Improved Algorithms for Detecting Negative Cost Cycles in Undirected Graphs

Xiaofeng Gu[1], Kamesh Madduri[2,⋆], K. Subramani[3,⋆⋆], and Hong-Jian Lai[1]

[1] Department of Mathematics,
West Virginia University,
Morgantown, WV
xgu@math.wvu.edu, hjlai@math.wvu.edu
[2] Computational Research Division,
Lawrence Berkeley National Laboratory,
Berkeley, CA
KMadduri@lbl.gov
[3] LDCSEE,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

Abstract. In this paper, we explore the design of algorithms for the problem of checking whether an *undirected* graph contains a negative cost cycle (UNCCD). It is known that this problem is significantly harder than the corresponding problem in directed graphs. Current approaches for solving this problem involve reducing it to either the b-matching problem or the T-join problem. The latter approach is more efficient in that it runs in $O(n^3)$ time on a graph with n vertices and m edges, while the former runs in $O(n^6)$ time. This paper shows that instances of the UNCCD problem, in which edge weights are restricted to be in the range $\{-K \cdot \cdot K\}$ can be solved in $O(n^{2.75} \cdot \log n)$ time. Our algorithm is basically a variation of the T-join approach, which exploits the existence of extremely efficient shortest path algorithms in graphs with integral positive weights. We also provide an implementation profile of the algorithms discussed.

1 Introduction

In this paper, we design and analyze algorithms for the problem of detecting a negative cost cycle in weighted, undirected graphs (UNCCD). Although UNCCD is in P, it is much harder than the corresponding problem in directed, weighted graphs. Indeed, approaches for this problem in the paper use variants of weighted matching in non-bipartite graphs. On an undirected, arbitrarily weighted graph

⋆ This work was supported in part by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

⋆⋆ This research was supported by the Air-Force Office of Scientific Research under contract FA9550-06-1-0050.

X. Deng, J.E. Hopcroft, and J. Xue (Eds.): FAW 2009, LNCS 5598, pp. 40–50, 2009.

$G = \langle V, E, c\rangle$, with n vertices and m edges, the b-matching approach runs in time $O((m+n)^3 = n^6)$, and the T-join approach runs in time $O(n^3)$. This paper focuses on the case in which edge weights are integers in the range $\{-K \cdot \cdot K\}$, where K is a fixed constant. We show that in this special case, the running time can be improved to $O(n^{2.75} \cdot \log n)$, which is an improvement over the previous bound for sparse graphs. We also provide a detailed implementation profile of the algorithms discussed in this paper.

The rest of this paper is organized as follows: Section 2 describes the preliminaries and notations that will be used in the paper. Section 3 reviews the b-matching technique and applies it to the UNCCD problem. Section 4 describes the T-join approach; this technique is then specialized to the case in which the edge weights are integers in the range $\{-K \cdot \cdot K\}$. We discuss implementation performance results in Section 5. We conclude in Section 6 by summarizing our contributions and discussing avenues for future research.

2 Preliminaries

Definition 1. *For an undirected graph $G = \langle V, E\rangle$, $|V| = n$, and a mapping $b : V(G) \to \mathbb{N}$, we say that a subgraph H of G is a perfect b-matching if each node i has exactly $b(i)$ incident arcs in H.*

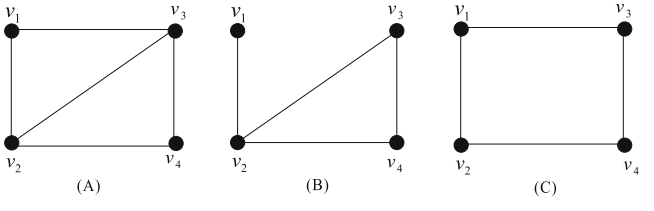


Fig. 1. Examples of perfect b-matchings

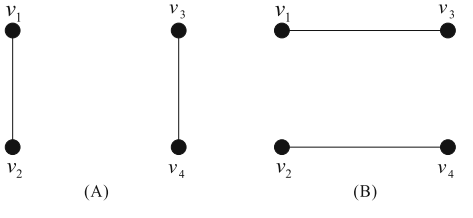


Fig. 2. Examples of perfect matchings

For example, an undirected graph G is shown in Figure 1(A). Figure 1(B) shows a perfect b-matching in G when $b : (v_1, v_2, v_3, v_4) \to (1, 3, 2, 2)$.

