

## An Optimal Algorithm for Raw Idea Selection under Uncertainty

Nadine Kempe  
Faculty of Computer  
Science, University of  
Magdeburg  
[kempe@sim-md.de](mailto:kempe@sim-md.de)

Graham Horton  
Faculty of Computer  
Science, University of  
Magdeburg; Zephram GbR  
[graham@sim-md.de](mailto:graham@sim-md.de)

Robert Buchholz  
Faculty of Computer  
Science, University of  
Magdeburg  
[robert@sim-md.de](mailto:robert@sim-md.de)

Jana Görs  
Zephram GbR,  
Magdeburg  
[jana.goers@zephram.de](mailto:jana.goers@zephram.de)

### Abstract

*At the first gate of an innovation process, a large number of raw ideas must be evaluated and those good enough to continue to the next phase be selected. No information about these ideas is available, so they have a high level of uncertainty. We present an algorithm that selects and ranks a set of alternatives in optimal time. The algorithm addresses uncertainty by allowing decision-makers to specify missing information that affect the outcome of their judgments. It generates multiple partial rankings efficiently according to the various possible combinations of missing items of information and identifies the set of items that are needed to obtain a unique result. In this manner, we can reduce the uncertainty in the selection procedure and make explicit expert knowledge that is relevant to the evaluation process. The algorithm is intended for use in a collaborative tool for corporations who utilize a structured innovation process.*

### 1. Introduction

In this paper, we consider a decision-making task with incomplete information. The application of interest is the idea phase of the innovation process, which is often referred to as the Front End of Innovation (FEI). The role of this phase is to develop ideas for new products and services which can be subsequently developed and marketed.

The FEI is an important interdisciplinary business process which we believe can benefit from collaborative tools. The FEI is described by managers as the weak link in corporate innovation activities [11]. It has been estimated, for example, that 50% of the money spent on innovation projects is wasted [2] and that out of 3000 ideas, only one becomes a commercial success [19]. This can be attributed in part, at least, to a lack of systematic innovation processes [8]. We find very little on this subject in the scientific literature, and a survey by Vorbach and

Perl [20] revealed that there are no commercially available tools for the FEI that cover the entire range of tasks that are involved.

The Front End of Innovation consists of two phases: a divergent phase, in which ideas are collected and generated, followed by a convergent phase, in which these ideas are enriched, evaluated and selected – usually several times. The function of the convergent phase is to identify a small number of ideas for implementation from a possibly very large number of initial ideas.

Evaluation and selection are usually carried out by a group of experts, who can contribute different perspectives on the ideas. In a large corporation, such experts' time is very valuable and it is therefore important to design an idea evaluation and selection meeting efficiently.

Uncertainty is one of the core problems of the FEI. Uncertainty is defined by Galbraith in [7] as "the difference between the amount of information required to perform a particular task, and the amount of information already possessed". In the case of the FEI, the task to be performed is the introduction of a new product or service to the market.

At the first stage of the innovation process, the ideas are only very briefly described. They may consist of only a few words or sentences, and no additional information about them is provided. They therefore possess a high degree of uncertainty. Herstatt and Verworn identify this lack of information as a "limiting factor" for the FEI [9]. In practice, this can lead to two significant difficulties: a hiatus in the selection process and/or uncertainty in the overall selection result.

If, during the course of the evaluation process, an expert discovers that he or she is lacking an item of information which they need in order to make a judgment, they have two options: either to put off the judgment until the information becomes available or to make the judgment anyway. In the first case, the selection cannot be completed, which can lead to

substantial extra costs. In the second case the selection result contains an uncertainty which is both hidden and unquantifiable. Both consequences are extremely undesirable for the Innovation Manager.

A selection step in the FEI would ideally perform two functions: first separate the ideas that are to remain in the process from those that are to be discarded, and second prioritize the winning ideas.

In this paper, we present a selection algorithm which performs both of these functions and which additionally addresses the problem of incomplete information. The method is based on a partial sorting algorithm, which has optimal complexity in the case of complete information.

The algorithm is based on pairwise comparisons, whereby the decision-maker (DM) is allowed to input a condition which determines the result of the comparison. This condition represents the unknown information. The algorithm then computes partial rankings for all possible combinations of such conditions, identifying those conditions which affect the winning set. These conditions can then be researched, yielding a unique selection result with a lower uncertainty than might otherwise have been the case. Conditions that do not affect the winning set do not need to be researched.

A further benefit of our approach can be seen from a knowledge management perspective. By giving experts the opportunity to input information they believe is relevant to the evaluation of potential innovations, they add to the explicit knowledge base of their organization. In particular, the information can be made available to the other participants in the collaborative selection task.

Although the selection task may in practice involve several criteria and experts, we restrict ourselves here to a single criterion and DM. Ordinal-scale outranking methods such as ORESTE [14] may be used to compute overall rankings from the individual ones, and simple schemes such as majority voting can be used to aggregate the judgements of several DMs.

