UNEDA Cardinal Alternative Ranking Layer Specification

Universal Engine for Decision Analysis

Version 7.21

The CAR (Cardinal Alternative Ranking) layer is implemented on top of UNEDA DTL. It is not a part of DTL and cannot access its data structures. It is, however, linked together with DTL into the UNEDA platform.

The layer enables the standard DTL evaluation functions after the CAR data has been entered using the functions below.

CONTENTS

CAR layer notes 3
Data types 4
Start CAR layer 4
Stop CAR layer 5
Set compatibility parameters 5
The CAR weight ranking process 5
Set CAR weight base (cardinal) 6
Open CAR partial weight hull process 7
Close CAR partial weight hull process 7
Check CAR partial weight hull statement 8
Add CAR partial weight hull statement 8
Set CAR probability base (cardinal) 9
Set CAR value base 9
Set CAR weight base (distance) 10
Set CAR probability base (distance) 10
Error handling 11

CAR layer notes

The CAR layer is a syntactic sugar layer, unlike UNEDA DTL which hosts the base functionality including data integrity and consistency checks. CAR has no knowledge about the inner functions of DTL, no more than any other caller has. CAR should be viewed as convenient sets of call sequences that run in user mode, not system mode. Thus, the layer has no access to the internal data in DTL. This leads to fewer error checks being possible in the CAR layer and more structural control being assumed of the API caller instead. For the same reason, make sure never to mix CAR-generated statements with non-CAR data. This can lead to inconsistencies from data having different origins and the CAR layer has no way of knowing where data comes from.

The CAR data for weights are entered in two stages, first as relative weights and then as partial hull weight verifications. The former should be generated by a relative procedure such as swing, but there is no way for CAR to ensure or check that. The latter stage works by pruning the orthogonal hull around the two weights supplied in the verification statement. It could be seen as a hull-trimming operator in a way similar to how ordinary normalisation works. And in analogy, ordinary normalisation does not guarantee that every pair of weight combinations taken from the hull are feasible. Similarly, there is no guarantee that weight combinations drawn from the pruned hull are together compliant with the verification statement. Here, the analogy does not continue since normalisation can also detect which combinations of weights are feasible while there is no similar operator to detect which combinations of partial weights have been verified or not. Such an operator would have to be non-linear by design since the verifications for intervals are ratios ($w_1/w_2 \geq v_2/v_1$) which are inherently non-linear. While the CAR layer can verify and prune the hull, it is up to the API caller to ensure that the weights ultimately are kept within the specifications supplied by the user.
Alternatively, the CAR layer can cut the hull to ensure that no points where the ratios do not hold are included.

Thus, the partial hull procedure consists of the following steps:

1. Start with value differences $v_1 = v_{12}-v_{11}$ and $v_2 = v_{22}-v_{21}$.
   Assume that the user states that v_1 is globally preferred to v_2.
   Note that they originate from different criteria Cr.1 and Cr.2.
2. Compute the ratio between the value diffs (v_1/v_2).
3. Establish the corresponding ratio between the weights (w_1/w_2)
   such that on the MC scale the ratio is 1.0 or more (i.e. that the difference on the scale for Cr.1 is overall worth at least as much as that on the scale for Cr.2 when projected onto the common [0,1] MC scale).
4. Adjust (prune) the hulls of the weights w_1 and w_2 accordingly.

Step 4 is carried out by shearing into the hull boundaries w_1^{min} and w_2^{max} so that $v_1 \cdot w_1 > v_2 \cdot w_2$ holds for consistent weight pairs w_1 and w_2, either somewhere within the hull (prune) or in its entirety (cut).
Thus, while pruning is conservative, cutting is more radical.

Data types

The data types are piggybacked onto DTL, except for the input vectors to the CAR calls which are defined locally in CAR.h.

```
#typedef int car_vector[MAX_CONS+1]
```

Start CAR layer

Call syntax: CAR_init(int method, int mode)

Return information:
OK   -
ERROR – state error
        input error

Call semantics: Perform initialization of the CAR layer. This must be the first call to CAR. The parameter 'method' controls the method for generating surrogate weights and probabilities. 0 = 'keep default method' is strongly advised as the call parameter except for advanced research. The default method is an adaptive cardinal version of the RX surrogate method. If optional methods are desired, they should be set as follows (see research papers for explanations):

  1 = CRS (cardinal version of the RS surrogate method)
  2 = CRR (cardinal version of the RR surrogate method)
  3 = CXR (cardinal version of the RX surrogate method)
  4 = CSR (cardinal version of the SR surrogate method)
  5 = CRC (cardinal version of the ROC surrogate method)
  5 = CXR (cardinal version of the XR surrogate method)

The parameter 'mode' controls the compatibility back to Excel versions of CAR for DTL. 'mode' is also the toggle flag for some options. If optional modes are desired, they should be set as follows:

  0 = standard modern CAR
 +1 = backward compatible weights with older Excel models
 +2 = backward compatible values with older Excel models
 +4 = CAR light mode -> partial weight hull runs without midpoints

These modes are configurational rather than user-specific.

