Design, Implementation and Evaluation of Multi-terminal Binary Decision Diagram based Binary Fuzzy Relations

Hamid Alavi Toussi
Department of Computer Science, Aarhus University, Aarhus, Denmark
hamid@cs.au.dk
Bahram Sadeghi Bigham*
Department of Computer Sciences, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran
b_sadeghi_b@iasbs.ac.ir

Received: 25/Jan/2015 Revised: 16/Mar/2015 Accepted: 02/Feb/2016

Abstract

Elimination of redundancies in the memory representation is necessary for fast and efficient analysis of large sets of fuzzy data. In this work, we use MTBDDs as the underlying data-structure to represent fuzzy sets and binary fuzzy relations. This leads to elimination of redundancies in the representation, less computations, and faster analyses. We also extended a BDD package (BuDDy) to support MTBDDs in general and fuzzy sets and relations in particular. Representation and manipulation of MTBDD based fuzzy sets and binary fuzzy relations are described in this paper. These include design and implementation of different fuzzy operations such as max, min and max-min composition. In particular, an efficient algorithm for computing max-min composition is presented. Effectiveness of our MTBDD based implementation is shown by applying it on fuzzy connectedness and image segmentation problem. Compared to a base implementation, the running time of the MTBDD based implementation was faster (in our test cases) by a factor ranging from 2 to 27. Also, when the MTBDD based data-structure was employed, the memory needed to represent the final results was improved by a factor ranging from 37.9 to 265.5. We also describe our base implementation which is based on matrices.

Keywords: Boolean Functions; BDD; MTBDD; Binary Fuzzy Relations; Fuzzy Connectedness; Image Segmentation.

1. Introduction

ROBDDs have been used in hardware community for model checking and circuit verification [1]. It has a variety of applications in other areas as well. For example, it is used in compiler community for efficient points-to analysis. In points-to analysis, it is either used as a compact representation of large sets (points-to sets in this case) [24,25,11] or, the whole analysis is encoded as Boolean functions (or relations) and performed by using Boolean operators (BDD is compact representation for a Boolean function) [2,3,4]. It is also used in image processing [5,6].

Efficient representation of fuzzy sets and relations can be of great importance for analysing large sets of fuzzy data. In this work, design and implementation of a MTBDD based data-structure for representing fuzzy sets and binary fuzzy relations is investigated. Our main idea is to use MTBDDs [7] as the underlying data-structure.

MTBDDs have been used to represent arrays and graphs [7,8]. Clark et al. discussed representation of 2-dimensional arrays and vectors by using MTBDDs. However, they did not provide any implementation. Also they used shadow nodes to simplify their algorithms. Our idea is incorporated into a modern BDD library (BuDDy [9]) without shadow nodes. Shadow nodes increase the size of MTBDDs and thus make the implementation less efficient. Instead, level attribute already presented in the BDD library, is used.

R. Iris Bahar used MTBDDs to perform matrix multiplication and also solve all pairs shortest paths problem [8]. D. Yu. Bugaychenko and I. P. Soloviev proposed MRBDD (Multi-root decision diagram) data-structure to represent integer functions. In their representation, a finite-valued function is represented by a list of k different ROBDDs (or k roots as they suggested) thus an assignment maps to a binary string of 0s and 1s of length k instead of just a 0 or 1. The resulted binary string should be decoded to a certain value. This value is the output of the function for the assignment. The list of ROBDDs which constitute the MRBDD share isomorphic sub-graphs (every sub-graph is also a ROBDD) [12].

In our implementation, because of the way that BuDDy allocates nodes, no two isomorphic ROBDD is ever allocated twice so our work does not just share sub-graphs among a set of ROBDDs belonging to a single MRBDD but among all ROBDDs in memory. This is also discussed in Section 2.2. Generally speaking, sharing is more pervasive in MRBDDs compared to MTBDDs since there are just two terminals instead of a set of terminals. However, implementation of operations on MTBDDs is straightforward because terminals are shown explicitly. This is not the case with MRBDD and for any operation, a correspondence between operation on output values and equivalent operation on binary encoding of the output values must be defined. Summation and multiplication on matrices which are represented by MRBDDs are explained in [12].

* Corresponding Author

We have evaluated our data-structure by solving fuzzy connectedness over different binary fuzzy relations. These relations are obtained from different images. Results are compared with a base implementation which uses 2-dimensional arrays to represent binary fuzzy relations. MTBDD based implementation was 18 – 27× faster when number of distinct membership values that can appear in relations is limited to 11 values. In all cases we have far better memory consumption when MTBDD based implementation is used. See Section 6 for more detail.