## 2. Related Work

Models for the FEI have been suggested by Koen et al. [13] and by Cooper [2]. In the first case, a cyclic process is proposed, in which the phases interact non-linearly in order to develop an idea to maturity. In the latter case, a bipartite model is used, in which ideas progress linearly from generation to maturity. In both cases, idea enrichment and selection play a key role.

Methods of multiple criteria decision analysis differ in their underlying models for the decision problem. These differences lead to different approaches to both the types of judgment used and the method of aggregating judgments to obtain an overall result.

In decision analysis models based on utility theory [4], uncertainty is mapped onto risk; the degree of uncertainty is required to be represented by a probability, enabling an expected value for the utility to be computed. Utility-based methods are therefore based on a cardinal scale. For reasons described below, we discount utility-based methods for decision-making in the FEI.

Proponents of fuzzy techniques in decision-making suggest using fuzzy measures to represent uncertainty [16]. Here, linguistic classifications of the degree of uncertainty such as "very uncertain" or "quite certain" are required of the DM. In our application, we prefer however to treat uncertainty as a named information deficit rather than as a "degree of knownness", since our goal is to identify specific items of information to be acquired in a subsequent phase of the process.

Decision methods based on outranking [6], [14] represent alternatives on an ordinal, rather than a cardinal scale. Here, alternatives are assigned relative positions, and the fundamental unit of user input is a preference judgment, or pairwise comparison. Some of these methods, such as ORESTE [14] only assume minimal preference judgments of the form "alternative A is preferable to alternative B", whereas others allow an intensity of preference expressed either verbally "A is strongly preferable to B", or numerically "A is seven times as good as B". The well-known Analytical Hierarchy Process by Saaty [18] is an example of the latter class.

One fundamental assumption common to all methods that require cardinal or scaled ordinal judgments is that the DMs are in a position to make such differentiated assessments. This is the case in many applications, where quantitative data on the alternatives is available. However, in the FEI, no data is available on the raw ideas, and judgments must be made largely intuitively. For this reason, we claim that such judgments are not appropriate for our application. In this question, we concur with Bana e Costa et al. [1], who state, "*They require [the decision maker] to be able to produce, either directly or indirectly, numerical representations of his or her strengths of preferences, which is not a natural cognitive task*".

For these and further reasons given by Saaty [17], we conclude that the only appropriate type of

judgment for our application is the minimal preference judgment.

Theoretical work on the field of decision making (see for example [15]) suggest incorporating the concept of “not preference” into the model (representing indifference, incomparability etc.). For the sake of simplicity, we did not include this option for the DM. See the last paragraph of Section 4.3 for a suggestion about how to handle those cases anyway.

### 3. Description of the problem

The convergence phase of the FEI consists of three different operations: enrichment, evaluation and selection.

Enrichment is the means by which ideas are extended and grown. Raw ideas that enter the innovation process are often little more than a few sentences. By the end of the FEI, these will have developed into multi-page documents that may contain the results of technical studies, market research, financial estimates and customer interviews. The goal of enrichment is to reduce the uncertainty in an idea and thus enable an accurate evaluation.

Evaluation refers to the estimation of the ability of an idea to achieve a given goal. Evaluations can be made either on an absolute scale, such as with utility-based approaches [4] or on an ordinal scale [14], which assigns relative values to ideas. In general, evaluation is carried out with respect to several criteria; in the case of the FEI, these criteria may be related to competitors, customer needs, technical feasibility or profit margins. The goal of evaluation is to enable the correct selection to be made.

Selection is the operation which culls ideas from the innovation process. The ideas that are deemed best suited to attaining the innovation goals are promoted to the next phase, while those that are not fit enough are eliminated. Cooper et al. [3] describe additional selection options, which we do not consider here. The goal of selection is to maximize the business benefit from the idea portfolio.

The second author is a partner in a consulting company specializing in the FEI. In face-to-face idea evaluation workshops, one very common problem is the unwillingness of experts to judge ideas because their value depends on a key piece of information which is unavailable. The expert is encouraged to note this down, and care is taken in the subsequent phase of the innovation process to ensure that the concern is addressed. The missing information may take many different forms, for example technical (“*Can we find a material that has property X?*”),

market-oriented (“*Does competitor X have an offer that does Y?*”), political (“*Is our CEO prepared to support projects of type X?*”) or simply informational (“*What is the current price of lithium?*”).

We assume in the following that such missing information can be formulated as statement which can be either true or false. The judgements will then depend on the truth value of these statements. The fourth example above would be phrased, for example, as “*We are able to purchase lithium at less than X dollars per kilogram.*”

The algorithm to be developed has to solve the following tasks:

**Selection.** Select the set of ideas (the “winning set”) deemed to be good enough by the DM. This enables the inadequate ideas to be eliminated.

**Ranking.** Rank the ideas in the winning set. This gives an indication of which ideas to prioritize in the following phase of the innovation process.

**Conditions.** Allow the DM to input missing information as conditions when entering his or her judgments. Compute selections and rankings for each possible combination of conditions. Indicate which conditions are needed to uniquely determine the selection and ranking.

