Supplementary material of manuscript:

"Exact Permutation and Bootstrap Distribution of Generalized Pairwise Comparison statistics"

R and SAS Code

R-code Permutation

```
############################################################
# William N Anderson
# R code for the permutation distribution, based on real skew matrices
# Compute means and variances for trial arm wins,
# based on the full permutation distribution,
# using an already computed win matrix and trial arms.
# The algorithm is O(N^2) in time and space.
#
# February 2023, based on earlier code for the 0-1 case
# This file is self contained
#####################################################################


############ utility functions. For package use would put into a separate file
checkmatrixandarms <- function(winmatrix, trialarms){
  if (length(trialarms) != nrow(winmatrix)) stop("Incompatible dimensions in win matrix and trial arms")
```

```
  if (any(winmatrix + t(winmatrix)  != 0)) stop("Win matrix not skew")
  if (!all(trialarms %in% c(0, 1))) stop("Invalid trial arms")
}

# Evaluate wins for a U matrix and trial arms
wincomputations_real <- function(winmatrix, trialarms){
  # test rows is the vector of rows corresponding to the treatment arm
  testrows <- which(trialarms == 1)
  comparisonmatrix <- winmatrix[testrows, -testrows]
  # K and L correspond to the definitions in the Dong paper -- we might want those someday
  K <- pmax(comparisonmatrix, 0) # Test wins are the positive matrix entries; test losses and ties are 0
  L <- pmax(-comparisonmatrix, 0) # Control wins are the positive matrix entries; control losses and
ties are 0
  winssum <- c(sum(K), sum(L)); names(winssum) <- c("Test Win Sum", "Control Win Sum")
  # might want K and L some day, but not today
  # list(`Test win matrix` = K, `Control win matrix` = L, winssum = winssum)
  winssum
}
##############end utility functions ########################

############ The practical algorithm ###################
# Count Cases 1-5 at a specific vertex
onevertex_PR <- function(ins, insq, outs, outsq){
  # function to compute needed terms for a specific vertex,
  # using already computed invalues and outvalues
  # subscripts correspond to case numbering in manuscript
  # note that we do not count case 6 directly
  c(insq, ins^2 - insq, outs^2-outsq,
    ins*outs, ins*outs)
}
```

```
winsmeanandvariance_real_P <- function(winmatrix, trialarms){
  # function to compute the expected number of treatment and control wins, and their variance
  # the actual trial assignments are not relevant for this function, just the count
  #
  # winmatrix is the skew matrix of wins, perhaps from a hierarchical evaluation.
  #
  checkmatrixandarms(winmatrix, trialarms) # will stop if data are bad
  m <- sum(trialarms == 1); n <- sum(trialarms == 0); N <- m + n

  # compute various vertex values
  # computations are O(N^2), because the matrix multiplications are by component
  # sum of values of edges pointing in to the vertex -- real analog of indegree
  invalues <- colSums(winmatrix*as.integer(winmatrix > 0))
  # sum of squares of values of edges pointing in to the vertex
  insqvalues <- colSums(winmatrix^2*as.integer(winmatrix > 0))
  # sum of values of edges pointing out of the vertex -- real analog of outdegree
  outvalues <- rowSums(winmatrix*as.integer(winmatrix > 0))
  # sum of squares of values of edges pointing out of the vertex
  outsqvalues <- rowSums(winmatrix^2*as.integer(winmatrix > 0))
  edgesum <- sum(invalues) # could also use outvalues

  # count the cases at each vertex v -- case definitions in manuscript
  # the case matrix has 5 rows (1 for each case), and N columns
  # case 6 is not represented in this matrix, because case 6 situations do not belong to a specific
vertex
  # the matrix is perhaps useful in understanding the algorithm, but only the rowSums are actually
used
  casematrix <- mapply(onevertex_PR, invalues, insqvalues, outvalues, outsqvalues)
