Improved Bi-directional Dynamic Programming For Shortest Path With Resource Constraints in DAG Report

Terli Sai Jaya Chandra[1], Neerati Bhuvanesh[2], and Vadithya Sanjay[2]

[1]21CSB0A60 , B.Tech CSE , NITW
[2] 21CSB0A39 , B.Tech CSE , NITW
[3] 21CSB0A64 , B.Tech CSE , NITW

1-04-2024

Abstract

In this work , we present labeling algorithm in boost graph library to solve shortest path with resource constrainsts in general graphs . How this algorithm can be improved using skyline algorithm , bidirectional dynamic programming and efficient usage of memory for DAGs

Keywords: labeling algorithm , boosted graph library , DAGs, skyline , shorest path , dynamic proramming , optimization

1. Problem Statement

The shortest path problem with resource constraints (SPPRC) searches for a shortest path in a directed graph subject to some additional resource constraints defined along the path, e.g. the total time should not be greater than some bound.

SPPRC is formulated on a graph with a set of nodes and a set of arcs. Each arc is associated with a cost and one or more nonnegative resources. The cost and resource consumption of a path are simply those of all arcs along the path combined. SPPRC then seeks a path from a given source to a given destination such that the cost is minimized and the consumption of each resource is not greater than its given upper bound

2. Labeling Algorithm in Boosted Graph Library

A generic and open-source implementation of the standard labeling algorithm for SPPRC could be found in the Boost Graph Library (BGL), which is part of the well-known Boost C++ libraries

2.1 Labels

A label on a node represents a path from the source to this node, as well as the cost and resource consumptions along the path. From the initial label representing the trivial path with a single node, i.e. the source, the dynamic programming approach extends paths into all directions in case they are resource feasible. The algorithm is also referred as the multi-label algorithm because in contrast to the shortest path problem without resource constraints, multiple labels at the same node could be incomparable

2.2 Dominanace Test

One path may cost less but consume more in (at least) one resource than a second path. In this case, the former path may become infeasible after extension while the latter one remains feasible On the other hand, one path may be better off in all criteria. In this case, the other path is dominated and should be removed by the idea of optimal substructure in dynamic programming The efficiency of the multi-label algorithm heavily relies on the label dominance tests to filter pareto-optimal labels and remove dominated labels which cannot be extended to an optimal solution.

2.3 Descriptions of Functions in the Labeling Algorithm

Following are descriptions of some functions used in the labeling algorithm:

- **rc**: The resource container of the initial label (usually zeroes for the cost and all resources).
- **ref**: The resource extension function defined on each arc in the graph, which controls how the cost and resource consumptions are aggregated along the path and the resource feasibility.
- **dominance**: The condition when one label is dominated by another one at the same node.

Algorithm 1 Labeling Algorithm for Shortest Path with Resource Constraints

Require: First label, current label, new label
Require: Set of unprocessed labels
Ensure: Optimal labels
Initialize first label with resource constraints
INSERT(unprocessed labels, first label)
while unprocessed labels is not empty **do**
    cur label ← EXTRACTMIN(unprocessed labels)
    if cur label is not dominated **then**
        node i ← ResidentNode(cur label)
        Perform pairwise dominance check over labels resident at i
        Mark all dominated labels as dominated
        DELETE all labels which are dominated AND processed
    end if
    if cur label is not dominated **then**
        Mark cur label as processed
        for all arc (i, j) of node i **do**
            new label ← ref(cur label)
            if new label is feasible **then**
                INSERT(unprocessed labels, new label)
                INSERT(set of labels resident at j, new label)
            end if
        end for
    else
        DELETE(cur label)
    end if
end while

3. DAG

The algorithm in BGL is designed for a general directed graph. In many applications, the underlying graph is a directed acyclic graph (DAG) by definition e.g. a time-space network where each arc represents an activity in a certain period of time; apparently, no arc goes from the future to the past. Thus in this work, we concentrate the discussion for DAG. The special structure of DAG can usually reduce the complexity of shortest path problems.

4. Improvement from References

This section tells how can labelling algorithm can be improved by using bidirectional dynamic programming and skyline algorithm.

4.1 Skyline Algorithm

Dominance tests among labels at the same node play a significant role in the algorithm. In pursuit of better performance, we may expect a "smarter" way to do the test than the naive implementation of two nested loops of labels. In fact, such problems have been considered in the database community known as the maximum vector problem or the skyline operator . We may reduce the number of label comparison following the idea of skyline algorithms.

A naive implementation of the skyline operator is composed of two nested loops of labels. For a pair of labels l1 and l2, we perform the dominance test in both directions: we check whether l1 is dominated by l2, and whether l2 is dominated by l1 Several skyline algorithms are proposed to accelerate the process. We choose to apply the skyline algorithm called sort-filter-skyline proposed in [10] for both its simplicity and efficiency.

The idea of the algorithm is that, if we presort the labels such that the former l1 can never be dominated by the latter l2 in the ordering, then we can perform the dominance test in just one direction. Thus we replace a certain amount of label comparison in dominance test by label comparison in sorting. Then we can benefit from the well-studied sorting algorithms. For the classical SPPRC we discuss, a lexicographical ordering of resources (including cost) can do the trick. If l1 comes before l2 in such an ordering, at least one resource of l1 is less than l2 unless they are exactly the same, thus l1 is never dominated by l2.

