Ternary-Decimal Exclusion Algorithm for Multiattribute Utility Functions

Yerkin G. Abdildin[0000-0002-7074-4377]

Nazarbayev University, 53 Kabanbay Batyr Ave., Nur-Sultan, 010000, Kazakhstan
yerkin.abdildin@nu.edu.kz

Abstract. We propose methods to eliminate redundant utility assessments in decision analysis applications. We abstract a set of utility assessments such that the set is represented as a matrix of ternary numbers. To achieve efficiency, the matrix is converted to a decimal vector for further processing. The resulting approach demonstrates excellent performance on random sets of utility assessments. The method eliminates the redundant questions for the decision maker and can serve for consistency check.

Keywords: Uncertainty, decision analysis, decision maker, multiattribute utility problem, redundant utility assessments.

1 Introduction

In multi-objective decisions under uncertainty, the decision maker's trade-offs over the different attributes (e.g. increase output, but decrease expenses [1]) can be expressed through a multiattribute utility function (MUF). In complex decision problems, e.g. power plant siting [2-4], the construction of a MUF [5] becomes difficult with the number of attributes. For example, Keeney and Raiffa [6] discuss the siting of nuclear power facilities utilizing ten attributes while Keeney and Sicherman [7] model alternatives for generating electricity using 14 attributes. To simplify the assessment of a MUF from the decision maker [8], it can be decomposed into lower-order terms [9]. The lower-order terms of MUF may contain redundancies. The elimination of possible redundancy in utility assessments saves time significantly for the decision maker.

Although in different context, the task to eliminate redundant expressions in a program may arise in compiler optimization. The example of a redundant expression here could be a computation that is performed twice. Different algorithms have been discussed to address this problem [10-12]. Redundancy is also met in biomedical science. In particular, if some biological functions (or terms) in the gene ontology (GO) hierarchy do not provide additional information, then they are redundant and must be excluded from the GO lists [13].

In this paper, we consider the problem of determining the minimal set of utility assessments (lower-order terms) needed to construct a MUF from a given set of assessments. We introduce a novel and efficient method for eliminating duplicate and redun-

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_22

dant terms in a MUF. The efficiency of the method is achieved by transferring the computational burden from a two-dimensional input (ternary matrix) to a decimal vector. The algorithm sorts the vector, eliminates possible duplicates and redundancies, and returns the unique set of utility assessments. The method can help to exclude redundant questions for the decision maker. It can also be used to check the consistency of the utility assessments, which are required for a construction of multiattribute utility functions.

2 Preliminaries

Consider a decision problem with a set of n attributes $X=\{X_1, X_2, ..., X_n\}$, where the domain of the attribute X_i is represented by instantiations: x_i^0, x_i^*, x_i, such that the former two are the subsets of the latter and distinct. Namely, x_i^0 and x_i^* are the *least* and *most* preferred values of attribute X_i, and x_i denotes *all* values. Let $U=\{U_1, U_2, ..., U_m\}$ be a given set of utility assessments, where U_j is a function of X. The goal is to return U without redundant assessments.

For example, consider the five utility assessments presented in Figure 1(a). The first two terms, $U(x_1, x_2, x_3^0, x_4^0)$ and $U(x_1^*, x_2, x_3, x_4^0)$, are *redundant* due to the terms $U(x_1, x_2, x_3^0, x_4)$ and $U(x_1^*, x_2, x_3, x_4)$, respectively. The fifth term is a *duplicate* of $U(x_1^*, x_2, x_3, x_4)$. Therefore; a unique set of utility assessments will contain only two terms.

To simplify the encoding and decoding of utility assessments, Abdildin and Abbas [14] introduced the ternary matrix of utility assessments, so that the given set can be encoded as a 5×4 ternary matrix M as shown in Figure 1(b).

