TU Wien – WS 2023/24
Heuristic Optimization Techniques
Programming Assigmnent 1

Matteo Migliarini 12306959
Nicola Maestri 12306354

Group 30

Weighted s-plex Editing Problem and its applications

The Weighted s-Plex Editing Problem (WsPEP) consists of finding the optimal way to edit edges of a weighted undirected graph to transform it into a graph in which connected components are s-plexes. Although reaching the best solutions is difficult for large instances, solving this problem could have many real-world applications due to the numerous settings that can be modeled using graphs.
For instance, in the field of Social Media Analysis our graph represents a network of people where nodes are the users and edges the connections among them. We can also assign a weight to each edge according to some features we are interested in, for example the frequency of messages between users. Editing this graph, as in WsPEP, corresponds to creating clusters of people who all know each other, except for at most s users. This could be useful for suggesting new people for users to know or for targeting advertisements.

Deterministic Heuristic Construction

This greedy algorithm constructs an admissible solution for the WsPE Problem. Solutions are built in two steps: first, nodes of the original graph are subdivided into clusters; then, edges are adjusted to guarantee that each cluster is an s-plex.
These two operations are carried out in two stages:

1. **Clustering**

The algorithm receives a graph as input and assigns each node to a different cluster. An iterative procedure tries to merge clusters until a condition is satisfied. In particular, at each step, the sum of edges from one cluster to another is computed, and the two clusters with the minimum sum are merged. The procedure goes on until either the sum of weights between clusters is positive, or there are only two clusters.

2. **s-plex improvement**

For each cluster, we assume that each node is connected to each other, and then we iteratively remove the heaviest edge until the cluster remains an s-plex.

Algorithm 1 Greedy Heuristic Construction

$G = (V, E) \leftarrow$ input graph
Assign each node to a different cluster
repeat
  % Find two clusters to merge
  $min\_weight \leftarrow 0$
  $clusters\_to\_merge(A, B) = None$
  for each couple of clusters (X, Y) **do**
    if $weight\_between\_(X, Y) < min\_weight$ **then**
      $min\_weight \leftarrow weight\_between\_(X, Y)$
      $(A, B) = (X, Y)$
    end if
  end for

  % Adjust edges inside the new s-plex
  Connect each node with each other inside the new cluster
  repeatRemove heaviest edge between two nodes of the new cluster
  until removal of the edge make the cluster no an s-plex
until all original edges are reinserted or there are only two clusters

Settings

Solutions are represented as an adjacency matrix. To efficiently compute the sum of edge weights between two clusters, we use a squared matrix with a size equal to the number of clusters, which stores these values. Initially, this matrix is identical to the weights matrix, and at each step, we update it by merging the two rows and columns corresponding to merged clusters. This procedure allows us to save unnecessary computations.
Throughout the following, weights are considered positive when we add an edge not present in the original graph or remove an edge part of the original graph. On the other hand, weights are considered negative when we add an edge already present in the original graph or remove an edge that is not part of the original graph.

We also implemented another version of heuristic construction following this approach, but we didn't use it in the following development.

Algorithm 2 Alternative Greedy Heuristic Construction

$G = (V, E) \leftarrow$ input graph
Assign each node to a different cluster
% Try to reinsert all original edges preserving admissibility
for each edge $(i, j) \in E$ **do**
  if (i, j) alreay inserted **then**
    continue
  else if i, j in the same cluster **then**
    add the edge
  else add minimum number of edges to merge s-plexes of i and j
  end if
end for

Randomized Heuristic Construction

Algorithm 3 Randomized Heuristic Construction

$G = (V, E) \leftarrow$ input graph
Assign each node to a different cluster
repeat
  % Pick two clusters with some probability and merge them
  for each couple of clusters (X, Y) **do**
    $Probability\_of\_merge(X, Y) = \frac{1}{weight\_between\_(X,Y)}$
  end for
  Normalize probabilities to sum to one
  Pick two a couple of two clusters according to this probabilities

  % Adjust edges inside the new s-plex
  Connect each node with each other inside the new cluster
  repeatRemove heaviest edge between two nodes of the new cluster
  until removal of the edge make the cluster no an s-plex

until all original edges are reinserted or there are only two clusters

The greedy construction is randomized adding randomness to how we assign nodes to clusters. In particular, we no longer merge clusters with the minimum sum of weights; instead, we assign to each couple of clusters a probability to be merged proportional to this value. This means that good solutions are encouraged, but all solutions are possible. To add more randomness, we could also act on the way the algorithm modifies edges inside each cluster but we didn't explore further this possibility.

