**This separate file is only for referees, it is not included in the paper.**

We present here the algorithmic developments that we performed, and that are mentioned in footnote 17 of the submitted paper. We present in the order general considerations about algorithms, then we present tables summarizing the performance of the algorithms, and finally we provide the matlab programs used to generate the tables.

# 1 Algorithmic considerations

The problem faced by the principal being NP-hard under low intensity of interaction as shown in the paper, it can be useful to know whether algorithms generate good approximations of the solution. The problem is isomorphic to the densest $k$ subgraph problem for fixed group size, which guaranties that the so-called greedy algorithm performs rather well if the group size is given (no less than $1 - 1/e \sim 63$ percent of maximum efficiency for large societies - see details in Ballester et al (2009, Appendix A, Proposition 10) for instance. However, for the unconstrained problem with contracting cost and endogenous group size, the level of complexity is larger, and faster algorithms give a lower bound to the ratio of inefficiency of one half.

We performed two algorithms on Erdös-Renyi random networks under various linear contracting cost, one being the greedy algorithm, the other being a very simple algorithm that we call *centrality-based* algorithm.

The greedy algorithm works as follows. In step 1, it determines the best singleton; in step 2 it determines the best pair containing the best singleton and stops if the singleton performs better than the pair; in step 3 it determines the best triplet containing the best pair and stops if the pair performs better than the triplet; etc, until reaching the best group. This algorithm therefore converges in no more than $n!$ steps.

The centrality-based algorithm is based on the ranking of centralities. In step 1 the agent with largest centrality is selected and the algorithm computes the performance of this singleton. In step 2, the algorithm computes the performance of the pairs composed of the two agents with highest centralities, and stops if the singleton performs better than the pair; in step 3 the algorithm computes the performance of the triplet containing the three agents with highest centralities, and stops if the pair performs better than the triplet, etc. This algorithm converges in no more than $n$ steps. Hence, the centrality-based algorithm is faster than the greedy algorithm in general.

We implemented both algorithms with the following initial parameters.

We fixed $t = 1$ in all simulations.

First, we examined their performance in proportion of the optimal objective attained with the optimal target, which requires to compute the optimal target. We initiate $n = 16$. Indeed, there combinatorial concerns regarding the computation of the optimal target become very important for larger values of $n$, since we have to browse all possible groups. We selected low versus high intensities of interaction $\delta \in \{0.01, 0.06\}$. We tested $c \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ (for these parameter sets, larger costs induce trivial solution). In each scenario, we performed a simulation generating 1000 Erdö-Renyi random networks. For the uniform probability $p$ of link existence, we set $p = 0.5$.[1]

On each generated network, we identified the optimal group over all possible groups (with $n = 16$, the program searches through $2^{10} - 1 = 65535$ groups) and obtained its performance. Then, we ran the greedy algorithm, and we computed the relative error of approximation of the greedy algorithm in percentage of the optimal performance. These results are presented in Table 1. The numbers in the table are the average relative error of approximation (in percentage) over the 1000 networks in each scenario. Roughly speaking, the greedy algorithm performed very well for these values in general, the average relative error of approximation being less than 1 percent in any case.[2]

Next, we turned to the comparisons between the the greedy and the centrality-based algorithms, without reference to the optimal solution. This allowed us to run programs with a larger society, and we were able to set $n = 30$, restricting then to average performances over 100 random networks, and setting $c \in \{0.2, 0.4, 0.6, 0.9\}$. Here what is time consuming is the greedy algorithm under low cost values. Results are in Tables 2 for $n = 16$ and in Table 3 for $n = 30$. The numbers in each table represent the ratio of the average performance of the greedy algorithm over the average performance of the centrality-based algorithm, in percentage. For instance, the number 99.85 means that the greedy performance is 99.85 percent of that of the centrality-based algorithm. One striking message from these numerical computations is that *the centrality-based algorithm performs always slightly better than the greedy algorithm*, whereas it is far less time consuming.

We also explored the impact of clustering coefficient and degree assortativity on the performances of both algorithms. For both algorithms, we found that clustering increases performance, while degree assortativity decreases performance. Precisely, we computed the correlation coefficient be-

[1] We also tested alternative values of $p \in \{0.25, 0.75\}$, to test the impact of network density. They do not qualitatively affect results.