Definition 1 [14]: *The ternary matrix of utility assessments,* M, *is an* $m\times n$ *matrix storing integers 0, 1, and 2, representing respectively the least preferable value, the most preferable value, and all values of an attribute, where* m *is the number of utility assessments, and* n *is the number of attributes of the decision-making problem.*

$$\begin{array}{c} U(x_1, x_2, x_3^0, x_4^0), \\ U(x_1^*, x_2, x_3, x_4^0), \\ U(x_1^*, x_2, x_3, x_4), \\ U(x_1, x_2, x_3^0, x_4), \\ U(x_1^*, x_2, x_3, x_4). \end{array}$$

2	2	0	0
1	2	2	0
1	2	2	2
2	2	0	2
1	2	2	2

Fig. 1. (a) Utility assessments; (b) Ternary matrix M.

To eliminate duplicate and redundant terms from the ternary matrix M in a brute-force approach, every row of M has to be compared with all other rows elementwise. The brute-force approach can be implemented in different ways, the pseudo-code of the one utilized in this paper is given in the appendix. Another approach [14] is to compare only parts of the rows called twos-complement. The twos-complement part represents the set of columns in a row containing ones and zeros, i.e. complements of twos. We now propose a novel and more efficient approach for eliminating redundant utility assessments.

3 Proposed Approach

3.1 Main Idea

The main idea of the *ternary-decimal exclusion algorithm* [15] can be described by small example when $n = 2$. In this case, the nine values of the ternary number, *T*, can vary from [0 0] to [2 2] (see Table 1).

We can observe that:

- If ternary number [2 2] exists in the input matrix (or $max = 3^n - 1 = 8$ in decimal), then all other numbers must be excluded from the ternary matrix.
- If ternary number [2 1] (or $max - 1 = 7$ in decimal) is present, then decimal numbers 1 and 4 can be eliminated (note, a modulo 3 division (%) returns the remainder 1). Similarly, 6 covers 0 and 3, because, for example, 6 % 3 = 0.
- The decimal 2 covers all numbers below it and 5 covers 3 and 4.
- In addition, 2 and 5 divide the decimal numbers into three sub-intervals of equal length.

Table 1. Coverage table for *n* = 2.

Ternary number *T* for *n* = 2	Decimal number *D* corresponding to *T*	Coverage: What can *D* eliminate?								
2 2	8	0	1	2	3	4	5	6	7	8
2 1	7		1			4			7	
2 0	6	0			3			6		
1 2	5				3	4	5			
1 1	4					4				
1 0	3				3					
0 2	2	0	1	2						
0 1	1		1							
0 0	0	0								

Therefore, we will convert the input ternary matrix $M(m, n)$ into a decimal vector $V(m)$, sort the vector, eliminate duplicate numbers, and then (recursively) eliminate redundancies using the observations discussed above. The recursion utilized in the proposed approach improves the performance of the algorithm. It recursively divides the sorted vector into three sub-vectors using the right bounds: $lb = \lfloor min + (max - min)/3 \rfloor$ and $mb = \lfloor min + 2(max - min)/3 \rfloor$, for the left and mid sub-vectors, respectively, where initially $min = 0$ and $max = 3^n - 1$.

Note that the ternary numbers in Table 1 are unique and their number grows fast with n.

Definition 2: *For a decision problem with* n *attributes, the cardinality of the ternary matrix of utility assessments is the total number of possible unique ternary numbers that it can contain and is equal to* 3^n.

So, the amount of unique ternary numbers (i.e. utility assessments) that can be formed from the rows of matrix $M(m,n)$ is limited by its cardinality. The cardinality of the ternary matrix of utility assessments will be used later in our simulations.

3.2 Pseudocodes

The algorithm consists of the following steps. First, we convert the ternary matrix $M(m,n)$ to a decimal vector $V(m)$. Recall, a d -digit ternary number, T, can be converted to a decimal as follows: $D = t_d R^d + t_{d-1} R^{d-1} + \ldots + t_1 R^1 + t_0 R^0$, where R denotes radix (base 3), and t_d denotes the digits of T. As an example, $T = 102$ in decimal is $D = 1*3^2 + 0*3^1 + 2*3^0 = 11$.

