Optimizing in graphs with expensive computation of edge weights

F. Noé[1], M. Oswald[2], and G. Reinelt[2]

[1] DFG Research Center "Matheon", FU Berlin, Arnimallee 6, 14195 Berlin, Germany, noe@math.fu-berlin.de,

[2] University of Heidelberg, Institute for Computer Science, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

1 Introduction

Much research has gone into the development of fast graph optimization algorithms and many problems, such as shortest-path [1] or minimum-cut [2], can now be routinely solved for large graphs. In many practical applications, *e.g.* in bioinformatics or computational chemistry [3, 4], the solution of graph optimization problems is very important, but hampered by the fact that the graph is not completely known. Especially edge weights, which may represent, for example, reaction rates in a reactive network, are often unknown or only known within some error bounds. Usually, methods for determining these edge weights are available, but the precise determination of a single edge weight may require long computations or expensive experiments. The objective is thus not to minimize the runtime of the graph optimization problem with a given set of edge weights, but instead to minimize the number of edge weights that need to be determined in order to be able to solve the graph optimization problem.

To our knowledge, this problem has so far not been addressed in operation research literature. In the present paper we present a simple heuristic for solving the graph problem while trying to compute exact weights only for few edges and to avoid determining the weights of other edges which have no or little impact on the problem. The algorithms are applied to shortest paths and minimum cuts in biomolecular reaction networks and the results demonstrate that only few edge weights need to be determined in order to solve these graph optimization problems. This encourages further research in this area.

In the following, let $G = (V, E, c)$ denote a graph with node set V, edge set E and a vector of edge weights c. The weight of an edge $e = uv$ is denoted by c_e or c_{uv}. In a general problem setting we have a set $\mathcal{I} \subseteq 2^E$ of feasible solutions and an objective function $f : 2^E \to \mathbb{R}$. The optimization problem consists of finding $I^* \in I$ such that $f(I^*) = \min\{f(I) \mid I \in \mathcal{I}\}$. We denote by $f^*(G)$ the optimum objective function value and by $\mathrm{OPT}(G)$ an optimal edge set. In this paper we require that the objective function f has the following *edge-monotonous property*: if the weight c_e of one edge $e \in E$ is increased, then f remains the same or increases. If c_e is decreased, then f remains the same or decreases. We start with no exact edge weights being given, but instead some finite lower and upper bounds l_e and u_e are available for every edge e. Correspondingly we can define the two graphs $G_l = (V, E, l)$ and $G_u = (V, E, u)$. Due to the monotonicity property, $f^*(G_l) \leq f^*(G) \leq f^*(G_u)$. Furthermore, we are given some means for refining bounds by replacing l_e and u_e of a given edge e by new bounds l'_e and u'_e such that $l'_e \geq l_e$ and $u'_e \leq u_e$.

2 Algorithms

2.1 Basic Algorithm

From the given lower and upper bounds on the edge weights of $G = (V, E, c)$ we can construct the two graphs $G_l = (V, E, l)$ and $G_u = (V, E, u)$. We assume that we have an algorithm to compute $\mathrm{OPT}(G_l)$ and $\mathrm{OPT}(G_u)$ and a method for refining edge weight bounds as described above. The following iterative algorithm determines $\mathrm{OPT}(G)$.

Algorithm 1 Compute $\mathrm{OPT}(G)$

```
(1) Compute $\mathrm{OPT}(G_l)$ and $\mathrm{OPT}(G_u)$. Output $f^*(G_u)$ and $f^*(G_l)$.
(2) If $f^*(G_u) - f^*(G_l) \leq \Delta$, then return($\mathrm{OPT}(G_l)$).
(3) Select an undetermined edge $h \in \mathrm{OPT}(G_l)$ with maximum weight $l_h = \max\{l_e \mid e \in \mathrm{OPT}(G_l) \text{ and } l_e < u_e\}$. (We call such an edge a critical edge). Refine $h$ and goto (1).
```

Theorem 1. *Assuming that at most n_{refine} edge refinements are necessary to exactly determine its weight ($l_e = u_e$), then, for $\Delta = 0$, Algorithm 1 computes $OPT(G)$ in finitely many steps.*

Proof. Edges with determined weight are never selected as critical edges in step (3). Therefore each edge can only be determined once. The algorithm terminates in step (2) at the latest when all edges are determined. (Although it is expected to terminate earlier.) Thus at most $|E| \cdot n_{\text{refine}}$ edge refinements are required.
The correctness of the algorithm follows immediately from the fact that $f^*(G_l) \leq f^*(G) \leq f^*(G_u)$ always holds. Thus the theorem is proved. □

For $\Delta > 0$ the algorithm is no longer exact, but will return an approximate solution I with $f^*(I) \leq f^*(G) + \Delta$. In many practical cases, such an approximate solution may be sufficient and save considerable amounts of CPU time.

