Brief Announcement: A Parallel (Δ, Γ)-Stepping Algorithm for the Constrained Shortest Path Problem

Tayebeh Bahreini
Wayne State University
tayebeh.bahreini@wayne.edu

Nathan Fisher
Wayne State University
fishern@wayne.edu

Daniel Grosu
Wayne State University
dgrosu@wayne.edu

ABSTRACT

We design a parallel algorithm for the Constrained Shortest Path (CSP) problem. The CSP problem is known to be NP-hard and there exists a pseudo-polynomial time sequential algorithm that solves it. To design the parallel algorithm, we extend the techniques used in the design of the Δ-stepping algorithm for the single-source shortest paths problem.

CCS CONCEPTS

• **Theory of computation → Shared memory algorithms**; **Shortest paths**.

KEYWORDS

constrained shortest path; delta-stepping; parallel algorithm

1 INTRODUCTION

Given a directed graph $G = (V, E)$, with non-negative *cost* c_{ij}, and non-negative *weight* ω_{ij} for all edges $(i, j) \in E$, the *Constrained Shortest Path* (CSP) problem is to find the shortest path (in terms of cost) from a source node v_1 to a destination node v_n, such that the total weight of the path is less than a given positive integer W. The CSP is an NP-hard problem, that is, it is not possible to find an optimal solution in polynomial time, unless $P = NP$. Several algorithms for solving the CSP problem are based on node labeling techniques [1, 3]. In this paper, we design a parallel (Δ, Γ)-stepping algorithm for solving the CSP problem. Several parallel Δ-stepping and radius-stepping algorithms have been developed for the single-source shortest paths problem [2]. However, none of these algorithms have been extended to solve the CSP problem. To design the parallel algorithm we extend the bucket-based approach used in the design of the Δ-stepping algorithm for the single-source shortest paths problem [4]. To the best of our knowledge, this is the first parallel algorithm for the CSP problem.

2 SEQUENTIAL (Δ, Γ)-STEPPING ALGORITHM

We extend the idea of Δ-stepping algorithm [4] developed for the single-source shortest paths problem to design an algorithm for CSP. We define a 2-dimensional array of buckets in which one dimension corresponds to the range of cost of labels and the other dimension corresponds to the range of weight of labels. Each bucket contains a set of labels with cost and weight in a range of Δ and Γ, respectively. That is, $B[j][k], \forall j \in \{1, \ldots, \lceil \frac{L}{\Delta} \rceil\}, \forall k \in \{1, \ldots, \lceil \frac{W}{\Gamma} \rceil\}$, contains a set of labels $\{a_i^l = (\pi_i^l, c_i^l, w_i^l)\}$, where π_i^l is the corresponding path from source node v_1 to node v_i and c_i^l and w_i^l are the cost and the weight of that path which are within intervals $[(j-1)\cdot\Delta, j\cdot\Delta)$ and $[(k-1)\cdot\Gamma, k\cdot\Gamma)$, respectively,

$$B[j][k] = \{(\pi_i^l, c_i^l, w_i^l) \mid ((j-1)\cdot\Delta \le c_i^l < j\cdot\Delta, (k-1)\cdot\Gamma \le w_i^l < k\cdot\Gamma\}$$

Here, L is an upper bound for the cost of the optimal path which is obtained by finding a feasible solution for the CSP problem.

Algorithm 1 shows the proposed sequential (Δ, Γ)-stepping algorithm. The aim of the algorithm is to find all non-dominated labels on every node. Then, among all the non-dominated labels on the destination node, it picks the one that has the minimum cost. The dominance relation is defined based on the weight and the cost on each label. For a given node i, let us assume the algorithm finds two labels (π_i^l, c_i^l, w_i^l) and $(\pi_i^{l'}, c_i^{l'}, w_i^{l'})$ such that $w_i^l \le w_i^{l'}$, and $c_i^l < c_i^{l'}$. Then, path $\pi_i^{l'}$ cannot be a part of the optimal solution, because we could replace it with path π_i^l which has a lower cost and a lower or equal weight. Therefore, we can disregard this path. In this case, we say that label (π_i^l, c_i^l, w_i^l) *dominates* label $(\pi_i^{l'}, c_i^{l'}, w_i^{l'})$ and denote it by $(\pi_i^l, c_i^l, w_i^l) \rhd (\pi_i^{l'} c_i^{l'}, w_i^{l'})$.

