Randomly Generated Outranking Digraphs and Dissimilarity Graphs

Raymond Bisdorff, Alexandru Olteanu

1 Introduction

An important factor in any research is to provide significant results for the proposed methods. Such a thing is only possible with a large and diverse set of instances for the problem being solved. With the case of Multicriteria Decision Aid and Multicriteria Clustering we need bipolar-valuated digraphs with precisely controled kernels (clusters). Randomly generating a digraph, for large instances will lead to very similar digraphs with 50% arc probability and large, weakly defined kernels (clusters).
This paper will present several approaches to generating outranking digraphs and dissimilarity graphs with fixed positive arc densities and kernels and clusters within a size range and with previously specified characteristics. Let us note that this paper will only concern itself with creating the digraphs (graphs) and not the performance values for each individual action (object). Such a relation between the performance table and the actual digraph will be analized at a later stage.

2 Multicriteria Decision Aid

We will start with Multicriteria Decision Aid and the generation of outranking digraphs.

2.1 General

Let X be the set of all actions and $\tilde{S}$ the outranking relation between the items of X. The corresponding digraph is $G(X, \tilde{S})$ [Bis06]. We define the valuation domain of G as the pair $(\tilde{S}_{min}, \tilde{S}_{max})$ with the property that $\forall x, y \in X \ : \ \tilde{S}_{min} \leq \tilde{S}(x,y) \leq \tilde{S}_{max}$. In our case we will use $(\tilde{S}_{min}, \tilde{S}_{max}) = (-100, +100)$.
The independence of a set $Y \subseteq X$ is defined as:

$$\Delta_Y^{ind} = \begin{cases} +1.0, & \text{when } Y = \{x\} \\ min_{x \in Y - \{y\}} min_{y \in Y}(-\tilde{S}(x,y)), & otherwise \end{cases} \tag{1}$$

The dominance and absorbance of the same set are:

$$\Delta_Y^{dom} = \begin{cases} +1.0, & \text{when } Y = X \\ min_{x \in Y - X} max_{y \in Y}(\tilde{S}(y,x)), & otherwise \end{cases} \tag{2}$$

$$\Delta_Y^{abs} = \begin{cases} +1.0, & \text{when } Y = X \\ min_{x \in Y - X} max_{y \in Y}(\tilde{S}(x,y)), & otherwise \end{cases} \tag{3}$$

When generating our digraphs we wish to embed a number of kernels inside it with predefined characteristics in terms of size, independence, absorbence and dominance. We also wish the positive arc count to hold a fixed value.
Let $ad \in (0,1)$ be called the arc density for $G(X,\tilde{S})$ iff $|\{(x,y)|\tilde{S}(x,y) \geq 0, \forall x,y \in X, x \neq y\}| = ad \cdot |X| \cdot (|X|-1)$.
We define $K = \{K_1, K_2, ..., K_k\}$ as the set of generated kernels of digraph $G(X,\tilde{S})$. A kernel is a set which is both independent and dominant or absorbent. The first are called outranking kernels and the second outranked kernels.
We call $KSz = (KSz_{min}, KSz_{max})$ the kernels size range iff $KSz_{min} \leq |K_i| \leq KSz_{max}$.
Let P be called the polarization classes with $P = \{(P_{1,1}, P_{1,2}), (P_{2,1}, P_{2,2}), ..., (P_{p,1}, P_{p,2})\}$ and $0 \leq P_{1,1} \leq P_{1,2} \leq P_{2,1} \leq ... \leq P_{p,2} \leq 100$.
In this digraph the outranking(outranked) kernels are set of nodes which are both outranking(outranked) and independent, or internally stable.
With that in mind we will present in the following sections the process of imposing the dominance, absorbance and independence characteristics on the sets chosen as kernels.

2.2 Imposing independence, dominance and absorbance

In order to set a certain value of independence of $K_i \in K$ to ind_thr we need all the $|K_i| \cdot (|K_i|-1)$ arcs inside K_i to be set to a negative value lower than $-ind\_thr$. This impacts the arc density negatively in that we are overall lowering the possible maximum arc density. If the desired result is to impose a weak independence, then only one arc needs to be set to a positive value equal to $-ind\_thr$.

