The Number of Link-Disjoint Paths in a Graph

Piet Van Mieghem* Eguzki Astiz Lezaun

Delft University of Technology
January 2006

Abstract

The computation of the number of link-disjoint paths in an optical network is addressed. We present an exact algorithm MAXFLOW. Due to the excessive worst-case computational complexity of MAXFLOW, analytic bounds are derived which are based on prime number theory. We regard these results as a first step to estimate the capacity of optical networks expressed in the number of link-disjoint paths.

1 Introduction

The presented work is motivated by the initially thought simple question: "How many optical connections with a certain hopcount can be established in a given network?" A simplified version of the question leads us to consider a network in which each link consists of one optical fibre. The number of light paths between a given set of source-destination pairs that can be set-up with a certain hopcount – number of links traversed – is the major subject of this article. Roughly speaking, we may consider the problem as related to the computation of the capacity of a network, in the sense that each link can only be used by one connection or light path.

Link-disjoint paths between two nodes are paths that do not have any link in common. Link-disjointness is represented by the symbol "$\|$". Thus, if $\mathcal{P}_1$ and $\mathcal{P}_2$ are two arbitrary paths,

$$\mathcal{P}_1 \texttt{ and } \mathcal{P}_2 \texttt{ are disjoint } \Leftrightarrow \mathcal{P}_1 \| \mathcal{P}_2$$

We first present an exact algorithm, MAXFLOW, that computes all possible link-disjoint paths in a given set $\mathcal{Q}$ of paths. The difficult nature of the problem is shown to lead to an infeasibly high computational complexity, even for relatively small graphs with a few tenths of nodes. A second part of the article gives analytic bounds to the number of link-disjoint paths M_h with fixed hopcount between two nodes. Our main result is Theorem 1 in which prime number theory pops up. To some extent, the problem we deal with, is a generalization of the well-known max flow-min cut problem, that has tight upperbounds as shown by Goemans and Williamson [2].

*email: P.VanMieghem@ewi.tudelft.nl

2 The MAXFLOW algorithm

2.1 Functionality

The MAXFLOW algorithm finds all M link-disjoint paths between any pair (u, v) of nodes in a graph G. The basic identity in MAXFLOW is a path, a sequence of consecutive links in a graph. As input, MAXFLOW assumes a given set of n paths $\mathcal{Q}$,

$$\mathcal{Q} = \{\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_n\}$$

An arbitrary number n of paths can be given. We define the set, $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_m\}$, whose elements $\mathcal{T}_i$ are mutually link-disjoint paths belonging to the set $\mathcal{Q}$,

$$\mathcal{T} = \{\mathcal{T}_i : \mathcal{T}_i \in \mathcal{Q} \text{ and } \mathcal{T}_i \| \mathcal{T}_j\}_{\forall i \neq j \in [1,m]}$$

The cardinality – the number of elements – of the set $\mathcal{T}$ is denoted as $|\mathcal{T}| = m$. Obviously, the set $\mathcal{T}$ is not unique. We define $\mathcal{D}$ as the set of *all* sets with the same properties as $\mathcal{T}$. Thus, $\mathcal{D}$ is the set of all link-disjoint paths of $\mathcal{Q}$. With this preparation, we are ready to formulate the MAXFLOW algorithm. Given $\mathcal{Q}$, MAXFLOW computes the set of paths $\mathcal{P}$, with the following properties:

1. $\mathcal{P} \in \mathcal{D}$
2. $\forall \mathcal{T} \in \mathcal{D} : |\mathcal{T}| \leq |\mathcal{P}| = M$

If the given set of paths $\mathcal{Q}$ contains all the paths between one certain source-destination pair (u, v), MAXFLOW will then compute the maximum number M of link-disjoint paths between *one* pair (u, v) of nodes. Depth-first search (see e.g. [1]) computes all possible paths between two nodes in a graph. If we appropriately select the set $\mathcal{Q}$ of paths between more source-destination pairs, MAXFLOW returns the maximum set of link-disjoint paths between more source-destination pairs. Furthermore, we can impose that all paths of $\mathcal{Q}$ have the same number of links or hopcount. Then, MAXFLOW computes the maximum set of link-disjoint, equi-hopcount paths.

2.2 Backtracking Technique

The MAXFLOW algorithm is based on the *backtracking* technique [1]. *Backtracking* finds a solution by trying one of several choices. If the choice proves incorrect, the computation backtracks or restarts at the point of choice and tries another choice. It is often convenient to maintain choice points and alternate choices using recursion. In our particular case, MAXFLOW uses backtracking by applying the *depth-first search* on the tree of choices (see subsection 2.3). Since our tree of choices is likely to be wide but not deep, the *depth-first search* technique is more convenient than *breadth-first search* technique [1].