In Algorithm 1, buckets of labels are treated in increasing lexicographic order of their indices. Here the lexicographic order is defined as $(j, k) <^{\text{lex}} (j', k')$ iff $j < j'$ or $(j = j'$ and $k \le k')$, where (j, k) and (j', k') are the indices of two buckets. The labels in each bucket are treated as in the case of the label correcting approach. Let A_i be the set of labels on node i. The algorithm sets all A_i except A_1 to the empty set. Set A_1 is the set of labels of the source node v_1 and is initially set to $\{(v_1, 0, 0)\}$. It also initializes all the buckets with the empty set (except $B[1][1]$) (Lines 1-4). In each iteration of the outer loop (Lines 5-15) the labels of the first non-empty bucket that has the lexicographically smallest indices, are treated. Once all buckets have been treated, the algorithm stops and the solution is stored in $(p, cost, weight)$, where p is the optimal path and *cost* and *weight* are its corresponding cost and weight.

We group the edges of the graph into the set of *light* edges E_{light} and the set of *heavy* edges E_{heavy}. The light edges are the edges with cost and weight less than Δ and Γ, respectively. The edges that are not light are heavy edges. In each phase, i.e., each iteration of the inner while loop, the algorithm removes all labels a_i^l from the current bucket $B[j][k]$ and relaxes all light edges (i, i') out of node i (Lines 12-13). The current bucket contains labels with the cost and the weight within intervals $[(j-1)\cdot\Delta, j\cdot\Delta)$ and $[(k-1)\cdot\Gamma, k\cdot\Gamma)$, respectively. Thus, by relaxing the light edges, some new labels with the cost and the weight within intervals $[(j-1)\cdot\Delta, (j+1)\cdot\Delta)$ and $[(k-1)\cdot\Gamma, (k+1)\cdot\Gamma)$, respectively, may be added to their corresponding buckets (i.e., $B[j][k]$, $B[j][k+1]$,

SPAA '22, July 11–14, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9146-7/22/07.
https://doi.org/10.1145/3490148.3538555

Algorithm 1 Sequential (Δ, Γ)-stepping Algorithm

Input: $G(V, E)$, W, L, Δ,Γ
Output: p, $cost$, $weight$
1: $A_i \leftarrow \emptyset \quad \forall i \in V$
2: $A_1 \leftarrow \{(v_1, 0, 0)\}$
3: $B[j][k] \leftarrow \emptyset \quad \forall j \in \{1, \ldots, \lceil \frac{L}{\Delta} \rceil\}, \forall k \in \{1, \ldots, \lceil \frac{W}{\Gamma} \rceil\}$
4: $B[1][1] \leftarrow \{(v_1, 0, 0)\}$
5: **while** $\sim$**isEmpty**(B) **do**
6: $(j, k) \leftarrow \min^{\text{lex}}\{(j, k) | B[j][k] \neq \emptyset\}$
7: $R \leftarrow \emptyset$
8: **while** $B[j][k] \neq \emptyset$ **do**
9: $R \leftarrow R \cup B[j][k]$
10: $tmp \leftarrow B[j][k]$
11: $B[j][k] \leftarrow \emptyset$
12: **for each** $a_i^l \in tmp$ and $(i, i') \in E_{light}$ **do**
13: Relax$(a_i^l, i'))$
14: **for** each $a_i^l \in R$ and $(i, i') \in E_{heavy}$ **do**
15: Relax$((a_i^l, i'))$
16: **if isEmpty** (A_n) **then**
17: $(p, cost, weight) \leftarrow (\emptyset, \infty, \infty)$
18: **else**
19: $(p, cost, weight) \leftarrow$ `min-cost`(A_n)

$B[j+1][k]$, or $B[j+1][k+1]$). The labels that are added to the current bucket will be deleted in the next phase. Once the current bucket is finally empty, the algorithm relaxes the heavy edges and sequentially searches for the next nonempty bucket. Once all labels have been settled, it checks the destination node. If there is no label on node n, it returns an empty path (i.e., the problem has no feasible solution). Otherwise, it finds the label with the minimum cost on node n and stores the solution to $(p, cost, weight)$ (Lines 16-19).