  # add the cases from all the vertices
  casecounts <- rowSums(casematrix)
  casecounts[6] <- edgesum^2 - sum(casecounts) # now we have case 6
```

```
 expected =  rep(edgesum*m*n/(N*(N - 1)), 2)
 names(expected) <- c("Test Win Sum", "Control Win Sum")
 TTerms <- c(m*n/(N*(N - 1)), m*n*(m - 1)/(N*(N - 1)*(N - 2)),
          m*n*(n - 1)/(N*(N - 1)*(N - 2)), 0, 0,
          m*n*(m - 1)*(n - 1)/(N*(N - 1)*(N - 2)*(N - 3)))
 CTerms <- c(m*n/(N*(N - 1)), m*n*(n - 1)/(N*(N - 1)*(N - 2)),
          m*n*(m - 1)/(N*(N - 1)*(N - 2)), 0, 0,
          m*n*(m - 1)*(n - 1)/(N*(N - 1)*(N - 2)*(N - 3)))
 TCTerms <- c(0, 0, 0, m*n*(n - 1)/(N*(N - 1)*(N - 2)),
           m*n*(m - 1)/(N*(N - 1)*(N - 2)),
           m*n*(m - 1)*(n - 1)/(N*(N - 1)*(N - 2)*(N - 3)))
 CTTerms <- c(0, 0, 0, m*n*(m - 1)/(N*(N - 1)*(N - 2)),
           m*n*(n - 1)/(N*(N - 1)*(N - 2)),
           m*n*(m - 1)*(n - 1)/(N*(N - 1)*(N - 2)*(N - 3)))
 expectedforvariance <- c(sum(casecounts*TTerms), sum(casecounts*TCTerms),
                          sum(casecounts*CTTerms), sum(casecounts*CTerms))
 expectedforvariance <- matrix(expectedforvariance, nrow = 2)
 variance <- expectedforvariance - expected %*% t(expected)
 rownames(variance) <- c("Test Win Sum", "Control Win Sum")
 colnames(variance) <- c("Test Win Sum", "Control Win Sum")
 # The observed wins are not relevant for this algorithm
 # But would be nice to have in the output
 Observed <- wincomputations_real(winmatrix, trialarms)
 list(Observed = Observed, Expected = expected, Variance = variance)
}


manuscriptexample_P <- function(){
manuscriptexample <-
          matrix(c( 0, -2,  0, 0,  1,
            2,  0,  3, 0, -5,
```

```
            0, -3,  0, 4,  0,
            0,  0, -4, 0, -1,
           -1,  5,  0, 1,  0),
          nrow = 5, byrow = TRUE)
manuscriptarms <- c(1, 1, 0, 0, 0)
winsmeanandvariance_real_P(manuscriptexample, manuscriptarms)

}
```

R-code Two-sample bootstrap

```
############################################################
# William N Anderson
# R code for the two-sample bootstrap distribution, based on real skew matrices
# Compute means and variances for trial arm wins,
# based on the full bootstrap distribution,
# using an already computed win matrix and trial arms.
# The algorithm is O(N^2) in time and space.
#
# The algorithm computes the mean win sums and variance that would be observed
# if all bootstrap samples were generated, without the necessity of actually
# computing all the bootstrap samples
#
# February 2023, based on earlier code for the 0-1 case
# This file is self contained
#####################################################################



############ utility functions. For package use would put into a separate file
checkmatrixandarms <- function(winmatrix, trialarms){
  if (length(trialarms) != nrow(winmatrix)) stop("Incompatible dimensions in win matrix and trial
arms")
  if (any(winmatrix + t(winmatrix)  != 0)) stop("Win matrix not skew")
  if (!all(trialarms %in% c(0, 1))) stop("Invalid trial arms")
}


# Evaluate wins for a U matrix and trial arms
wincomputations_real <- function(winmatrix, trialarms){
  # test rows is the vector of rows corresponding to the treatment arm
  testrows <- which(trialarms == 1)
```

```
  comparisonmatrix <- winmatrix[testrows, -testrows]
  # K and L correspond to the definitions in the Dong paper -- we might want those someday
  K <- pmax(comparisonmatrix, 0) # Test wins are the positive matrix entries; test losses and ties are 0
  L <- pmax(-comparisonmatrix, 0) # Control wins are the positive matrix entries; control losses and ties are 0
  winssum <- c(sum(K), sum(L)); names(winssum) <- c("Test Win Sum", "Control Win Sum")
  # might want K and L some day, but not today
  # list(`Test win matrix` = K, `Control win matrix` = L, winssum = winssum)
  winssum
}
##############end utility functions ########################