Our major contributions in this work are:

- Describing representation and manipulation of fuzzy sets and binary fuzzy relations based on MTBDDs
- Extending BuDDy library to support MTBDDs in general and fuzzy sets and binary fuzzy relations in particular
- Evaluation of our implementation by solving fuzzy connectedness problem

An introduction to ROBDDs, BuDDy and MTBDDs comes in Section 2. The way that BuDDy is extended is discussed in Section 3. Representation and manipulation of MTBDD based fuzzy sets and binary fuzzy relations are discussed in Sections 4 and 5. The empirical evaluation is given in Section 6 and finally, in Conclusion, we discuss possible future directions. A draft version of this work has been published in arXiv [23].

2. Background

2.1 Binary Decision Diagrams

BDD is a data structure for representing Boolean functions compactly. A completely unreduced BDD is shown in Figure 1 which represents the Boolean function *f = x1 .x2 .x3 + x1 . x2 .x3 + x1 .x2 .x3.* Behind some of the nodes, their associated functions are presented.

The representation which is shown in Figure 1 is canonical. However, it is completely inefficient since it takes $O(2^n)$ space to represent a Boolean function with n variables in memory. ROBDD (or BDD for short) tries to address this problem by eliminating redundancies in unreduced BDDs. For eliminating redundancies and having a canonical representation, following two constraints should be always satisfied in any ROBDD:

1. A ROBDD should be ordered, that is, variables should respect a given total order on any path in a ROBDD. 2. A ROBDD should be reduced which means that there are no two sub-graphs in a ROBDD that are isomorphic and also for any node in a ROBDD its low-child should be different from its high-child.

Note that within every node in a ROBDD, level, a pointer to its low child, and a pointer to its high child are saved. Every node of a ROBDD which is also a ROBDD can be identified uniquely by a triple (level, low, high). Figure 2 shows the same Boolean function as in Figure 1 but in reduced form. We can also see this BDD and its associated Boolean function as a set which contains 011, 101 and 111 strings.

It is very common to use the term BDD to refer to ROBDD and we follow this practice in the rest of this paper.

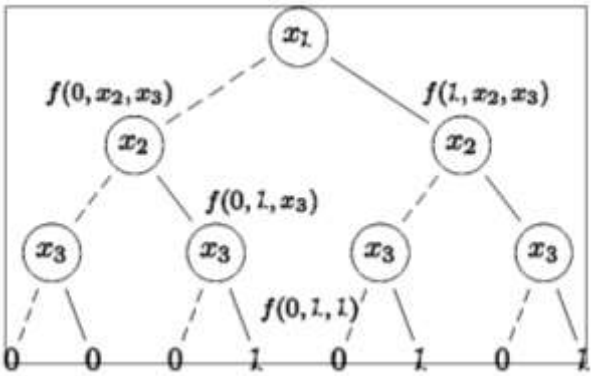


Fig. 1. A completely unreduced BDD which represents the Boolean function f = ¬x1 .x2 .x3 + x1 .¬x2 .x3 + x1 .x2 .x3.

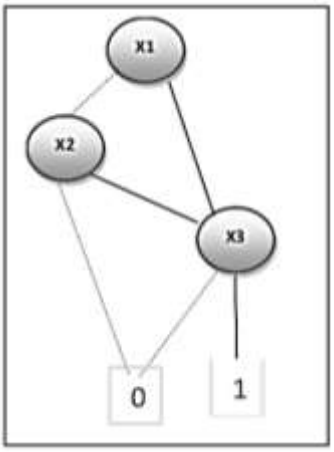


Fig. 2. The reduced version of the BDD previously shown in Figure 1.

2.2 BuDDy

BuDDy [9] is a library for creating and manipulating BDDs. It is written in C and also offers a C++ interface. Since we extended this library, it is useful to know some of its internals which affected our design.

In BuDDy all nodes (BDDs) are stored in an array which is named bddnodes. Every slot in this array has four fields namely level, low, high which are used to identify the BDD stored in the slot, and, the fourth field hash which is used to make searching the array more efficient by using hashing.

