating a new candidate set in *H*, we check the intersection of *H* and every element in *C* using the *Intresect* procedure. The *Intresect* procedure returns the number of sets in *C* that have non empty intersection with *H*. Therefore, *Intresect* returns $|C|$ if *H* is a hitting set.

Generating new candidate sets of size *d* continues until finding a hitting set (Intresect returns $|C|$) or checking all possible sets of size *p* and getting a negative value in *i*. After terminating the *while* loop, if $i \geq 0$, then we have found a hitting set of size *p*. In this case, if $q>p$, we continue the search for finding a set of size $(k+q)/2$. Otherwise, we return *H* as the final minimum hitting set. If $i < 0$, then the size of the minimum hitting set is larger than *p*. In this case, we set *k* to the current *p* value to determine the lower bound and increase *p* by a factor of *d*. If increasing *p* by a factor of *d* exceeds *q*, we set *p* as the average of its current value and *q*.

Table 1. Properties of the Accident dataset

Name	$\lvert U\rvert$	$\lvert C\rvert$
ac200k	64	81
ac150k	64	447
ac130k	81	990
ac110k	81	2000
ac90k	336	4322
ac70k	336	10968
ac50k	336	32207
ac30k	442	135439

The *GenerateNextSubset* procedure accepts the set size *r*, the universe size *n*, the next set index *i* and generates a set in *H*. This procedure finds the *i*'th combination of size *r* from the set of size *n*. This function stores the index of the selected items in *H*. The *Intresect* procedure finds the intersection of every set in *C* with *H*, and returns the number of sets with non empty intersections. Set *H* contains the indices of the selected elements from *U* for the candidate hitting set.

4. Experimental Results

This section presents the results of the proposed algorithm for computing the minimum hitting sets on random and real-world data. We generate 240 random instances with respect to three parameters, which are the cardinality of universe $|U|$, the number of sets $|C|$, and the cardinality of the minimum hitting set $|H|$. The intervals for these parameters are given below:

- $|U|$ changes between 20 and 100
- $|C|$ changes between 2000 and 10000
- $|H|$ changes between 1 and 10

The real-world data, which is called as "accidents", present anonymized information about several hundred thousand accidents in Flanders during the period 1991–2000. This data set is retrieved from the FIMI repository [20]. The properties of the data set is given in Table 1.

All computations were executed on a 2.5 GHz single processor PC computer with 16 GB RAM

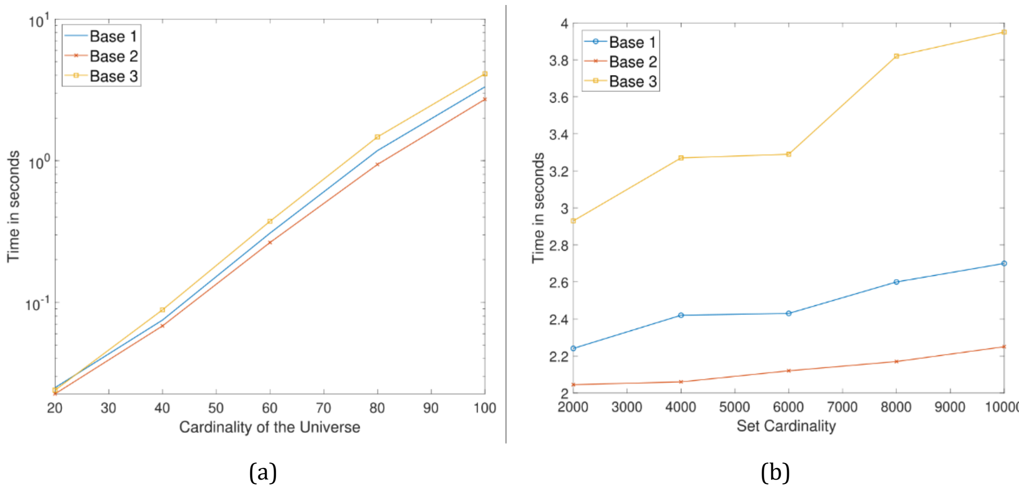

Figure 2. (a) Cardinality of Universe vs Time (in seconds) and (b) Set Cardinality vs Time (in seconds).

under Ubuntu operating system. The proposed algorithm was coded in C programming language. Three different progress factors called as *Base*1, *Base*2, *Base*3 where the progress factor d equals 1,2 and 3 respectively, are used to analyze the proposed progressive search algorithm. The proposed algorithm achieves 100% minimum hitting set calculation accuracy, and the results are evaluated based on run time.