[2] The performance of the algorithms should be lower for larger network size.

tween each statistics and average algorithm performance. The statistics are given as follows. We define $g = \mathbf{1}^T\mathbf{G}\mathbf{1}$, $\mathbf{D} = \mathbf{G}\mathbf{1}$, $\mathbf{J}$ the $n$-square matrix of ones, and $\mathbf{H} = \frac{\mathbf{GJG}}{g}$:

Global clustering coefficient $C(\mathbf{G})$:

$$C(\mathbf{G}) = \frac{trace(\mathbf{G}^3)}{\mathbf{1}^T(\mathbf{G}^2 - \mathbf{G})\mathbf{1}}$$

Degree assortativity Pearson coefficient $DA(\mathbf{G})$[3]:

$$DA(\mathbf{G}) = g\frac{\mathbf{D}^T(\mathbf{G} - \mathbf{H})\mathbf{D}}{\mathbf{D}^T\mathbf{D} - \frac{g^2}{n}}$$

For $n = 30$, we computed the global clustering coefficient, the degree assortativity, the performance of the greedy algorithm and the performance of the centrality-based algorithm on 100 randomly generated networks, from which we deduced the following correlation coefficients for $c \in \{0.2, 0.3, 0.4\}$:

- For $c = 0.2$,
  - correlation coefficient clustering/greedy: 0.8977
  - correlation coefficient assortativity/greedy: $-0.2213$
  - correlation coefficient clustering/centrality-based: 0.9099
  - correlation coefficient assortativity/centrality-based: $-0.2273$
- For $c = 0.3$,
  - correlation coefficient clustering/greedy: 0.9149
  - correlation coefficient assortativity/greedy: $-0.2193$
  - correlation coefficient clustering/centrality-based: 0.9192
  - correlation coefficient assortativity/centrality-based: $-0.2257$
- For $c = 0.4$,
  - correlation coefficient clustering/greedy: 0.8941
  - correlation coefficient assortativity/greedy: $-0.1519$
  - correlation coefficient clustering/centrality-based: 0.8995
  - correlation coefficient assortativity/centrality-based: $-0.1478$

[3]The formulae given below has been proved in the paper entitled 'The value of network information: assortative mixing makes the difference' (Belhaj & Deroïan, mimeo 2018).

# 2 Tables

| $\delta$ | $\mathbf{c = 0.1}$ | $\mathbf{c = 0.2}$ | $\mathbf{c = 0.3}$ | $\mathbf{c = 0.4}$ | $\mathbf{c = 0.5}$ | $\mathbf{c = 0.6}$ |
|---|---|---|---|---|---|---|
| $\delta = \mathbf{0.01}$ | 100 | 99.85 | 99.94 | 99.97 | 100 | 100 |
| $\delta = \mathbf{0.06}$ | 99.97 | 99.68 | 99.78 | 99.65 | 99.84 | 99.82 |

Table 1: $n = 16$; Targeting through greedy algorithm: Average over 1000 random networks of the relative error of approximation in percentage of the optimal performance.

| $\delta$ | $\mathbf{c = 0.1}$ | $\mathbf{c = 0.2}$ | $\mathbf{c = 0.3}$ | $\mathbf{c = 0.4}$ | $\mathbf{c = 0.5}$ | $\mathbf{c = 0.6}$ | $\mathbf{c = 0.9}$ |
|---|---|---|---|---|---|---|---|
| $\delta = \mathbf{0.01}$ | 99.97 | 99.93 | 99.92 | 99.96 | 100 | 100 | 100 |
| $\delta = \mathbf{0.06}$ | 99.99 | 99.78 | 99.73 | 99.80 | 99.87 | 99.92 | 100 |

Table 2: $n = 16$; Targeting through greedy vs centrality-based algorithm: Ratio of the respective average performances of greedy over centrality-based algorithm over 1000 random networks in percentage.

| $\delta$ | $\mathbf{c = 0.2}$ | $\mathbf{c = 0.4}$ | $\mathbf{c = 0.6}$ | $\mathbf{c = 0.9}$ |
|---|---|---|---|---|
| $\delta = \mathbf{0.01}$ | 99.87 | 99.97 | 100 | 100 |
| $\delta = \mathbf{0.06}$ | 99.93 | 99.91 | 99.93 | 99.96 |

Table 3: $n = 30$; Targeting through greedy vs centrality-based algorithm: Ratio of the respective average performances of greedy over centrality-based algorithm over 100 random networks in percentage.

# 3 Matlab programs

He present in the order the program of determining the optimal solution, then we present the greedy algorithm, which computes the average performance over a given set of randomly generated networks, and finally we present the centrality-based algorithm (which also provides the average performance over randomly generated networks). Beforehand, we present two simple functions. Function *Perfo* computes the performance of a given group, and function *Subset* checks group inclusion (this latter function is needed for the greedy algorithm).