# Count Cases 1-10 at a specific vertex
# The subscripts on the returned vector are the case numbers
onevertex_B2R <- function(D, starT, starsqT, starC, starsqC){
  # function to compute needed terms for a specific vertex.
  # subscripts correspond to case numbering in manuscript.
  # note that we do not count cases 11-14 directly,
  # since they do not correspond to specific vertices
  c(D*starsqT, (1 - D)* starsqC, D*(starT^2 - starsqT), (1 - D)*(starT^2 - starsqT), D*(starC^2 - starsqC), # cases 1-5
    (1 - D)*(starC^2 - starsqC), D*starT*starC, (1 - D)*starT*starC, D*starT*starC, (1 - D)*starT*starC ) # cases 6-10
}



winsmeanandvariance_real_B2 <- function(winmatrix, trialarms){
  checkmatrixandarms(winmatrix, trialarms) # will stop if data are bad
  N <- length(trialarms) # must be consistent with winmatrix
  treatmentarms <- trialarms == 1; m <- sum(treatmentarms)
  controlarms <- trialarms == 0; n <- sum(controlarms)
```

```
# compute vectors giving the numbers of treatment and control wins for each vertex
treatmentwinmatrix <- outer(treatmentarms, controlarms)*winmatrix*as.integer(winmatrix > 0)
treatmentwinmatrix <- treatmentwinmatrix + t(treatmentwinmatrix)
controlwinmatrix <- outer(controlarms, treatmentarms)*winmatrix*as.integer(winmatrix > 0)
controlwinmatrix <- controlwinmatrix + t(controlwinmatrix)
# using manuscript notation as much as possible
`#T_v` <- rowSums(treatmentwinmatrix) # sum of treatment wins adjacent to vertex
`#Tsq_v` <- rowSums(treatmentwinmatrix^2) # sum of squares of treatment wins adjacent to vertex
`#C_v` <- rowSums(controlwinmatrix)   # sum of control wins adjacent to vertex
`#Csq_v` <- rowSums(controlwinmatrix^2) # sum of squares of treatment wins adjacent to vertex
Deq1 <- trialarms; Deq0  = 1 - trialarms # manuscript notation
W_T <- `#T_v` %*% Deq1; W_C <- `#C_v` %*% Deq0 # Test and Control win sums
expected = c(W_T, W_C); names(expected) <- c("Test Win Sum", "Control Win Sum")

casematrix <- mapply(onevertex_B2R, trialarms,`#T_v`, `#Tsq_v`, `#C_v`, `#Csq_v`)
casecounts <- rowSums(casematrix)
casecounts[11] <- W_T^2 - (casecounts[1] + casecounts[3] + casecounts[4])
casecounts[12] <- W_C^2 - (casecounts[2] + casecounts[5] + casecounts[6])
casecounts[13] <- W_T*W_C - (casecounts[7] + casecounts[8])
casecounts[14] <- W_T*W_C - (casecounts[9] + casecounts[10])
factors <- c((2*m - 1)*(2*n - 1), (2*m - 1)*(2*n - 1), (2*m - 1)*(n - 1), # cases 1:3
        (m - 1)*(2*n - 1), (2*m - 1)*(n - 1), (m - 1)*(2*n - 1), # cases 4:6
        (2*m - 1)*(n - 1), (m - 1)*(2*n - 1), (2*m - 1)*(n - 1),  # cases 7:9
        (m - 1)*(2*n - 1),(m - 1)*(n - 1), (m - 1)*(n - 1), # cases 10:12
        (m - 1)*(n - 1), (m - 1)*(n - 1)) # cases 13:14
factors <- factors/(m*n)
# indicators for cases contributing to variance terms
caseindicatorTT <- c(1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0) # Cases 1, 3, 4, 11
caseindicatorTC <- c(0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0) # Cases 7, 8, 13
caseindicatorCT <- c(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1) # Cases 9, 10, 14
caseindicatorCC <- c(0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0) # Cases 2, 5, 6, 12
```

```
  expTT <- sum(casecounts*factors*caseindicatorTT)
  expTC <- sum(casecounts*factors*caseindicatorTC)
  expCT <- sum(casecounts*factors*caseindicatorCT)
  expCC <- sum(casecounts*factors*caseindicatorCC)
  exp <- matrix(c(expTT, expTC, expCT, expCC), nrow = 2)
  variance <- exp - expected%*%t(expected)
  rownames(variance) <- c("Test Win Sum", "Control Win Sum")
  colnames(variance) <- c("Test Win Sum", "Control Win Sum")
  # The observed wins are not equal to the expected
  # But are computed separately for validation purposes
  Observed <- wincomputations_real(winmatrix, trialarms)
  list(Observed = Observed, Expected = expected, Variance = variance)
}

manuscriptexample_B <- function(){
manuscriptexample <-
            matrix(c( 0, -2,  0, 0,  1,
              2,  0,  3, 0, -5,
              0, -3,  0, 4,  0,
              0,  0, -4, 0, -1,
             -1,  5,  0, 1,  0),
            nrow = 5, byrow = TRUE)
manuscriptarms <- c(1, 1, 0, 0, 0)
winsmeanandvariance_real_B2(manuscriptexample, manuscriptarms)

}
```

SAS-code Permutation

```
/*This code creates an example U matrix*/
Data U;
   input U1 U2 U3 U4 U5;
   datalines;
0 -2 0 0 1
2 0 3 0 -5
0 -3 0 4 0
0 0 -4 0 -1
-1 5 0 1 0
;

/*Assign values to be used later*/
%let nT=2; %let nC=3; %let N=%eval(&nT+&nC);

/*Calculate the indegree, outdegree and counts for ID, 2, 3, 4 and 5*/
Data example;
set U;
array U U1-U&N;
in=0;
ins=0;
out=0;
outs=0;
do i=1 to &N;
      if U[i]<0 then do; in= in-U[i]; ins=ins+U[i]**2; end;
    if U[i]>0 then do; out = out+U[i];outs=outs+U[i]**2; end;
end;
drop i;
in2_ins=in**2-ins;
out2_outs=out**2-outs;
inout=in*out;
run;

/*Sum over all rows */
proc means data=example (keep=out in outs ins in2_ins out2_outs inout)
noprint;
  output out=variance sum=;
run;

/*Calculate cases and the variance for the number of wins for the
treatment arm (V_WT), the variance for the number of wins for the control
arm
(V_WC) and their covariance (V_WTWC). Finally calculate the expectation
(Exp_WD) and variance for the win
difference (Var_WD)*/
data variance;
set variance;
factor1=&nT*&nC/(&N*(&N-1));
factor2=&nT*&nC*(&nT - 1)/(&N*(&N - 1)*(&N - 2));
factor3=&nT*&nC*(&nC - 1)/(&N*(&N - 1)*(&N - 2));
factor4=&nT*&nC*(&nT - 1)*(&nC - 1)/(&N*(&N - 1)*(&N - 2)*(&N - 3));
WT=out*factor1;
WC=out*factor1;
V_WT= factor1*ins + factor2*in2_ins + factor3*out2_outs + factor4*(in**2-
(ins+in2_ins+out2_outs+2*inout)) - (factor1*out)**2;
V_WC= factor1*outs + factor2*out2_outs + factor3*in2_ins + factor4*(out**2-
(outs+out2_outs+in2_ins+2*inout)) - (factor1*out)**2;
V_WTWC= factor1*inout + factor4*(in**2-(ins+in2_ins+out2_outs+2*inout)) -
(factor1*out)**2;
Exp_WD=WT-WC;
```

```
Var_WD=V_WT+V_WC-2*V_WTWC;
run;

proc print data=variance (keep=WT Exp_WD Var_WD V_WT V_WC V_WTWC); run;
```

SAS-code Two-sample bootstrap

```
/*This code creates an example U matrix*/
Data U;
   input U1 U2 U3;
   datalines;
0 0 1
3 0 -5
;