First, we evaluate time results on random data. To perform a precise analysis of the search methods, we average the run time results for each size of universe $|U|$ and the number of sets $|C|$. Figure 2a shows the time of *Base*1, *Base*2 and *Base*3 against the number of items $|U|$. The performances of all search methods are similar and the times from all search methods fit a linear function of the number of items. Yet, *Base*2 is slightly faster than *Base*1 and *Base*3, and the differences increase as the number of items n increases. Figure 2b shows the time results of *Base*1, *Base*2 and *Base*3 against the number of sets $|C|$. It can be seen that *Base*2 performs better than *Base*1 and *Base*3. *Base*1 and *Base*2 can find solutions around 2 seconds, whereas *Base*3 requires 2x more time.

Average of the run time results of *Base*1, *Base*2, and *Base*3 are presented in Figure 3 as the cardinality of the hitting set changes. For $|U|$=20, when the size of the hitting set is 4, Base3 is the fastest. However, as the size of the set increases, *Base*2 becomes faster than *Base*1 and *Base*3. When the number of items $|U|$>20, the search methods fail to find solutions for $|H|$=10 in 1200 seconds. Thus, we report the results when the size of hitting sets equals to 1, 4 and 7. For $|U|$=40, although *Base*2 is a bit faster, the search methods have similar performances for the obtained solutions. For $|U|$=60, when the size of hitting set equals to 1 and 7, all search methods have comparable performances, however when the size of hitting set equals to 4, *Base*2 is faster than *Base*1 and *Base*3. For $|U|$=80, *Base*2 is 1 s. faster than Base1 and 2 s. faster *Base*3 when the size of hitting sets equals to 4 and 7. Lastly, for $|U|$=100, *Base*2 is again faster than *Base*1 and *Base*3, and the difference between *Base*1 and *Base*3 increases as the size of the hitting sets increases. In conclusion, when the size of the hitting sets increases, *Base*2 seems to perform better than *Base*1 and *Base*3.

Taking everything into account, *Base*3 has the worst performance on the random data set. Although *Base*1 and *Base*2 have a comparable performance as the number of the items increases, when the number of the sets and the size of the hitting sets increase, *Base*2 has a better performance than *Base*1.

Next, we evaluate time results on the real-world data. Time results for the accident dataset are shown in Table 2. The proposed method is able to find the minimum hitting set accurately even if the cardinality of the universe and the number of sets are high. Note that the time results for *Base*1 are shown only since the cardinality of the hitting set is 1 for this dataset.