The boundary conditions which the algorithm should adhere to are:

**Size of winning set.** The number of ideas in the winning set may either be a parameter of the algorithm or be determined on-the-fly by the DM as the selection progresses.

**Complexity.** Of course, as for any algorithm, the computational complexity should be as low as possible. In this case, we measure the complexity in terms of the number of judgments that the DM is required to make.

**Appropriate judgments.** The algorithm should require judgments from the DM that are appropriate to the situation. From the arguments presented in the previous Section, this condition leads to the minimal preference judgment.

The algorithm to be developed combines the selection and the prioritization of the winning set into a single step. This is known in Computer Science as *partial sorting* [12]. An everyday example of partial sorting is determining the top ten matches from an online search request and presenting these ten matches in ranking order while ignoring the rest.

### 4. Our approach

Our approach combines two concepts. The first of those is the above-mentioned *partial sorting*, which will be described in Section 4.1.

The second concept, namely the *Conditions* and the way we modeled them based on our experience in the FEI is the subject of Section 4.2.

How we incorporated both of them into an algorithm is shown in Section 4.3, together with an example in Section 4.4.

#### 4.1. Optimal partial sorting: Tournament Sort

Sorting a set of alternatives  $A = \{a_1, a_2, \dots, a_n\}$  using pairwise comparisons in the format  $a_i > a_j$  is a well-known task in Computer Science. For example the book by Knuth [12], consists of over 750 pages discussing only the question of “Sorting and Searching”.

It is proven that the lower complexity bound for completely sorting a set of  $N$  alternatives is  $N \log N$  [12]. This is much faster than the naïve approach of comparing each alternative against every other, which needs  $N^2/2$  comparisons: Sorting 50 alternatives needs 282 comparisons instead of 1250. As the comparisons have to be made by the DM, this savings potential is important for us.

Our application is designed to deal with a single criterion for the comparisons made by a single DM. Using those restrictions, it is safe to assume that the DM makes rational judgments, which translates into the transitive property of pairwise comparisons:

$$(a_i > a_j) \wedge (a_j > a_k) \Rightarrow a_i > a_k$$

Transitivity is important, because it allows us to use optimal algorithms which have no redundant judgments. By contrast, other decision-making methods (such as AHP [18]) do not make this assumption, but require additional judgments

One algorithm that computes a partial ranking using the lower bound for the number of necessary comparisons is Tournament Sort, which is extensively explained in [12]. Besides being optimal in the number of pairwise comparisons, Tournament Sort has a very important feature for our application: it fulfills our requirement of the flexible *size of winning set* stated in Section 3.

The first output that Tournament Sort generates is the winner of the tournament. Then, additional tournaments are carried out to determine the 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> and so on alternatives. This process can be stopped at any rank or when a preset cutoff rank has been reached. This scheme has the advantage of selecting the alternatives in descending order; with the knowledge that the remaining alternatives are worse than the ones already ranked, the DM can stop when he or she thinks the alternatives are getting too bad to remain in the process.

Tournament Sort organizes the necessary comparisons into rounds. The winners of each round move on to the next round until there is only the final winner left (the same principle holds for example for the playoffs of the National Football League of the USA). See Figure 1 for an example where the winner of the letters A – H is determined using the lexicographical order.

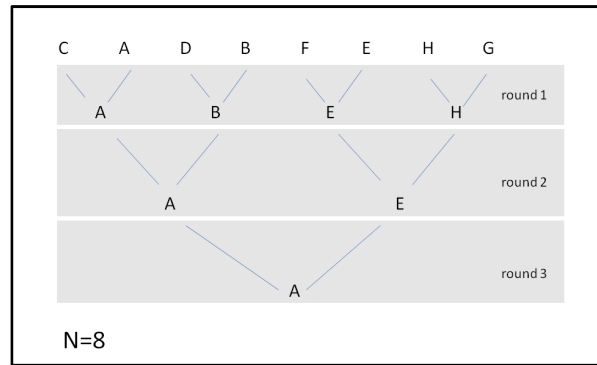


Figure 1. Tournament sort example

Note that the losers of each comparison are not excluded entirely from consideration but remain in the tournament for the following runs.

In order to get the partial ranking we need, the 2<sup>nd</sup> best alternative has to be selected now. To do so, the winner has to be removed from the current tournament and a new run has to be started, re-using the already known comparisons results.

That means we have to update the branch of the tree the winner was in during the original tournament run. We do this by eliminating the former winner in the input set and then updating each of the following comparisons in its branch.

This process will stop when either the winner is reached or when one comparison along the branch stays the same as it was before.

In the example in Figure 1, looking for the 2<sup>nd</sup> best alternative would mean A is eliminated and C is directly promoted in round 1. From there, C has to be compared against B in round 2 and the winner of this comparison (which is B) has to be compared against E in the final round. That means the 2<sup>nd</sup> best alternative here is B.

This scheme has to be repeated until the size  $k$  ( $1 \leq k \leq N$ ) of the winning set is reached.