Every BDD can be uniquely identified by using its level, low and high attributes. In another word, we can associate a triple (level, low, high) with every BDD. At the core of BuDDy is a routine named bdd_makenode which is used for allocating BDDs. This routine only creates one entry for every distinct triple in bddnodes array and if it is asked to create a triple which is already inserted in bddnodes, it simply returns the index of the existing entry. This index represents the BDD in BuDDy. Also, if the triple sent to this routine contains the same value as its low and high, no new BDD will be allocated and the low will be returned.

In this way all the BDDs are always reduced and share sub-graphs that are isomorphic. Sub-graphs of any BDD are BDDs themselves and are allocated only once for any distinct triple. This brings some of the advantages of MRBDDs to our implementation. As described in [12], BDDs which constitute a MRBDD share isomorphic sub-graphs, but in BuDDy and as a consequence in our implementation any two BDDs share isomorphic sub-graphs.

2.3 Apply Operation

BDDs represent Boolean functions so one way to manipulate them is through Boolean operations (**or, and, xor**, etc). In general, there is a routine (bdd_apply in BuDDy) that takes two BDDs (which represent two Boolean functions) and makes a new BDD out of them by applying a Boolean operator. For further details see [13, 9].

2.4 Multi-Terminal Binary Decision Diagrams

A Multi-Terminal Binary Decision Diagram (or MTBDD for short) is a data-structure which has all the features of BDD and it allows more than two terminals. In Sections 3, 4 and 5, we explain how we have extended BuDDy to add support for MTBDDs as well as other functionalities which were needed.

3. Extending BuDDy to Support MTBDDs

In BuDDy, BDD type is defined as int. The integer representative of a BDD can be used to index into bddnodes array and retrieve its root. However, terminals are not required to be stored in bddnodes array explicitly since integers zero and one are reserved to show them. Integers greater than one are used to show non-terminals.

We chose not to define any new type to show MTBDD and simply used integer as their representative to comply with existing design. As a result, integers were also used to show terminals other than zero and one. However, the routine bdd_makenode can use any slot with index greater than one in bddnodes array to store a BDD (a non-terminal) and returns the slot's index as the BDD's representative. Thus using integers greater than one for showing terminals could introduce new complexities in this routine. To overcome this problem, negative integers were used to show terminals other than zero and one (-1 cannot be used to show a terminal since it indicates an uninitialized slot in bddnodes). In this way, all the existing routines continue to work (except bdd_apply in cases that it encounters terminals other than zero and one).

The BuDDy library was extended in a generic way so it can be used in similar scenarios. Major routines which are added to the library are mtbdd_apply and mtbdd_maxmin_compose (mmc for short). The former was added to handle maximum and minimum operators for MTBDDs, and the latter is simply a new functionality which was added to do max-min compositions of two binary fuzzy relations. See Subsection 5.1 for further detail.

Floating points were not used to show the membership values since imprecision in floating-points was not acceptable for us and we would like to have fully deterministic results. A C struct which has a field of type Integer is used to represent membership values. For example, an instance of this struct with an integer set to 25 shows 0.025 when precision of three digits is used. It may be worth noting that the precision should be known in advance to interpret a membership value. We used three different precisions in our benchmarks. Precision of three digits which can show 1001 different membership values, precision of two digits which can show 101 different membership values and precision of a single digit which can show 11 different membership values (Note that 1 is considered to be a membership value).

4. MTBDDs as Fuzzy Sets

In the MTBDD representation of a fuzzy set, there are as many terminals as there are different membership values in the fuzzy set (including zero and one). Different paths (including those which are reduced or are not represented explicitly) show different members of the fuzzy set. The terminal each path ends at, shows its membership value. In Figure 3, membership value 0.3 is represented by -3 so the MTBDD shows the fuzzy set {0.3/0000, 0.3/0001, 0.3/0010, 0.3/0011, 1/0100, 1/0110, 1/1000, 1/1010}. Strings 0000, 0001, 0010, ... correspond to numbers 0, 1, 2, ... respectively.