Basic Local Search

Algorithm 4 Randomized Heuristic Construction

```
S ← Randomized Heuristic Costruction
N ← selected neighborhood structure
repeat
    choose S' ∈ N(S)
    if f(S') < f(S) then
        S ← S
    end if
until
```

We implemented basic local search with different neighbor structures and step functions. Implemented step functions are:
- **First improvement**: first better solution is taken
- **Best improvement**: best solution in the neighborhood is taken
- **Random neighbor**: a random neighbor is compared with the current solution

In particular, in the code is possibble to uncomment the desired step function to use for choosing the next neighbor to visit. In practice, we observed that it was harder to coverge to a solutions both with best improvement and random step function due to the size of the neighbor structures adopted.

We developed more neighbor structures to cope with different aspects of the problem:
1-Flip Neighborhood
Given a solution S, neighbors are obtained by adding or removing one edge in an s-plex of S
Swap-edges Neighborhood
Given a solution S, neighbors are obtained by swapping two edges in an s-plex of S.
Swap-nodes Neighborhood
Given a solution S, neighbors are obtained by considering two nodes in different clusters, swapping them, and reconnecting these nodes to the neighbors of the substituted node.
Move one node
Given a solution S, neighbors are obtained by moving one node from one cluster to another and adjusting connections in the cluster. Each node of the modified s-plex is connected to at least $n - s$ nodes, if needed adding the edges with minimum weight.
Break s-plex:
Given a solution S, neighbors are obtained by subdividing a cluster into two subclusters. Each node of the new s-plexes is connected to at least $n - s$ nodes of its new cluster, if needed adding the edges with minimum weight.

The first two structures address the problem of improving the quality of each s-plex, whereas the last three structures help find a good way for clustering nodes in s-plexes. A limitation of these neighborhoods is their size, which makes a full exploration prohibitive for large instances. Therefore, for some instances it could be convenient to either use the first improvement as step function or to explore only a subset of them in a more deterministic way.

Despite we implemented all these structures, we prefered to avoid the use of *Swap-edges Neighborhood* and *Break s-plex* due to complexity reasons and uncleared side effects of our code. Nevertheless, *1-Flip Neighborhood*, *Swap-nodes Neighborhood* and *Move one node* appear to fit well the problem and we used them for *Variable Neighborhood Descent*

Variable Neighborhood Descent (VND)

Algorithm 5 Greedy Heuristic Construction

$S \leftarrow$ *Randomized Heuristic Costruction*
$\{\mathcal{N}_i\}_1^{\ell_{max}} \leftarrow$ *neighborhood structure*
$\ell \leftarrow 1$
repeat
  find $S' \in \mathcal{N}_\ell(S)$ with $f(S') < f(S''), \forall S'' \in \mathcal{N}_\ell(S)$
  if $f(S') < f(S)$ **then**
    $S \leftarrow S'$
    $\ell \leftarrow 1$
  else $\ell = \ell + 1$
  end if
until $\ell > \ell_{max}$

Variable Neighborhood Descent (VND) is an improvement of basic Local Search that tries to avoid local minima by considering more neighbor structures.
We implemented an algorithm for VND that takes a list of neighbor structures as input, and at each iteration, it attempts to improve the current solution. Neighborhoods are searched sequentially, and the algorithm terminates when it reaches a local minimum for all the considered structures.
Our tests suggest that a good order for neighbor structures is [*Swap node*, *Move node*, *1-Flip*]; all the results are reported in Table 2.

GRASP

Algorithm 6 Greedy Heuristic Construction

$S \leftarrow None$
repeat
  $S \leftarrow$ *Randomized Greedy Heuristic*
  $S' \leftarrow Local\_Search(S)$
  if $f(S') < f(S)$ **then**
    $S \leftarrow S'$
  end if
until

Grasp implemantation follows naturally from aformentioned *Randomized Greedy Heuristic* and *Local Search*. This method increases significantly the quality of the final solution and leads to results comparable or even slightly better respect to VND and Simulated Annealing which will be introduced in the subsequent section. All results are reported in Table 2.