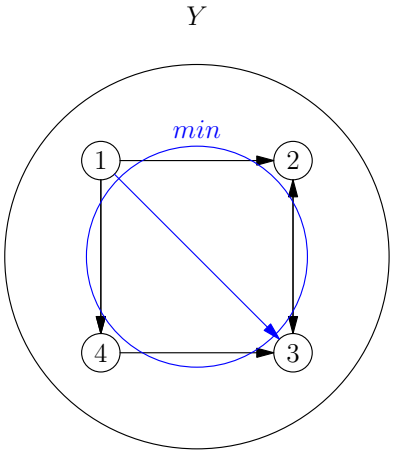

Figure 1: Independence of Y;

Let dom_thr be the desired value for the dominance of $K_i \in K$.
For each $x \in X - K_i$ the maximum value for $\tilde{S}(y,x)$, where $y \in K_i$, needs to be higher than dom_thr. The process of imposing the dominance of a choice needs $|X - K_i|$ positive arcs, as shown in the picture below. Similarly we conclude that the same number of arcs are needed for achieving a preset level of absorbance abs_thr.

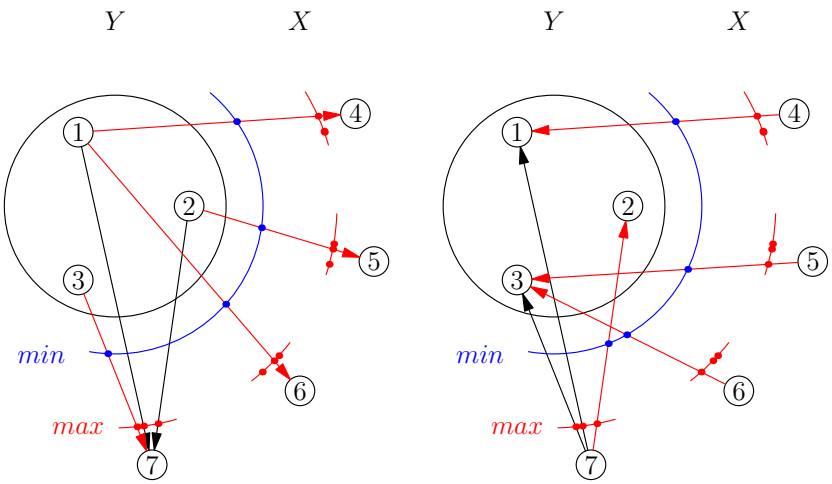

Figure 2: Dominance of Y; Absorbence of Y

2.3 Method 1

The first method uses a simplification, in that the set of all kernels will contain only independent subsets of X. This in no way means that all possible kernels are disjunctive, but that the ones we will construct are. This step was taken in order to ease the process of imposing the corresponding attributes to each kernel, and also to limit the interferences to other kernels when doing so. Method 2 will look at constructing overlaping kernels. Bear in mind that each problem instance is required to keep a fixed positive arcs density value.
The steps of the algorithm are:

1. Fix the parameters ad, KSz and P.
2. Start with a digraph filled with negative values (in order to start with a $ad = 0\%$);
3. Split X into independent sets with sizes concordant to KSz;
4. Cycle through each set and impose the internal stability and absorbance or dominance characteristics to values inside a chosen polarization class, while the arc density does not exceed ad;
5. Have at least one of the kernels absorbant and one dominant, or the some sets might fail to meet desired characteristics;
6. Fill in positive arcs, if needed in order to reach the desired arc density, among the outgoing arcs for dominant kernels and ingoing arcs for absorbant kernels;
7. Fill in positive arcs, if needed in order to reach the desired arc density, among the ingoing arcs for dominant kernels and outgoing arcs for absorbant kernels, but ignoring at least one node in order to keep the dominance(absorbance) to the desired level;
8. Fill in positive arcs among the rest of the available arcs, destroying the dominance, absorbance, independence relations one kernel at a time, but achieving the needed arc density.