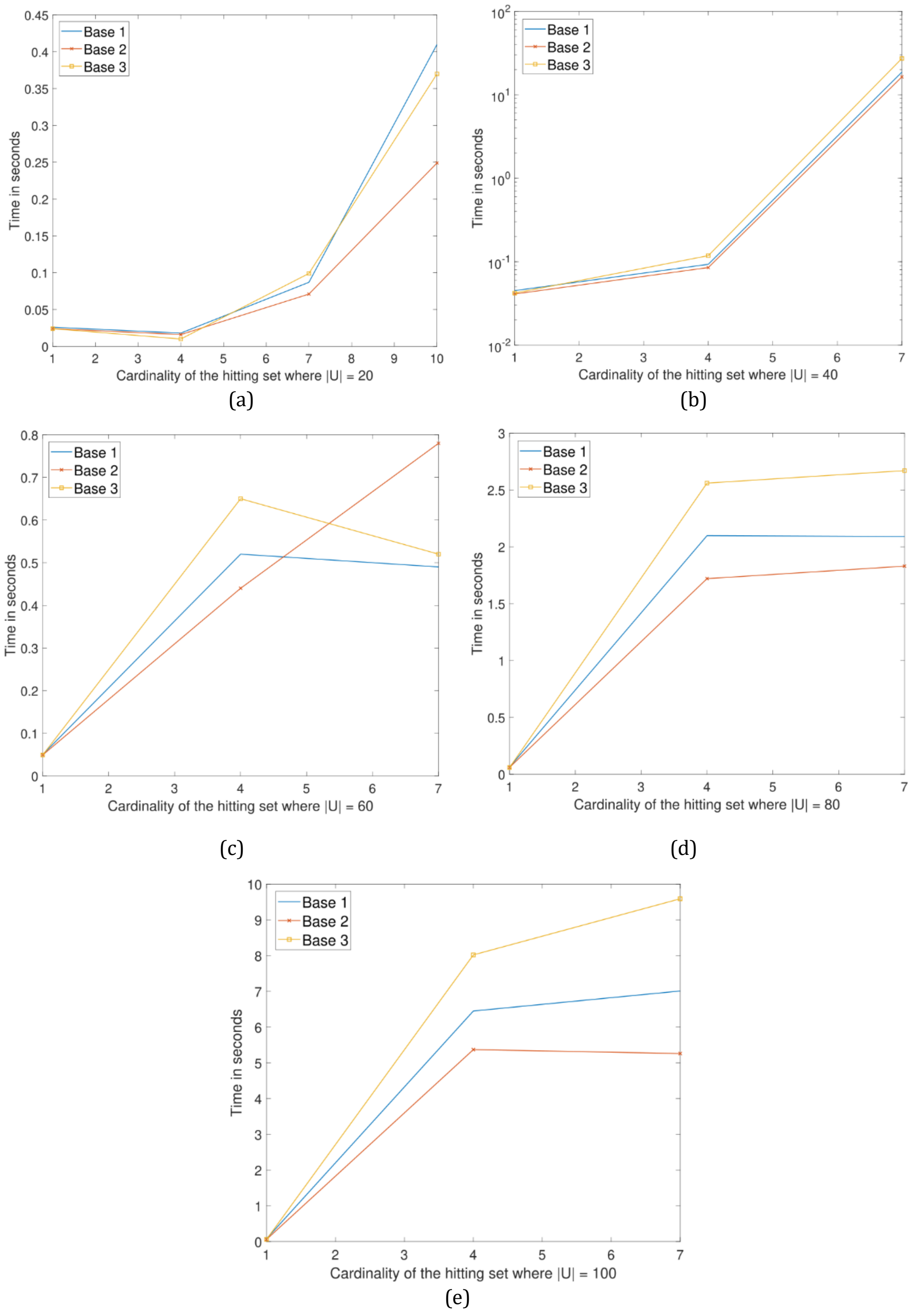

Figure 3. Time results with respect to cardinality of the hitting sets where |U| = 20, 40, 60, 80, 100.

Table 2. Time results in seconds

Name	time in seconds
ac_200k	0.002
ac_150k	0.009
ac_130k	0.03
ac_110k	0.05
ac_90k	1.08
ac_70k	2.74
ac_50k	8.08
ac_30k	54.2

5. Discussion and Conclusion

In this paper, we study the minimum hitting set problem. A hitting set for a collection of finite sets is a set that has at least one common element with each set in the collection. The minimum hitting set problem seeks for a hitting set of minimum size. We present a fast exact algorithm for finding the minimum hitting set in a given collection of sets. The proposed algorithm performs a progressive search on solution space. We analyze the performance of three different progress factors, namely *Base*1, *Base*2 and *Base*3, where the progress factor *d* are equals 1,2 and 3, respectively. We conduct a comprehensive computational experiment on large-scale instances of the problem to test the performance of these search methods. Considering the performance of the search methods on the obtained solutions, *Base*2 is found to be the most effective search for finding the minimum hitting set with regard to execution times. Designing efficient reduction rules to improve the performance of progressive search might be an interesting topic as future work.