2.3 Search Tree

The *Search Tree* is also commonly known as the *Tree of Choices*. In general, the *Search Tree* may represent all the possible results if it is properly designed. The backtracking algorithm will search in the *Search Tree*. The *Search Tree* is designed by MAXFLOW itself together with the search method,

i.e. *depth-first search*. The remainder of this section will describe the particular *Search Tree* of the MAXFLOW problem.

Every vertex of the MAXFLOW *Search Tree* contains one particular set of link-disjoint paths that the algorithm computes during the execution process. Vertices are located at levels depending on the recursion degree and edges only exist between adjacent level vertices (property of a tree). The starting vertex is the root vertex, $\mathcal{R}$, which contains the initial set $\mathcal{Q}$ of paths. For every path $\mathcal{Q}_i$ contained in the vertex $\mathcal{R}$ we compute the set of paths that are disjoint to $\mathcal{Q}_i$. This new set of paths, namely $\mathcal{T}_{Q_i}$ is placed in a vertex on the next level in the search tree. The edge, labelled as $\mathcal{Q}_i$, that connects vertices $\mathcal{Q}$ and $\mathcal{T}_{Q_i}$ is associated to the path $\mathcal{Q}_i$. All the paths contained in the set $\mathcal{T}_{Q_i}$ are disjoint to $\mathcal{Q}_i$, but they might not be mutually link-disjoint. For example, let $\mathcal{T}$ be a set of $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ paths and let $\mathcal{T}_2$ and $\mathcal{T}_3$ share one link and both be link-disjoint to $\mathcal{T}_1$. The resulting vertex, link-disjoint to $\mathcal{T}_1$, is $\{\mathcal{T}_2, \mathcal{T}_3\}$ where $\mathcal{T}_2$ and $\mathcal{T}_3$ are link-disjoint to $\mathcal{T}_1$, but they are not disjoint to each other. Further, $\mathcal{T}_{Q_i}$ only contains elements of $\mathcal{Q}$ with an index higher than i,

$$\mathcal{T}_{Q_i} = \{\mathcal{Q}_j : \mathcal{Q}_j \in \mathcal{Q} \text{ and } \mathcal{Q}_j \| \mathcal{Q}_i\}_{1 \leq i < j \leq n}$$

and, obviously,

$$\mathcal{T}_{Q_n} = \{\emptyset\}$$

Hence, $\mathcal{Q}_1$ does not appear in the set $\mathcal{T}_{Q_3}$, even though $\mathcal{Q}_1 \| \mathcal{Q}_3$ but, for this particular case, $\mathcal{Q}_3 \in \mathcal{T}_{Q_1}$. Using this tree building process, MAXFLOW "walks" over all the edges looking for the longest branch in terms of edges. The paths associated to consecutive edges of the *Search Tree* are disjoint by definition. Thus, the longest branch forms the largest set of mutually link disjoint paths.

Figure 1 shows an example of a *Search Tree*. From a certain level i, the vertices $\mathcal{T}_{P_1}$ and $\mathcal{T}_{P_5}$ on level $i+1$ are computed. Vertex $\mathcal{T}_{P_1}$ contains all the paths of the ancestor that are disjoint to P_1 and, likewise, the vertex $\mathcal{T}_{P_5}$ contains all paths link-disjoint to the path P_5. Even though paths P_1 and P_5 are link-disjoint because P_5 is contained in vertex $\mathcal{T}_{P_1}$ on level $i+1$, the path P_1 is not included in vertex $\mathcal{T}_{P_5}$.

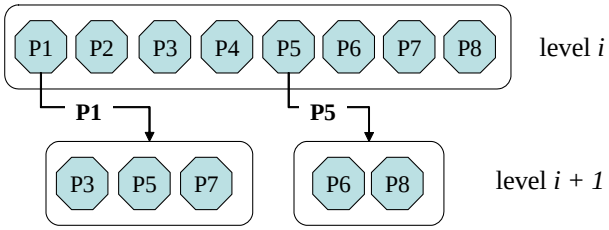


Figure 1: Construction of level $i+1$ the choice tree.

An edge in the *Search Tree* that connects two levels implies that there exists at least one pair of paths that are disjoint to each other. There are two extreme cases. First, the *Search Tree* is just a single branch with $|\mathcal{Q}|$ levels when all the paths from the $\mathcal{Q}$ set are mutually disjoint. Second, if all the paths share at least one certain link, which means that there is not a single pair of paths that are disjoint, the *Search Tree* turns out to be a single root vertex.