#### 4.2. Missing information and its influence

As described in Section 3, we model the missing information influencing the DMs judgments using a m-tuple of conditions  $C = (c_1, c_2, \dots, c_m)$  where

$c_i \in \{true, false\}$ . This means the DM may state that alternative  $a_j$  is better than  $a_k$  if condition  $c_i$  is true:

$$\text{if } c_i = true \text{ then } (a_j > a_k)$$

This concept states some requirements for the missing information we model. First of all, the verbalization as a statement implies that only the options  $c = true$  or  $c = false$  are possible.

The second requirement is the logical implication of the stated condition. In order to be able to calculate the impact a condition has, we need the opposite of the comparison to be true when the condition is reversed:

$$\begin{aligned} \text{if } c_i = true \text{ then } (a_j > a_k) &\Rightarrow \\ \text{if } c_i = false \text{ then } (a_j < a_k) & \end{aligned}$$

In order to be able to describe different kinds of missing information, we allow individual conditions to influence multiple pairwise comparisons.

The implication of this for the DM is that he/she can re-use conditions which have already been entered into the system. This means that given two alternatives, the DM has three options:

1. Compare the two alternatives without imposing any condition.
2. Compare the two alternatives by making the result dependent on a new condition which has to be entered.
3. Compare the two alternatives by making the result dependent on an already existing condition.

Note that one specific comparison may also be influenced by multiple conditions.

Using this approach, we can describe different kinds of missing information based on the impact they have on the partial ranking.

The first is a piece of information that relates only to a single idea. This could be uncertainty due to ambiguity about the idea itself or missing details about its implementation. In this case, the condition only influences the pairwise comparisons that involve the corresponding alternative. Conditions of this type influence the position the alternative will occupy in the final ranking while leaving the relationships between the remaining alternatives unchanged. This kind of condition we will refer to as *single-impact*.

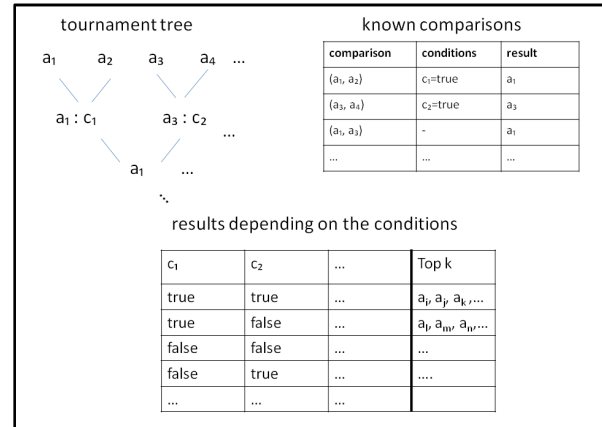
On the other hand, there are conditions which influence multiple alternatives. For example, this could be the price of a raw material which affects the profitability of more than one idea. These conditions change the rank of multiple alternatives relative to

each other and relative to the rest of the ranking. This kind of condition will be called *multiple-impact*.

Another dimension of distinction for the conditions is by the influence they have on the winning set. As we are only looking for the ranks of the first  $k$  alternatives (*Top k*), a condition is *irrelevant*, if its being true or false only influences alternatives which are not within the *Top k*. Conversely, we call a condition *relevant*, if its boolean value changes either the rank of one of the *Top k* or if it determines whether one alternative will leave or enter the *Top k*. In other words, the condition is needed to uniquely determine the final partial ranking.

### 4.3. Our algorithm: sorting under the influence of missing information

After describing the sorting scheme we use and the different kinds of missing information with their impact, we now want to present the algorithm that incorporates both of them.



**Figure 2. Schematic description of the three main data structures of our algorithm**

The algorithm works on three data structures which are shown schematically in Figure 2. The first structure is the tournament tree already introduced in Figure 1. In this tree, the current tournament with the winner of each comparison is saved and if this winner depends on a condition, the winner is flagged accordingly. If a winner  $a_i$  depends on a condition  $c_j$  being true we will use the notation  $a_i : c_j$ .

The second structure is a list of the already-known comparisons along with their outcome and if applicable the conditions under which they were made. We will refer to this structure by  $Q$ . As we will show later, there is a considerable number of comparisons which are needed at several points during the algorithm and thus can be reproduced

without having to ask the DM the same question twice. Note that using the definitions in Section 4.2, we can deduce the comparison result of  $c_j = false$  from the case  $c_j = true$ .

The third structure represents the result of the algorithm, namely the *Top k* under all possible combinations of true and false for the conditions the DM provides. As the explicit conditions that influence the result of the partial ranking we are looking for are not known beforehand, our algorithm has to first, identify them, and second calculate their impact on the ranking.

We will use  $\bar{C}$  to denote one specific combination of those boolean values for all known conditions. Therefore, our algorithm will carry out the Tournament Sort scheme presented in Section 4.1 for all possible  $\bar{C}$ . As  $C$  is empty in the beginning of the algorithm, we will start without any condition.

The alternatives are compared with each other using the Tournament Sort scheme. Whenever a pairwise comparison result is needed, it is first checked, if we already know the outcome under the current  $\bar{C}$  (i.e. the comparison is element of  $Q$  under the current  $\bar{C}$ ).