### 3.1 Function Perfo

function F = Perfo($n, S, B, O, t, M, cost, size$)

%%%%%%%%%%%%
% We create '$MS$': principal submatrix of **M** associated with group S %
%%%%%%%%%%%%

$MS1 = zeros(1, n)$;
for $io = 1 : size$
if $S(io) > 0$
$MS1 = [MS1; M(S(io), :)]$;
end
end
$MS1bis = MS1(2 : size + 1, :)$;
$MS1 = MS1bis$;
$MS2 = zeros(size, 1)$;
for $jo = 1 : size$
if $S(jo) > 0$
$MS2 = [MS2\ MS1(:, S(jo))]$;
end
end
$MS2bis = MS2(:, 2 : size + 1)$;
$MS2 = MS2bis$;
$MS = MS2$;

%%%%%%%%%%%%
% We create '$BS$': vector of centalities of agents in group S %
%%%%%%%%%%%%

$BS = 0$;
for $io = 1 : size$
if $S(io) > 0$
$BS = [BS; B(S(io), 1)]$;
end
end
$BS2 = BS(2 : size + 1, :)$;
$BS = BS2$;

F = $transpose(B) * O + sqrt(t) * sqrt(transpose(BS) * inv(MS) * BS) - cost * size$;

end

## 3.2 Function Subset

%%%%% The function Subset returns $-1$ if 'winner' is not included in 'S', and it returns 1 if 'winner' is included in 'S' %%%%%

function F $= Subset(winner, S, size)$
$F = -1;$
$ko = 1;$
while $(ko < size)$ && $(winner(ko) == S(ko))$
$ko = ko + 1;$
end
if $(ko == size)$
F $= 1;$
end
end

## 3.3 Optimal solution on a given network

clear all

%%%%% Parameters %%%%%

$n = 10;$
$t = 1;$
$cost = 0.5;$
$p = 0.5;$ % this is the probability of link formation in Erdos Renyi process
%
$delta = 0.01;$

%%%%%%%%%%%

$J = ones(n, n);$
$id = eye(n, n);$
$O = ones(n, 1);$
$N = O;$
for $i = 1 : n$
$N(i) = i;$
end

%%%%%%%%%%%
% We generate network **G** %
%%%%%%%%%%%

$Grand = zeros(n, n);$
for $i = 1 : n$
for $j = i + 1 : n$
$random_link = rand;$
if $random_link < p$
$Grand(i, j) = 1;$
$Grand(j, i) = 1;$
end
end
end
$G = Grand;$

%%%%%%%%%%%

$D = G * O;$
$M = inv(id - delta * G);$
if $min(M) < 0$
disp('Min M negatif')
return
end
$B = M * O;$ % Bonacich centrality vector %
Perfmax = 0; % 'Perfmax' is the optimal performance %
for $size = 1 : n$ %'size' is the group cardinality %

%%%%%%%%%%%
% We generate the set of groups of cardinality 'size' %
% We use the matlab function 'nchoosek' %
%%%%%%%%%%%

$C = nchoosek(N, size);$
$c = factorial(n)/factorial(size)/factorial(n - size);$
for $index = 1 : c$
$S = C(index, :);$ % 'S' is the profile of agents in a given group, sorted by increasing label %

%%%%%%%%%%%
% 'Perf' computes the performance of group S by function 'Perfo' %

%%%%%%%%%%%

Perf = Perfo($n, S, B, O, t, M, cost, size$);
if Perf > Perfmax
Perfmax = Perf;
end
end
end
Perfmax
disp('end')

## 3.4 Greedy algorithm

clear all

%%%%% Parameters %%%%%

$n = 10$;
$t = 1$;
$cost = 0.5$;
$nbg = 100$; % this is the number of random networks generated %
$p = 0.5$; % this is the probability of link formation in Erdos Renyi process %
$delta = 0.01$;

%%%%%%%%%%%

$J = ones(n, n)$;
$id = eye(n, n)$;
$O = ones(n, 1)$;
$N = O$;
for $i = 1 : n$
$N(i) = i$;
end