If $b(v) = 2 \ \forall \ v \in V(G)$, then we call it a perfect 2-matching. Figure 1(C) shows a perfect 2-matching in G. Notice that a perfect 2-matching in G is just a Hamiltonian cycle or a set of disjoint cycles that cover all the vertices in $V(G)$.

When $b(v) = 1 \ \forall \ v \in V(G)$, it is a perfect matching in G. In Figure 2, both examples (A) and (B) are perfect matchings in G.

If graph G is weighted, we may consider the problem of finding a minimum perfect b-matching in G, which is closely related to the UNCCD problem. We will show this in Section 3.

Definition 2. *Given an undirected graph $G = \langle V, E\rangle$ and a set $T \subseteq V(G)$ of even cardinality. A set $J \subseteq E(G)$ is a T-join if, $|J \cap \delta(x)|$ is odd if and only if $x \in T$, where $\delta(x)$ is the set of edges incident with x.*

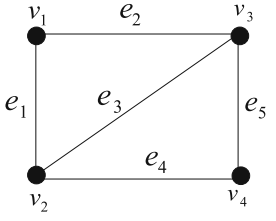


Fig. 3. Examples of T-joins

In Figure 3, we have an undirected graph G. Given $T = \{v_2, v_4\}$, $J_1 = \{e_4\}$ is a T-join in G. $J_2 = \{e_3, e_5\}$ is also a T-join. When $T = \emptyset$, $J_3 = \{e_1, e_2, e_4, e_5\}$ and $J_4 = \{e_3, e_4, e_5\}$ are two possible $\emptyset$-joins.

Notice that $J = \emptyset$ is always a $\emptyset$-join. An $\emptyset$-join is an edge set of some cycles in G or just $\emptyset$.

If graph G is weighted, we may consider the problem of finding a minimum cost T-join in G, which is also related to the UNCCD problem when $T = \emptyset$. We will show this in Section 4.

Definition 3. *Given a graph G with weights $c : E(G) \to \mathbb{R}$ and no negative cycles. The metric closure of G is a graph $\bar{G}$ with weights $\bar{c}$, where $\bar{G}$ is the simple graph on $V(G)$ that, for $x, y \in V(G)$ with $x \neq y$, contains an edge $e = (x, y)$ with weight $\bar{c}(e)$ equal to the shortest distance from x to y in G if and only if y is reachable from x in G.*

3 UNCCD Algorithm Based on *b*-matching

Algorithm 1 describes a b-matching method to solve the UNCCD problem. Before proving the correctness of Algorithm 1, some useful lemmas and theorems are introduced.

Lemma 1. *There is a perfect b-matching M_b in G' with negative cost, if and only if G contains a negative cycle C, and $c(C) = c(M_b)$.*

Proof: Notice that $b(i) = 2$, for $i = 1, 2, \ldots, n$, and there is a loop (i, i) with zero cost for each node i. So, there always exists a perfect b-matching with zero

UNCCD Algorithm based on b-matching
Input: An undirected graph G with edge costs $c : E(G) \to \mathbb{R}$.
Objective: Find a negative cycle.

1. Construct a graph G' by adding a loop (i, i) of zero cost for each node $i \in V(G)$. Construct the perfect b-matching with $b(i) = 2$ for $i = 1, 2, \ldots, n$ in the resulting graph G'.
2. Insert two additional nodes, say nodes k and l, in the middle of each arc $(i, j), i \neq j$. We set $b(k) = b(l) = 1$ and edge cost $c_{ik} = c_{lj} = c_{ij}/2$ and $c_{kl} = 0$. Let G'' denote the resulting graph.
3. Construct a third graph G''' from G'' by splitting each node i with $b(i) = 2$ into two nodes i' and i''. For each arc (i, j) in G'', we introduce two arcs (i', j) and (i'', j) with the same cost as (i, j). loop (i, i) convert to (i', i'').
4. Find a minimum perfect matching M in G'''. If $c(M) < 0$, then the original graph G has a negative cycle.