Branch prunning is used in backtracking to improve the efficiency. Obviously, prunning a branch saves time because there is no longer need to check/search the branch. A bounding function estimates if a search over a certain branch can improve the already obtained solution. If the estimate is not positive, the algorithm prunes that branch avoiding useless walks over the child branches.

The boundary function in MAXFLOW checks two numbers: the cardinality α of the resulting set to be returned and the number β of the remaining number of paths to be checked in the current vertex. If the current vertex has n paths and if the path $\mathcal{Q}_i$ for $1 \leq i \leq n$ is currently being checked, then $\beta = n - i$. Indeed, if $\alpha \geq \beta$, then the remaining β paths are insufficient to compute a set that contains more that α disjoint paths; the maximum achievable is β. The bounding function is recursively applied at every vertex of the *Search Tree*.

2.4 Source Code

The meta-code of MAXFLOW is given below:

Algorithm: $MAXFLOW(\mathcal{Q})$
Input: the set of paths $\mathcal{Q} = \{\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_n\}$
Output: the set of paths $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_m\}_{1 \leq m \leq n}$ belonging to $\mathcal{Q}$, whose elements are mutually link-disjoint paths. There does not exist another set containing more elements than $\mathcal{P}$.

```
01  FUNCTION MAXFLOW(Q): Set of paths)  RETURN P:Set of Paths
02  QTMP    : Set of Paths;
03  QNEW    : Set of Paths;
04
05   P ← ∅;
06  WHILE({|P| < |Q|} ∧ {Q not empty }) DO
07
08       Q_active: Path;
09
10       Q_active ← Q[0];     // Get first element of Q
11       Q ← Q\{Q_active};  // remove Q_active path from the set Q
12
13       QNEW ← GetDisjointPaths(Q, Q_active);
14       if (QNEW is empty )
15           if (P is empty )
16               P ← {Q_active};
17       else
18           if (QNEW equals Q)
19               Q ← ∅;
20           QTMP ← MAXFLOW(QNEW); // Recursion
21           if ( |QTMP| ≥ |P|)
22                P ← QTMP ⋃ {Q_active};
```

The MAXFLOW algorithm "walks" over all the edges of the *Search Tree*. This *Search Tree* can be built from the input information $\mathcal{Q}$ using the method described in Section 2.3. At every vertex of

the tree the algorithm is executed. The paths contained at the vertex can be accessed through the set $\mathcal{Q}$ (line 01) and the result is stored in $\mathcal{P}$. The main idea is that for *every* path on the *active* vertex, we compute the set of disjoint paths (line 13) after removing the path (line 11). If the returned set, $\mathcal{Q}NEW$, is empty (line 14), MAXFLOW ends the recursion since there are no link-disjoint paths. On the other hand, if the set is not empty, we jump to a new vertex at the next level containing $\mathcal{Q}NEW$ using the recursion (line 20). Lines 18 and 19 ensure that in case $\mathcal{Q}NEW$ equals $\mathcal{Q}$, then after the recursion call, the algorithm finishes at the current vertex of the tree. For, in that case, $\mathcal{Q}$ is going to be processed as $\mathcal{Q}NEW$ at the next recursion level.

The selected search method in this algorithm, *Depth-first search*, first jumps to the next level exploring as far as possible along each branch before backtracking (continuing with the next path of $\mathcal{Q}$). When the recursion returns, we compare the returned set of disjoint paths, $\mathcal{Q}TMP$, with the maximum obtained at this vertex (line 21). When the returned set of paths contains at least the same number of paths as $\mathcal{P}$, the algorithm replaces its partially computed maximum set, $\mathcal{P}$, with the union of $\mathcal{Q}TMP$ and $\mathcal{Q}_{active}$ (line 22).

The *bounding function* is implemented at the lines 6 and 18-19. If every path were checked, the WHILE statement would be replaced by FOR EACH. Every iteration we remove an element from $\mathcal{Q}$. If $|\mathcal{P}|$ becomes larger than $|\mathcal{Q}|$, there is no need to continue searching since $\mathcal{P}$ is a subset of $\mathcal{Q}$.