The RELAX procedure is given in Algorithm 2. The procedure gets a relaxation request (a_i^l, i') as input, where a_i^l is a label on node i, and i' is the end node of edge $(i, i') \in E$, and generates a new label $(\pi_i^l | i', c_i^l + c_{ii'}, w_i^l + \omega_{ii'})$ by relaxing edge (i, i'). The algorithm adds the new label to its corresponding bucket and to the list of labels on node i' if the weight of the label is less than W, the cost of the label is less than L, and the label is not dominated by any label on node i' (Lines 1-4). Furthermore, if there is a label $a_{i'}^{l'}$ in $A_{i'}$ that is dominated by the new label, the algorithm removes it from both $A_{i'}$ and the corresponding bucket (Lines 5-7). The algorithm is not processing labels with weight greater than W, because treating these labels only leads to a source-destination path with the total weight greater than W, while the goal of CSP is to find the shortest path with the total weight not greater than W. Similarly, the algorithm does not treat labels with cost greater than L, because the paths of these labels cannot be a part of the optimal solution as their cost is greater than the upper bound L. Next, we discuss the properties of the sequential (Δ, Γ)-stepping algorithm.

Theorem 2.1. *The sequential (Δ, Γ)-stepping algorithm finds the optimal solution, if there exists a feasible solution for the problem.*

Proof. We show that if the labels of a bucket $B[j][k]$ are settled, their values are optimal and cannot be dominated by the labels obtained by treating upcoming buckets. The proof is based on the fact that the buckets are processed in increasing lexicographic order. Since the cost and the weight of the edges are non-negative, the labels of upcoming buckets cannot dominate the treated labels.

Algorithm 2 RELAX(a_i^l, i')

1: $(\bar{c}, \bar{\omega}, \bar{\pi}) \leftarrow (c_i^l + c_{ii'}, w_i^l + \omega_{ii'}, \pi_i^l | i')$
2: **if** $\bar{\omega} \leq W$ **and** $\bar{c} \leq L$ **and** $(\bar{\pi}, \bar{c}, \bar{\omega}) \rhd a_{i'}^{l'} \quad \forall l' \in A_{i'}$ **then**
3: $A_{i'} \leftarrow A_{i'} \cup \{(\bar{\pi}, \bar{c}, \bar{\omega})\}$
4: $B[\lceil \frac{\bar{c}}{\Delta} \rceil][\lceil \frac{\bar{\omega}}{\Gamma} \rceil] \leftarrow B[\lceil \frac{\bar{c}}{\Delta} \rceil][\lceil \frac{\bar{\omega}}{\Gamma} \rceil] \cup \{(\bar{\pi}, \bar{c}, \bar{\omega})\}$
5: **if** $\exists a_{i'}^{l'} \in A_{i'} \,|\, (\bar{\pi}, \bar{c}, \bar{\omega}) \rhd a_{i'}^{l'}$ **then**
6: $A_{i'} \leftarrow A_{i'} \setminus \{a_{i'}^{l'}\}$
7: $B[\lceil \frac{c_{i'}^{l'}}{\Delta} \rceil][\lceil \frac{w_{i'}^{l'}}{\Gamma} \rceil] \leftarrow B[\lceil \frac{c_{i'}^{l'}}{\Delta} \rceil][\lceil \frac{w_{i'}^{l'}}{\Gamma} \rceil] \setminus \{a_{i'}^{l'}\}$