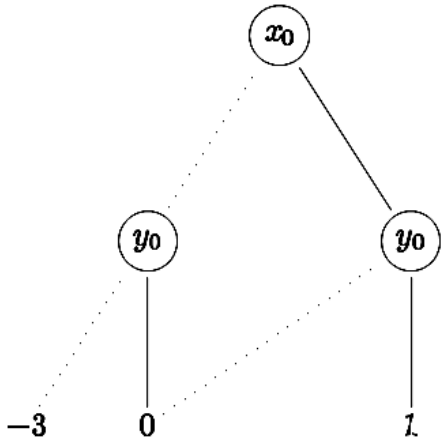


Fig. 3. A MTBDD which represents the binary fuzzy relation {1/(1, 1), 0.3/(0, 0)}

4.1 Intersection and Union Operations

Maximum and minimum operators were used as fuzzy set intersection and fuzzy set union respectively [14]. The general apply routine which was mentioned in Section 2 takes two BDDs as its operands and another parameter as its operator, then, it applies the operator to the operands. A slightly modified apply routine (mtbdd_apply) which handles MTBDDs and maximum/minimum operators was added to the BuDDy library. Our implementation can be easily extended to include other t-norm operators.

5. MTBDDs as Binary Fuzzy Relations

Any binary fuzzy relation has two domains, two disjoint sets of BDD variables are used. Every set of BDD variables is mapped to one of the domains. For example, if a domain has eight objects, three variables would be needed to show all its members (i.e. $2^3 = 8$).

In Figure 4, there are two domains and each one has two objects so all objects can be encoded by using one variable for each domain. Variable *x0* is used to encode the objects in the first domain and variable *y0* is used to encode objects in the second domain. Suppose that non-terminal −3 is mapped to 0.3. In this way (0, 1) and (1, 0) are associated with 0 membership value. (1, 1) is associated with 1 membership value and (0, 0) is associated with 0.3 membership value. You can also see it as the 2-dimensional array:

0.3	*0*
0	*1*

Since we implemented an algorithm similar to 4-way block-multiplication to compute the max-min composition of two binary fuzzy relations which are represented as MTBDDs, it is desirable to partition each relation into four blocks and access each one in constant time. In order to make this possible, interleaved variable ordering was used [7]. This means that if variables *xi* constitute domain *x* and variables *yi* constitute domain *y*, the ordering of variables would be *x0* , *y0* , *x1* , *y1* , *x2* , *y2* , ... See Figure 5 for an example. Binary fuzzy relations can be seen as square matrices of size $n \times n$. In a binary fuzzy relation A that $n \neq 2^k$, an identity matrix with smallest possible size is attached to A to meet this requirement. This technique is also used in [7]. This is working for max-min composition since minimum of zero and any other membership value is zero (similar to matrix multiplication and multiplication of zero by other element).

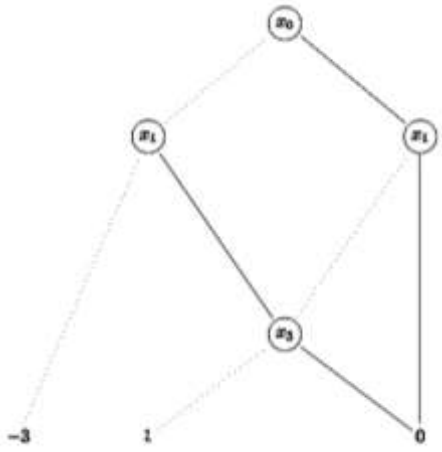


Fig. 4. A MTBDD representing a fuzzy set

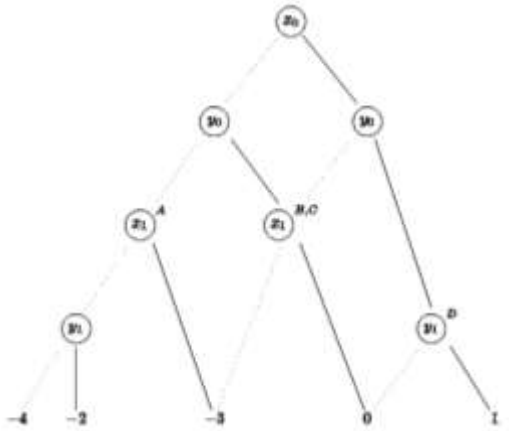


Fig. 5. A binary fuzzy relation represented as a MTBDD. Nodes A, B, C and D show four partitions of this MTBDD. Both B and C correspond to the same node.

