Guided Local Search
导向式本地搜索

By 吴婷

Contents

- Introduction
- Implement
- Application
- Conclusions

“Introduction of GLS

objective function *g* : maps every candidate solution *s* to a numerical value
function *h* : be used by LS (replacing *g*):

$$h(s) = g(s) + \lambda \times \sum (p_i \times I_i(s)) \qquad (1)$$

The utility of penalizing feature *i*, under a local optimum is defined as follows:

$$\text{util}_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \qquad (2)$$

- GLS focuses its search effort on more promising areas of the search space: areas that contain candidate solutions that exhibit “good features”.
- Penalties help to prevent the search from directing all effort to any particular region of the search space.

General Procedure of GLS

```
Procedure GLS (L, g, I, λ, c)
/* g is an objective function; L is a LS strategy; I and c are arrays of
features and their costs; λ is a scaling number*/
1. Generate a starting candidate solution randomly or heuristically;
2. Initialize all the penalty values to 0;
3. Repeat the following until a termination condition has been reached:
   3.1. Perform local search until a local optimum LM has been reached;
   3.2. For each feature i which is exhibited in LM compute utility
                $util_i = c_i/(1+p_i);$
   3.3. Penalize every feature i such that util(i) is maximum:
                $p_i = p_i + 1;$
4. Return the best candidate solution found so far according to the
objective function g.
```

“Introduction of FLS

The principle : ignore neighbors that are unlikely to lead to improving moves in order to enhance the efficiency of a search.

Break the choosen neighborhood down into a number of small sub-neighborhoods and attach an activation bit to each one of them.
The idea is to scan continuously the sub-neighborhoods in a given order, searching only those with the activation bit set to 1.
The neighborhood search process does not restart whenever we find a better solution but it continues with the next sub-neighborhood in the given order.
Initially, all sub-neighborhoods are active. If a sub-neighborhood is examined and does not contain any improving moves then it becomes inactive.

“ GLS and other metaheuristics

GLS and Tabu Search

GLS is closely related to Tabu Search (TS). For example, penalties in GLS can be seen as soft taboos in TS that guide LS away from local minima. There are many ways to adopt TS ideas in GLS.

GLS and Genetic Algorithms

As a meta-heuristic method, GLS can also sit on top of genetic algorithms (GA). This has been demonstrated in Guided Genetic Algorithm (GGA) (Lau and Tsang, 1997 1998; Lau, 1999).

“

GLFS

Combining GLS with FLS is straightforward. The key idea is to associate features to subneighborhoods. By reducing the size of the neighbourhood, one may significantly reduce the amount of computation involved in each local search iterations.

Pseudo-code for Guided Local Search

```
procedure GuidedLocalSeach(p, g, M, λ, I )
        k = 0;
        s0 = ConstructionMethod(p);
        /* set all penalties to 0 */
        for i = 1 to M do
                p(i)=0
        /* define the augmented objective function */
                $h \leftarrow g + \lambda * \sum p_i * I_i;$
        while StoppingCriterion do
                s(k+1) = ImprovementMethod(s(k), h)
                /* compute the utility of features */
                for i = 1 to M do
                $util_i \leftarrow I_i(s_{k+1}) * c_i/(1 + p_i);$
                /* penalize features with maximum utility */
                for each i such that util(i) is maximum do
                        p(i)=p(i+1)
                k = k + 1;
        s* best solution found
        return s*;
```

$\lambda = \alpha * g(\text{local minimum})/(\text{no. of features present in the local minimum}),$

Pseudo-code for Fast Local Search

```
procedure FastLocalSeach(s, h, bit, L)
while bit(i)==1 exists /* i.e. while active sub-neighborhood exists */
        for i = 1 to L do
                if bit(i)==1 /* search sub-neighborhood i */
                Moves = MovesForSubneighborhood(i) ;
                for each move m in Moves do
                        s' = m(s) /* s' is the result of move m */
                        if h(s') < h(s) then /*minimization case is assumed here*/
                                /* spread activation */
                                ActivateSet = SubneighborhoodsForMove(m);
                                for each sub-neighborhood j in ActivateSet
                                        bit(j) = 1
                                s = s'
                                goto ImprovingMoveFound
                        bit(j) = 0 /* no improving move found */
        ImprovingMoveFound: continue;
return s;
```