If the comparison itself or its outcome under the current  $\bar{C}$  is unknown, the DM has to provide the judgment and the result is added to  $Q$ . If the DM did not state a new condition for the comparison, the Tournament Sort scheme is carried on.

If a new condition is stated, it is added to  $C$ . In this case, we have to interrupt the Tournament scheme. The reason for this is the new condition. So far, we computed the ranking assuming one specific true / false combination for the known conditions. As we now discover a new condition, we have to consider it in our combination. Therefore, the Tournament Sort is started anew.

This will end when, for all combinations of conditions, the partial ranking is completed without the need of any unknown condition.

Note that by progressing within the algorithm, more and more comparisons are known to us and therefore can be reproduced without asking the DM. More precisely, up to each interruption, all previously needed comparisons are now known to us and furthermore, we know they do not depend on an unknown condition. Therefore the process can be reproduced up to the point of interruption without the interaction of the DM.

During the algorithm, the order of combinations which is used does not influence the final outcome or the number of comparison the DM has to provide.

Note that a known comparison result that is not influenced by any conditions is automatically known for all condition combinations (i.e. if one specific

comparison does not depend on any condition, the conditions becoming true or false do not change the outcome of this comparison).

Summarized from the explanation above, the pseudo code for the proposed algorithm follows.

```

C = {} //known conditions
Q = {} //known comparisons
label restart
for each true/false combination  $\bar{C}$  of C
  for each required comparison  $q_i$ 
  in Tournament Sort
    if  $q_i \in Q$  under current  $\bar{C}$ 
      get result from Q
    else
      get result from
      decision-maker,
      potentially using  $\bar{C}$ 
      add  $q_i$  under  $\bar{C}$  to Q
      if result depends on a
      new condition  $c_j$ 
        add  $c_j$  to C
        goto restart
    perform Tournament Sort
    step using the result
  save Top k for current
  combination

```

As mentioned in Section 2, there might be the need for the DM to express any kind of “not preference” during the algorithm. One way handling those cases would be to phrase the respective comparison as a condition (e.g.  $a_j$  is better than  $a_i$  if  $a_j$  is better than  $a_i$ ). Doing this, the decision is postponed until the algorithm is finished and can be cared for afterwards (if it is relevant).

#### 4.4. Example

For this example, we will use the letters A – H with lexicographical order from Figure 1 to represent alternatives, and the task is to obtain the *Top 3*. As our algorithm proceeds, the DM will stipulate two conditions which we denote by X and Y. Those two conditions are specified as follows:

- If X is true, then alternatives D and E are superior to all other alternatives. Relative to each other, D is better than E.
- If Y is true, then H is better than G.

At the beginning of the algorithm we are not aware of any condition and therefore, the current combination of conditions is empty. Using the order of the input set as shown in Figure 1, the first comparison we need is (C,A). The DM provides the outcome without any condition. For the next

comparison (D,B), a condition will be stated. That means we interrupt the sorting. We now choose an (arbitrary) boolean value for the condition. Let us assume X is true.

We start again by comparing C to A. As we already asked the DM for the outcome, we now can reproduce it. The same holds for (D,B). We know the result assuming X is true. The next comparison is still unknown to us and therefore has to be provided by the DM. Finally, the last comparison in this round is (H,G) which again depends on a condition and therefore interrupts the sorting. Again, we start anew, now assuming some specific boolean values for both X and Y.

Figure 3 shows the first step of the algorithm for X and Y having the value true. After the initial building of the tree and calculation of the winner D, the 2<sup>nd</sup> and 3<sup>rd</sup> alternatives are selected (Figure 4).

Note that grey letters were eliminated in an earlier tournament run and that the grey arrows indicates which set of comparisons had to be made for which result.

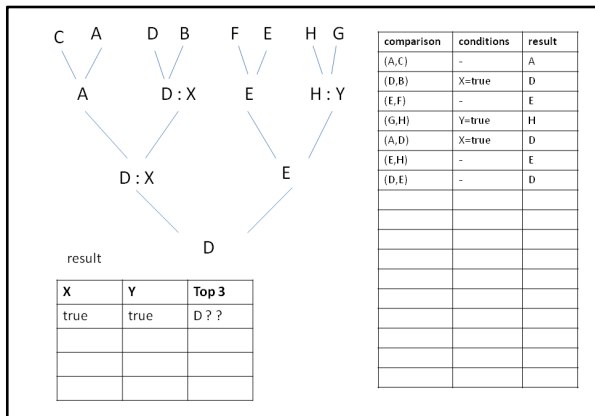


Figure 3. Initial tree

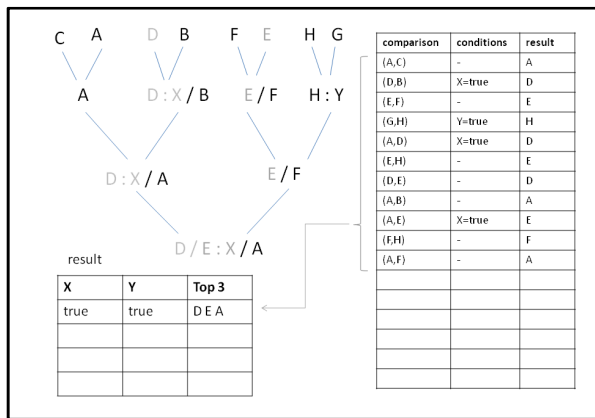


Figure 4. Initial tree after eliminating D and E

We now want to know the Top 3 for the remaining three combinations of boolean values of the two conditions we have. We start by using the (arbitrary) combination of X being false (we denote this case by  $\bar{X}$ ) and Y being true.