3 An example

The MAXFLOW operation is illustrated by Figure 2. From the topology shown Figure 2 (a), we compute all the paths with the initial node S and destination node D. That set of paths $\mathcal{Q}$ is shown in the table of Figure 2 (c). In Figure 2(b) the *Search Tree* is drawn. Edges show a label of the path they are associated with. We explain the algorithm just before $\mathcal{Q}_{active}$ turns to $\mathcal{P}_4$. At that moment, the state of the partial solution, $\mathcal{P}$, is $\{\mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5\}$ and $\mathcal{Q}$ contains $\{\mathcal{P}_4, \mathcal{P}_5\}$. The bounding function test fails and the MAXFLOW does not walk over the edge $\mathcal{P}_4$. The vertex hanging on $\mathcal{P}_4$ is not drawn as solid, but as a dashed box. Despite the fact that $\mathcal{P}_3$ is link-disjoint to $\mathcal{P}_4$, it is not included because $\mathcal{P}_4$ is already included in the vertex coming from $\mathcal{P}_3$. Thus, the longest branch is composed by $\{\mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5\}$ which is returned by the algorithm as the maximum number of link-disjoint paths in the topology of Figure 2 (a).

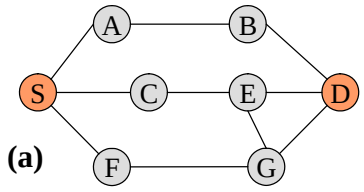
3.1 Complexity

The complexity will be computed depending on the number $n = |\mathcal{Q}|$ of the input set of paths $\mathcal{Q}$. The computation of the original input set $\mathcal{Q}$ also requires computational effort, which we omit here.

At line number 13, the MAXFLOW algorithm executes the GetDisjointPaths function that selects the paths of $\mathcal{Q}$ that are link-disjoint relative to the path $\mathcal{Q}_{active}$. If N is the maximum number of links of a path in $\mathcal{Q}$, the GetDisjointPaths function has a complexity

$$K(n, N) = O(n \cdot N^2)$$

Checking that two paths are disjoint requires $O(N^2)$ elementary operations, namely compairing if a link of one path occurs in the other path. For the special cases, the line tree and the root only,



Paths between S and D	
P1	S-F-G-E-D
P2	S-C-E-G-D
P3	S-F-G-D
P4	S-C-E-D
P5	S-A-B-D

(c)

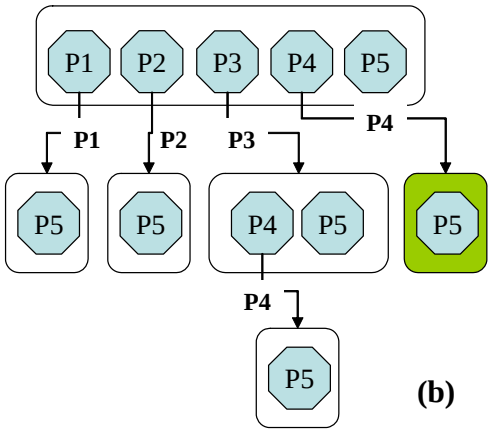


Figure 2: An example of a topology (a) in which in table (c) all paths between source S and destination D are given. Those paths are the input of MAXFLOW. The (pruned) search tree is shown in (b)

discussed in subsection 2.3 the overall complexity is straightforward. For the line tree (case A), where all the paths are mutually disjoint,

$$T_A(n, N) = T_A(n-1, N) + K(n-1, N)$$

Recall that MAXFLOW algorithm walks over a single branch because at any level the partial solution is not lower than the remaining set of paths. Therefore, no other branches start at any node. The solution of the recurrence equation is

$$T_A(n, N) = O\left(\frac{n(n-1)}{2}N^2\right) = O(n^2N^2) \tag{1}$$

In the root only case (case B), all the paths have a certain link in common. Hence, it is not possible to find a pair of paths that are disjoint. If there are no disjoint paths, the tree does not have levels and is just composed by a root vertex. However, the MAXFLOW algorithm checks all possible pairs $\binom{n}{2}$ of paths. If each check requires $K(1, N)$, then the solution is

$$T_B(n, N) = O\left(\frac{n(n-1)}{2}N^2\right) = O(n^2N^2) \tag{2}$$

Surprisingly, the required computation complexity is the same for both extreme cases. The rest of the tree topologies lay in between of these two extreme trees. Unfortunately, the complexity does not fall in between. It is difficult to figure out what the worst case for this algorithm is. Due to the bounding function, if the partial solutions gives us a set of i paths at a certain level, then, when the remaining set of paths to be checked is lower or equal to i, all the remaining branches are pruned. This implies, that it is impossible to build a tree with an arbitrary number of edges and vertices. However, we are able to compute an upper bound for the last case, when the tree has the maximum number of edges and vertices. This upper bound represents the situation of the extreme case A when the bounding function does not work.

