

# Optimizing the Cyclic K-conflict-free Shortest Path Problem in a Network-on-chip

Boureima Zerbo<sup>1</sup>, Marc Sevaux<sup>2\*</sup>, Andr'e Rossi<sup>3</sup> and Jean-Charles Cr'eput<sup>4</sup>

<sup>1</sup>Universit'e de Ouagadougou, Burkina-Faso

<sup>2</sup>Universit'e de Bretagne-Sud –Lab-STICC, CNRS UMR 6285, Lorient, France

<sup>3</sup>Universit'e d'Angers, LERIA, Angers, France

<sup>4</sup>Universit'e de Technologie de Belfort-Montb'eliard, France

## Abstract

We study a combinatorial optimization problem for conflict-free routing in a Network-on-Chip. Based on time division multiplexing and cyclic emission, the problem consists in finding a set of  $K$  shortest paths, such that packets will never conflict through the network but can use shared communication links in an efficient way. The model allows to avoid collisions and deadlocks in irregular network topologies, while minimizing latency. A time-expanded graph approach is retained for the solution process. First, we present a mixed integer linear programming model for the problem. Second, a set of shortest paths operators are combined within three iterated local search schemes able to quickly generate admissible solutions for the problem. To evaluate the method, experiments are conducted on a set of five real-life problem instances, and on many artificial unstructured random instances derived from them. We detail the problem of traffic instance generation, that also illustrates the designer's task of flow decomposition between communicating components. Intensive simulations illustrate the accuracy of the solution method.

## Introduction

Network-on-chip (NoC) is an emerging approach in multi-processor system-on-chip (MPSoC) technology in which the design of efficient communication routing schemes is an important challenge [1-3]. In such systems, traditional solutions with shared-buses are replaced by interconnections with short links. As in computer networks or terrestrial transportation networks, a critical issue is to allow guarantee of traffic bandwidth, avoid collisions, deadlocks and livelocks. In such networks, Guaranteed Traffic (GT) approaches are often opposed to Best Effort (BE) methods [4]. A key feature of GT is to address conflict-free routing at the time of route computation, whereas BE deals with these problems only at the execution time of the routes. It is admitted that BE networks achieve good average performances, but that worst case performances are very hard to predict [5]. Furthermore, avoiding deadlocks in BE networks implies restrictions on routing and/or extra-costs due to virtual channel splitting. On the contrary, GT methods ensure the application real-time requirements and avoid the possibility of contention and deadlocks while using irregular topologies that allow significant power savings.

In this paper, we present a combinatorial optimization problem that allows guaranteed traffic with conflict-free routing. The communication is wormhole. Wormhole routing operates by advancing the head of a message directly from incoming to outgoing links. Packets are stored in the links while advancing through the networks in a pipeline fashion. The header packet contains the specification of the path to follow, and packets must remain contiguous in the network links. No memory buffer is needed in the routers to store packets. Transmissions are synchronous with a common clock that defines the time unit, called a time-slot. To efficiently schedule messages through shared links, we adopt the technique of time division multiplexing. Messages are emitted periodically with a period of length  $T$ . The value of time  $T$  is determined according to the communications bandwidth constraints during the design phase. The size of  $T$ , for all interconnected Intellectual Property

(IP) components, is standardized according to the minimum of the maximum number of packets received by the IPs [6,7]. The number of packets by message defines a given bandwidth between a source and a destination node. At each source node, a TDMA (time division multiple access) table specifies the departure times of each message within the possible time-slots  $0, \dots, T-1$ . This technique of time-multiplexing was previously presented and used in [6,8-11]. But the emphasis was mainly put on architecture considerations, and the combinatorial optimization problem succinctly addressed with some ad hoc heuristics. No reproducible benchmarks were proposed to allow comparative evaluations between the methods. In this paper, and for the first time, we state the optimization problem formally, and relate it to other standard routing problems. We focus on its solution using standard local search techniques, and provide a set of reusable benchmarks to compare this work's results to future works on this problem. It can be used to calculate the best communication paths for implantation in the NoC during the design.

Other works deal with different resolution techniques. The interested reader is referred to Ge and Wu [12]; Lin et al. [13]. A recent survey of these routing methods in NoC is presented in Benmessouad and Koudil [14].