Let's first demonstrate with a small example why we need at least one of the kernels to be absorbant. Assuming for instance that we have 6 nodes and we have partitioned them into 2 sets of 3. If we try to impose a strong dominance for $K1$ we can see the result on the left side of Fig. 3. The red lines

show the order in which the dominance is computed. For each node from outside $K1$ we take the maximum value of the arcs going from $K1$ to it, and from these maximums we take the minimum. The fact that the lines are red is used in order to show that the values for the maximums are high. We therefore impose a high dominance for $K1$. Now if we check the absorbence of $K2$ we see that it has also become strong. All the maximums among the arcs going inside $K2$ for each node outside of $K2$ are also high resulting in a high absorbence for $K2$. Trying to impose a high dominance for $K2$ will now lead to having the two kernels both strongly dominant and absorbent.

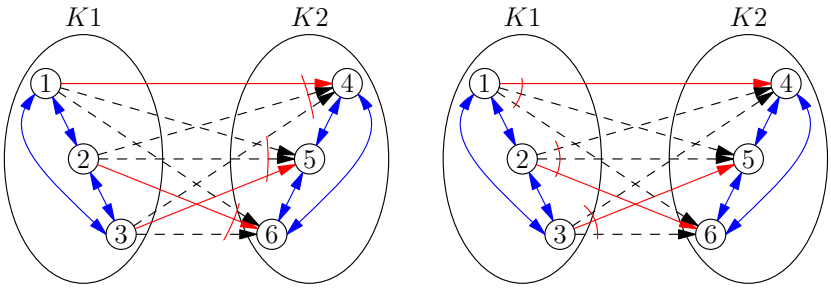

Figure 3: Example of 2 kernels of size 3;
Dominance of K1(left) and absorbance of K2(right);

Let's add another node to the graph and keep it outside of the 2 clusters. In this case, even if we impose a strong dominance for $K1$, we see that the absorbance of $K2$ will remain low because of the negative arcs going from the extra node to all of the ones in $K2$. Notice the blue line crossing the arcs going from the node outside the two kernels inside $K2$.

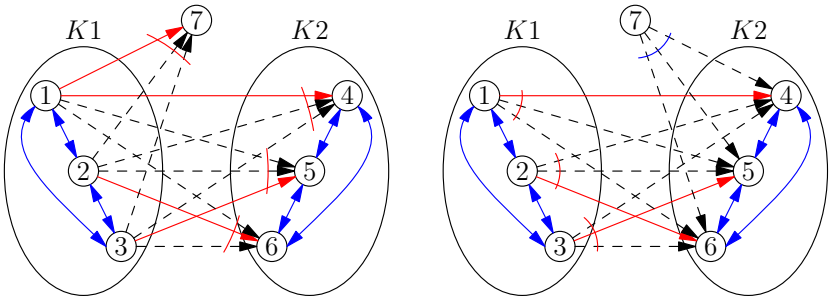

Figure 4: Example of 2 kernels of size 3 and an extra node outside them;
Dominance of K1(left) and absorbance of K2(right);

Algorithm 1

def GenerateKernels(**In:** N, arc_dens, KSz_range, K_type_seq ; **Out:** G)

$G \leftarrow GenerateNegativeDigraph(N)$
$K \leftarrow GenerateSets(G,\ KSz\_range)$
$k\_type\_seq\_ispolarized \leftarrow IsPolarized(K\_type\_seq)$
$type \leftarrow 0$
$k\_index \leftarrow 0$
$while\ \ k\_index < len(K)$:
    $for\ \ type\ in\ K\_type\_seq$:
        $for\ \ k\ in\ range(type\_repeats)$:
            $SetCharacteristics(K[k\_index],\ type)$
            $k\_index \leftarrow k\_index + 1$
            $if\ \ k\_index \geq len(K)\ or\ (k\_index == len(K) - 1\ and\ k\_type\_seq\_ispolarized)$

```
                break
            type ← type + 1
            if type >= type_no
                type ← 0
            if k_index ≥ len(K) or (k_index == len(K) − 1 and k_type_seq_ispolarized)
                break
    if GetArcDensity(G) < arc_dens :
        FillArcsWithoutDamagingRelations()
    if GetArcDensity(G) < arc_dens :
        FillArcsDamagingRelations(G)
    return G
```