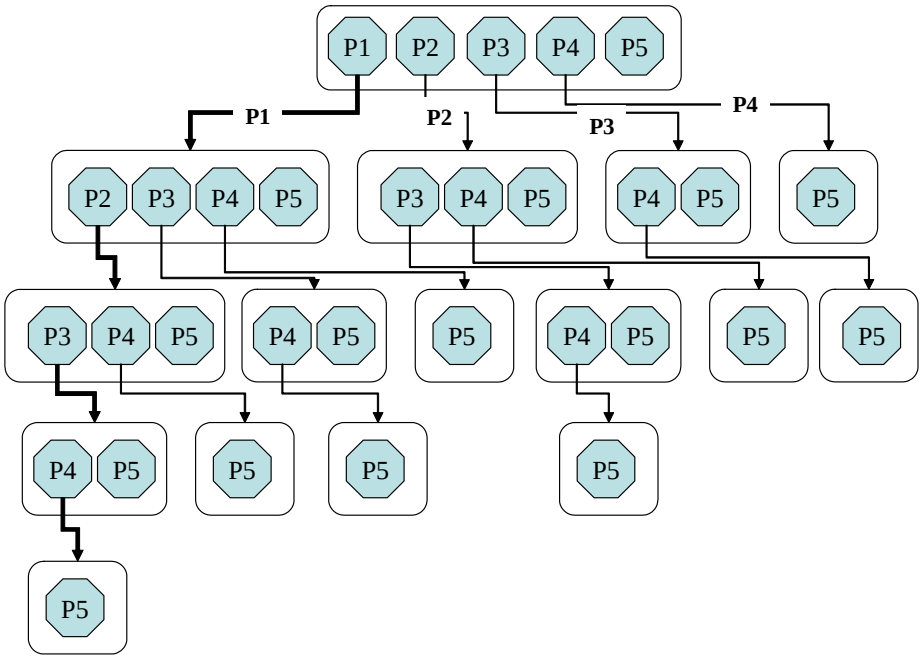


Figure 3: An example of a full search tree

An example of such tree is shown in Figure 3. In that example, the root vertex has only five paths. For a general case of n paths, we can compute the complexity using the following recurrence function,

$$\begin{aligned} T(n,N) &= K(n-1,N) + T(n-1) + K(n-2,N) + T(n-2) + \ldots + K(1,N) + T(1) \\ &= \sum_{i=1}^{n-1} (T(i) + K(i)) \end{aligned}$$

The difference $T(n,N) - T(n-1,N)$ obeys

$$T(n,N) - T(n-1,N) = K(n-1,N) + T(n-1,N)$$

which leads to the recursion

$$T(n,N) - 2T(n-1,N) = K(n-1,N)$$

whose solution is

$$T(n,N) = 2^n - N^2(n+1)$$

Although the complexity of MAXFLOW is in most case of $O\left(n^2N^2\right)$, the problem of computing all link-disjoint paths between a set of node pairs effectively requires as input $\mathcal{Q}$ a hugh set, where $n = O\left((N-1)!\right)$. For, the total number of paths between a source-destination pairs in any graph is bounded by $[e\,(N-2)!]$ (see e.g. [4, p. 322]) where $[x]$ denotes the integral part of x. This means that an exact algorithm as MAXFLOW is impractical for $N > 10$ because $C_{\text{MAXFLOW}} = O\left(\left((N-1)!\right)^2\right)$.

4 Theory

The unattractive complexity MAXFLOW has motivated an analytic study. Obviously, the number of link-disjoint paths with $h > 1$ hops between two (different) nodes, M_h, is bounded by

$$0 \leq M_h \leq N - 2 \tag{3}$$

Indeed, the upperbound follows from the maximum degree of a node minus the direct (1 hop) link. The lower bound indicates that, although the graph is connected, for some values of the hopcount h, there does not exist such a path as is readily observed from a star or ring topology. In the sequel, we investigate the number of link-disjoint paths in the complete graph K_N. Since any graph is a subgraph of the complete graph, the number of link-disjoint paths in any other graph will be upperbounded by that in K_N. We assume that links are undirected.