NOTE: Never mix CAR-generated statements with non-CAR data. This can lead to inconsistencies from data having different origins. Remember that CAR is a syntactic sugar layer.

NOTE2: If CAR data has been generated with CAR_light=OFF, tornados with odd mode numbers will automatically be used (no midpoints), since a full set of midpoints cannot move around and would only result in empty tornados.

Stop CAR layer

Call syntax: CAR_exit()

Return information:
OK -
ERROR – state error

Call semantics: Perform closedown of the CAR layer. This function is void-valued since nothing can go wrong.

Set compatibility parameters

Call syntax: CAR_set_compat(int w_unc, int v_unc)

Return information:
OK -
ERROR – state error
input error

Call semantics: Sets the uncertainty levels for Excel compatibility with older IIASA models. 'w_unc' is for weight statements and 'v_unc' is for value statements. Valid input is between 0.01 and 0.20 (for 'w_unc') or 0.10 (for 'v_unc'). Defaults are 0.10 for 'w_unc' and 0.05 for 'v_unc'.

The CAR weight ranking process

The weight ranking process in CAR is a *two-stage process* in which the weights are entered in the first stage and then verified/consolidated in the second stage. The first stage is carried out in the user interface and the results are added to the weight base using CAR_set_W_base. It is up to the discretion of the API caller to ensure that the weights have actually been generated using a relative process. This cannot be checked by the CAR layer. In the second stage, the results from the first stage are verified and consolidated by comparing parts of the scales in the user interface (the first stage concerns the full scales) and then entering the results into CAR. A structural pseudo-code for the two-stage process might look like this (where jcall is a generic error-handling routine):

```
/* User value-setting interaction */
...
if (jcall(CAR_set_V_base(…)) < CAR_OK)
    handle_error();
/* Values completed – weight elicitation can commence */
...
/* User weight elicitation interaction */
...
/* Set the weight base */
if (jcall(CAR_set_W_base(…)) < CAR_OK)
    handle_error();
/* Loop through partial weight session */
if (jcall(CAR_start_W_phull()) < CAR_OK)
    handle_error();
while (user_wants_to_verify) {
    show_phull_opportunities();
    /* Establish which partial scales to compare */
    ask_for_phull_to_verify();
```

```
    load_uwstmt(&uwstmt);
    tradeoff = 0.0;
    if (jcall(CAR_check_W_phull(…)) < CAR_OK)
        handle_error();
    if (tradeoff < 1.0) {
        display("partial hull impossible - inconsistent due to too weak trade");
        display_graphics_feedback_1();
        }
    else
        rc = CAR_ prune/cut/equal_W_phull(&uwstmt);
    if ((rc < CAR_OK) && (rc != CAR_INCONSISTENT)) {
        jcall(rc);
        handle_error();
        }
    else if (rc == CAR_INCONSISTENT) {
        display("partial hull impossible - inconsistent due to mix of statements");
        display_graphics_feedback_2();
        }
    else {
        display("partial hull statement added to the weight base");
        display_graphics_feedback_3();
        }
    }
if (jcall(CAR_finish_W_phull()) < CAR_OK)
    handle_error();
/* Continue after partial hull session */
...
```

Allowing *impure* criteria trees (trees with mixed real and intermediate nodes having the same parent) is not allowed from a conceptual point of view. It is inconceivable that a decision-maker is able to compare worst-to-best for compound weights -> those comparisons will instead invariably and unfortunately be *absolute weights*, a fallacy shared with all other products and tools on the market. The immediate and obvious remedy to this dire situation is to have the UI draw the weights of an impure structure in a tree format but keeping it a one-level structure internally. (Earlier, it was feasible to allow the ranking of impure stakeholder weight trees because CAR was also the mother layer for CSR rankings. This was, in reality, the raison d'être for keeping the mode option.) Likewise, allowing partial hull verification in impure trees is in theory highly problematic for the above-given reasons. And while theoretically invalid, it still works reasonably well in the code library. It is included for the tidiness of the UI but is basically unsound.

NOTE: For CAR_set_W_base as well as CAR_check_W_phull and CAR_cut_W_phull it is up to the API caller to ensure that the input data conforms to an elicitation model where relative weights are obtained. The API caller must keep track of the statements made by the user, and their integrity and applicability. Plus ensure that they are part of a sound and solid relative elicitation process such as swing. The CAR layer is not a part of DTL but only a syntactic service layer with no more privileges than any other DTL caller. CAR cannot inspect the DTL data structures for integrity or conformity.