5.1 Max-min Composition

In general max-min composition of two binary fuzzy relations R1 (D × D) and R2 (D × D) is defined as follows:

$$R_3(a,b) = max_{c \in D} min\big(R_1(a,c), R_2(c,b)\big)$$

The max-min composition procedure is similar to block matrix multiplication. A binary fuzzy relation can be viewed as a 2-dimensional matrix. This matrix is partitioned into four sub-matrices (blocks) in the procedure (Figure 5).

max-min composition was implemented as a recursive procedure which is shown in Figure 6. During the recursion, at each call, parameters of the call (MTBDD a and MTBDD b) should be interpreted based on the depth of the recursion which is passed as the third parameter (call_level). This is because, the partitioning does not create four new MTBDDs but returns four sub-graphs of the original MTBDD as its partitions so a hypothetical level (root_level) is assumed. The mentioned hypothetical level indicates the smallest possible level of the resulted partitions (Figure 6). For example, consider the MTBDD shown in Figure 5, four partitions would be created after partitioning namely A, B, C and D. The hypothetical root level for these partitions is two which corresponds to the variable *x1*. This technique (introducing and using a hypothetical root level) avoids creation of new matrices (MTBDDs) and makes the max-min composition procedure more efficient. In order to compute the max-min composition of MTBDDs a and b, we have to call mmc(a, b, 0).

6. Evaluation

To evaluate our representation, we extended the BuDDy library to represent and manipulate binary fuzzy relations by using MTBDDs. Also, we implemented binary fuzzy relations based on two dimensional arrays. The array implementation was used as a baseline (base implementation). Images in our input set are obtained from UIUC image database [15] and The Berkeley Segmentation Dataset and Benchmark [16].

In our experiment, the fuzzy-connectedness problem is solved for different images in the input set. Results of these experiments and further details come next.

6.1 Evaluation Results and Further Details

Results are given in tables 2, 3 and 4. Input relations for our experiments, are affinity relations, which are created from various images. Affinity relation is a symmetric and reflexive fuzzy relation which assigns a membership value to a pair of pixels based on their local properties [17]. We initialized this relation only for pairs of pixels which are neighbour in a given image and membership value for any other pair of pixels in the image is set to zero. This leads to sparsity of affinity relations. The following similarity measure which was used in [18] has been employed to initialize affinity relations. δ is the largest diff (diff is computed for every pair of pixels) and $c_r, c_g, c_b, d_r, d_g, d_b$ show color intensities associated with c and d pixels respectively:

$$simil(c,d) = 1 - \sqrt[2]{diff}/\delta \; where \; diff = (c_r - d_r)^2 + \left(c_g - d_g\right)^2 + (c_b - d_b)^2$$

Images used for creating affinity relations are shown in Table 1. The first one is from UIUC image database [15] and the next three images are obtained from The Berkeley Segmentation Dataset and Benchmark [16]. The last image is a synthetic image from reference [18]. All experiments are run on a machine with 2.8 GHz Intel CPU and 4 GB of RAM running Fedora 14.

In the rest of this Section, problem of fuzzy-connectedness is investigated. Input to this algorithm is an affinity relation which is extracted from an image, and final output is a relation that assigns a membership value to every pair of pixels in the image. The output can be used to create different clusters of pixels [17]. The final goal of fuzzy connectedness is to calculate FC relation which is a reflexive, symmetric and transitive relation. It is basically max-min transitive closure of the initial affinity relation. The FC relation is a full binary fuzzy relation. It assigns a membership value to every pair *(c, d)*. This value is the maximum strength of all possible paths from *c* to *d*. The strength of a path is the smallest membership value along the path. FC relations are obtained by computing max-min transitive closure of affinity relations.

We implemented two different versions to compute the transitive closure. Our base implementation used two dimensional arrays to represent binary fuzzy relations and, it employed Floyd-warshall algorithm as shown in Figure *7*. *n* is the number of pixels in the image. *c* stores the affinity relation initially and represents fuzzy connectedness (FC) relation at the end. Our second implementation used MTBDDs to represent binary fuzzy relations and, it computed FC relation by using Repeated Squaring algorithm as shown in Figure 8. Affinity relation is the input to this algorithm and at the end, res would be a MTBDD that represent the fuzzy connectedness relation (FC). Because of the MTBDD special structure we could not use the Floyd-warshall algorithm in conjunction with this data-structure efficiently.

Table 2 shows running-times of our base and MTBDD based programs which compute max-min transitive closure of affinity relations obtained from our test images. Three versions are shown in the table. Column base shows the base implementation and the other two columns under mtbdd show the MTBDD based implementation with one and two digits of precision. The MTBDD based implementation is significantly faster than the base implementation when precision of one digit is used (it is 18 – 27× faster). Compared to base implementation, using 2-digits precision improved running time in all cases, but one, which was our smallest image (40 × 27). In this particular case all running times were under one minute. In other cases MTBDD based implementation with two digits precision is faster by a factor ranging from 2 to 7.