3 Multicriteria Clustering

3.1 General

Let X be the set of all actions and $\tilde{D}$ the dissimilarity relation between the objects of X. We consider the dissimilarity graph $G(X, \tilde{D})$. We define the valuation domain of G as the pair $(\tilde{D}_{min}, \tilde{D}_{max})$ with the property that $\forall x, y \in X \ : \ \tilde{D}_{min} \leq \tilde{D}(x, y) \leq \tilde{D}_{max}$. In our case we will use $(\tilde{D}_{min}, \tilde{D}_{max}) = (-100, +100)$.
The internal stablility, or similarity of a set $Y \subseteq X$ is defined as:

$$\Delta_Y^{sim} = \begin{cases} +1.0, & \text{when } Y = \{x\} \\ min_{x \in Y - \{y\}} min_{y \in Y}(-\tilde{D}(x, y)), & otherwise \end{cases} \tag{4}$$

The external stability, or dissimilarity of the same set is:

$$\Delta_Y^{dis} = \begin{cases} +1.0, & \text{when } Y = X \\ min_{x \in Y - X} max_{y \in Y}(\tilde{D}(x, y)), & otherwise \end{cases} \tag{5}$$

When generating our graphs we wish to embed a number of clusters inside it with predefined characteristics in terms of size, internal and external stability. We also wish the positive arc count to hold a fixed value.
Let $ad \in (0, 1)$ be called the arc density for $G(X, \tilde{D})$ iff $|\{(x, y) | \tilde{D}(x, y) \geq 0, \forall x, y \in X, x \neq y\}| = ad \cdot |X| \cdot (|X| - 1)$.
We define $C = \{C_1, C_2, ..., C_k\}$ as the set of generated clusters of graph $G(X, \tilde{D})$. A cluster is a set which is both internally and externally stable.
We call $CSz = (CSz_{min}, CSz_{max})$ the clusters size range iff $CSz_{min} \leq |C_i| \leq CSz_{max}$.
Let P be called the polarization classes with $P = \{(P_{1,1}, P_{1,2}), (P_{2,1}, P_{2,2}), ..., (P_{p,1}, P_{p,2})\}$ and $0 \leq P_{1,1} \leq P_{1,2} \leq P_{2,1} \leq ... \leq P_{p,2} \leq 100$.
We will also use the notion of neighbourhood in our second algorithm.
For a node x in $\tilde{G}(X, \tilde{D})$ we call its open and closed similarity neighbourhoods:

$$N^-(x) = \{y \in X | \tilde{D}(x, y) < 0\} \quad , \quad N^-[x] = N^-(x) \cup \{x\} \tag{6}$$

Similarly we define the open and closed dissimilarity neighbourhoods:

$$N^+(x) = \{y \in X | \tilde{D}(x, y) > 0\} \quad , \quad N^+[x] = N^+(x) \cup \{x\} \tag{7}$$

Extending the neighbourhood concept to a set we have:

$$N^-[Y] = \bigcap_{x \in Y} N^-[x] \quad , \quad N^-(Y) = \bigcap_{x \in Y} N^-(x) \tag{8}$$

$$N^+[Y] = \bigcup_{x \in Y} N^+[x] \quad , \quad N^+(Y) = \bigcup_{x \in Y} N^+(x) \tag{9}$$

3.2 Imposing internal and external stability

In order to set a certain value of internal stability of $C_i \in C$ to sim_thr we need all the $|C_i| \cdot (|C_i| - 1)$ arcs inside C_i to be set to a negative value lower than $-sim\_thr$. This impacts the arc density negatively in that we are overall lowering the possible maximum arc density. If the desired result is to impose a weak similarity, then only one arc needs to be set to a positive value equal to $-sim\_thr$.