Set CAR weight base (cardinal)

Call syntax: CAR_set_W_base(int n_nodes, car_vector ord_crit, car_vector rel)

```
Return information:
OK    - number of statements added (also in ord_crit[0])
ERROR – not activated
        frame not loaded
        wrong frame type
        input error
        illegal tree
        inconsistent
```

Call semantics: Statements for all criteria with the same parent node are added to the weight base at the same time. 'n_nodes' is the number of nodes being compared in total. 'ord_crit' is a vector of node numbers from most important to least important criterion. 'rel' is the relation between the current node and the next, expressed as 0 for '=' and 1-3 for '>', '>>', and '>>>' (entering -1 signifies that the rest of the nodes are nullified, i.e. given the weight zero). It is presupposed that the relations have been obtained by a *relative elicitation process*, thus representing relative weights. Absolute weights should not be entered here (unless the option mode +4 is activated) and cannot be subsequently verified by partial hull statements later in the process. The number of added statements is returned in ord_crit[0]. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base.

Open CAR partial weight hull process

Call syntax: CAR_open_W_phull()

```
Return information:
OK    -
ERROR – not activated
        frame not loaded
        wrong frame type
        not allowed
        inconsistent
```

Call semantics: The partial hull verification process is initialised. This call must be made before any other partial hull function calls.

Close CAR partial weight hull process

Call syntax: CAR_close_W_phull()

```
Return information:
OK    -
ERROR – not activated
        frame not loaded
        wrong frame type
        not allowed
        inconsistent
```

Call semantics: The partial hull verification process is completed. This call must be made after all other partial hull function calls.

Check CAR partial weight hull statement

Call syntax: CAR_check_W_phull(struct user_w_stmt_rec* swp, real *tradeoff)

Return information:
OK - tradeoff updated
ERROR – not activated
frame not loaded
wrong frame type
illegal tree
input error
inconsistent

Call semantics: A partial hull verification statement is checked against the weight base. This must be done after the CAR_set_W_base() call, which is step one in the weight ranking procedure. The check will do two things. It will reveal any input errors in the fields and also whether a subsequent call to CAR_add_W_phull() or CAR_cut_W_phull() will succeed. If the trade-off is greater than 1, the verification will succeed provided the weight base is *clean*, i.e. not mixed with other statements such as links. If, when calling, the 'tradeoff' parameter contains -1.0 (or -2.0) then the maximal (or minimal) global MC-scale trade-off for each weight is returned in their respective fields in swp (which thusly are overwritten). For all other values of 'tradeoff', the swp structure is left unchanged. If 'tradeoff' contains a negative value on return, the weight base contains elements that originate from non-CAR calls.

Add CAR partial weight hull statement

Call syntax: CAR_add_W_phull(struct user_w_stmt_rec* swp)
Call syntax: CAR_cut_W_phull(struct user_w_stmt_rec* swp)
Call syntax: CAR_equal_W_phull(struct user_w_stmt_rec* swp)

Return information:
OK - number of statements added
ERROR – not activated
frame not loaded
wrong frame type
illegal tree
input error
inconsistent

Call semantics: Partial weight verification information is added to the weight base. This must be done after the CAR_set_W_base() call, which is step one in the elicitation procedure. It is presupposed that the relations in step one have been obtained by a relative process, thus representing relative weights. The verification statements entered through this call constitute step two of the same procedure. The statements are of the type $a \cdot w_i > b \cdot w_j$ for weights w_i and w_j, and constants a and $b \in (0,1]$ from the elicitation process. The statement record is filled in the usual way, except that a is stored in swp->lobo and b is stored in swp->upbo. *Pruning* ensures that $a \cdot w_i > b \cdot w_j$ holds for some consistent pairs w_i and w_j within their respective hulls while *cutting* ensures that $a \cdot w_i > b \cdot w_j$ holds for all consistent pairs w_i and w_j within the hulls. The *equal* call verifies that $a \cdot w_i \approx b \cdot w_j$ holds instead. In case of success, the number of statements added to the weight base is returned. In case of success, the number of

statements added to the weight base is returned. In case of inconsistency, nothing is added to the base. It is not left in an inconsistent state. This call is for acknowledging a global value difference between consequence midpoints or between numerically determined parts of two local value scales.