2.2 Parallelization

Algorithm 1 can be parallelized in a rather straightforward way. To establish a communication between the individual processes, a "database" of bounds is required which every process has read and write access to. The database contains the vectors L and U of the current lower and upper edge weight bounds, and an edge is marked "busy" when a process is about to change its weight. Every processor keeps own graphs G_l and G_u and executes algorithm 2.

Algorithm 2 Parallel computation of OPT(G)

```
(1) Let $F = \emptyset$. ($F$ is a set of edges with estimated weights.)
(2) Remove every member $e$ from $F$ that is not *busy*.
(3) Update $l_e = L_e$ and $u_e = U_e$, for all edges $e \in E$.
(4) Compute OPT($G_l$) and OPT($G_u$).
(5) If all edge weights of OPT($G_l$) are determined, *i.e.*, $l_e = u_e$, for all $e \in$ OPT($G_l$), then
    (5.1) If $F = \emptyset$ return(OPT($G_l$)), otherwise wait for some time interval $\tau$.
(6) Select an undetermined edge $h$ with maximum weight from OPT($G_l$). If no such edge exists, goto 2.
(7) Distinguish the following cases:
    (7.1) If $h$ is *busy*, then assign a hypothetical edge weight to $h$ and set $F = F \cup \{h\}$.
    (7.2) If $h$ is not *busy*, then mark $h$ as *busy*, refine $h$, set $U_h := u_h$, $L_h := l_h$, and mark $h$ as not *busy*.
    Then goto 2
```

Theorem 2. *The parallel algorithm computes $OPT(G)$ in finite time.*

Proof. Assume that each edge weight refinement takes finite time. When iterating the loop (2)–(7), one edge weight is refined in each iteration and since each edge is determined only once in a given process, this will terminate after at most $|E| \cdot n_{\text{refine}}$ cycles. Loop (2)–(6) is iterated only if all $e \in \text{OPT}(G_l)$ are determined. If $F = \emptyset$, then the algorithm will terminate in the next iteration in step (5). If $F \neq \emptyset$, then the process will wait in (5) and other processes are currently in step (6), determining the edges $e \in \text{OPT}(G_l)$ which are also in F. As these other process will finish the determination in finite time and afterwards update L and U, eventually $F = \emptyset$ and the process terminates in the next iteration in (5). □

Only in the first part of step (7) possibly incorrect edge weights are assigned and added to F. They are only removed from F in step (2) if their true values have been determined. Thus, $F = \emptyset$ only if all weight bounds L and U are correct, and only in this case the algorithm returns. Thus algorithm 2 produces the same result as Algorithm 1 and the theorem is proven. □

3 Applications to molecular transition networks

We describe an application of shortest paths in the computation of the dynamics of biomolecules where it is very expensive to obtain exact edge weights [3, 4], typically requiring minutes to hours of CPU time for each edge weight. Biomolecules, such as proteins, undergo transitions between metastable "end-states" which correspond to different atomic coordinates and have different biological functions. For example, `Ras p21` is a cell signaling protein which exists in an *active* state that promotes cell growth and an *inactive* state that inhibits cell growth. These states are "connected" via intermediate states which are typically short-lived and have no particular biological function.

In a *transition network*, a state is modeled as a vertex $v \in V$, and a possible transition between a pair of states is modeled by an (undirected) edge $uv \in E$.

3.1 Shortest paths

When the mean residence times are used as edge weights in a transition network, the shortest paths between two given vertices u and v represent the most populated transition pathways for the molecule to change between the two associated structures [4].