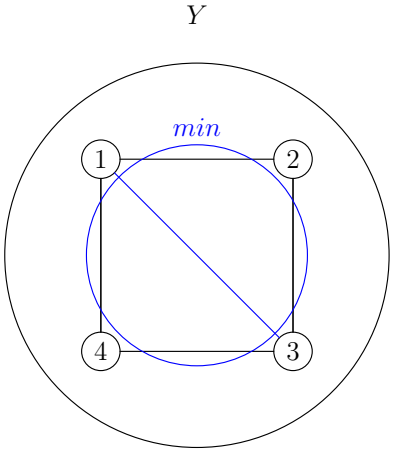

Figure 5: Internal stability of Y;

Let dis_thr be the desired value for the external stability of $C_i \in C$.
For each $x \in X - C_i$ the maximum value for $\tilde{D}(x, y)$, where $y \in C_i$, needs to be higher than dis_thr. The process of imposing the dissimilarity of a choice needs $|X - C_i|$ positive arcs, as shown in the picture below.

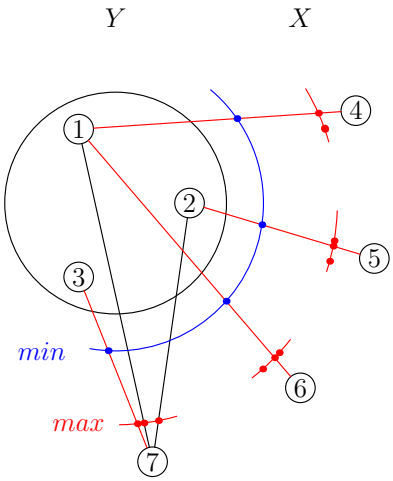

Figure 6: External stability of Y;

3.3 Method 1

The first method is almost exactly the same as the one used for the MCDA problem. It's even simpler because we use a nondirected graph. Because of this we don't need to concern ourselves

like in the previous case with having both dominant and absorbent kernels. Here we only have one measure between the set considered and the rest of the objects. As before, we use independent subsets to start building clusters and impose certain values for their internal and external stability measures. The steps of the algorithm are:

1. Fix the parameters ad, CSz and P.
2. Start with a graph filled with negative values (in order to start with a $ad = 0\%$);
3. Split X into independent sets with sizes concordant to CSz;
4. Cycle through each set and impose the internal and external stability characteristics to values inside a chosen polarization class, while the arc density does not exceed ad;
5. Fill in positive arcs among the rest of the available arcs, destroying the stability measures, but achieving the needed arc density.

Algorithm 1
def GenerateClusters1(**In:** N, arc_dens, C_range, $Sim\_polarization$, $Dis\_polarization$; **Out:** G)
$G \leftarrow GenerateNegativeGraph(N)$
$X \leftarrow G.objects$
while $GetArcDensity(G) < arc\_dens$:
$C \leftarrow GetCluster(X, C\_range)$
if C *is null*
break
$X \leftarrow X - C$
$SetCharacteristics(G, C, GetSimPol(Sim\_polarization), GetDisPol(Dis\_polarization))$
if $GetArcDensity(G) < arc\_dens$:
$FillArcsDamagingRelations(G)$
return G

For our benchmark set we generate instances of orders from 100 to 1000. The arc densities range from 10% to 90%. We use 4 cluster classes $CSz \in \{(1,3),(4,10),(11,100),(1,100)\}$ and 3 polarization classes $P = \{(0,33),(33,66),(66,100)\}$.

3.4 Method 2

Our second method adds a layer of complexity to the firat. We are no longer looking for independent set to become clusters, but also allow clusters to overlap. The process is the same as before, building independent sets and imposing the characteristics on them. These clusters are called main clusters. After each main cluster is built we then generate all possible overlapping clusters, restricted to the size range imposed at start. The steps of the algorithm are:

1. Fix the parameters ad, CSz and P.
2. Start with a graph filled with negative values (in order to start with a $ad = 0\%$);
3. Split X into independent sets with sizes concordant to CSz;
4. Cycle through each set and impose the internal and external stability characteristics to values inside a chosen polarization class, while the arc density does not exceed ad;
5. For each of the previous sets builditeratively all possible overlaping sets that can become clusters; (sets for which the internal and external stability measures can be still be imposed without damaging measures of overlapping clusters)

6. Fill in positive arcs among the rest of the available arcs, destroying the stability measures, but achieving the needed arc density.