AvPerfGreedy = 0;
for nbgraphs = $1 : nbg$
nbgraphs
%%%%%%%%%%%
% We generate network **G** %

```
%%%%%%%%%%%

Grand = zeros(n, n);
for  i = 1 : n
for  j = i + 1 : n
random_link = rand;
if  random_link < p
Grand(i, j) = 1;
Grand(j, i) = 1;
end
end
end
G = Grand;

%%%%%%%%%%%

D = G * O;
M = inv(id − delta * G);
if  min(M) < 0
disp('Min M negatif')
return
end
B = M * O; % Bonacich centrality vector %

PerfGreedy = 0;
PerfGreedyPrecSize = 0;
PerfGreedyCurrentSize = 0;
size = 1;
Currentwinner= 1;
winnerPrec = 1;
C = nchoosek(N, 1);
c = factorial(n)/factorial(1)/factorial(n − 1);

for  index = 1 : c
S = C(index, :);
Perf = Perfo(n, S, B, O, t, M, cost, size);
if  Perf > PerfGreedyCurrentSize
PerfGreedyCurrentSize = Perf;
Currentwinner = S;
end
```

end

PerfGreedy = PerfGreedyCurrentSize;
PerfGreedyPrecSize = PerfGreedyCurrentSize;
winner = Currentwinner;
winnerPrec = Currentwinner;

for $size = 2 : n$
PerfGreedyCurrentSize = 0;
$C = nchoosek(N, size)$;
$c = factorial(n)/factorial(size)/factorial(n - size)$;

for $index = 1 : c$
S = $C(index, :)$;
if Subset$(winner, S, size) > 0$ % selects group S if winner is included in S %
Perf = Perfo$(n, S, B, O, t, M, cost, size)$; % 'Perf' computes the performance of group S by function 'Perfo' %
end
if Perf > PerfGreedyCurrentSize
PerfGreedyCurrentSize = Perf;
Currentwinner = S;
end
end

if PerfGreedyCurrentSize >= PerfGreedyPrecSize
PerfGreedyPrecSize = PerfGreedyCurrentSize;
PerfGreedy = PerfGreedyCurrentSize;
winner = Currentwinner;
winnerPrec = Currentwinner;
end
if PerfGreedyCurrentSize < PerfGreedyPrecSize
PerfGreedy = PerfGreedyPrecSize;
winner = winnerPrec;
break
end
end % end of the loop 'for $size = 2 : n$' %

PerfGreedy;
AvPerfGreedy = AvPerfGreedy + PerfGreedy;

end % end of the loop 'for nbgraphs = 1 : $nbg$' %

AvPerfGreedy = AvPerfGreedy /$nbg$
disp('end')

## 3.5 Centrality-based algorithm

clear all

%%%%% Parameters %%%%%

$n = 10$;
$t = 1$;
$cost = 0.5$;
$nbg = 100$; % this is the number of random networks generated %
$p = 0.5$; % this is the probability of link formation in Erdos Renyi process
%
$delta = 0.01$;

%%%%%%%%%%%

$J = ones(n, n)$;
$id = eye(n, n)$;
$O = ones(n, 1)$;
$N = O$;
for $i = 1 : n$
$N(i) = i$;
end

AvPerfCentrality = 0;
for nbgraphs = 1 : $nbg$
nbgraphs
%%%%%%%%%%%
% We generate network **G** %
%%%%%%%%%%%

$Grand = zeros(n, n)$;
for $i = 1 : n$
for $j = i + 1 : n$
$random_link = rand$;
if $random_link < p$

$Grand(i,j) = 1;$
$Grand(j,i) = 1;$
end
end
end
$G = Grand;$

%%%%%%%%%%

$D = G * O;$
$M = inv(id - delta * G);$
if $min(M) < 0$
disp('Min M negatif')
return
end
$B = M * O;$ % Bonacich centrality vector %

Bon = $[BN]$;
BonSorted = $sortrows(\text{Bon}, 1, 'descend')$;
BonSorted = BonSorted$(:, 2)$;
PerfCentrality = 0;
PerfCentralityPrec = 0;

for $size = 1 : n$
S = BonSorted$(1 : size)$;
Perf = $Perfo(n, S, B, O, t, M, cost, size)$; % 'Perf' computes the performance of group S by function 'Perfo' %
if Perf >= PerfCentralityPrec
PerfCentrality = Perf;
PerfCentralityPrec = PerfCentrality;
end
if Perf < PerfCentralityPrec
PerfCentrality = PerfCentralityPrec;
break
end
end % end of the loop 'for $size = 1 : n$' %

PerfCentrality;
AvPerfCentrality = AvPerfCentrality + PerfCentrality;
end % end of the loop 'for nbgraphs $= 1 : nbg$' %

AvPerfCentrality = AvPerfCentrality $/nbg$
disp('end')