Simulated Annealing

Algorithm 7 Greedy Heuristic Construction

```
T ← T_init
epoch ← 0
S ← initial solution
cooling ← 0.95
repeat
    repeat
        S' ← random neighbor out of N(S)
        p ← random value in [0, 1]
        if f(S') < f(S) then
            S ← S'
        else
            % Metropolis criterion
            if p < e^{-|f(S')-f(S)|/T} then
                S ← S'
            end if
        end if
        epoch = epoch + 1
    until
    T ← T * cooling
until T < T_min  or  epoch > max_epoch
```

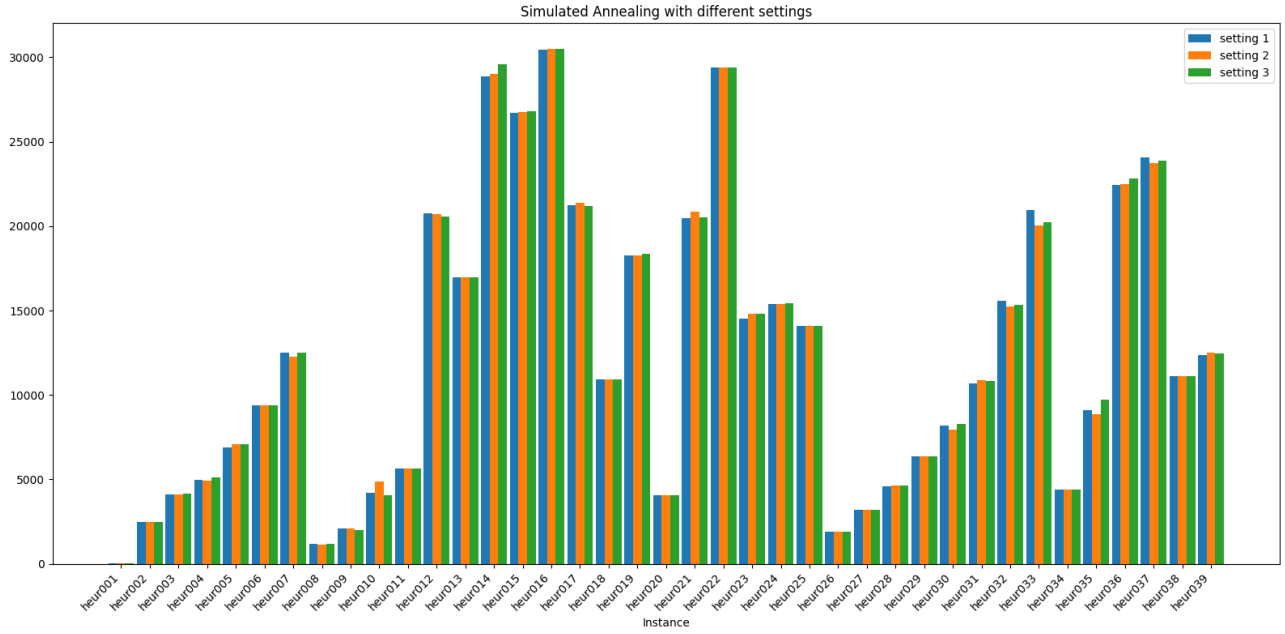
Simulated annealing is implemented following this pseudocode. We use a *Swap-node* and *1-Flip* as the neighborhood because it empirically seems to be a better choice compared to other neighbor structures. To fit the general setting to the specific task, we worked with different settings of temperature and cooling. Results and comparisons are analyzed in the next section.

Parameter Tuning

In our general setting of Simulated Annealing we set these parameters:
- **Initial Temperature**: 10^5
- **Minimum Temperature**: 10^{-3}
- **Cooling Schedule**: $alpha = 0.92$
- **Maximum number of iterations**: 5000

This setting proved to give results comparable with other methods as shown in Table 2, however we tried to explore how changes of those values impact on the quality of the solution. In particular we report in Table 1 results obtained with different setting of initial Temperature and cooling schedule. In the first experiment, we decreased the initial temperature to 10^3 and we observed that the overall quality of solutions is on the level of our original setting. In the second experiment, we tried to set the parameter α to a different value. This has a significant impact on the performance and running time of the algorithm, in particular if we

increase the value of α the cooling is much slower and we observed that for $\alpha = 0.96$ the running time was significantly higher also for small instances. On the other hand, decreasing α leads to a quicker cooling and faster convergence but the exploration of solution space is reduced. We empirically observed that under the value of $\alpha = 0.85$ there was almost no improvement respect to the solution proposed by the greedy construction heuristic.
Results of our experiments are reported in the graphic below while the results are listed in Table 1.