When images are larger and their pixels are more homogeneous, MTBDD based implementation becomes faster relative to the base implementation. Note that running-time in base implementation only depends on size of input image (i.e. number of pixels). In contrast, MTBDD based implementation's running-time depend both on size of image and values stored in every pixel of the image. For example, running-times for image3 and image4 are the same when base implementation is used but it takes less time for MTBDD based implementation to compute the transitive closure when it takes image4 as input. This is because image4 leads to MTBDDs which are more compact (this is described in more detail in the next paragraph).

As described in Section 5, different paths in MTBDD representation of a relation show different pairs in the relation. More commonalities among paths lead to more reductions and, a more compact MTBDD representation of the relation. Computations on a smaller MTBDD take less time. Two different images even with the same size result in different MTBDDs with different sizes. An image which results in a MTBDD with more commonalities in its paths occupies less memory and results in fewer computations in the MTBDD based solver (Sparsity in input relation and also images with homogeneous pixels leads to more compact MTBDDs).

In Table 3, number of entries in FC relations, number of terminals and number of nodes in MTBDD representation of these relations are shown. Algorithms which were used to compute FC relations are show in Figure 7 (base version) and Figure 8 (MTBDD version).

Number of nodes in MTBDD base implementation is extremely lower than number of entries in base implementation which leads to a far improved memory consumption when MTBDDs are used to represent binary fuzzy relations. Every array entry in base implementation is three bytes (two bytes for a short integer and one byte for a flag) and the size of every bddnode is 20 bytes [9]. In Table 4 size of array and MTBDD based representation of FC relations are shown (in KB). The column size (r) indicates number of entries in the array representation of the relation (base implementation), column size (array) shows the amount of memory allocated for representing

arrays in the base implementation and, the other two columns indicate the amount of memory allocated for representing MTBDDs in the MTBDD based implementation (precision of one and two digits). Considering this table, MTBDD based representation takes 37.9 – 265.5× less memory than the array representation depending on the input image.

It is also noteworthy that shape and number of nodes in BDDs (and MTBDDs as well) also depend on variable ordering beside data they are representing since different variable ordering leads to different paths with different degree of sharing. However, a fixed variable ordering is used in our implementation.

Number of terminals is also shown in Table 3. This gives the number of distinct (hard) clusterings that can be obtained from the resulted relation. When number of terminals is limited to 11 (1-digit precision), it is five or six and in the other case, when number of terminals is limited to 101 (2-digit precision), it is usually around 25.

```
procedure mmc(a, b, callLevel)

  if both a and b are terminals then return min(a, b)
  if (r = mmc_cache[a, b, callLevel]) != NULL then return r
  if (r = mmc_cache[b, a, callLevel]) != NULL then return r

  rootLevel = callLevel * 2

  partition a into sa[0], sa[1], sa[2] and sa[3] based on rootLevel
  partition b into sb[0], sb[1], sb[2] and sb[3] based on rootLevel

  t1 = mmc(sa[0], sb[0], callLevel + 1)
  t2 = mmc(sa[1], sb[2], callLevel + 1)
  l = mtbdd_apply(t1, t2, mtbddopFuzzymax)
  t1 = mmc(sa[0], sb[1], callLevel + 1)
  t2 = mmc(sa[1], sb[3], callLevel + 1)
  h = mtbdd_apply(t1, t2, mtbddopFuzzymax)
  L = bdd_makenode(rootLevel * 1, l, h)

  t1 = mmc(sa[2], sb[0], callLevel + 1)
  t2 = mmc(sa[3], sb[2], callLevel + 1)
  l = mtbdd_apply(t1, t2, mtbddopFuzzymax)
  t1 = mmc(sa[2], sb[1], callLevel + 1)
  t2 = mmc(sa[3], sb[3], callLevel + 1)
  h = mtbdd_apply(t1, t2, mtbddopFuzzymax)
  H = bdd_makenode(rootLevel * 1, l, h)

  r = bdd_makenode(rootLevel, L, H)
  mmc_cache[a, b, callLevel] = r
  return r

end procedure
```

Fig. 6. Max-min composition (mmc) routine in pseudo code. a and b are MTBDDs and callLevel indicates the depth of the recursion.

```
for k = 1 to n do
  for i = 1 to n do
   for j = 1 to n do
     c[i, j ] = max(c[i, j ], min(c[i, k], c[k, j]))
```

Fig. 7. Using Floyd-warshall algorithm to compute max-min transitive closure of affinity relation.

```
res = affinity
repeat
  old = res
  res = mmc(res, res, 0)
until res == old
```

Fig. 8. Using MTBDDs and Repeated Squaring algorithm to compute max-min transitive closure of affinity relation.