For the comparisons that do not depend on conditions, the outcome stays the same and we can

reproduce them using the table with the known comparison. For the comparisons which depend on conditions, the outcome changes. As we can see in Figure 5, this means that the winner of D – B is no longer D but B. Now, B has to be compared with A, which will yield A, and finally A has to be compared with E, where A also wins.

Note that the results of all three of these comparisons are already known, as we can see from the table of known comparisons in Figure 4. This means we obtain the winner for the case Y is true and X is false without requiring any additional comparison effort from the DM. The result of the selection of the 2<sup>nd</sup> and 3<sup>rd</sup> places is shown in Figure 5.

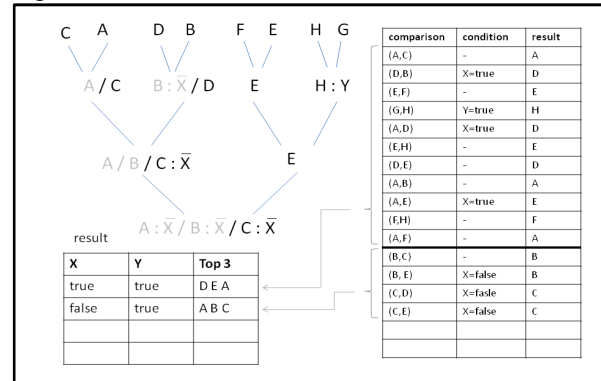
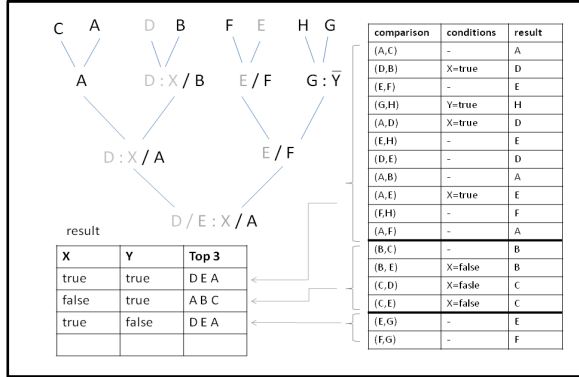


Figure 5. Result for X false and Y true

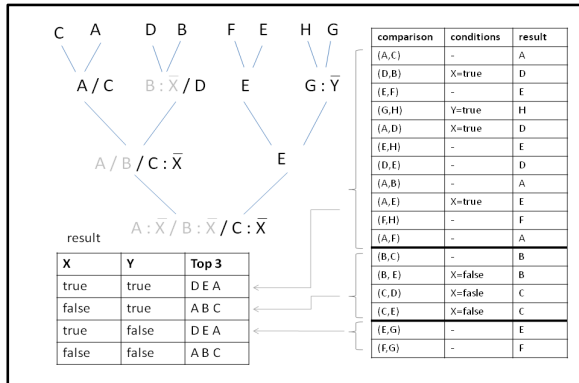
The next combination we investigate is the one where X is true and Y is false. We therefore start again with the initial tree and set the outcome of the comparisons affected by conditions according to the new combination. This means that the winner from G and H is no longer H but G. By comparing G against E we see that E still wins. That means the condition Y has no influence on the current winner. As E becomes the 2<sup>nd</sup> best alternative in this tournament, it has to be eliminated in order to find the 3<sup>rd</sup> best one. Next, G has to be compared to F and again loses. We therefore obtain the Top 3 for X true and Y false with only two additional comparisons. The result for this case is shown in Figure 6.



**Figure 6. Result for X true and Y false**

The last case we have to investigate is the *Top 3* for both X and Y being false. The result is shown in Figure 7. Note that this can be achieved without any further comparisons by the DM.

Our result consists of four ranked *Top 3* sets of alternatives which are contingent on the values of X and Y. Although 11 pairwise comparisons by the DM are needed to obtain the first result, only 6 more are sufficient to obtain all four variants.



**Figure 7. Result for X and Y both false**

Using this final result we now can determine which conditions are *relevant* and which are *irrelevant*. As can be seen in Figure 7, the final *Top 3* are only affected by the value of X. If X is true, the *Top 3* ranking is DEA and if X is false, the *Top 3* ranking is ABC. In contrast to this, no such statement can be made about the condition Y. That means that in this example, X is *relevant* and Y is *irrelevant* and therefore that Y does not have to be investigated in order to uniquely determine the final ranking.