The case $h = 2$, is particularly simple and follows directly from the fact that all two hop paths between two nodes in K_N are link disjoint. Hence,

$$M_2 = N - 2 \tag{4}$$

The case $h = 3$ also equals $N - 2$ (for $N > 4$),

$$M_3 = N - 2 \tag{5}$$

A 3-hop path between 1 and N is characterized by $(1 \rightarrow i)(i \rightarrow j)(j \rightarrow N)$. The set of 3-hop paths $(1 \rightarrow i)(i \rightarrow i+1)(i+1 \rightarrow N)$ for $2 \leq i \leq N-2$ consists of $N-3$ link disjoint paths. This set can be complemented with the path $(1 \rightarrow N-1)(N-1 \rightarrow 2)(2 \rightarrow N)$ to a total of $N-2$ link-disjoint 3-hop paths.

4.1 Link-disjoint paths with $N-1$ hops

The number M_{N-1} of link-disjoint paths with $h = N-1$ hops between two nodes is determined.

Theorem 1 *The number of link-disjoint paths with $h = N-1$ hops between two nodes is lower bounded by*[1]

$$M_{N-1} \geq \frac{\varphi(N-1)}{2} \tag{6}$$

where $\varphi(m)$ is Euler's totient function [3, Section 5.5],

$$\varphi(m) = m \prod_{q|m} \left(1 - \frac{1}{q}\right)$$

and q denotes here a prime.

Proof: The maximum possible number of links is $\frac{N(N-1)}{2} - 1$ which equals the total number of links in the complete graph minus the direct link between the two nodes. Hence, M_{N-1} can never be larger than $\left[\frac{N}{2} - \frac{1}{N-1}\right] = \left[\frac{N-1}{2}\right]$. If q is a prime, then $\varphi(q) = q\left(1 - \frac{1}{q}\right) = q - 1$. The equality sign in (6) is

[1] If the links are directed, in which case there are $N(N-1)$ links in the complete graph, $M = \varphi(N-1)$.

reached in a complete graph K_N with $N = q+1$ nodes where q is prime, because then $\frac{\varphi(N-1)}{2} = \left[\frac{N}{2} - 1\right]$ and the highest number of link-disjoint paths is $\max(M_{N-1}) = \left[\frac{N}{2} - \frac{1}{N-1}\right] = \left[\frac{N}{2} - 1\right]$ since N is even.

Figure 4 illustrates that a link-disjoint path, indexed by $k \in [1, N-2]$, between node 1 and N is characterized by the node list

$$P_{1\to N;k} = \{1 + (kx) \bmod (N-1)\}_{0\le x\le N-1} \tag{7}$$

where the path ends in node $1 \equiv N$. The number k of these paths $P_{1\to N;k}$ then equals the number of modulo $N-1$ classes that is complete, which equals $\varphi(N-1)$. Since every path can be traversed in the opposite direction, half of the number $\varphi(N-1)$ is link disjoined. This also follows from (7) because

$$\begin{aligned} P_{N\to 1;(N-1-k)} &= \{1 + (N-1-k)\,(N-1-x) \bmod (N-1)\}_{0\le x\le N-1} \\ &= \{1 + (kx) \bmod (N-1)\}_{0\le x\le N-1} = P_{1\to N;k} \end{aligned}$$

Hence, for different link-disjoint paths, we may restrict k to the interval $\left[1, \left[\frac{N-1}{2}\right]\right]$.

Apart from the "trivial" $P_{1\to N;1}$ path that always exists for any N, the other ($k > 1$) individual paths $P_{1\to N;k}$ that consists of $N-1$ hops are those for which k is *not* a divisor of $N-1$. Hence, if $N-1$ is prime, all $k < N-1$, in total $N-2$, satisfy this condition. Indeed, in that case is $\varphi(N-1) = N-2$. But, the first half set of $N-1$-hop paths indexed by $1 \le k \le \frac{N-2}{2}$ consumes already all available links. Thus, we can draw in two different complete graphs K_N, precisely two different sets of $N-1$-hops paths provided $N-1$ is prime.

If $k|N-1$, the hopcount of $P_{1\to N;k}$ is precisely $\frac{N-1}{k}$. Indeed, the hopcount h of the path $P_{1\to N;k}$ is the smallest solution $h = x$ of the equation $(kx) \bmod (N-1) \equiv 0$, which, if $k|N-1$, is equivalent to $x \bmod (\frac{N-1}{k}) \equiv 0$ and satified for $x = \frac{N-1}{k}$. If the greatest common divisor of k and $N-1$ is g, then the hopcount of $P_{1\to N;k}$ is $\frac{N-1}{g}$.