Table 1. Images which are used for creating affinity relations.

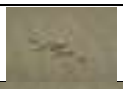
Image	Image name	Size
	*Image0*	80x65
	*Image1*	40x27
	*Image2*	60x40
	*Image3*	90x60
	Image4	90x60

Table 2. Running times of Fuzzy Connectedness problem solver for both base and MTBDD based implementations (in seconds).

Image	Base	mtbdd 1	mtbdd 2
Image0	3574	134.61	2236
Image1	32.01	1.78	41.90
Image2	350.72	13.88	285.48
Image3	4000	214.64	1898.21
Image4	same as image3	148.95	533.52

Table 3. Number of entries in FC relations and number of nodes which are allocated to represent the relation in its MTBDD representation

Image	Size(r)	mtbdd terminals 1	mtbdd terminals 2	Nodes in mtbdd 1	Nodes in mtbdd 2
Image0	27040000	6	28	25683	216461
Image1	1166400	6	25	2212	30753
Image2	5760000	5	27	4007	87306
Image3	29160000	5	26	16469	581770
Image4	same as image3	5	10	34995	57439

Table 4. Size of array and MTBDD representation in KB

Image	size(r)	size(array)	size(mtbdd 1)	size(mtbdd 2)
Image0	27040000	81120	513	4329
Image1	1166400	3499	44	615
Image2	5760000	17280	80	1746
Image3	29160000	87480	329	11635
Image4	29160000	87480	699	1148

7. Conclusion

In this work, we designed and implemented a MTBDD based data-structure to represent fuzzy sets and relations. Also, the BuDDy library was extended to support MTBDDs, and it was employed to implement our idea. Promising results were obtained in evaluation of our method. In particular, considering the fuzzy connectedness problem and compared to the base implementation, when MTBDD based implementation was used, the running-time was improved by a factor ranging from 2 to 27, and, when the memory needed to represent final results was improved by a factor ranging from 37.9 to 265.5.

In the future we would like to apply our data-structure to other problems in fuzzy systems which involve manipulating binary fuzzy relations and fuzzy sets, specially, problems with very large sets of fuzzy data such as the use of fuzzy sets in data mining, approximate reasoning and information retrieval based on fuzzy logic [19, 20, 21, 22].

Extending our current implementation to a framework for research in fuzzy systems is another direction we would like to follow. In particular, researchers would be able to add t-norms operators of their interest, and, design and run new experiments on top of our framework.

References

[1] E. Clarke, O. Grumberg, and D. Long. “Symbolic Model Checking for Sequential Circuit Verification.” In IEEE Transactions on Computer Aided Design, 1994.

[2] Marc Berndl, Ond˙rej Lhot´ak, Feng Qian, Laurie Hendren, and Navindra Umanee. “Points-to analysis using BDDs.” In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Inplementation, 2003, pp 103–114.

[3] John Whaley and Monica Lam. “Clonning-based context-sensitive Pointer Alias analysis using Binary Decision Diagrams.” In Proceeding of PLDI, 2004.

[4] Ond˙rej Lhot´ak, Stephen Curial, and Jos´e Nelson Amaral. “Using XBDDs and ZBDDs in points-to analysis.” Software, Practice and Experience, vol 39, Issue 2, pp 63–188, 2009.

[5] Watis Leelapatra, Kanchana Kanchanasut, and Chidchanok Lursinsap. “Displacement BDD and geometric transformations of binary decision diagram encoded images.” Pattern Recognition Letters, vol 29, Issue 4, pp 438–456, March 2008.

[6] Mike Starkey, Randy Bryant, and Y Bryant. “Using ordered binary-decision diagrams for compressing images and image sequences.” Technical report, CMU-CS, 1995.