Algorithm 1. UNCCD Algorithm based on b-matching

cost. If there are no negative cost cycles in G, then the b-matching of zero cost is minimum. If there is a negative cost cycle C in G, then there must be a perfect b-matching M_b with negative cost, and $c(M_b) = c(C) + \sum_{i \notin C} c((i, i)) = c(C)$.□

Now we construct the equivalence of b-matchings in G' and G'' by

(a) (i, j) in a b-matching M'_b in $G' \Leftrightarrow (i, k)$ and (l, j) in a b-matching M''_b in G''.
(b) $(i, j) \notin M'_b \Leftrightarrow (k, l) \in M''_b$.

Lemma 2. *The equivalence between M'_b and M''_b is well-defined, and so M'_b and M''_b are equivalent.*

Lemma 3. *A perfect b-matching M''_b in $G'' \Leftrightarrow$ a perfect matching M in G'''.*

Proof: Notice that $\forall\ (p, q) \in E(G'')$, either $b(p) = 1$ or $b(q) = 1$. Each node i with $b(i) = 2$ is split into two nodes i' and i''. For each (i, j), $b(j) = 1$(because $b(i) = 2$), and so any perfect matching in G''' will contain at most one of the arcs (i', j) and (i'', j), corresponding to $(i, j) \in M''_b$. □

3.1 Correctness and Analysis

Algorithm 1 solves the undirected negative cost cycle detection problem. We can reduce this to an instance of the shortest paths problem in an undirected network [1]. From Lemma 1, 2, and 3, we know that there is a negative cost cycle in G if and only if there is a perfect matching with negative cost in G'''. So the algorithm terminates with a minimum perfect matching M in G'''. If $c(M) < 0$, there is a negative cycle in G. If $c(M) > 0$, then we declare that there is no negative cycle in G.

Theorem 1. *The UNCCD Algorithm based on b-matching can be solved in $O(n^6)$ time.*

Proof: From [2], we know that the Minimum Perfect Matching Algorithm on G''' runs in $O(|V(G''')|^3)$ time. G has n nodes and m arcs, and from the construction of G', G'', and G''', each node in G is split into two nodes in G'''. Also, for each arc in G, there are two nodes in G'''. So, $|V(G''')| = 2(m+n)$. Therefore, the running time of the algorithm is $O((m+n)^3)$. As for the simple graph, $m \leq \frac{n\cdot(n-1)}{2}$. Hence the total running time is $O(n^6)$. □

4 UNCCD Algorithm Based on *T*-join

Notice that when $T = \emptyset$, a T-join in a graph G is exactly an edge set of some cycles in G, or just $\emptyset$. We have the following lemma.

Lemma 4. *There is a negative cycle in G if and only if there exists an $\emptyset$-join with weight < 0.*

Thus, in order to determine if the graph has negative cycles, we need to find a minimum weight T-join when $T = \emptyset$.

Lemma 5. *Let G be a graph, $c : E(G) \to \mathbb{R}^+$, and $T \subseteq V(G)$ such that $|T|$ is even. Every optimum T-join in G is the disjoint union of the edge sets of $\frac{|T|}{2}$ paths, whose ends are distinct and in T.*

According to this lemma, the minimum weight T-join problem can be solved in the case of non-negative weights.

Minimum weight T-join problem with non-negative weights
Input: An undirected graph G, weight $c : E(G) \to \mathbb{R}^+$, and a set $T \subseteq V(G)$ with even cardinality.
Objective: Find a minimum weight T-join in G.

1. Solve an **All pairs shortest paths problem** in (G, c).
2. From the output of the All Pairs Shortest Paths problem, construct the **metric closure** $(\bar{G}, \bar{c})$.
3. Find a **minimum weight perfect matching** M in $(\bar{G}[T], \bar{c})$.
4. Consider the shortest x-y-path in G for each edge $(x, y) \in M$. Let J be the symmetric difference of the edge sets of all these paths. Then J is optimum.

Algorithm 2. Minimum weight T-join problem with non-negative weights

Theorem 2. *The minimum weight T-join problem with non-negative weights can be solved in $O(n^3)$ time.*

Proof: Steps 1 and 2 (which require all-pairs shortest paths) can be executed in $O(n^3)$ time. Step 3, Minimum Weight perfect matching, can be solved in $O(m' \cdot n + n^2 \cdot \log n)$ time using Gabow's algorithm [2,3], where m' is the number of edges in $(\bar{G}[T], \bar{c})$, which is $O(n^2)$. Hence, the total running time is $O(n^3)$. □

This method no longer works if we allow negative weights. However, we next show that the minimum weight T-join problem with arbitrary weights can be reduced to that with non-negative weights.