Given a graph that represents a network topology, and a set of  $K$  source-destination messages of variable sizes that are periodically emitted based on a temporal cycle of length  $T$ , the goal is to compute  $K$  source-destination paths and to set their message emission dates,

**Corresponding Author:** Prof. Marc Sevaux, Universit'e de Bretagne-Sud –Lab-STICC, CNRS UMR 6285, Lorient, France; E-mail: [marc.sevaux@univ-ubs.fr](mailto:marc.sevaux@univ-ubs.fr)

**Citation:** Zerbo B, Sevaux M, Rossi A, Cr'eput JC (2017) Optimizing the Cyclic K-conflict-free Shortest Path Problem in a Network-on-chip. Int J Comput Softw Eng 2: 115. doi: <https://doi.org/10.15344/2456-4451/2017/115>

**Copyright:** © 2017 Zerbo. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

such that packets will never clash through the network. The objective is to minimize the paths total length in order to minimize total power consumption and network use rate. The length of a path is expressed in time-slots.

One way to address such a problem of multi-path findings where time plays an important role is by using a time-expanded graph. A time-expanded graph (TEG) contains one copy of the node and arc sets of the original static graph for each discrete time step considered. As well as for time-division multiplexing [9], this structure is often useful in terrestrial transportation problems, particularly in routing with time-windows. As reviewed by K'ohler et al. [15], the TEG concept allows solving a variety of flow-over-time problems by applying algorithmic techniques developed for static network flows. As an example, we use a modified Dijkstra algorithm able to compute a single shortest path in the TEG in pseudo-polynomial time. We will see how such a procedure drastically improves performance of the heuristic approach for the whole problem. Based on the TEG structure, we first present an integer linear program (ILP) for the problem in order to address it in an exact way with a standard solver. We also propose three heuristics to address the problem based on a standard iterated construction-improvement scheme. They are an iterated random search, and an iterated local search declined within two versions, greedy descent and steepest descent respectively. The basic operators are a greedy parallel construction method, a neighborhood operator, a modified Dijkstra algorithm, and management date operators. Even though deciding feasibility is NP-hard, our heuristic approaches attempt to generate feasible solutions as fast as possible. In experiments, we illustrate how the progressive introduction of operators drastically improves performances. Intensive experiments are performed on a set of real-life instances and on artificial random traffic configurations. We present the problem of traffic instance generation as a maximum flow problem. It illustrates the designer's problem of bandwidth decomposition and casts a light on the prerequisites of a guaranteed traffic approach. Intensive simulations illustrate the robustness of the approach when addressing a wide range of possible traffic configurations.

In Section 2, we state the routing problem, called Cyclic K-conflict-free shortest Paths Problem (CKPP). We relate the problem to the literature of shortest paths routing. In the same section, we justify the use of a time-expanded graph structure to solve the problem and present an ILP model of CKPP. In Section 3, we present the local search methods retained for the solution process. Based on an iterated local search scheme, we detail three versions of conducting the improvement search. Section 4 is devoted to empirical evaluations of the algorithms on benchmarks representative of the application domain, and also on unstructured random instances. Then, the last section is devoted to the conclusion and further research.

### Definition of the Routing Problem

We first state the routing problem and relate it to similar combinatorial optimization problems in the literature. We justify the use of a time-expanded graph approach for the solution method, and also present a mixed integer linear program for the problem.

#### Problem statement

A NoC-based architecture consists of a set of interconnected Intellectual Property (IP) components. Such IPs are typically general purpose or specific processors, dedicated hardware accelerators,

and peripheral controllers or memories in a single chip. They are emitters and/or receivers of messages. The components are connected by routers according to a given network topology. Each IP is associated with a unique a router through its network interface (NI). More formally, a NoC can be modeled by a directed graph  $G = (V,A)$ , where the set  $V$  of vertices represents the routers and IPs, and the set of arcs  $A \subseteq V \times V$  represents the directed transmission links between them. Here, we consider wormhole routing. Each message  $\mu$  is made of a sequence of  $l(\mu)$  contiguous packets, with  $l(\mu) = l\mu > 0$ , that are transmitted along the arcs of the network. The header packet contains a specification of the message origin/destination path, while other packets are application-dependent data. A router has no memorization capacity, it only retransmits packets as specified by the header packet. The NoC is synchronous, it is cadenced by a common clock shared by all its components. Each arc has a capacity of 1 packet by unit of time, called a time-slot or time-step. The packets are transmitted in a contiguous way. Hence, if a physical occurrence of a message emitted at time  $t(\mu)$  follows the path  $i, i+1, \dots, i+n$ , with  $i \in V$ , its packets will cross the arc  $(i, i+1)$  at the consecutive time-steps  $t(\mu)+i+q, q = 0, \dots, l_\mu - 1$ . We say that an arc may be "occupied" by a packet at a given time-slot, otherwise it is said "free" at a given time-slot.