Suppose label $a_i^l = (\pi_i^l, c_i^l, w_i^l)$ in $B[j][k]$ is dominated by treating label $a_{i'}^{l'} = (\pi_{i'}^{l'}, c_{i'}^{l'}, w_{i'}^{l'})$ in $B[j'][k']$, where $(j, k) <^{\text{lex}} (j', k')$, through an edge (i', i). As a result of $(j, k) <^{\text{lex}} (j', k')$, there are two possible cases: case I: $j < j'$; case II: $j = j', k < k'$. Here, we provide the proof for case I, but a similar proof can be obtained for case II. If $(\pi_{i'}^{l'} | i, c_{i'}^{l'} + c_{i'i}, w_{i'}^{l'} + \omega_{i'i}) \rhd a_i^l$, then $c_{i'}^{l'} + c_{i'i} \leq c_i^l < \Delta \cdot j \Longrightarrow c_{i'}^{l'} < \Delta \cdot j$. On the other hand, $\Delta \cdot (j' - 1) \leq c_{i'}^{l'}$ and since $j < j'$, we have $\Delta \cdot j \leq c_{i'}^{l'}$, which is a contradiction.

To analyze the time complexity, we define a (Δ, Γ)-*path* as a path with cost at most Δ and weight at most Γ without edge repetitions. We denote the set of pairs of nodes (i, i') that are connected through a (Δ, Γ)-path by $C_{\Delta\Gamma}$ and define $n_{\Delta\Gamma} = |C_{\Delta\Gamma}|$. Define $C_{\Delta\Gamma}^+$ as the set of triples (i, i', h) such that $(i, i') \in C_{\Delta\Gamma}$ and (i', h) is a light edge and $m_{\Delta\Gamma} = |C_{\Delta\Gamma}^+|$.

Lemma 2.2. *The time complexity of the (Δ, Γ)-stepping algorithm for relaxing the light edges is bounded by $O(W^2 \cdot m_{\Delta\Gamma})$.*

Proof. To determine the number of labels that are added to a bucket; but are deleted in the next phases, we give a mapping from the set of deleted labels into $C_{\Delta\Gamma}$. Consider a label a_i^l which is dominated by a label in phase t. There must be a most recent phase $t' \leq t$ when a_i^l was generated and was not dominated by any label. Consider a settled label (obtained in any phase) on node i with its associated path $\pi = (v_1, \ldots, v_{i'}, ..., v_i)$. Let us assume that $a_{i'}^{l'}$ is a label on node i' that is the first unsettled label on π immediately before phase t'. Hence, $a_{i'}^{l'}$ is settled in phase t' (due to the edge relaxation of settled labels in the previous phase). Since both a_i^l and $a_{i'}^{l'}$ are in the same bucket in phase t' and $a_{i'}^{l'}$ has been settled, $(v_{i'}, \ldots, v_i)$ is a (Δ, Γ)-path. Since $a_{i'}^{l'}$ becomes settled, the deletion of a_i^l in phase t can be uniquely mapped to a label $a_{i'}^{l'}$ where $(i, i') \in C_{\Delta\Gamma}$. Since there are at most W labels on each node i and i', the maximum number of deleted labels is $O(W \cdot |C_{\Delta\Gamma}|) = O(W \cdot n_{\Delta\Gamma})$. On the other hand, the number of relaxations of the dominated labels is identified based on the number of light edges coming out of dominated labels which is $O(W \cdot m_{\Delta\Gamma})$. In the RELAX procedure, testing the dominance relationship on the labels takes $O(W)$. Thus, the time complexity of relaxing the light edges is $O(W^2 \cdot m_{\Delta\Gamma})$.

Theorem 2.3. *The time complexity of the sequential (Δ, Γ)-stepping algorithm is $O\left(W \cdot (n + m) + W^2 \cdot m_{\Delta\Gamma} + \frac{L}{\Delta} \cdot \frac{W}{\Gamma}\right)$.*

Proof. The complexity of the algorithm mainly includes $O(n \cdot W)$ for the while loops, $O(\frac{L}{\Delta} \cdot \frac{W}{\Gamma})$ for scanning the nonempty buckets, $O(W^2 \cdot m_{\Delta\Gamma})$ for identifying light edges and generating labels, and $O(m \cdot W)$ for relaxing heavy edges.