Next, we sort the decimal vector $V(m)$ and eliminate duplicates. One can use any efficient algorithm (see comparisons in [16]) to sort the decimal vector and then eliminate the duplicates in linear time.

Then, we recursively exclude redundancies from the vector. The pseudocode of the exclude_redundant subroutine is presented in Table 2. A modulo division is denoted by %, an empty vector by [], and elements of vector V from 1 to idx by V(1:idx). The exclude_redundant procedure uses the search_bound subroutine (Table 3).

Now, we convert the decimal vector to the ternary matrix. Recall, a decimal number D can be converted to a ternary T by continually dividing D by three to have a quotient and a remainder until the former is equal to zero. Then the remainder is read in a reverse order.

Complex interdependence conditions in a decision problem may lead to specific redundancies in utility assessments. The last step of the proposed algorithm, exclude_specific subroutine (Table 4), is needed to eliminate such redundancies.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_22

Table 2. Pseudocode of the exclude_redundant subroutine.

Procedure exclude_redundant(V, min, max):
Input: A vector of *m* sorted unique nonnegative decimal integers, V[*m*], and the minimal and maximal possible values of *m* in V[*m*].
Output: A new vector of sorted unique positive decimal integers, V.

```
m ← length(V)
if m < 2
    return V    // base condition
if (V(m) = max)
    return V ← max
k ← 1
if (V(m) = max - 1) AND (V(m – 1) = max - 2)
    while k < m - 1
        if (V(k) % 3) = 1 OR (V(k) % 3) = 0
            delete V(k)
elseif (V(m) = max - 1)
    while k < m
        if (V(k) % 3) = 1
            delete V(k)
elseif (V(m) = max - 2)
    while k < m
        if (V(k) % 3) = 0
            delete V(k)
m ← length(V)
lb ← floor(min + (max-min)/3)         // left bound
mb ← floor(min + 2*(max-min)/3)     // mid bound
idxL ← search_bound(V, 1, m, lb)     // find elm ≤ lb to create Left sub-vector
if idxL = 0
    L ← []
elseif V(idxL) = lb
    L ← lb
else
    L ← V(1 : idxL)
idxM ← search_bound(V, idxL + 1, m, mb)  // find elm ≤ mb to create Mid sub-vector
if idxM = idxL
    M ← []
elseif V(idxM) = mb
    M ← mb
else
    M ← V(idxL + 1 : idxM)
if m = idxM     // create Right sub-vector
    R ← []
else
    R ← V(idxM + 1 : m)
[L] ← exclude_redundant(L, min, lb)
[M] ← exclude_redundant(M, lb, mb)
[R] ← exclude_redundant(R, mb, max)
[V] ← [L M R]     // merge L M R
return V
```

Table 3. Pseudocode of the search_bound subroutine.

```
Procedure search_bound(V, left, right, x):
Input:    A sorted vector of unique nonnegative decimal integers, V, its first and end indices,
and number x to search.
Output:   idx, an index of x if it exists in V, index of element lower than x, otherwise.
  if left > right
    idx ← right
    return idx
  else
    mid ← floor(left + (right - left)/2)
    if x < V(mid)
      idx ← search_bound(V, left, mid - 1, x)
    elseif x > V(mid)
      idx ← search_bound(V, mid + 1, right, x)
    else
      idx ← mid
      return idx
```

Table 4. Pseudocode of the exclude_specific subroutine.

```
Procedure exclude_specific(M):
Input:    Ternary matrix M.
Output:   Matrix M without redundant rows.
  r ← 1
  while r < size(M, 1) + 1
    tc ← find(M(r, :) ≠ 2)  // indices of 0s and 1s in row r
    i ← r + 1
    while i < size(M,1) + 1
      if isequal(M(r, tc), M(i, tc))  // compare elms of r and i by indices in tc
        M(i, :) ← []  // delete row i
        i ← i - 1
      i ← i + 1
    r ← r + 1
  return M
```

3.3 An illustrative example

Let us consider a three-attribute decision problem in which attributes X_1 and X_2 are utility independent [6] from attribute X_3, and attribute X_3 is utility independent from attributes X_1 and X_2. These partial utility independence conditions imply that:

- $U(x_1 \mid x_2, x_3) = U(x_1 \mid x_2, x_3^0)$
- $U(x_2 \mid x_1, x_3) = U(x_2 \mid x_1, x_3^0)$
- $U(x_3 \mid x_1, x_2) = U(x_3 \mid x_1^0, x_2^0)$.