[7] Edmund M.and Fujita Clarke, M., McGeer, P C., McMillan, K., Yang, J C-Y, and X Zhao. “Multi-Terminal Binary

Decision Diagrams: An Efficient Data-Structure for Matrix Representation." Formal Methods in System Design, 1997.
[8] R. I. Bahar, E. A. Frohm, C. M. Gaona, E. Macii, A. Pardo, and F. Somenzi. "Algebraic Decision Diagrams and Their Applications. Formal Methods in System Design," 10, 1997.
[9] Jorn Lind-Nielsen. BuDDy library. http://sourceforge.net/projects/buddy/, 2002.
[10] D. Bugaychenko. "On application of multi-rooted binary decision diagrams to probabilistic model checking. In Verification, Model Checking, and Abstract Interpretation," pp 104–118. Springer, 2012.
[11] Vaclav Dvorak. "Branching program-based programmable logic for embedded systems." In Proceedings of ICONS 2012, pp 109–115. International Academy, Research, and Industry Association, 2012.
[12] D. Yu. Bugaychenko and I. P. Soloviev. "Application of multiroot decision diagrams for integer functions." MATHEMATICS, vol 43, Issue 2, pp 92–97, 2010.
[13] Randal E. Bryant. "Graph Based Algorithm for Boolean function manipulation." In IEEE Transactions on Computers, 1985.
[14] L. A. Zadeh. "Fuzzy Sets." Information and Control, pp 338–353, 1965.
[15] Shivani Agarwal. "UIUC Image Database for Car Detection." http://cogcomp.cs.illinois.edu/Data/Car/, April 2002. Accessed: 2012-08-20.
[16] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics." In Proceeding of 8th International Conference on Computer Vision, vol 2, pp 416–423, July 2001.
[17] Jayaram K. Udupa and Punam K. Saha. Fuzzy Connectedness and Image Segmentation. In Proceeding of the IEEE, vol 91, pp 1649-1669, 2003.
[18] Pedro F. Felzenszwalb. "Efficient Graph-Based Image Segmentation." Journal of Computer Vision, vol 59, Issue 2, pp 167-181, 2004.
[19] M. Delgado, N. Mann, M. Martn-Bautista, D. Snchez, and M. Vila. "Mining fuzzy association rules: An overview." Soft Computing for Information Processing and Analysis, vol 164, pp 351–373, 2005.
[20] Ulrich Bodenhofer, Eyke Hllermeier, Frank Klawonn, and Rudolf Kruse. "Special issue on soft computing for information mining." Soft Computing - A Fusion of Foundations, Methodologies and Applications, vol 11, pp 397–399, 2007.
[21] Amel Borgi and Herman Akdag. "Knowledge based supervised fuzzy-classification: An application to image processing." Annals of Mathematics and Artificial Intelligence, vol 32, Issue 1, pp 67–86, 2001.
[22] Ariel Gmez, Carlos Len, Jorge Ropero, Alejandro Carrasco, and Joaqun Luque. Sabio. "Soft agent for extended information retrieval. "Applied Artificial Intelligence, vol 27, pp 249–277, 2013.
[23] Hamid A. Toussi and Bahram Sadeghi Bigham, Design, Implementation and Evaluation of MTBDD based Fuzzy Sets and Binary Fuzzy Relations, preprint arXiv:1403.1279 [cs.DS], [Online]. Available: http://arxiv.org/abs/1403.1279
[24] Hamid A. Toussi and Abbas Rasoolzadegan, "Flow-sensitive points-to analysis for Java programs using BDDs," In Proceeding of 4th International Conference on Computer and Knowledge Engineering (ICCKE), pp.380-386, 29-30 Oct. 2014.
[25] Ben Hardekopf and Calvin Lin. "Semi-sparse flow-sensitive pointer analysis," In ACM SIGPLAN Notices, vol. 44, no. 1, pp. 226-238. ACM, 2009.

Hamid Alavi Toussi obtained the M.Sc in Computer Science in 2011 form University of Sistan and Baluchestan, Zahedan, Iran. He also holds a B.Sc in Computer Engineering (obtained in 2009 from Islamic Azad University of Mashhad, Iran). Currently, he is a Ph.D student in Computer Science at Aarhus University (Computer Science department), working on program analysis for web applications.

Bahram Sadeghi Bigham is an Assistant Professor in Computer Sciences at the Institute for Advanced Studies in Basic Sciences (IASBS). His research interests are in the areas of Medical Applications of AI, Computational Methods, Data Mining, and Robotics. Prior to arriving at IASBS, Dr. Sadeghi worked as a Postdoctoral Fellow at the University of Cardiff in the School of Computer Science. In June 2008, He completed his Ph.D at Amirkabir University of Technology (Tehran Polytechnic), where he also completed a M.Sc in 2000. His B.Sc is from University of Birjand in Mathematics.