3 PARALLEL (Δ,Γ)-STEPPING ALGORITHM

Algorithm 3 shows the parallel (Δ,Γ)-stepping algorithm for CSP. Here, we assume the CRCW PRAM model. The nodes with their corresponding labels are randomly assigned to the processors (Line 2). Set U_q stores all labels assigned to processor q and ind_i stores the index of the processor that is responsible for node i. Each processor q has its own buckets $B[j][k] \cap U_q \; \forall j,k$. The algorithm sequentially treats the next nonempty bucket; but performs the operations for a phase of the current bucket in parallel. In Line 4, it finds the first nonempty bucket in parallel through a reduction operation. During the treatment of bucket $B[j][k]$, each processor q treats its labels. To process each *relaxation request* (a_i^l, i') correctly, the requests must be redistributed to the processor responsible for the target node i' (which is $ind(i')$). For this purpose, we define a buffer $BUFF_q$ for each processor q. Each process stores its relaxation requests in the buffer of the processor responsible for the target nodes by using the randomized dart throwing technique [5] (Lines 11-12). Then, each processor scans its buffer and sequentially performs the relaxations (Lines 13-14). Once all light edges are relaxed, the algorithm relaxes the heavy edges (Lines 15-18).

We now determine the time complexity of Algorithm 3. For each pair of nodes $(i,i') \in C_{\Delta\Gamma}$, we call a path *efficient* if (i) its cost and its weight cannot be both improved by any other path from node i to node i'; and (ii) among all paths with the same cost and the same weight, it has the minimum length (minimum number of nodes). Let $Q_{\Delta\Gamma}$ be the set of all (Δ,Γ)- paths that are efficient and $l_{\Delta\Gamma}$ be the number of nodes of the longest path (in terms of the number of nodes) in $Q_{\Delta\Gamma}$, i.e., $l_{\Delta\Gamma} = \max\{|\pi| : \pi \in Q_{\Delta\Gamma}\}$.

Lemma 3.1. *The number of phases of the sequential (Δ,Γ)-stepping algorithm is bounded by $O(\frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot l_{\Delta\Gamma})$.*

Proof. First, we show that each bucket needs at most $l_{\Delta\Gamma}$ phases to become empty. Let us assume that label $a_i^l = (\pi_i^l, c_i^l, w_i^l) \in B[j][k]$, where $i \neq 1$ and $(\pi_i^l = (v_1, ..., v_{h_0}, v_{h_1}, ..., v_{h_r}) | h_r = i)$, is a label settled at the end of the phases of the current bucket and π_i^l is an efficient path. For more readability, we denote the corresponding label of each node v_{h_u} on the path π_l^i by a_{h_u}. We assume that labels a_{h_1} through a_{h_r} are the only labels settled by iterations of $B[j][k]$. By definition, before the first label removal from bucket $B[j][k]$, the label a_{h_0} must have been settled in a previous bucket. Hence, the edge (v_{h_0}, v_{h_1}) has been relaxed and the label a_{h_1} has already reached its final value. Thus, v_{h_1} will be settled in the first phase for bucket $B[j][k]$. Similarly, it can be concluded inductively that in phase u, v_{h_u} is settled. At each phase at least one of the labels is settled, and the cost and the weight on the path between v_{h_1} to v_{h_r} is at most Δ and Γ; thus, after at most $l_{\Delta\Gamma}$ phases, all labels are settled. Furthermore, the algorithm has to traverse $\lceil \frac{L}{\Delta} \rceil \cdot \lceil \frac{W}{\Gamma} \rceil$ buckets. Thus, the total number of phases is $O(\frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot l_{\Delta\Gamma})$.

Theorem 3.2. *The parallel (Δ,Γ)-stepping algorithm for an arbitrary graph with maximum degree d, requires $O(d \cdot \frac{L}{\Delta} \cdot \frac{W}{\Gamma} l_{\Delta\Gamma} \cdot \log(n \cdot W))$ time and $O(W \cdot (n + m + W \cdot m_{\Delta\Gamma}))$ work whp.*

Proof. (sketch) The time complexity of the algorithm mainly includes $O(\frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot \log(n \cdot W))$ for finding a globally nonempty bucket in each iteration and $O(d \cdot \frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot l_{\Delta\Gamma} \cdot \log(n \cdot W))$ for generating/assigning relaxation requests via the following.