The multiattribute utility function (MUF) can be decomposed into lower-order terms by expanding it through the attributes, which present some utility independence. For example, the expansion of a MUF [9,14] through the first attribute leads to the following functional form: $U(x_1, x_2, x_3) = U(x_1^*, x_2, x_3)U(x_1 \mid x_2, x_3) + U(x_1^0, x_2, x_3)\bar{U}(x_1 \mid x_2, x_3)$, where $\bar{U}(x_1 \mid x_2, x_3) = 1 - U(x_1 \mid x_2, x_3)$. A further expansion of the MUF through the remaining two attributes and incorporation of the partial utility independence conditions (see details in [9,14]) will result in the functional form with multiple lower-order terms some of which are duplicate or redundant.

The utility terms arising from the decomposition of the MUF for this decision problem can be encoded as an input ternary matrix for our algorithm (see Figure 2). Here, the input matrix is converted to a decimal vector (#1), the vector is sorted, and duplicates are eliminated (#2), redundancies are excluded (#3), and the vector is converted to a ternary matrix, which is then returned as an output (#4-5-Output). The output represents the five terms, which are required for construction of the multiattribute utility function, namely, $U(x_1, x_2, x_3^0)$, $U(x_1^*, x_2^*, x_3^*)$, $U(x_1^*, x_2^0, x_3^*)$, $U(x_1^0, x_2^*, x_3^*)$, $U(x_1^0, x_2^0, x_3)$. These utility terms should be assessed [8] from the decision maker. Thus, the proposed algorithm eliminates redundant questions for the assessor.

Input			#1	#2	#3	#4-5-Output		
1	1	1	13	24	24	2	2	0
2	2	0	24	15	13	1	1	1
1	2	0	15	13	10	1	0	1
0	0	2	2	12	4	0	1	1
1	1	0	12	10	2	0	0	2
0	0	2	2	9				
1	0	1	10	6				
1	0	0	9	4				
0	1	1	4	3				
2	2	0	24	2				
0	2	0	6	1				
0	1	0	3	0				
0	2	0	6					
0	0	1	1					
0	0	0	0					

Fig. 2. Illustration of the proposed algorithm on a three-attribute decision problem.

4 Simulation Results

The simulation results demonstrate very good performance of the proposed algorithm compared to the brute-force approach. The simulations were done in MATLAB R2017b on a machine with 8GB of RAM and Intel(R) Core(TM) i5-6400 CPU @2.70GHz 2.71GHz under Windows 10 Enterprise. Figure 3 illustrates performance of the proposed algorithm relative to the brute-force approach for different numbers of functions (utility terms) and fixed number of attributes for 10,000 runs of randomly generated input ternary matrices. The average running times of the brute-force approach were set to unit. Comparing to the brute-force approach, the ternary-decimal exclusion algorithm demonstrates excellent performance for various values of *m* for a fixed value of *n*.