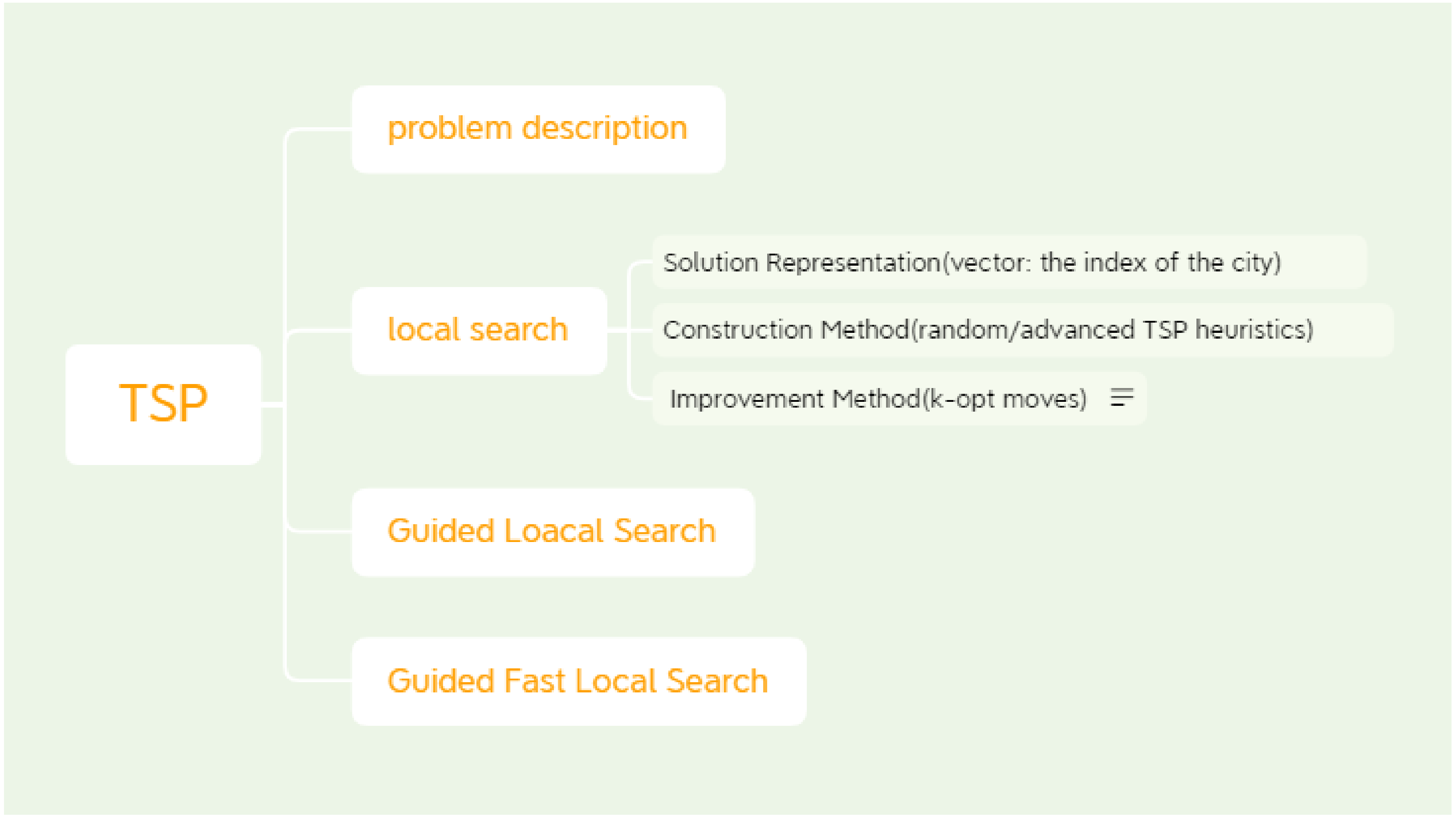
Pseudo-code for Guided Fast Local Search

```
procedure GuidedFastLocalSearch(p, g, λ, I, c, M, L )
k = 0
S(0) = ConstructionMethod(p);
for i = 1 to M
        p(i) = 0; /* set all penalties to 0 */
for i = 1 to L
        bit(i) = 1 /* set all sub-neighborhoods to the active state */
$h \leftarrow g + \lambda * \sum p_i * I_i$; /* define the augmented objective function */
while StoppingCriterion do
        s(k+1) = FastLocalSearch(s(k), h, bit, L)
        for i = 1 to M do
                $util_i \leftarrow I_i(s_{k+1}) * c_i/(1+p_i)$; /* compute the utility of features */
                /* penalize features with maximum utility */
                for each i such that util(i) is maximum do
                        p(i) = p(i) + 1
                        /* activate sub-neighborhoods related to penalized feature i */
                        ActivateSet = SubneighborhoodsForFeature(i);
                        for each sub-neighborhood j in ActivateSet do
                                bit(j) = 1
                        k = k + 1
s* best solution found;
return s*;
```

Applications

Routing/Scheduling category	Traveling Salesman Problem
Assignment Problem category	Quadratic Assignment Problem
Resource Allocation category	Workforce Scheduling Problem
Constraint Optimization category	Radio Link Frequency Assignment Problem

Application in TSP


TSP
problem description
local search
Solution Representation(vector: the index of the city)
Construction Method(random/advanced TSP heuristics)
Improvement Method(k-opt moves)
Guided Loacal Search
Guided Fast Local Search

conclusions

- Features of high cost are penalized more times than features of low cost: the diversification process is directed and deterministic rather than undirected and random
- several penalty cycles may be required before a move is executed out of a local minimum.
- GFLS significantly reduces the computation times required to explore the area around a local minimum to find the best escape route allowing many more penalty modification cycles to be performed in a given amount of running time.