Using the algorithms presented in Section 2, we have computed the best paths of four different transitions in biomolecules while determining only a small number of edge weights: the pathways for the $\alpha_L \rightleftharpoons \beta$, $\beta \rightleftharpoons \alpha_R$ and $\alpha_L \rightleftharpoons \alpha_R$ transitions in the $\texttt{Ala}_8$ peptide and the active$\rightleftharpoons$inactive transition in the `Ras p21` molecular switch. Detailed descriptions of these molecular systems can be found in [4, 3]. For computing the shortest paths for a given set of edge weights, Dijkstra's algorithm was employed [1].

We first used trivial initial edge weight bounds (0 and ∞). For the $\texttt{Ala}_8$ and `Ras p21` networks the number of edge weights required to be computes are up to three orders of magnitude below $|E|$. Table 1 displays in the second column the number of edges of the networks and in the third column the actual number of refinement steps of Algorithm 1.

	$\lvert E\rvert$	n_{ec}, normal	n_{ec}, highest weight
$\texttt{Ala}_8$, $\alpha_L \rightleftharpoons \beta$	772420	870	63
$\texttt{Ala}_8$, $\beta \rightleftharpoons \alpha_R$	772420	865	450
$\texttt{Ala}_8$, $\alpha_L \rightleftharpoons \alpha_R$	772420	1016	71
`Ras p21`, active$\rightleftharpoons$inactive	47404	2252	n/a

Table 1. Number of determined edge weights for shortest path computation

We have also used the algorithm in such a way as to provide an approximate result for the shortest path, with Δ being small enough so that at least the highest edge weight along the shortest path was unambiguously identified. The highest edge weight is biologically the most interesting one, as it provides the molecular structure corresponding to the bottleneck of the transition. The results shown in the fourth column of Table 1 are very encouraging. The numbers of edge weights required are three to four orders of magnitude less than $|E|$. With these savings, the times required for the best-path calculations are reduced from several CPU years to a few CPU days.

3.2 Minimum cuts

When the inverse mean residence times are used as edge weights in a transition network, the minimum (s,t)-cut is of special relevance as it yields the set of edges corresponding to the slowest, or rate-limiting, part of the transition connecting vertices s and t, also known as *transition state ensemble* [3].

We have computed the minimum cut for the `Ras p21` transition network using Algorithm 1. For the computation of a minimum (s, t)-cut with given edge weights the algorithm of Nagamochi and Ibaraki was employed. [2]. This minimum cut, which consists of 174 edges, required the computation of 1092 out of a total of $|E| = 47404$ edges. When choosing Δ such that only the highest-weighted edge was computed, $n_{ec} = 805$ edge computations were required, thus reducing the required CPU time to about 5% compared to the trivial solution.

4 Conclusions

In the present paper we have investigated the problem that an optimum solution $\mathrm{OPT}(G)$ needs to be computed for some edge weighted graph G, where initially the weights are not available and can only be obtained at high cost. This is different from the usual setting where complete information is given and the fastest optimization algorithm is sought for.

We have presented a serial and parallel version of a simple heuristic approach and shown that it is very successful for analyzing molecular dynamics using transition networks and that only a very small number of the edge weights need to be known exactly. The approach has reduced the computer time necessary to perform the graph optimization from several CPU years to a few days, which facilitates calculations that would otherwise be out of reach.

These results suggest that graph optimizations in the case where edge weights are not (fully) known is a worthwhile field for further research that may benefit many application areas.

Acknowledgements

F.N. kindly acknowledges funding from the DFG center Matheon and Landesstiftung Baden-Württemberg

References

1. E. Dijkstra. A note on two problems in connexion with graphs. *Num. Math.*, 1:269–271, 1959.
2. H. Nagamochi and T. Ibaraki. Computing edge connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5:54–66, 1992.

3. F. Noé and D. Krachtus and J. C. Smith and S. Fischer. Transition networks for the comprehensive characterization of complex conformational change in proteins. *J. Chem. Theory and Comput.*, 2:840–857, 2006.
4. F. Noé, M. Oswald, G. Reinelt, S. Fischer, and J. C. Smith. Computing Best Transition Pathways in High-Dimensional Dynamical Systems: Application to the $\alpha_L \leftrightharpoons \beta \leftrightharpoons \alpha_R$ Transitions in Octaalanine. *Multiscale Model. Sim.*, 5:393–419, 2006.