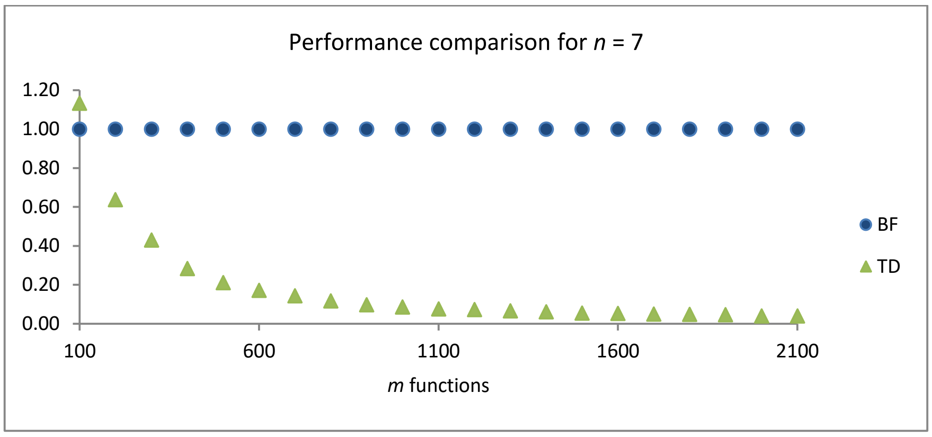

Fig. 3. Performance of the ternary-decimal exclusion algorithm (TD) relative to the brute-force approach (BF) for different numbers of functions.

The proposed algorithm also shows very good performance for different values of *n* (see Figure 4). Note that the value of *m* is proportional to *n* in this test and is equal to half of the cardinality of the ternary matrix. One limitation of the proposed approach is the case when *m* is disproportionately very low for a given *n*. For example, if $m < floor(0.05(3^n))$, then the efficiency of the proposed approach decreases. This gives a motivation for future work.

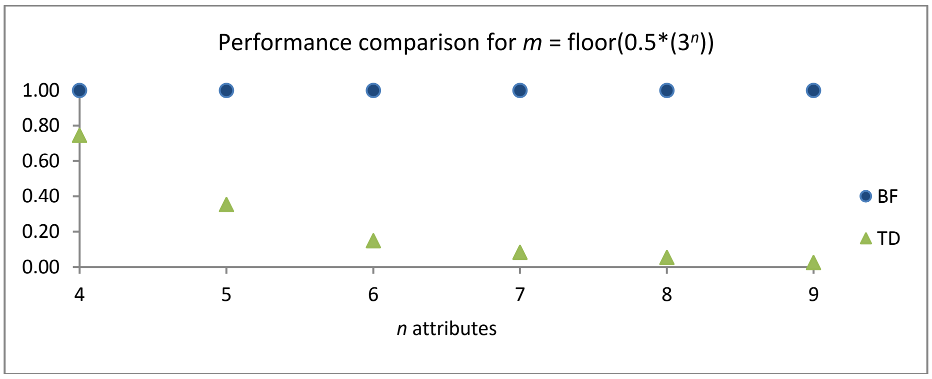

Fig. 4. Performance of the ternary-decimal exclusion algorithm (TD) relative to the brute-force approach (BF) for different numbers of attributes.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_22

5 Conclusion and Future Work

For solving problems under uncertainty, the decision maker often must consider many factors and use multiple criteria (i.e. attributes). The construction of a multiattribute utility function, in this case, may require assessments of various lower-order utility terms. The exclusion of utility terms, which are duplicate or redundant, can be quite challenging in complex decision problems.

This paper introduced a novel method for excluding redundant terms in a multiattribute utility function. The ternary-decimal exclusion algorithm demonstrated excellent performance on random sets. Future work may extend the analysis and possibly find better recurrence relations. The proposed algorithm can be used in decision support systems and can serve as a consistency check for the utility assessments.