The same distinction is possible for *single-impact* vs. *multiple-impact*. The condition Y only affects the relative position of G within the ranking – namely being before or after H in the ranking. That means Y has a *single-impact*. On the other hand, condition X changes the rank of more than one alternative (D and E) This means that X has a *multiple-impact*.

## 5. Discussion of the algorithm

In the discussion of our algorithm, there are two important questions to answer. The first question asks about the number of pairwise comparisons. As mentioned in Section 4.1, the algorithm is optimal regarding the number of comparisons in its basic form, meaning without any conditions. But how many additional comparisons do we have to invest in order to predict the impact of the conditions? This question will be answered in Section 5.1.

The second question concerns the savings potential by not having to investigate the irrelevant conditions. In other words, how many of the identified conditions will be irrelevant? This will be answered in 5.2.

### 5.1. Analysis of the number of pairwise comparisons

As already mentioned, the number of pairwise comparisons of the basic algorithm for the sorting of the whole set is proportional to  $N \log N$ , where  $N$  is the number of alternatives.

By only asking for a partial ranking, this complexity can be reduced to  $N - 1 + (k - 1) \log N$  for the *Top k* alternatives.  $N - 1$  comparisons are needed to build the initial tree and find the winner. Then for each of the remaining  $(k - 1)$  alternatives, the maximum number of necessary comparisons is given by the height of the tree,  $\log N$ . See Figure 1 for an example where the  $N=8$  alternatives built a tree of height  $3 = \log_2 8$ .

Now let us consider the impact of the conditions we use. The number of comparisons which have to be done when one condition  $c$  is reversed depends on the number of comparisons which are influenced by this condition. Let us denote this number by  $p(c)$ .

For each of those comparison reversals, at most  $k \log N$  new comparisons are necessary. This is the case when the change occurs in the first round of the tournament and the impact causes all following comparisons to change for each of the *Top k*.

In the worst case, this has to be done for each of the  $2^m$  true/false combinations of the conditions, where  $m$  is the number of conditions identified by the DM. Using these considerations, we can give the worst case scenario for additional pairwise comparisons caused by the conditions as

$$2^m \times p_{max} \times k \times \log N$$

where  $p_{max}$  is the maximum number of comparisons which are influenced by any condition stated in the process.

The above shown formulas give an impression about the overall number of comparisons that have to be made in order to complete the algorithm. That includes certain comparisons that are needed several times during the algorithm.

In our case, this will not be limiting factor as we rather measure the complexity using the number of comparisons which have to be provided by the DM. Examples have shown that this number is significantly smaller than the estimates above because comparisons can be reused.

In our example, the number of additional comparisons caused by the conditions would have been  $2^2 \times 2 \times 3 \times \log 8 = 72$ . Out of these 72 comparisons, only six had to be provided by the DM. The results of all the other comparisons were already known. This shows how using the already known comparisons can reduce the additional effort for the DM considerably.

## 5.2. Analysis of the number of irrelevant conditions

An indicator for the amount of investigation effort that can be saved is the probability for one condition being irrelevant.

For *single-impact* conditions, this probability can be calculated directly. As we are interested in the *Top k* out of  $N$  alternatives, the probability  $p$  for one random alternative being within this *Top k* can be described as

$$p = \frac{k}{N}$$

Using this, the probability for one alternative not being within the *Top k* is

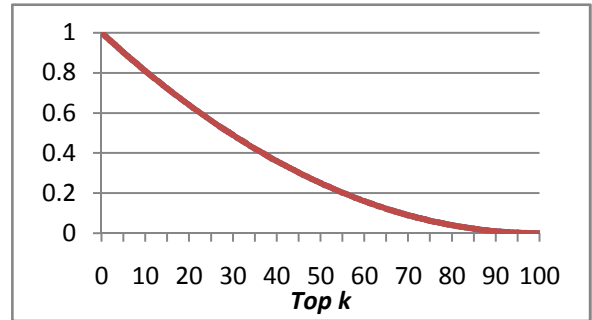
$$\bar{p} = 1 - p$$

As the definition states, a condition is irrelevant if the affected alternative is not within the *Top k*, regardless of whether the condition is true or false. As the rank for one alternative can be seen as random for both states of the condition, the probability  $p_i$  for one condition being irrelevant can be calculated using

$$p_i = (\bar{p})^2$$

Using this formula, the probability for one condition in 100 alternatives being irrelevant is plotted in Figure 8. As expected, for small  $k$ , the

probability for one condition being irrelevant is comparatively high.



**Figure 8. Probability for one single-impact condition in 100 alternatives being irrelevant dependent on the Top k**

For an example where the *Top 20* from 100 alternatives are needed and the DM stipulates 15 conditions, it is to be expected that about 10 of these conditions are irrelevant:

$$15 \times \left(1 - \frac{20}{100}\right)^2 = 9.6$$

The probability of being irrelevant for *multiple-impact* conditions is more complex. As they can influence an arbitrary number of alternatives, one cannot predict how this will change the ranking exactly. In the worst case, one condition could change the ranking completely, assigning a random new rank to all the alternatives. In this case, the probability of the *Top k* being the same for both states of the condition is very small.