Algorithm 2

```
def GenerateClusters2(In: N, arc_dens, CSz_range, Sim_polarization, Dis_polarization ; Out: G )
    G ← GenerateNegativeGraph(N)
    X ← G.objects
    Hist ← ∅
    while GetArcDensity(G) < arc_dens :
        MC ← GetMainCluster(G, X, CSz_range)
        if MC is null
            break
        SetCharacteristics(G, MC, GetSimPol(Sim_polarization), GetDisPol(Dis_polarization))
        X ← X − MC
        Hist ← Hist + MC
        while GetArcDensity(G) < arc_dens :
            OC ← GetOverlappingCluster(G, MC, CSz_range)
            if OC is null
                break
            SetCharacteristics(G, OC, GetSimPol(Sim_polarization), GetDisPol(Dis_polarization))
            Hist ← Hist + OC
    if GetArcDensity(G) < arc_dens :
        FillArcsDamagingRelations(G)
    return G

def GetMainCluster(In: G, X, CSz_range )
    found_cluster ← False
    MC ← ∅
    while notfound_cluster :
        CSz ← c ∈ CSz_range
        feasible_clusters ← EnumerateFeasibleClusters( G, CSz, X)
        if feasible_clusters = ∅ :
            CSz_range ← CSz_range − CSz
        else :
            found_cluster ← True
            MC ← cluster ∈ feasible_clusters
        if CSz_range = ∅ :
            break
    return MC

def EnumerateFeasibleClusters(In: G, CSz, X )
    partitions ← EnumeratePartitions( X, CSz)
    feasible_clusters ← ∅
    for x in partitions :
        if isClusterFeasible( G, x) :
            feasible_clusters ← feasible_clusters ∪ {x}
    return feasible_clusters

def isClusterFeasible(In: G, C )
    if N⁻[C] ⊉ C :
```

```
        return False
    else :
        feasible ← True
        for x ∈ X − C :
            tempC ← C
            for cluster in Hist :
                tempC ← tempC − cluster
            if tempC = ∅ :
                feasible ← False
                break
        return feasible

def GetOverlappingCluster(In: G, MC, CSz_range )
    found_cluster ← False
    X ← G.X
    OC ← ∅
    while notfound_cluster :
        feasible_clusters ← ∅
        CSz ← c ∈ CSz_range
        for CSz1 in (1, CSz − 1) :
            CSz2 ← CSz − CSz1
            C1 ← EnumeratePartitions( MC, CSz1)
            for c1 in C1 :
                C2 ← EnumeratePartitions( X − MC − N+(c1), CSz2)
                for c2 in C2 :
                    c ← c1 ∪ c2
                    if c ∉ Hist ∧ isClusterFeasible( G, c) :
                        clusters ← clusters ∪ {c}
        if clusters = ∅ :
            CSz_range ← CSz_range − {CSz}
        else
            found_cluster ← True
            OC ← cluster ∈ feasible_clusters
        if CSz_range = ∅ :
            break
    return MC

def SetCharacteristics(In: G, C, Sim_pol, Dis_pol )
    SetIntStablility( G, C, Sim_pol)
    SetExtStablility( G, C, Dis_pol)

def SetIntStability(In: G, C, Sim_pol )
    for x in C :
        for y in C :
            if x ≠ y :
                newval ← GetVal(Sim_pol)
                if G.relation[x][y] > newval :
                    G.D(x, y) ← newval

def SetExtStability(In: G, C, Sim_pol )
```

$first\_run \ = \ True$
$X \ \leftarrow \ G.objects$
$Y \ \leftarrow \ C$
$for \ \ x \ in \ X - Y :$
    $y \ \leftarrow \ k \in N^{+}(x) \cap C$
    $if \ \ y \ is \ null :$
        $y \ \leftarrow \ k \in N^{-}(x) \cap Y$
        $if \ \ y \ is \ null :$
            $y \ \leftarrow \ k \in N^{-}(x) \cap C$
    $newval \ \leftarrow \ GetVal(Dis\_pol)$
    $if \ \ G.D(x, y) \ < \ newval :$
        $G.D(x, y) \ \leftarrow \ newval$
    $if \ \ first\_run :$
        $Y \ \leftarrow \ Y - \{y\}$
        $if \ \ Y \ is \ \emptyset :$
            $Y \ \leftarrow \ C$
            $first\_run \ \leftarrow \ False$

References

[1] ...