To guarantee transfer rate, a source repeatedly emits messages based on a period or cycle  $T$ . A given bandwidth is specified by a message size, i.e., the number of packets emitted by period  $T$ . Hence, a message can be seen as a class of its physical occurrences at each cycle. This allows to consider classes of time-slots for arc occupations. If an arc is occupied at time-slot  $t$ , it will also be occupied at timeslots  $t + \lambda T$ , for all nonnegative integer  $\lambda$ . We say that an arc is occupied at  $t$  modulo  $T$  to express its recurrent occupation by a packet. We will often talk of a packet, or message, to refer to the class it represents. When transmitted through the network, two packets are said conflict-free if they never clash, i.e., cross an arc at the same time-slot. By extension, paths or messages are said "conflict-free" or "contention-free" when

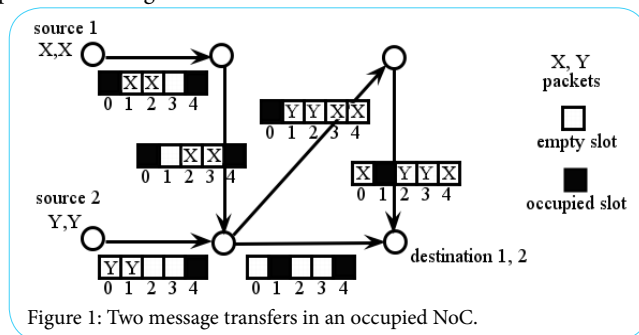


Figure 1: Two message transfers in an occupied NoC.

their related packets never conflict (two at a time). Since paths may share common communication links, and since we want to achieve guaranteed traffic, paths and emission dates have to be stated such that two packets will never clash. We assume that paths may contain circuits, only when those circuits do not contain the destination or the origin of the path. Indeed, if the origin appears in a circuit, then the message can be send later without that circuit, and if the destination appears in a circuit, then all the corresponding traffic is useless and can be removed. However, a circuit that does not involve the destination nor the origin may be useful to wait for a region of the network to be less heavily loaded. Figure 1 illustrates two conflict-free source/destination paths in an already occupied NoC, with an emission cycle of length  $T = 5$ . Two packets are emitted from source 1, and two packets from source 2. A table of the available time-slots (modulo  $T$ ) that packets can use as they advance through the network is shown

next to each arc, and filled squares are unavailable time-slots. We can see that packets cross the shared arcs at distinct time-slots modulo  $T$  and never conflict. We can now precisely state the combinatorial optimization problem called "Cyclic K-conflict-free shortest Paths Problem" that we address in this paper.

### Cyclic K-conflict-free shortest Paths Problem (CKPP)

A problem instance consists in a directed graph  $G = (V,A)$ , an emission cycle of length  $T$ , and a set of  $K$  messages defined by

$$M = \{\mu_k = (s_k, d_k, l_k) : s_k \in V, d_k \in V, l_k > V, k = 1, \dots, K\}$$

where  $(s_k, d_k)$  is an origin/destination pair, and  $l_k$  the number of packets of the message. The goal is to find the departure times of the messages  $t(\mu_k) \in [0, T - 1]$ ,  $\mu_k \in M$ , at each source node and the origin/destination paths of minimum total length for conveying the messages, such that the paths are conflict-free.

We should note that the objective of the problem is the total length of the paths, whereas the constraints reside in the temporal occupation of the arcs such that paths are conflict-free. It is worth noting that the problem is NP-hard in the strong sense for general graphs as well as for planar graphs or grid networks. This can be seen by relating the problem to the K-Edge-Disjoint Shortest Paths problem (EDP) [16]. This well known problem has different versions depending whether we ask for vertex or edge disjoint paths between a set of  $K$  source-sink pairs. This is one of Karp's [17] original NP-hard problems. By restricting CKPP to only those instances for which  $T = 1$ , we retrieve the EDP. This problem is known to be NP-hard in the case of planar graphs [18], even when stated in the grid and when path lengths are constrained by a constant [19]. Then, since CKPP remains NP-hard when the number  $T$  is bounded by a constant, it follows that it is NP-hard in the strong sense. Similarly, the CKPP can be seen as an extension of a Bin Packing problem, where arcs stand for bins of capacity  $T$ , and the  $K$  messages for items of size  $q_k$ .