References

[1] Lin, L. and Jiang, Y. 2003. The computation of hitting sets: Review and new algorithms. In Information Processing Letters, 177-184.

[2] de Kleer, J. 2011. Hitting set algorithms for model-based diagnosis. In 22th International Workshop on Principles of Diagnosis (DX-11).

[3] Carastan-Santos, D., Camargo, R. Y., Martins, D. C., Song, S. W., and Rozante, L. C. S. 2017. Finding exact hitting set solutions for systems biology applications using heterogeneous GPU clusters. In Future Generation Computer Systems, 67:418-429.

[4] Liu, J., Xu, H. and Xie, C. 2007. A New Statistical Hitting Set Attack on Anonymity Protocols, In 2007 International Conference on Computational Intelligence and Security (CIS 2007), pp. 922-925, doi: 10.1109/CIS.2007.73.

[5] Bailey J., Stuckey P.J. (2005) Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In: Hermenegildo M.V., Cabeza D. (eds) Practical Aspects of Declarative Languages. PADL 2005. Lecture Notes in Computer Science, vol 3350. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30557-6_14

[6] Kavvadias D.J., Stavropoulos E.C. (1999) Evaluation of an Algorithm for the Transversal Hypergraph Problem. In: Vitter J.S., Zaroliagis C.D. (eds) Algorithm Engineering. WAE 1999. Lecture Notes in Computer Science, vol 1668. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48318-7_8

[7] Gainer-Dewar, A. and Vera-Licona, P. 2016. The Minimal Hitting Set Generation Problem: Algorithms and Computation, 31(1):63-100.

[8] Chan, T. M., Har-Peled, S. 2012. Approximation algorithms for maximum independent set of pseudo-disks. In Discrete Comput. Geom., 48 (2): 373-392.

[9] Reiter, R. 1987. A theory of diagnosis from first principles. Artificial Intelligence, 32(1):57- 95.

[10] Wotawa, F. 2001. A variant of reiter's hitting-set algorithm. Information Processing Letters, 79(1):45-51.

[11] Pill, I.H. and Quaritsch, T. 2012. Optimizations for the boolean approach to computing minimal hitting sets. In Luc de Raedt, editor, ECAI 2012 - 20th European Conference on Arti cial Intelligence, 27{31 August 2012, Montpellier, France Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, volume 242 of Frontiers in Artificial Intelligence and Applications. 2012. Netherlands, 648-653.

[12] Zhao, X., and Ouyang, D. 2015. Deriving all minimal hitting sets based on join relation. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(7):1063-1076.

[13] Nyberg, M. 2011. A generalized minimal hitting-set algorithm to handle diagnosis with behavioral modes. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 41(1):137-148.

[14] Zhou, G., Feng, W., Jiang, B., Li, C. 2014. Computing minimal hitting set based on immune genetic algorithm. International Journal of Modelling, Identication and Control, 21(1):93-100. 2014.

[15] Liu, J., Ouyang, D.T., Wang, Y., Wang, Y. Zhang, L. 2015. Computing minimal hitting set based on immune genetic algorithm. Acta Electr. Sinica, 43(5):841-845.

[16] Hu, K., Liu, Z., Huang, K., Dai, C., Gao, S. 2016. Improved diferential evolution algorithm of model-based diagnosis in traction substation fault diagnosis of high-speed railway. IET Electrical Systems in Transportation, 6(3):163-169.

[17] Gao, S., Dai, C., Liu, Z., Geng, X. 2014. Geng. Application of bpso with ga in model-based fault

diagnosis of traction substation. In 2014 IEEE Congress on Evolutionary Computation (CEC), 2063-2069.

[18] Wang, Q., Jin, T., Mohammed, M. A. 2019. An innovative minimum hitting set algorithm for model-based fault diagnosis in power distribution network. IEEE Access, 7:30683-30692.

[19] Ouali, M. E., Fohlin, H., Srivastav, A. 2014. A randomised approximation algorithm for the hitting set problem. In Theoretical Computer Science, 555:23-34.

[20] Frequent itemset mining dataset repository. http://fimi.cs.helsinki./data/, May accessed: 2020.