Algorithm 2 Topological Ordering and Labeling Algorithm

```
1: Perform topological sorting of nodes to get the order.
2: for all node i in topological order do
3:     Initialize first label with resource constraints.
4:     INSERT(set of labels resident at source, first label).
5:     Perform the SKYLINE operation on the set of labels resident at node i.
6:     for all current label cur_label resident at node i do
7:         for all arc (i, j) of node i do
8:             Create a new label new_label by referencing cur_label.
9:             if new_label is feasible then
10:                INSERT(set of labels resident at j, new_label).
11:            else
12:                DELETE(new_label).
13:            end if
14:        end for
15:    end for
16: end for
```

4.2 Memory Management

Dominance tests are only performed between labels at the same node. Therefore, it is very attractive to organize labels in the memory in a way that labels resident at the same node are close to each other to improve memory data locality. In case of large instances, the number of labels and dominance tests can be vast. If the two labels in the dominance test are located close to each other in the memory, we can accelerate memory access to take advantage of CPU caching.

At the beginning of the algorithm, every single node is associated with its own memory block. When a new label is generated, it is contained in the associated memory block of its resident node. Once the

block of a node is full, a new memory block is allocated to this node. Blocks are occasionally re-organized if necessary such that blocks associated with the same node are located close to each other. In this way, the data locality of two labels at the same node are significantly improved.

4.3 Bi-directional Search

Bi-directional dynamic programming to further reduce the number of labels. Bi-directional dynamic programming has been considered to accelerate algorithms for the classical shortest path problems and the elementary SPPRC bi-directional dynamic programming search is composed of three stages.

First is a forward pass from the source to nodes "somewhere in the middle" to generate forward labels, just as the one directional labeling algorithm we discussed; then a symmetric backward pass from the destination generates backward labels; and finally at each node with both forward and backward labels, it joins any pair of labels to form a feasible complete path.

If the searches do not stop appropriately, redundant labels will be generated and duplicate paths will be obtained many times at multiple nodes. For DAG with a topological ordering, however, we can pick any node (usually right in the middle) as a milestone to stop the path extension. Specifically, a forward label should stop extension when it reaches a node after the milestone in the topological ordering for the first time; symmetrically, a backward label stops when it reaches a node before the milestone for the first time.

Lastly, we join labels resident at only one half of the nodes, say after the milestone. The other half of the nodes should be ignored since they only provide duplicate paths.

5. Our Improvements

We can observe that forward and backward dynamic programming are independent to each other , so we can parallelize forward and backward dynamic programming. We can also see that labels can be joined paralleling , because joining of a label at a node is independent of labels on other nodes. This can help us improving run time for large instances.

6. Result

In this section , we present the results obtained from our codes. We implemented labeling algorithm in bgl, improved version of it using skyline algorithm , bi-directional dynamic programming algorithm , parallel version of bi-directional algorithm. We implemented these in c++ in windows. We used openmp library for parallelizing.

Table 1: Comparison of Generated Labels for SPPRC Instances

Instance	**BGL**	**DAG (skyline)**	**DAG (Bi-dir)**	**DAG (parallel Bi-dir)**
Instance 1	14	11	19	19
Instance 2	19	14	18	18

Table 2: Comparison in time

Instance	**BGL**	**DAG (skyline)**	**DAG (Bi-dir)**	**DAG (parallel Bi-dir)**
Instance 1	0.33302	0.1928	0.28734	0.2736
Instance 2	0.4217	0.25942	0.3246	0.35966

Notice that improvements on skyline algorithm and memory management have no impact on the number of labels. There is no difference between number of labels in "Bi-dir" and its parallel version."Bi-

Algorithm 3 Topological Ordering and Labeling Algorithm with Milestone

Require: Set of nodes, milestone
Ensure: Complete labels at each node

Perform topological sorting of nodes to get the order.
for all node i less than milestone in order **do**
    Initialize forward label with forward resource constraints, backward label with backward resource constraints.
    INSERT(set of forward labels at source, forward label).
    INSERT(set of backward labels at destination, backward label).
    Perform the SKYLINE operation on the set of forward labels at node i.
    for all forward label at i **do**
        for all arc (i, j) from node i **do**
            Create a new label *new_label* by referencing forward label.
            if *new_label* is feasible **then**
                INSERT(set of forward labels at j, *new_label*).
            else
                DELETE(*new_label*).
            end if
        end for
    end for
end for
for all node i greater than milestone in reversed order **do**
    Perform the SKYLINE operation on the set of backward labels at node i.
    for all backward label at i **do**
        for all arc (j, i) to node i **do**
            Create a new label *new_label* by referencing backward label.
            if *new_label* is feasible **then**
                INSERT(set of backward labels at j, *new_label*).
            else
                DELETE(*new_label*).
            end if
        end for
    end for
end for
for all node i greater than or equal to milestone **do**
    for all forward label at i **do**
        for all backward label at i **do**
            Create a new label *new_label* by joining forward label and backward label.
            if *new_label* is feasible **then**
                INSERT(set of complete labels at i, *new_label*).
            else
                DELETE(*new_label*).
            end if
        end for
    end for
    Perform the SKYLINE operation on the set of complete labels at node i.
end for

dir" indicates the number of labels of the bi-directional search, including forward, backward, and joined labels.

7. Conclusion

We have presented different improvements of labeling algorithm using skyline algorithm and bi-directional dynamic programming for shortest path with resource constraints in DAG. We further tried to improve it by parallelizing the bi-direction dynamic programming.

8. References

Improved Dynamic Programming for the Shortest Path Problem with Resource Constraints in DAG (2018 IEEE 4 th International Conference on Computer and Communications)