istance	$T_{init} = 10^5, \alpha = 0.92$	$T_{init} = 10^3, \alpha = 0.92$	$T_{init} = 10^5, \alpha = 0.85$
heur001	35	35	35
heur002	2472	2472	2472
heur003	4113	4097	4169
heur004	4993	4935	5095
heur005	6872	7099	7089
heur006	9381	9383	9398
heur007	12531	12256	12515
heur008	1192	1128	1192
heur009	2075	2075	1977
heur010	4217	4867	4081
heur011	5661	5661	5660
heur012	20778	20718	20584
heur013	16962	16962	16962
heur014	28888	28996	29604
heur015	26727	26740	26813
heur016	30470	30484	30498
heur017	21248	21372	21184
heur018	10918	10918	10918
heur019	18248	18271	18360
heur020	4053	4053	4053
heur021	20481	20837	20509
heur022	29410	29410	29410
heur023	14524	14805	14814
heur024	15376	15406	15420
heur025	14070	14077	14085
heur026	1912	1912	1912
heur027	3187	3187	3187
heur028	4571	4628	4629
heur029	6381	6381	6381
heur030	8170	7967	8294
heur031	10661	10855	10846
heur032	15556	15264	15321
heur033	20969	20038	20216
heur034	4393	4393	4393
heur035	9099	8862	9707
heur036	22459	22497	22813
heur037	24071	23733	23896
heur038	11117	11117	11117
heur039	12365	12510	12478

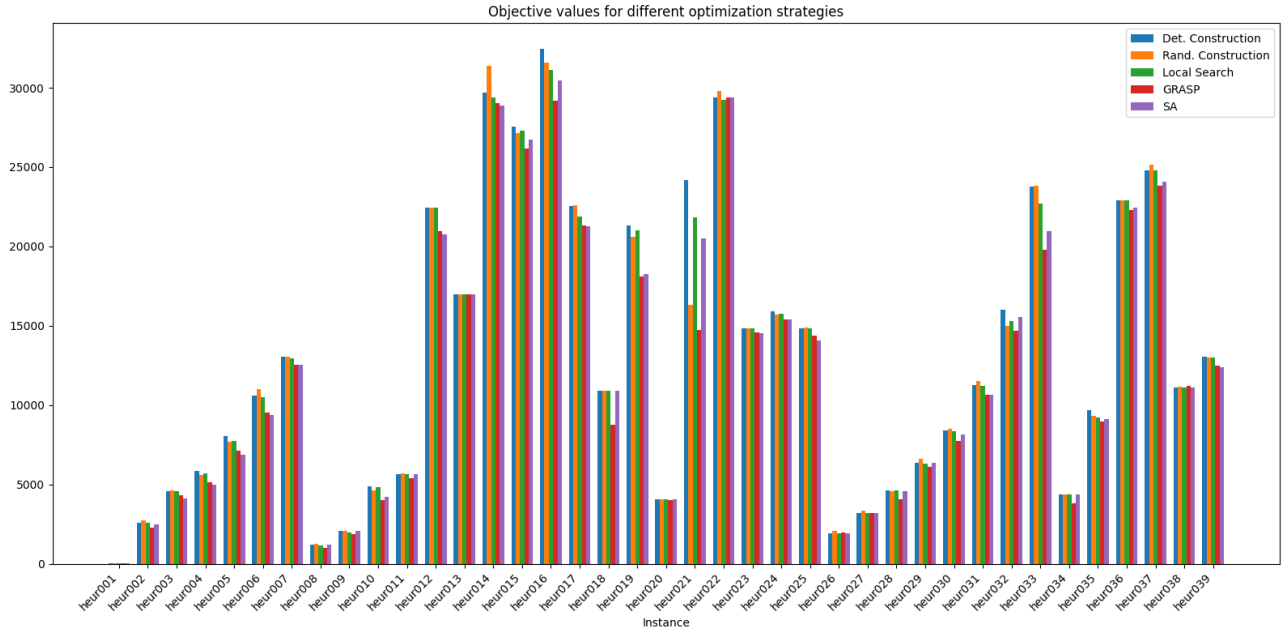
Table 1:

Delta Evaluation

In the Weighted s-Plex Editing Problem the objective function to minimize corresponds to the sum over weights of edited edges and these values are retrievable from adjacency matrix and weights matrix. Delta evaluation plays a fundamental role to avoid repetitive and useless operations for computing the objective value of neighbors of a current solution.
In particular, we can compute the difference between these two values according to the neighborhood the modified solution belongs to:
1-Flip Neighborhood:
$\Delta_f(S, S') = w(edited\ edge)$, which could be positive or negative according to the edge we are editing.
Swap-edges Neighborhood
$\Delta_f(S, S') = (w(e_{1'}) + w(e_{2'})) - (w(e_1) + w(e_2))$ with e_1, e_2 removed edges and $e_{1'}, e_{2'}$ added edges.
Swap-node Neighborhood
$\Delta_f(S, S') = (\sum_{i \in I} w(e_{v,i}) + \sum_{j \in J} w(e_{uj})) - (\sum_{i \in I} w(e_{u,i}) + \sum_{j \in J} w(e_{vj}))$
with v, u swapped nodes, $I := \{nodes\ connected\ with\ u\ in\ S\}$, $J := \{nodes\ connected\ with\ v\ in\ S\}$
Move one node
$\Delta_f(S, S') = \sum_{i \in I'} w(e_{v,i}) - \sum_{i \in I} w(e_{vj}) + \sum_{e \in X} w(e)$
with v moved node, $I := \{nodes\ connected\ with\ v\ in\ S\}$, $I' := \{nodes\ connected\ with\ v\ in\ S'\}$,
$X := min_{Y \subseteq E}\{new\ cluster\ of\ v\ remains\ a\ s-plex\}$
Break s-plex:
$\Delta_f(S, S') = \sum_{e \in X} w(e) - \sum_{e \in Y} w(e)$, with
$X := \{added\ edges\ to\ make\ the\ two\ new\ clusters\ s-plexes\}$
$Y := \{removed\ edges\ to\ create\ the\ two\ clusters\}$

In our implementation this difference is an attribute of the neighbor, therefore it is straighforward to evaluate the objective function on the new candidate solution.

Results and Comparisons



We now compare results obtained with different algorithms on the same istance.
In particular, we focus on *Deterministic Heuristic Construction*, *Randomized Heuristic Costruction*, *Local Search*, *VND, GRASP* and *Simulated Annealing*. We present all the values obtained in Table 2.
The graph above shows the best values of the objective function for each instance and it is visible that using GRASP and SA lead to a better solution compared to both the Greedy Construction and Basic Local Search. GRASP, VND and Simulated Annealing all prove to be valid solutions and there is no a best method for all the instances as shown in the second graphic. Nevertheless we observed that on average GRASP is faster than VND and SA and solutions are slightly better on these instances. VND can take a lot of time in realation to the neighborhood considered.
We report here neighborhood used in each algorithm:
- **Local Search**: *swap one node*
- **VND**: *[swap one node, 1-flip]*
- **GRASP**: *swap one node*
- **Simulated Annealing**: *swap one node*

With this choice we tried to compare algorithms in a more fair way avoiding bias due to differences in the neighborhood structure

istance	Det. Constr.	Rand. Constr.	Local Search	VND	GRASP	SA
heur001	35	38	35	35	35	35
heur002	2567	2737	2567	2567	2267	2472
heur003	4595	4621	4583	4543	4320	4113
heur004	5838	5605	5716	5664	5132	4993
heur005	8057	7701	7725	7472	7125	6872
heur006	10621	11000	10506	10446	9510	9381
heur007	13059	13026	12958	12427	12532	12531
heur008	1192	1269	1158	1094	1002	1192
heur009	2075	2081	1972	1805	1861	2075
heur010	4902	4642	4819	4719	4025	4217
heur011	5661	5681	5658	5658	5377	5661
heur012	22441	22441	22441	22441	20957	20778
heur013	16962	16962	16962	16962	16962	16962
heur014	29684	31398	29379	28815	29052	28888
heur015	27548	27139	27272	27272	26197	26727
heur016	32447	31563	31116	30405	29178	30470
heur017	22528	22608	21901	-	21297	21248
heur018	10918	10918	10918	-	8776	10918
heur019	21347	20598	21031	-	18114	18248
heur020	4053	4079	4053	-	4030	4053
heur021	24177	16296	21827	-	14712	20481
heur022	29410	29797	29229	-	29403	29410
heur023	14817	14846	14817	-	14592	14524
heur024	15909	15725	15771	-	15414	15376
heur025	14840	14877	14840	-	14378	14070
heur026	1912	2077	1912	-	1982	1912
heur027	3187	3361	3187	-	3173	3187
heur028	4629	4575	4609	-	4068	4571
heur029	6381	6637	6320	-	6130	6381
heur030	8411	8526	8376	-	7759	8170
heur031	11257	11532	11210	-	10638	10661
heur032	16024	15013	15322	-	14686	15556
heur033	23758	23845	22721	-	19789	20969
heur034	4393	4395	4393	-	3810	4393
heur035	9707	9301	9211	-	8976	9099
heur036	22925	22922	22925	-	22307	22459
heur037	24814	25138	24814	-	23827	24071
heur038	11117	11181	11115	-	11198	11117
heur039	13028	13022	12999	-	12495	12365

Table 2: