Appendix A. Local search algorithm

```
k = 0.05;                                         \\ initialize annealing constant k
tabuList = [];                                    \\ initialize tabuList as empty

while ContinueSearch~=0                           \\ continue until stopping condition is met
  ContinueSearch = 0;                             \\ initialize stopping condition
  Counter = 0;                                    \\ initialize count of runs since updating k

  while Counter < CounterMax                      \\ if the count is less than our
                                                  \\ prespecified max count
    List = setdiff(1:length(ListOfMoves),tabuList);   \\ find rows from the ListOfMoves not in tabuList
    MoveNo = randsample(List,1);                  \\ select random MoveNo from those not in tabuList

    tabuList(end+1) = MoveNo;                     \\ add selected MoveNo to the tabuList
    if numel(tabuList) > tabuTenure               \\ if the tabuList is full
      tabuList = tabuList(2:end);                 \\ let the top row expire
    end

    [AICnew,MatchNew] = Rearrange(MoveNo,...      \\ find the best rearrangement
                                  MatchOld,AICold);   \\ of the selected segment

    if AICnew > AICold                            \\ if the new AIC is worse than the old AIC
      if exp(-k*(AICnew-AICold)) > rand()         \\ should we accept the worse match?
        AICold = AICnew;   MatchOld = MatchNew;   \\ yes, update the old combination and AIC
        ContinueSearch = 1;                       \\ ammend stopping condition
      end
    else                                          \\ if the new AIC is better than the old AIC
      AICold = AICnew;    MatchOld = MatchNew;    \\ update the stored old combination and AIC
      if AICnew < AICbest                         \\ if the new AIC is better than the best AIC
        AICbest = AICnew;   MatchBest = MatchNew; \\ update the best combination and AIC
      end
    end
    Counter = Counter + 1;                        \\ add 1 to the count since updating k
  end
  k=alpha*k;                                      \\ update the annealing constant.
end

MatchBest                                         \\ return the optimal solution
```