DP-ISO说明文档

概述

DP-ISO是基于本组对于图匹配的接口需求以及效率需求设计的基于DAF思想的子图同构算法

算法概述

DP-ISO的主要优化部分分为以下三个部分

DAGDP

对于query的每个弱连通子图 构建一个DAG并根据邻接关系减少candidate set的规模

Adaptive Matching Order

采用了candidate size order作为match order，同时采用动态更新candidate set的方式

Fail Set

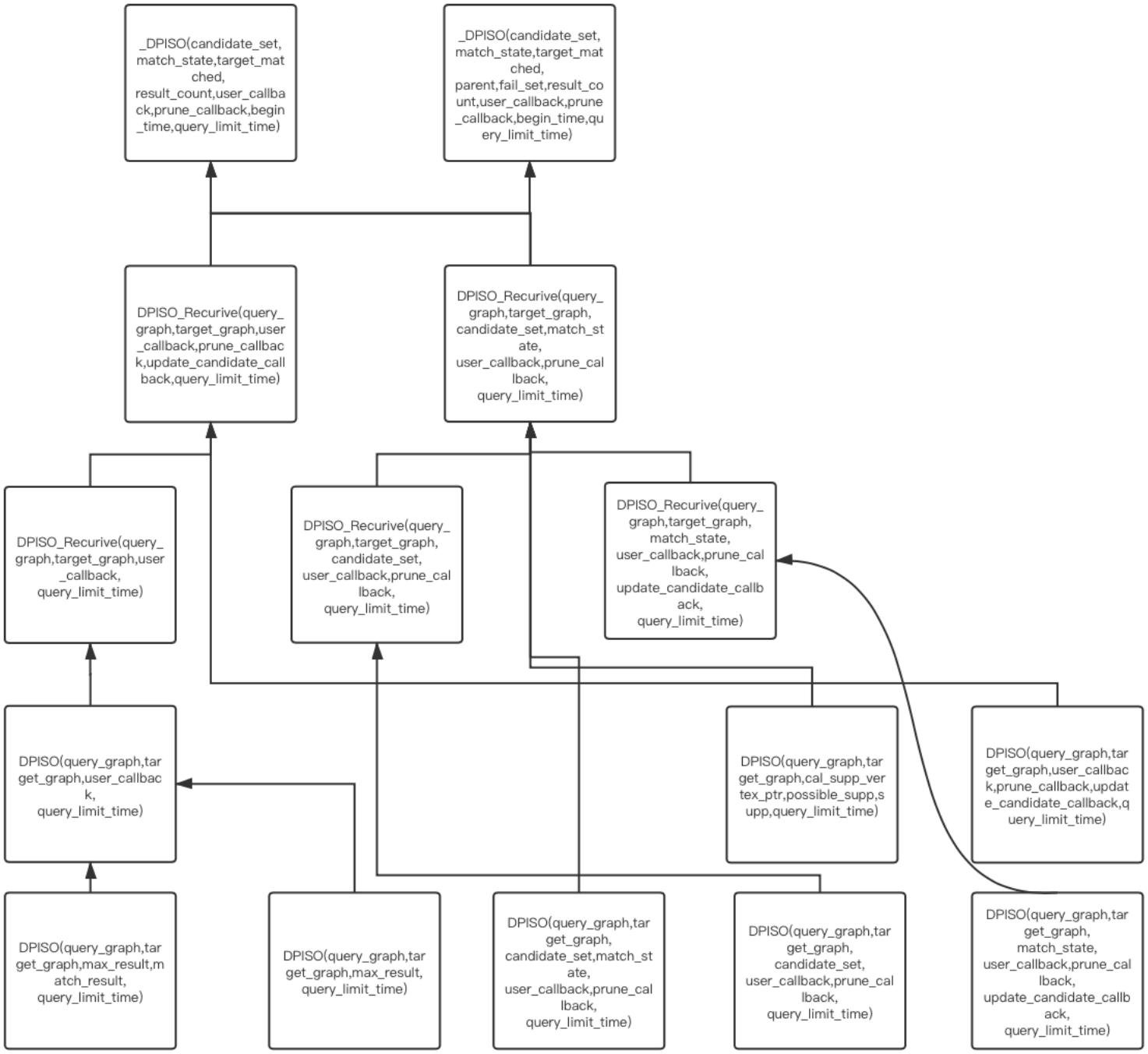
利用匹配时的先验知识对当前状态进行剪枝，从而减少搜索空间

与DAF的不同

1.不采用拓扑序而采用邻接节点，可以支持部分匹配
2.考虑到Graph相关的遍历性能，不构建candidate space
3.增加了匹配的自适应能力，若query较小则不采用fail set优化
4.可匹配带有多个连通块的query

代码设计

箭头表示调用关系



QueryGraph和TargetGraph要求

QueryGraph

使用GUNDAM::SmallGraph,GUNDAM::LargeGraph或GUNDAM::LargeGraph2都可以

TargetGraph

使用GUNDAM::LargeGraph或GUNDAM::LargeGraph2

可用接口及其相关参数说明

constexpr size_t large_query_edge = 6

若query的边数超过这个数目，则采用fail set

constexpr size_t adj_vertex_limit = 200000

在更新Candidate Set时，如果某个点以edge_label为边label的邻接点个数超过这个数目，则不更新与该部分点有关的canddiate set
数字设置：实际上是如果某个点的邻接点是这个图的大部分点（比如超过80%），那么更新邻居节点的candidate set时，这部分的更新大概率没用，还会费时间。根据实际需要自己调整

DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,MatchCallback user_callback, double query_limit_time = 1200)