Theorem 3. *Let G be a graph with weights $c : E(G) \to \mathbb{R}$, and $T \subseteq V(G)$ with $|T|$ even. Let E^- be the set of edges with negative weights, T^- the set of vertices that have an odd number of negative weighted edges incident on them, and $d : E(G) \to \mathbb{R}^+ : d(e) = |c(e)|$.*
Then J is a minimum c-weight T-join if and only if $J \triangle E^-$ is a minimum d-weight $(T \triangle T^-)$-join, where $\triangle$ means the symmetric difference.

Proof: Please refer to [2]. □

Corollary 1. *J is a minimum c-weight $\emptyset$-join if and only if $J \triangle E^-$ is a minimum d-weight T^--join.*

Because $d \geq 0$, we can find a minimum d-weight T^--join J' through the above algorithm. According to Theorem 3, there must be a c-weight $\emptyset$-join J such that $J' = J \triangle E^-$. So $J = J' \triangle E^-$ is the minimum weight $\emptyset$-join.

UNCCD Algorithm based on T-join
Input: An undirected graph G with edge costs $c : E(G) \to \mathbb{R}$.
Objective: Find a negative cycle in G.

1. Let E^- be the set of edges with negative cost, T^- the set of vertices that are incident with an odd number of negative edges, and graph G^d with edge costs $d : E(G^d) \to R_+ : d(e) = |c(e)|$.
2. Implement Algorithm 2 on G^d with edge costs d to find the minimum T^--join J'.
3. We state $J = J' \triangle E^-$ is a $\emptyset$-join of G.
4. **if** $c(J) < 0$ **then**
5. There is a negative cycle formed by some edges in J.
6. **else**
7. There is no negative cycle in G.
8. **end if**

Algorithm 3. UNCCD Algorithm based on T-join

4.1 Correctness and Analysis

Algorithm 3 describes the method of T-join to solve the UNCCD problem. From Lemma 4, we know that to detect a negative cycle, we just need to find a $\emptyset$-join with negative cost. Then, from Theorem 3, we can reduce $\emptyset$-join with arbitrary costs in G to a T^--join with non-negative costs in G^d.

The algorithm terminates with a minimum perfect matching M in $\bar{G^d}[T^-]$. From the construction of the graph $\bar{G^d}$, each arc (x, y) in M is a shortest path from x to y in G^d. So, M corresponds to a union of edge sets (each shortest path

is an edge sets) in G. Let J be the symmetric difference of all these edge sets. Then, J' be the minimum T^--join in G^d. From Theorem 3, we know that the minimum $\emptyset$-join $J = J' \triangle E^-$ according to $J' = J \triangle E^-$.

Theorem 4. *The UNCCD Algorithm based on T-join can be solved in* $O(n^3)$ *time.*

4.2 A Special Case of UNCCD

Now we consider a special case of the UNCCD problem where integer weights belong to $\in \{-K \cdot\cdot K\}$, where K is a positive integer. According to Algorithms 2, 3 and Theorems 2, 4, we know that the running time of the T-join approach is mainly dominated by the time for all pairs shortest path problem with non-negative weights, and the minimum perfect matching problem.

Currently, $O(m \cdot n + n^2 \cdot \log n)$ is the best time bound for the shortest path problem with non-negative weights in undirected graphs, as well as the minimum perfect matching problem. When the edge weights are integers, however, we can use faster algorithms.

For all pairs shortest paths problem with integer weights in undirected graphs, an $\tilde{O}(N \cdot n^{2.38})$ time bound algorithm was described by Shoshan and Zwick in [4], where $\tilde{O}(f(n))$ is the abbreviation for $O(f(n) \cdot (\log n)^t)$ for some constant t and N is the largest edge weight in the graph.

As for minimum perfect matching with integer weights, Gabow presents an $O(n^{\frac{3}{4}} m \log N)$ in [5] by scaling techniques, where N is the largest edge weight in the graph.

So, we can still use the T-join approach. The only difference is that now edge weights are integers ranging in $\{-K \cdot\cdot K\}$. The algorithm is described as follows.