/*Define the sample size per treatment arm (needs to be compatible with the
U matrix*/
%let nT=2; %let nC=3;

/*create treatment win matrix (UT) and control win matrix (UC) from U for
both row and column sums*/
Data UT_Dv1;
set U;
array Ut U1-U&nC;
do i=1 to &nC;
      if Ut[i]<0 then Ut[i]=0;
end;
drop i;
run;

Data UC_Dv1;
set U;
array Uc U1-U&nC;
do i=1 to &nC;
      if Uc[i]>0 then Uc[i]=0; else Uc[i]=-Uc[i];
end;
drop i;
run;

proc transpose data = U prefix=U out= U_Dv0 (drop=_NAME_);
run;

Data UT_Dv0;
set U_Dv0;
array Ut U1-U&nT;
do i=1 to &nT;
      if Ut[i]<0 then Ut[i]=0;
end;
drop i;
run;

Data UC_Dv0;
set U_Dv0;
array Uc U1-U&nT;
do i=1 to &nT;
      if Uc[i]>0 then Uc[i]=0; else Uc[i]=-Uc[i];
end;
drop i;
run;

/*sum the wins and squares per row and column for both win matrices*/
Data ST_Dv1;
set UT_Dv1;
array Ut U1-U&nC;
T_Dv1=0;
Ts_Dv1=0;
```

```
do i=1 to &nC;
      if Ut[i]>0 then do;
            T_Dv1= T_Dv1+Ut[i];
        Ts_Dv1 = Ts_Dv1+Ut[i]**2;
      end;
end;
drop i;
run;

Data SC_Dv1;
set UC_Dv1;
array Uc U1-U&nC;
C_Dv1=0;
Cs_Dv1=0;
do i=1 to &nC;
      if Uc[i]>0 then do;
        C_Dv1= C_Dv1+Uc[i];
        Cs_Dv1 = Cs_Dv1+Uc[i]**2;
      end;
end;
drop i;
run;

Data ST_Dv0;
set UT_Dv0;
array Ut U1-U&nT;
T_Dv0=0;
Ts_Dv0=0;
do i=1 to &nT;
      if Ut[i]>0 then do;
            T_Dv0= T_Dv0+Ut[i];
        Ts_Dv0 = Ts_Dv0+Ut[i]**2;
      end;
end;
drop i;
run;

Data SC_Dv0;
set UC_Dv0;
array Uc U1-U&nT;
C_Dv0=0;
Cs_Dv0=0;
do i=1 to &nT;
      if Uc[i]>0 then do;
        C_Dv0= C_Dv0+Uc[i];
        Cs_Dv0 = Cs_Dv0+Uc[i]**2;
      end;
end;
drop i;
run;

/*merge the Dv1 and Dv0 sums and calculate cases */
data S_Dv1;
merge ST_Dv1(keep=T_Dv1 Ts_Dv1) SC_Dv1(keep=C_Dv1 Cs_Dv1);
T2_Ts_Dv1=T_Dv1**2-Ts_Dv1;
C2_Cs_Dv1=C_Dv1**2-Cs_Dv1;
TC_Dv1=T_Dv1*C_Dv1;
run;

data S_Dv0;
merge ST_Dv0(keep=T_Dv0 Ts_Dv0) SC_Dv0(keep=C_Dv0 Cs_Dv0);
```

```
T2_Ts_Dv0=T_Dv0**2-Ts_Dv0;
C2_Cs_Dv0=C_Dv0**2-Cs_Dv0;
TC_Dv0=T_Dv0*C_Dv0;
run;

proc means data=S_Dv1 noprint;
output out=Sum_Dv1 sum=;
run;

proc means data=S_Dv0 noprint;
output out=Sum_Dv0 sum=;
run;

data Sum_U;
merge Sum_Dv1 (drop= _FREQ_ _TYPE_) Sum_Dv0 (drop= _FREQ_ _TYPE_);
run;

/*Calculate case 1-14 and the variance for the number of wins for the
treatment arm (V_WT), the variance for the number of wins for the control
arm
(V_WC) and their covariance (V_WTWC). Finally calculate the expectation
(Exp_WD) and variance for the win
difference (Var_WD)*/
data variance;
set Sum_U;
WT=T_Dv1; /*=T_Dv0 */
WC=C_Dv1;  /*=C_Dv0 */
factor1 = (2*&nT-1)*(2*&nC - 1)/(&nT*&nC);
factor2 = (2*&nT-1)*(&nC-1)/(&nT*&nC);
factor3 = (&nT-1)*(2*&nC-1)/(&nT*&nC);
factor4 = (&nT-1)*(&nC-1)/(&nT*&nC);
V_WT = factor1*Ts_Dv1 + factor2*T2_Ts_Dv1 + factor3*T2_Ts_Dv0 +
factor4*(WT**2-Ts_Dv1-T2_Ts_Dv1-T2_Ts_Dv0)-WT**2;
V_WC = factor1*Cs_Dv1 + factor2*C2_Cs_Dv1 + factor3*C2_Cs_Dv0 +
factor4*(WC**2-Cs_Dv1-C2_Cs_Dv1-C2_Cs_Dv0)-WC**2;
V_WTWC = factor2*TC_Dv1 + factor3*TC_Dv0 + factor4*(WT*WC-TC_Dv1-TC_Dv0) -
WT*WC;
Exp_WD=WT-WC;
Var_WD=V_WT+V_WC-2*V_WTWC;
run;

proc print data=variance (keep=V_WT V_WC V_WTWC Exp_WD Var_WD); run;
```