给定Query Graph和Target Graph的情况下，通过user_callback进行匹配结果的处理，默认设定的执行时间为1200s（若超过这个时间，则强制结束）

user_callback

一个简单的user_callback可以写成如下形式：

```
  using QueryVertex = typename QueryGraph::VertexConstPtr;
  using TargetVertex = typename TargetGraph::VertexConstPtr;
  using MatchMap = std::map<QueryVertex, TargetVertex>;
  using MatchResult = std::vector<MatchMap>;

  MatchResult match_result;
  auto match_callback = [&match_result](const MatchMap &match_map) {  //也可以写成const a
uto &match_map
    match_result.emplace_back(match_map); //将match存进match result中
    return true;  //返回值要为bool类型
  };
```

其中，MatchMap为DPISO运行过程中的match数据结构

DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,int max_result, double query_limit_time = 1200)

给定Query Graph和Target Graph的情况下，计算匹配数量（不超过max_result），默认设定的执行时间为1200s（若超过这个时间，则强制结束）

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,int max_result, ResultContainer &match_result,double query_limit_time = 1200)

给定Query Graph和Target Graph的情况下，计算匹配数量（不超过max_result)并将其存储在match_result中，默认设定的执行时间为1200s（若超过这个时间，则强制结束）

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,typename QueryGraph::VertexConstPtr cal_supp_vertex_ptr,const std::vector &possible_supp,std::vector &supp,double single_query_limit_time = 100)

给定Query Graph和Target Graph的情况下，根据possible_supp的结果计算匹cal_supp_vertex_ptr的supp并将其存储在supp中，默认possible_supp内每个点的匹配计算时间为100s（若超过这个时间，则认为该点不为supp）

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,MatchCallback match_callback, PruneCallBack prune_callback,UpdateInitCandidateCallback update_initcandidate_callback,double query_limit_time = 1200)

给定Query Graph和Target Graph的情况下，通过user_callback进行匹配结果的处理，通过prune_callback进行剪枝处理，通过update_initcandidate_callback来人工对candidate_set进行一定的限制。默认设定的执行时间为1200s（若超过这个时间，则强制结束）

user_callback

一个简单的user_callback可以写成如下形式：

```
  using QueryVertex = typename QueryGraph::VertexConstPtr;
  using TargetVertex = typename TargetGraph::VertexConstPtr;
  using MatchMap = std::map<QueryVertex, TargetVertex>;
  using MatchResult = std::vector<MatchMap>;

  MatchResult match_result;
  auto match_callback = [&match_result](const MatchMap &match_map) {  //也可以写成const a
uto &match_map
    match_result.emplace_back(match_map); //将match存进match result中
    return true; //返回值要为bool类型
  };
```

其中，MatchMap为DPISO运行过程中的match数据结构

prune_callback

一个简单的prune_callback形式如下所示：

```
   auto prune_callback = [](const auto &match_state){
```

```
return true;   //空的prune callback  , 返回值要为bool类型
};
```

其中，match_state表示的是match,其具体数据结构在user_callback中说明

update_initcandidate_callback

一个简单的update_initcandidate_callback形式如下所示：

```
using QueryVertex = typename QueryGraph::VertexConstPtr;
using TargetVertex = typename TargetGraph::VertexConstPtr;
using CandidateSetContainer = std::map<QueryVertex,std::vector<TargetVertex>>;
 auto update_callback = [](CandidateSetContainer  &candidate_set){
 return true;   //空的update callback, 返回值要为bool类型
 };
```

其中，candidate_set表示这个pattern在该target graph下的candidate set

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,std::map<typename QueryGraph::VertexConstPtr,std::vector<typename TargetGraph::VertexConstPtr>> &candidate_set,std::map<typename QueryGraph::VertexConstPtr,typename TargetGraph::VertexConstPtr> &match_state,MatchCallback user_callback, PruneCallback prune_callback,double query_limit_time = 1200)

在已知candidate_set和match_state的情况下，给定Query Graph和Target Graph，通过user_callback进行匹配结果的处理，通过prune_callback进行剪枝处理。默认设定的执行时间为1200s（若超过这个时间，则强制结束）
tips:candidate_set和match_state在运行时会被修改，外部做好备份

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,std::map<typename QueryGraph::VertexConstPtr,std::vector<typename TargetGraph::VertexConstPtr>> &candidate_set,MatchCallback user_callback, PruneCallback prune_callback,double query_limit_time = 1200)

在已知candidate_set的情况下，给定Query Graph和Target Graph，通过user_callback进行匹配结果的处理，通过prune_callback进行剪枝处理。默认设定的执行时间为1200s（若超过这个时间，则强制结束）
tips:candidate_set在运行时会被修改，外部做好备份

int DPISO(const QueryGraph &query_graph, const TargetGraph

&target_graph,std::map<typename QueryGraph::VertexConstPtr,typename TargetGraph::VertexConstPtr> &match_state,MatchCallback user_callback, PruneCallback prune_callback,UpdateCandidateCallback update_candidate_callback,double query_limit_time = 1200)

在已知match_state的情况下，给定Query Graph和Target Graph，通过user_callback进行匹配结果的处理，通过prune_callback进行剪枝处理,通过update_initcandidate_callback来人工对candidate_set进行一定的限制。。默认设定的执行时间为1200s（若超过这个时间，则强制结束）
tips:match_state在运行时会被修改，外部做好备份

int DPISO(const QueryGraph &query_graph, const TargetGraph &target_graph,const SuppContainer &supp_list,UserCallBack user_callback)

在已知supp包含的节点的情况下，给定Query Graph和Target Graph，通过user_callback进行匹配结果的处理（只留supp里的节点还是留完整匹配）。
supp_list是一个包含QueryGraph::VertexConstPtr的容器，要求使用stl的容器（vector,list,set都可以）