The Theorem is proved if inequality is demonstrated. A single example suffices. In case $N = 7$, we have that $\frac{\varphi(N-1)}{2} = 1$ namely the "trivial" $P_{1\to 7;1}$ path, where $P = 1-3-5-2-6-4-7$ is another $N-1$ hop path and link disjoint to $P_{1\to 7;1}$. This observation illustrates the inequality sign in (6). $\square$

The procedure in the proof first removes all direct links ($k = 1$), then all second hop links ($k = 2$) if they possibly construct an $N-1$ hop path and so on. We call this the "equal spacing" method. the By any relabeling (i.e. permutation) of the nodes, this regular hop structure can be broken, but the total number of link-disjoint paths does not change by relabeling. On the other hand, procedure cannot always guarantee to find the maximum set of link disjoint paths. For example, in case $N = 8$, we find beside the "trivial" $P_{1\to N;1}$ path,

$$\begin{aligned} P_{1\to 8;2} &= 1-3-5-7-2-4-6-8 \\ P_{1\to 8;3} &= 1-4-7-3-6-2-5-8 \end{aligned}$$

whereas another possible set is

$$\begin{aligned} P_a &= 1-3-5-2-7-4-6-8 \\ P_b &= 1-4-2-6-3-7-5-8 \end{aligned}$$

Clearly, relabeling the node 2 by node 7 and, vice versa, node 2 by node 7 which we denote by $2 \longleftrightarrow 7$ and relabeling $3 \longleftrightarrow 6$ transforms both sets into the same. However, in that case, also the trivial path is relabeled and the resulting set of three link-disjoint paths is thus different. In case $N = 9$, our procedure does not lead to a maximum set. For, beside the "trivial" $P_{1\to N;1}$ path, we only find

$$P_{1\to 9;3} = 1 - 4 - 7 - 2 - 5 - 8 - 3 - 6 - 9$$

whereas another set

$$P_a = 1 - 4 - 2 - 7 - 5 - 8 - 3 - 6 - 9$$
$$P_b = 1 - 3 - 5 - 2 - 6 - 8 - 4 - 7 - 9$$

clearly contains 1 additional link-disjoint path. The relabeling $2 \longleftrightarrow 7$ transforms P_a into $P_{1\to 9;3}$ and P_b to

$$P_c = 1 - 3 - 5 - 7 - 6 - 8 - 4 - 2 - 9$$

and the "trivial" $P_{1\to N;1}$ path to

$$P_d = 1 - 7 - 3 - 4 - 5 - 6 - 2 - 8 - 9$$

We observe that P_c initial consists of second hop links ($k = 2$) up to roughly the half the path length – first four hops – and then a non-constant value of k is followed: $k = 7, 2, 4, 6, 7$. This observation shows that the "equal spacing" method is not exhaustive. In particular, only if $N = q + 1$ with q prime, the "equal spacing" method is exhaustive. In all other cases, where $N - 1$ is composite, non-equal spacing seems necessary.

As a corollary of Theorem 1, the average number of link-disjoint paths in $G_p(N)$ with $N - 1$ hops between two nodes is larger than or equal to $E[M_{h-1}] \geq \frac{\varphi(N-1)}{2} p^{N-1}$.

4.2 Link-disjoint paths with $N - 2$ hops

The question arises whether the previous method can be extended to other hopcount cases as well. Clearly, by omitting one node, the previous construction directly leads to at least $\frac{\varphi(N-2)}{2}$ link-disjoint $N - 2$ hop paths, while, at most there are $\left\lceil \frac{N(N-1)}{2(N-2)} - \frac{1}{N-2} \right\rceil$ such paths. Hence, if $N - 2 = q$ is prime, there holds that $\frac{N-3}{2} \leq M_{N-2} \leq \left\lceil \frac{N(N-1)}{2(N-2)} - \frac{1}{N-2} \right\rceil$ and $\left\lceil \frac{N(N-1)}{2(N-2)} - \frac{1}{N-2} \right\rceil = \left\lceil \frac{(q+2)(q+1)}{2q} - \frac{1}{q} \right\rceil = \left\lceil \frac{q+3}{2} \right\rceil = \frac{N+1}{2}$. But, since the difference of 2 is not sufficient for an additional $N - 2$ hops path (for $N > 4$), we conclude that $M_{N-2} = \frac{N-3}{2}$ if $N - 2$ is prime.