Set CAR probability base (cardinal)

Call syntax: CAR_set_P_base(int crit, int alt, int n_nodes, car_vector ord_nodes, car_vector rel)

Return information:
OK - number of statements added (also in ord_nodes[0])
ERROR – not activated
frame not loaded
input error
criterion unknown
not allowed
alternative unknown
inconsistent

Call semantics: Statements for all probabilities with the same parent node are added to the probability base at the same time. 'crit' is the criterion that the probabilities belong to. 'alt' is the alternative that all of the probabilities belong to. 'n_nodes' is the number of nodes being compared in total. 'ord_nodes' is a vector of node numbers from most probable to least probable event. 'rel' is the relation between the current node and the next, expressed as 0 for '=' and 1-3 for '>', '>>', and '>>>' (entering -1 signifies that the rest of the nodes are nullified, i.e. given the weight zero). The number of added statements is returned in ord_nodes[0]. The base is checked for consistency concerning all the new ranges. In case of inconsistency, nothing is added to the base.

Set CAR value base

Call syntax: CAR_set_V_base(int crit, car_vector ord_alts, car_vector ord_nodes, car_vector rel)

Return information:
OK - number of statements added (also in ord_nodes[0])
ERROR – not activated
frame not loaded
input error
criterion unknown
not allowed
alternative unknown
inconsistent

Call semantics: Statements for all values in a criterion are added to the value base at the same time. 'crit' is the criterion that the values belong to. 'ord_alts' is a vector with the alternative each consequence value belongs to. 'ord_nodes' is a vector of node numbers from most valuable to least valuable consequence. 'rel' is the relation between the current node and the next, expressed as 0 for '=' and 1-9 for '>', '>>', etc. The number of added statements is returned in ord_nodes[0]. The base is checked for

consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base.

Set CAR weight base (distance)

Call syntax: CAR_rank_W_base(int n_nodes, car_vector ord_crit, double dist)

```
Return information:
OK    - number of statements added (also in ord_crit[0])
ERROR – not activated
        frame not loaded
        wrong frame type
        input error
        illegal tree
        inconsistent
```

Call semantics: Statements for all criteria with the same parent node are added to the weight base at the same time. 'n_nodes' is the number of nodes being ranked in total. 'ord_crit' is a vector of node numbers from most important to least important criterion. 'dist' is the distance between the intervals of two adjacent weights. 'dist'=0 yields adjacent intervals, 'dist'>0 yields a gap between the intervals (with 1 yielding intervals of width zero), and 'dist'<0 yields an overlap between the intervals (with –1 yielding intervals of double width). The number of added statements is returned in ord_crit[0]. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base.

Set CAR probability base (distance)

Call syntax: CAR_rank_P_base(int crit, int alt, int n_nodes, car_vector ord_nodes, double dist)

```
Return information:
OK    - number of statements added (also in ord_nodes[0])
ERROR – not activated
        frame not loaded
        input error
        criterion unknown
        not allowed
        alternative unknown
        inconsistent
```

Call semantics: Statements for all probabilities with the same parent node are added to the probability base at the same time. 'crit' is the criterion that the probabilities belong to. 'alt' is the alternative that all of the probabilities belong to. 'n_nodes' is the number of nodes being compared in total. 'dist' is the distance between the intervals of two adjacent weights. 'dist'=0 yields adjacent intervals, 'dist'>0 yields a gap between the intervals (with 1 yielding intervals of width zero), and 'dist'<0 yields an overlap between the intervals (with –1 yielding intervals of double width). The number of added statements is returned in ord_nodes[0]. The base is checked for consistency concerning all the new ranges. In case of inconsistency, nothing is added to the base.

Error handling

All CAR calls return an integer that serves as an error code and information carrier at the same time. In the event of an error, a negative number is returned. The caller should interpret the error code and take action accordingly. The definitions are found in CAR.h.

CAR_INPUT_ERROR

One of the input parameters contained invalid information.

CAR_ILLEGAL_TREE

Impure trees not allowed in CAR.

CAR_FRAME_NOT_LOADED

An attempt to use frame commands while no frame is loaded.

CAR_WRONG_FRAME_TYPE

An attempt to issue a PS/PM-only command to a DM frame or vice versa.

CAR_STATE_ERROR

A call to DTL is made when CAR is in the wrong initialization state.

CAR_CRIT_UNKNOWN

The requested criterion does not exist.

CAR_ALT_UNKNOWN

The alternative does not exist.

CAR_NOT_ALLOWED

The call is not allowed since the frame was created without car_light set.

CAR_NOT_ACTIVATED

A call to CAR is made when CAR is in the wrong initialisation state.

CAR_INCONSISTENT

The supplied statement is inconsistent.

CAR_ILLEGAL_TREE

The tree structure supplied is invalid.

CAR_SYS_CORRUPT

The internal data structures of CAR are misaligned.