Algorithm 3 Parallel (Δ,Γ)-Stepping Algorithm

Input: $G(V,E)$, W, L, Δ,Γ
Output: p, $cost$, $weight$

1: Execute lines 1 to 4 from Algorithm 1
2: $(\{U_q\}, \{ind_i\}) \leftarrow$ RandAssign(A_i)
3: **while** ~**isEmpty**(B) **do**
4: $(j,k) \leftarrow \min^{lex}\{(j,k) | B[j][k] \neq \emptyset\}$
5: **for each** process q **do in parallel**
6: $R_q \leftarrow \emptyset$
7: **while** $B[j][k] \cap U_q \neq \emptyset$ **do**
8: $tmp \leftarrow B[j][k] \cap U_q$
9: $R_q \leftarrow R_q \cup tmp$
10: $B[j][k] \leftarrow B[j][k] \setminus U_q$
11: **for** each $a_i^l \in tmp$ and $(i,i') \in E_{light}$ **do**
12: Throw$(a_i^l, i', ind(i'))$
13: **for** each $(a_i^l, i') \in BUFF_q$ **do**
14: Relax(a_i^l, i')
15: **for each** $a_i^l \in R_q$ and $(i,i') \in E_{heavy}$ **do**
16: Throw$(a_i^l, i', ind(i'))$
17: **for** each $(a_i^l, i') \in BUFF_q$ **do**
18: Relax(a_i^l, i')
19: **if isEmpty**(A_n) **then**
20: $(p, cost, weight) \leftarrow (\emptyset, \infty, \infty)$
21: **else**
22: $(p, cost, weight) \leftarrow$ `min-cost`(A_n)

Chernoff bounds theorem. Given h subproblems $l_1, \ldots, l_h$ of size $(0,d]$ distributed uniformly at random over p processors. Let $H = \sum_{i=1}^{h} |l_i|$. The maximum expected load $\bar{H}$ received by any processors is bounded by

$$\bar{H} = \frac{H}{p} + O\left(\sqrt{\frac{d \cdot H}{p} \log(\frac{p \cdot H}{d})} + d \cdot \log(d \cdot p \cdot H)\right)$$

We denote the number of generating/assigning requests in phase t by H_t. According to Lemma 2.2, $\cup_t H_t = O(W \cdot (n + m + W \cdot m_{\Delta\Gamma}))$. Thus, by setting $p = \frac{W \cdot (n+m+W \cdot m_{\Delta\Gamma})}{\frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot l_{\Delta\Gamma} \cdot d \cdot \log(n \cdot W)}$, the maximum load on each node over all phases is, $\bar{H} = O\left(\frac{L}{\Delta} \cdot \frac{W}{\Gamma} \cdot l_{\Delta\Gamma} \cdot d \cdot \log(n \cdot W)\right)$. In the future work, we will investigate the impacts of Δ and Γ on the algorithm time complexity on random graphs and perform an empirical analysis of our algorithm.

ACKNOWLEDGMENTS

This research was supported in part by the US National Science Foundation under grant no. IIS-1724227.

REFERENCES

[1] Martin Desrochers and François Soumis. 1988. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR: Information Systems and Operational Research* 26, 3 (1988), 191–212.
[2] Xiaojun Dong, Yan Gu, Yihan Sun, and Yunming Zhang. 2021. Efficient Stepping Algorithms and Implementations for Parallel Shortest Paths. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. 184–197.
[3] Irina Dumitrescu and Natashia Boland. 2001. Algorithms for the weight constrained shortest path problem. *International Transactions in Operational Research* 8, 1 (2001), 15–29.
[4] Ulrich Meyer and Peter Sanders. 2003. Δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms* 49, 1 (2003), 114–152.
[5] Gary L. Miller and John H. Reif. 1989. Parallel Tree Contraction Part 1: Fundamentals. *Adv. Comput. Res.* 5 (1989), 47–72.