References

1. Abdildin, Y. G., Abbas, A. E.: Canonical Multiattribute Utility Functions: Enumeration, Verification, and Application. Procedia Computer Science, 18, 2288–2297 (2013). https://doi.org/10.1016/j.procs.2013.05.400
2. Gros, J.: Power Plant Siting: a Paretian Environmental Approach. Nuclear Engineering and Design, 34, 281-292 (1975). https://doi.org/10.1016/0029-5493(75)90125-9
3. Gros, J., Avenhaus, R., Linnerooth, J., Pahner, P.D., Otway, H.J.: A systems analysis approach to nuclear facility siting. Behavioral Science, 21 (2), 116-127 (1976). http://dx.doi.org/10.1002/bs.3830210206
4. Solomon, B.D., Haynes, K.E.: A survey and critique of multiobjective power plant siting decision rules. Socio-Economic Planning Sciences, 18 (2), 71-79 (1984). https://EconPapers.repec.org/RePEc:eee:soceps:v:18:y:1984:i:2:p:71-79
5. Abbas A.E.: Constructing multiattribute utility functions for decision analysis. In: Hasenbein J, editor. INFORMS tutorials in Operations research 62-98. Hanover, Maryland (2010). https://doi.org/10.1287/educ.1100.0070
6. Keeney, R.L., Raiffa. H.: Decisions with Multiple Objectives: Preferences and Value tradeoffs, Wiley, New York (1976).
7. Keeney, R.L., Sicherman, A.: Illustrative Comparison of One Utility's Coal and Nuclear Choices. Operations Research, 31 (1), 50-83 (1983). https://doi.org/10.1287/opre.31.1.50
8. Abdildin, Y.G., Abbas, A.E.: Analysis of decision alternatives of the deep borehole filter restoration problem. Energy, 114, 1306–1321 (2016). http://dx.doi.org/10.1016/j.energy.2016.08.034
9. Abbas, A.E.: General decompositions of multiattribute utility functions with partial utility independence. Journal of Multicriteria Decision Analysis, 17, 37-59 (2010). https://doi.org/10.1002/mcda.452
10. Knoop, J., Ruthing, O., Steffen, B.: Lazy code motion. ACM SIGPLAN Notices, 27 (7), 224-234 (1992). https://doi.org/10.1145/143095.143136
11. Morel, E., Renvoise, C.: Global optimization by suppression of partial redundancies. Communication of the ACM, 22 (2), 96-103 (1979). https://doi.org/10.1145/359060.359069
12. Paleri, V.K., Srikant, Y.N., Shankar, P.: Partial redundancy elimination: a simple, pragmatic, and provably correct algorithm. Science of Computer Programming, 48, 1-20 (2003). https://doi.org/10.1016/S0167-6423(02)00083-7

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_22

13. Jantzen, S.G., Sutherland, B.J.G., Minkley, D.R., Koop, B.F.: GO Trimming: Systematically reducing redundancy in large Gene Ontology datasets. BMC Research Notes, 4 (267), 1-9 (2011). https://doi.org/10.1186/1756-0500-4-267
14. Abdildin, Y.G., Abbas, A.E.: An Algorithm for Excluding Redundant Assessments in a Multiattribute Utility Problem. Procedia Computer Science, 9, 802–811 (2012). http://dx.doi.org/10.1016/j.procs.2012.04.086
15. Abdildin, Y. G.: Multiattribute utility functions for the deep borehole filter restoration problem. Ph.D. dissertation, University of Illinois at Urbana-Champaign (2014). http://hdl.handle.net/2142/50440, www.ideals.illinois.edu
16. Biggar, P., Nash, N., Williams, K., Gregg, D.: An experimental study of sorting and branch prediction. ACM Journal of Experimental Algorithmics, 12, Article 1.8, 1-39 (2008). https://doi.org/10.1145/1227161.1370599

Appendix

Table 5. Pseudocode of the brute-force approach.

```
Algorithm brute-force(M):
Input:    Ternary matrix M.
Output:   Matrix M without redundant rows.
  r ← 1
  while r < size(M, 1) + 1
    i ← 1
    while i < size(M, 1) + 1
      if i ≠ r
        if M(r, :) ≥ M(i, :)  // if all elms of row r ≥ the corresponding elms of row i
          redundant ← true  // then row i can be redundant due to row r
          for j ← 1:size(M, 2)
            if M(r, j) > M(i, j) AND M(r, j) = 1
              redundant ← false
              break
          if redundant = true
            M(i, :) ← []  // delete row i
            if i < r
              r ← r - 1
            i ← i - 1
      i ← i + 1
    r ← r + 1
  return M
```