Theorem 5. *UNCCD problem with integer weights* $\in \{-K \cdot\cdot K\}$ *can be solved in* $O(n^{2.75} \cdot \log n)$ *time.*

Proof: In Algorithm 4, step 2, all pairs shortest path problem in (G, d) can be computed in $\tilde{O}(K \cdot n^{2.38}) = O(K \cdot n^{2.38} \cdot (\log n)^t)$ time for a constant t. Step 3, minimum perfect matching problem in $(\bar{G}[T^-], \bar{d})$ needs $O(n^{\frac{3}{4}} \cdot m' \cdot \log N)$ time, where N is the largest edge weight in $(\bar{G}[T^-], \bar{d})$ and m' is the number of edges in $(\bar{G}[T^-], \bar{d})$, which is $O(n^2)$. We know that an edge in $(\bar{G}[T^-], \bar{d})$ corresponds to a shortest path in (G, d), where $d = |c| \leq K$ and K is a positive integer constant. The number of edges in any shortest path in (G, d) is not more than n, so $N \leq nK$. So the total running time is $O(K \cdot n^{2.38} \cdot (\log n)^t + n^{\frac{3}{4}} \cdot n^2 \cdot \log nK)$.

$\lim_{n \to \infty} \frac{K \cdot n^{2.38} \cdot (\log n)^t}{n^{\frac{3}{4}} \cdot n^2 \cdot \log nK} = 0$, and hence Algorithm 4 can be solved in $O(n^{2.75} \cdot \log n)$ time. □

5 Implementation

To empirically evaluate the performance of the T-join approach, we implement Algorithm 4 and experiment with networks from several different graph families.

UNCCD Algorithm with integer weights $\in \{-K \cdot\cdot K\}$
Input: An undirected graph G with edge weights $c : E(G) \to \mathbb{Z} \cap \{-K \cdot\cdot K\}$.
Objective: Find a negative cycle in (G, c).

1. Let E^- be the set of edges with negative cost, T^- the set of vertices that are incident with an odd number of negative edges, and graph (G, d) with edge costs $d(e) = |c(e)|$.
2. Solve an **All Pairs Shortest Paths Problem** in (G, d).
3. Let $(\bar{G}, \bar{d})$ denote the metric closure of (G, d) and $(\bar{G}[T^-], \bar{d})$ denote the subgraph of $(\bar{G}, \bar{d})$ induced by T^-.
4. Find a minimum weight perfect matching M in $(\bar{G}[T^-], \bar{d})$.
5. We consider the shortest x-y-path in (G, d) for each edge $(x, y) \in M$. Let J' be the symmetric difference of the edge sets of all these paths. Then J' is a minimum T^--join in (G, d).
6. We state $J = J' \triangle E^-$ is a $\emptyset$-join of (G, c).
7. **if** $c(J) < 0$ **then**
8. There is a negative cycle formed by some edges in J.
9. **else**
10. There is no negative cycle in (G, c).
11. **end if**

Algorithm 4. UNCCD Algorithm with integer weights $\in \{-K \cdot\cdot K\}$

We report performance on a 2.4 GHz Intel Xeon Linux system with 4 MB L2 cache and 8 GB DRAM memory. We build our code with the GNU C compiler version 4.2.4 and the optimization flags `-O3 -funroll-loops`.

Observe that the overall UNCCD running time is dominated by two steps: all pairs shortest paths (APSP) on the original graph, and minimum weight perfect matching (MWPM) on an induced subgraph of the graph closure. We reformulate the APSP computation as running n single-source shortest paths (SSSP), and use a fast SSSP implementation from the 9th DIMACS Challenge on shortest paths [6]. For MWPM, we modify an implementation of Gabow's algorithm [7], the source code for which is freely available online [8].

We experiment with several graph families used in the DIMACS Implementation Challenge for the single-source shortest paths problem with arbitrary edge weights [6]. Specifically, we give performance results for four different families in this paper:

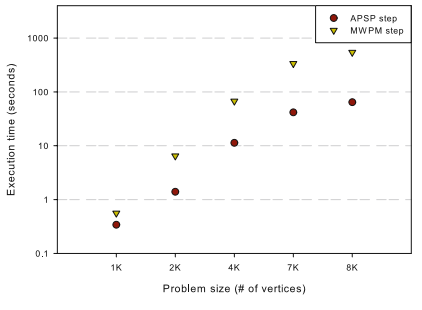
- `LMesh`: We generate synthetic two-dimensional meshes with grid dimensions x and y. $x = \frac{n}{10}$ and $y = 10$ for our experiments. The graphs are sparse with $m \approx 2n$. The diameter of these graphs is $O(n)$.
- `SqMesh`: Square meshes are fully-connected grids with $x = y = \sqrt{n}$. The graphs are sparse with $m \approx 2n$. The diameter of these graphs is $O(\sqrt{n})$.
- `SpRand`: We generate graphs according to the Erdos-Renyi random graph model, and ensure that the graph is connected. The ratio of the number of edges to the number of vertices is set to 10 to generate sparse graphs, and

K is set to $\sqrt{n}$. Random graphs have a low diameter and a Gaussian degree distribution.

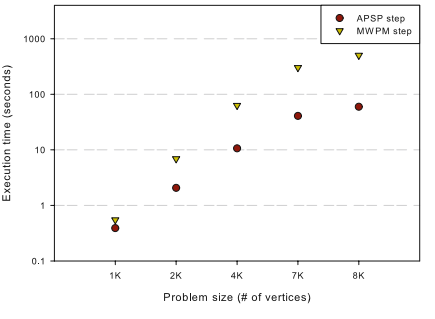
– `DRand`: We generate dense random graphs by increasing the ratio of edges to vertices in the above family. It is set to 100 (or $\sqrt{n}$ for the case of $n = 10000$.)

Figure 4 plots the execution time of the APSP and MWPM steps in the algorithm for various problem instances from these four families. The number of vertices is varied from $n = 1000$ to $n = 8000$ in our study. The average memory footprint is larger than the L2 cache size for all the problem instances, and the largest case uses 20% of the main memory. The execution time on the Y-axis is depicted in the logarithmic scale, as the time grows quadratically with the problem size. We observe that in general, the APSP step is faster than MWPM for large problem instances. The cross-over point varies depending on the problem size and sparsity. For instance, for the dense random graphs, MWPM starts to dominate only for $n > 4000$. We also observe that the running times of both the steps do not vary much across graph families, i.e., the running time does not seem to be sensitive to the graph topology.

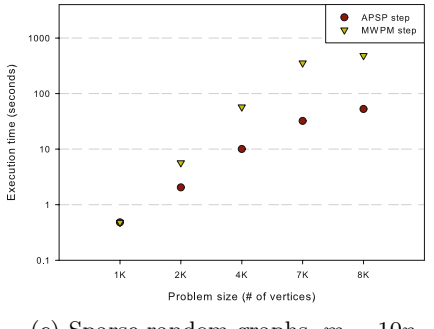
The reported running times relate closely with the asymptotic complexity bounds. We performed a linear regression analysis on the data given in Figure 4,



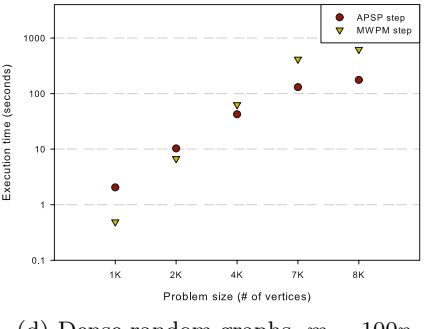
(a) Long mesh graphs ($\frac{n}{10} \times 10$)



(b) Square mesh graphs ($\sqrt{n} \times \sqrt{n}$)

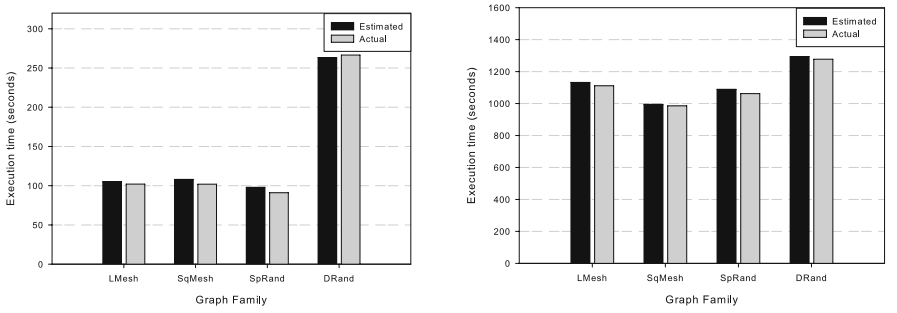


(c) Sparse random graphs, $m = 10n$



(d) Dense random graphs, $m = 100n$

Fig. 4. Execution time of the APSP and MWPM steps for various graph instances



(a) All-Pairs Shortest Paths (APSP) (b) Minimum Weight Perfect Matching (MWPM)