In general, the probability for one condition being irrelevant naturally increases, the fewer *Top k* are to be selected and ranked.

## 6. Conclusions and outlook

Idea selection in the Front End of Innovation is characterized by a large degree of uncertainty owing the lack of information contained in the ideas. We presented an algorithm for selecting and prioritizing a set of winning ideas which takes uncertainty into account. The algorithm allows the DM to assign conditions to his or her judgments, which represent missing information that affects the outcome of the judgment. The algorithm generates all possible results and indicates which subset of the conditions must be clarified in order to produce a unique prioritized selection.

In the case that no conditions are stipulated, the algorithm has a proven optimal complexity in the number of judgments required. In the general case,

the number of variants grows exponentially in the number of conditions stated by the DM. However, a large proportion of these additional judgments can be avoided by storing and re-using.

Our algorithm provides several benefits to the Innovation Manager. Firstly, the uncertainty in the overall selection result is reduced. This in turn reduces the danger of selection errors, which can be very costly. In addition, the algorithm makes information that is relevant to the evaluation explicit, and thus available to the organization. Finally, the selection can still be completed in one sitting, even though it depends on as yet unknown information.

There is future work planned for our approach in several directions. One direction to be investigated is how large the system can get before it becomes infeasible because of the number of comparisons required by the conditions or the cognitive load the DM is faced with when comparing alternatives under several conditions.

One tool for this will be a Monte Carlo simulation study which will give us insight into the patterns generated by multiple conditions and expected values for the corresponding numbers of judgments needed.

Furthermore, the question of multiple criteria and DMs was not addressed, although these both hold in FEI practice. Our next step will be to extend our approach to cover these cases.

A planned software implementation of the algorithm for more than one DM will allow us to explore issues of interest from the perspectives of group support systems and collaboration engineering. In addition, we will be able to gather experience in the type and frequency of conditions supplied by the DMs.

## 7. References

- [1] C. Bana e Costa, J.-M. De Corte, and J.-C. Vansnick, "On the Mathematical Foundations of Macbeth". In [5].
- [2] R. G. Cooper, "Stage-Gate Systems: A new tool for managing new products", *Business Horizons*, 3(3), May-June, 44-54 (1990).
- [3] R.G. Cooper, S.J. Edgett, and E.J. Kleinschmidt, *Portfolio Management for New Products*, Perseus Books, New York, 2001.
- [4] J. Dyer, "Multiattribute Utility Theory", in [5].
- [5] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys*, Springer, Boston, MA, 2005.
- [6] J. Figueira, V. Mousseau, and B. Roy, "ELECTRE Methods", in [5].
- [7] J. Galbraith, *Designing complex organizations*, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, 1973, pp. 5
- [8] *Global IBM CEO Study*, IBM Global Business Services, Somers, NY, 2008.
- [9] C. Herstatt, B. Verworn, "The 'Fuzzy Front End' of Innovation", in *Bringing Technology and Innovation into the Boardroom: Strategy, Innovation and Competences for Business Value*, Palgrave Macmillan, 2004.
- [10] J. Paasi, T. Luoma, "Systematic Strategic Decision Support for Innovation Development", *European Conference on Management of Technology*, Nice, France, 17-19 September 2008.
- [11] A. Khurana, S. R. Rosenthal: "Towards holistic 'front ends' in new product development", *The Journal of Product Innovation Management* 15 (1998) 1: 57-74
- [12] D. Knuth, *The Art of Computer Programming*, Volume 3, Addison-Wesley, 2nd Edition, 1998.
- [13] P. A. Koen, G. M. Ajamian, S. Boyce, A. Clamen, E. Fisher, S. Fountoulakis, A. Johnson, P. Puri and R. Seibert "Fuzzy Front End: Effective Methods, Tools and Techniques", *The PDMA toolbook for new product development*, John Wiley & Sons, Inc., New York, 2002, pp. 5-35
- [14] J.-M. Martel, B. Matarazzo, "Other Outranking Approaches", in [5].
- [15] M. Öztürk, A. Tsoukiàs, P. Vincke, "Preference Modeling", in [5].
- [16] S. Petrovic, R. Petrovic, "A New Fuzzy Criteria Methodology for Ranking of Alternatives", *International Transactions in Operational Research*, Volume 9, Issue 1, pages 73–84, January 2002.
- [17] T. Saaty, "Relative Measurement and Its Generalization in Decision Making: Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors: The Analytic Hierarchy/Network Process", *Rev. R. Acad. Cien. Serie A. Mat.*, 102 (2), 251–318.
- [18] T. Saaty, *The Analytic Hierarchy Process*. McGraw-Hill, New York, 1980.
- [19] G. A. Stevens, J. Burley, "3,000 raw ideas = 1 commercial success!", *Research Technology Management*, 40(3), 12-16 (1997).
- [20] S. Vorbach, E. Perl, "Software Based Support for Innovation Processes", *Proceedings of I-KNOW '05*, Graz, Austria, June 29 - July 1, 2005.