The other cases are more complex because the set of modulo $N - 2$ classes is not complete and the omitted node may be used to complete them. In addition to the $\frac{\varphi(N-2)}{2}$ link-disjoint $N - 2$ hop paths, for each $k \in \left[1, \left[\frac{N-1}{2}\right]\right]$ that satisfied the conditions $k|N - 1$ and $\frac{N-1}{k} + 2 = N - 2$, a new $N - 2$ link disjoined path is found. Indeed, with the previous modulo-construction, a path has hopcount equal to $h = \frac{N-1}{k}$ if $k|N - 1$. By including the omitted node, the path increases by 2 links. The number of those additional solutions is small, namely, that k for which $\frac{N-1}{N-4} = 1 + \frac{3}{N-4}$ is an integer, which only occurs if $N = 7$. In that case $k = \frac{N-1}{N-4} = 2$.

In conclusion, $M_{N-2} = \frac{\varphi(N-2)}{2}$ for all $N > 7$. If $N = 7$, $M_{N-2} = \frac{\varphi(N-2)}{2} + 1$. Most likely, the addition process that complements the equal spacing method does not exhaust all possibilities.

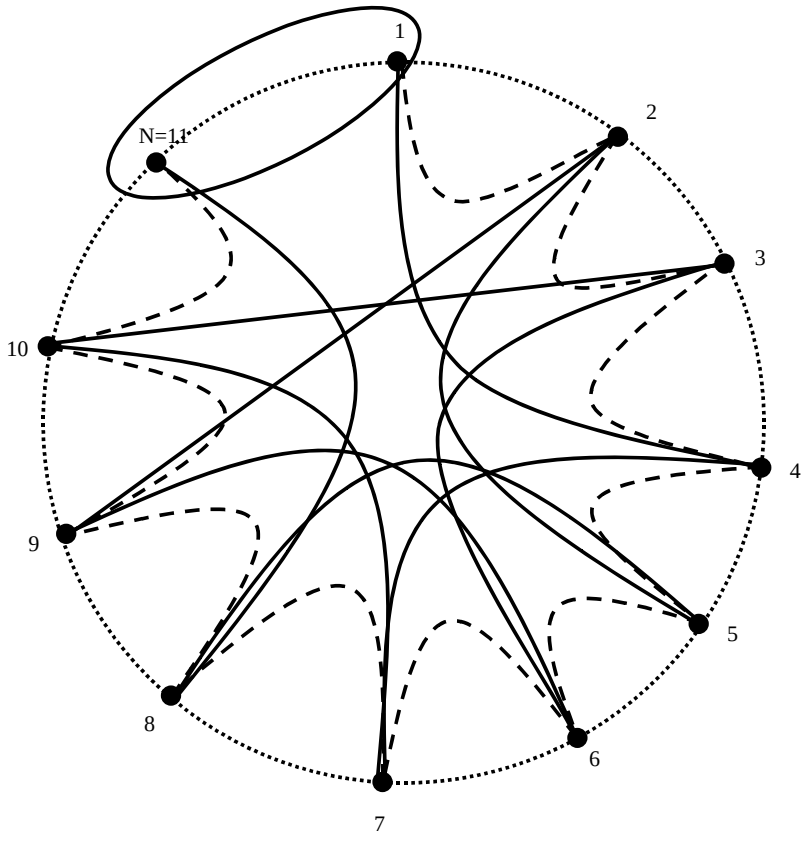


Figure 4: The nodes of the complete graph K_{11} are drawn on a circle. For $k = 1$ and $k = 3$, $P_{1\to 11;1}$ and $P_{1\to 11;3}$ are the only two "equal spaced" 10-hop paths because $\frac{\varphi(10)}{2} = 2$.

The cases $4 \le h \le N-3$ are challenging. Any path with $h = j$ hops between A and B is specified by $\mathcal{P}_{A\to B} = (n_1 \to n_2)(n_2 \to n_3)\cdots(n_j \to n_{j+1})$ with the property that the sum of the differences of the nodal identification numbers $S_j = \sum_{i=1}^{j}(n_{i+1} - n_i) = n_{j+1} - n_1 = B - A$ which is independent of the hopcount. Without loss of generality because we can also relabel the nodal ID numbers, we can choose $A = 1$ and $B = N$. Next, by adding the direct link between A and B, a loop is created for which $S_j = 0$. The paths in the modulo method (7) feature the additional property that $n_{i+1} - n_i = k$ or, via equal spacing, the total link space is scanned. If $h = N - 1$, the modulo method exhausts all possibilities because all nodes must be visited by each $N - 1$-hop path. But, in the other cases, the equal spacing might not be successful.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *An Introduction to Algorithms*. MIT Press, Boston, 1991.

[2] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, November 1995.

[3] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, London, 4th edition, 1968.

[4] P. Van Mieghem. *Performance Analysis of Communications Networks and Systems.* Cambridge University Press, Cambridge, U.K., 2006.