Fig. 5. The actual and expected (from regression analysis) execution times for the APSP and MWPM steps in the T-join algorithm, on a graph of 10,000 vertices. m is determined by the graph family, and $K = 1000$.

to compute the additive and multiplicative constants for the $m \cdot n$ complexity term, separately for the APSP and the MWPM steps. With these values, we can predict the running times for larger graph instances. Figure 5 plots the predicted (from the data obtained through regression analysis) and actual execution times for four large-scale instances from different graph families. The number of vertices is set to 10,000 in all four cases, and the number of edges is dependent on the specific graph family (see above discussion). We plot the running times for AWSP and MWPM separately. We observe that the actual running times are only slightly lower than the predicted values (the difference is less than 5% in most cases), confirming that we realize the asymptotic bounds in practice.

6 Conclusion

In this paper, we specialized the T-join approach for solving the UNCCD problem to the case in which the edge weights are integers in $\{-K \cdot \cdot K\}$, where K is a fixed constant. We were able to show that in this special case, the existing time bound of $O(n^3)$ could be improved to $O(n^{2.75} \cdot \log n)$. Our result is particularly relevant for sparse graphs. We also provided a detailed implementation profile of the algorithms discussed in this paper.

There are several open issues that will be addressed in future work:

(i) Improving the running time to $O(m \cdot n)$: Our improved algorithm depends on the existence of a fast all-pairs shortest path algorithm for fixed integral weights. It would be interesting to see if we can avoid the shortest paths approach altogether in order to obtain an $O(m \cdot n)$ time bound for the case of arbitrarily weighted graphs.

(ii) Parameterizing the running time in terms of edge weights: For the case of directed graphs, there exist a number of negative cycle detection algorithms

in the literature whose running times are parameterized by edge weights, for instance [9,10]. At this juncture, it is unknown whether similar algorithms exist for UNCCD.

(iii) Producing short certificates of non-existence: Recent work in algorithm design has increasingly emphasized the role the of certifying algorithms [11,12]. The idea here is that the algorithm provides a witness to its output, which is easily certifiable. In case of UNCCD, for a "yes" instance, a negative cost cycle can serve as a certificate. It is not clear what an easily verifiable certificate is, for "no" instances.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice-Hall, Englewood Cliffs (1993)
2. Korte, B., Vygen, J.: Combinatorial Optimization. Algorithms and Combinatorics, vol. 21. Springer, New York (2000)
3. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Association for Computing Machinery, pp. 434–443 (1990)
4. Shoshan, A., Zwick, U.: All pairs shortest paths in undirected graphs with integer weights. In: FOCS, pp. 605–615 (1999)
5. Gabow, H.N.: A scaling algorithm for weighted matching on general graphs. In: Proceedings 26th Annual Symposium of the Foundations of Computer Science, pp. 90–100. IEEE Computer Society Press, Los Alamitos (1985)
6. Demetrescu, C., Goldberg, A., Johnson, D.: 9th DIMACS implementation challenge – Shortest Paths (2005), `http://www.dis.uniroma1.it/~challenge9/`
7. Gabow, H.: Implementation of algorithms for maximum matching on non-bipartite graphs. PhD thesis, Stanford University (1974)
8. Rothberg, E.: Implementation of H. Gabow's weighted matching algorithm (1992), `ftp://dimacs.rutgers.edu/pub/netflow/matching/weighted/`
9. Goldberg, A.V.: Scaling algorithms for the shortest paths problem. SIAM Journal on Computing 24, 494–504 (1995)
10. Goldberg, A.: Shortest path algorithms: Engineering aspects. In: ISAAC: 12th International Symposium on Algorithms and Computation, pp. 502–513 (2001)
11. Guo, L., Mukhopadhyay, S., Cukic, B.: Does your result checker really check? In: Dependable Systems and Networks, pp. 399–404 (2004)
12. Kratsch, D., McConnell, R.M., Mehlhorn, K., Spinrad, J.: Certifying algorithms for recognizing interval graphs and permutation graphs. In: SODA, pp. 158–167 (2003)