A similar problem is the unsplitable flow problem (UFP). It is a generalization of EDP where every edge  $e$  has a positive capacity  $c_e$  and every pair has a demand  $f_k > 0$ . The demand from  $s_k$  to  $d_k$  has to be routed in an unsplitable manner, i.e., along a single path from  $s_k$  to  $d_k$ . For every edge  $e$  the total demand routed through that edge should be at most  $c_e$ . The problem adds a capacity constraint to the EDP. It is different from CKPP since it allows variable arc capacities, discarding temporal aspects on arc occupation. Generally, classical flow models only deal with static situation. A well-known problem that introduces time dependent transit constraints is the one-to-one shortest path problem with time windows (SPPTW) [20]. The aim is to compute a shortest path respecting the given time-windows on arcs occupation. This kind of problems often arises in terrestrial transportation, road traffic control, and vehicle routing applications. An example is Automated Guided Vehicles (AGVs) [21] technology for optimizing large scale production and logistic systems. The SPPTW is NP-hard but several pseudo-polynomial time algorithms are available to solve it exactly [20,22,23]. A sub-problem of our CKPP is to compute a one-to-one source-destination path in an already occupied NoC. The available ("free") time-slots on each arc stand for time-windows. Hence, a modified Dijkstra algorithm can be designed to compute such a single path in an efficient way. It will be used as an operator into the heuristic methods presented in this paper to address the whole CKPP.

### Time-expanded graph approach

We address the problem by using a spatiotemporal graph as a memory of temporal arc occupation. This memory specifies for each arc of the network, and each time-slot in interval  $[0, T - 1]$ , whether or not the arc is occupied by a given packet at the given time slot. We say that an arc is "free" or "occupied" at time  $t$  modulo  $T$ . Hence, an array of booleans of size  $T$  is associated to each arc. Such a spatiotemporal graph is generally called a time-expanded graph (TEG)[15]. In its standard definition, a TEG contains one copy of the node set for each discrete time step. This is a time layer. For each arc in the original graph, there is an arc copy between each pair of time layers in the TEG. When time occupation is cyclic, as in our case, each layer  $t$  is connected to its next  $(t + 1)$  modulo  $T$  layer. The notion was presented by Ford and Fulkerson [24] in order to model flow over time. The memory size needed is  $O(N^2 \times T)$  for general graphs or  $O(N \times T)$  for planar graphs, with  $N$  the number of nodes. As reviewed by Köhler et al. [15], the TEG concept allows to solve a variety of flow-over-time problems by applying algorithmic techniques developed for static network flows. It is worth noting that the size of the time-expanded network is only pseudo-polynomial in the input size, since the number  $T$  is encoded with  $\log(T)$  bits. Hence, we should address only instances of moderate size  $T$ . Fortunately, since  $T$  is always small (maximum of 47 time-slots in our applications), the TEG approach is a natural way to deal with shortest path findings that would guarantee traffic in a NoC.

### Mixed integer linear programming model

We first propose an ILP of the CKPP. The model is packet oriented. A NoC is composed of a set  $\mathcal{I}$  of IPs ( $|\mathcal{I}| = P$ ) and a set  $\mathcal{R}$  of routers ( $|\mathcal{R}| = N$ ). A set  $\mathcal{P}$  of packets ( $|\mathcal{P}| = Q$ ) with a specific origin and destination (taken from  $\mathcal{I}$ ) has to travel across the NoC without conflicts and in the shortest possible time. The set of IPs and routers form the nodes of the graph  $V = \{0, \dots, P - 1, P, \dots, P + N - 1\}$ . The set of arcs is denoted by  $A \subseteq V \times V$ . There are  $|Q|$  packets indexed by  $q$ ,  $q \in \{1, \dots, Q\}$  such that where  $l_k$  is the size of message as stated in the CKPP definition. Each packet  $q$  starts and ends in an IP, i.e. has an origin  $s_q$  in  $\mathcal{I}$ ,  $s_q \in \{0, \dots, P - 1\}$  and a destination  $d_q$  in  $\mathcal{I}$ ,  $d_q \in \{0, \dots, P - 1\}$  with  $s_q \neq d_q$ . Time is discretized,  $t \in \{0, \dots, T - 1\}$  is a time slot, and  $T$  is the TDMA size. The system is pipeline-based, thus, time slots  $t$  and  $t + T$  are identical for all  $t$ . Let  $x_{i,j,q,t}$  be a boolean variable set to 1 if packet  $q$  uses arc  $(i, j)$  during time slot  $t$ , 0 otherwise. With the previous definition, we can state the constraints and the objective function as a 0-1 integer linear program in Equations (1-11) below.

**1. Arc capacity:** Each arc is used for conveying at most one packet per unit of time.

$$\sum_{q \in \{1, \dots, Q\}} x_{i,j,q,t} \leq \forall (i, j) \in A, \forall t \in \{0, \dots, T - 1\} \quad (1)$$

**2. Packet origin:** For all  $q \in \{1, \dots, Q\}$ , packet  $q$  originates from IP  $s_q \in \{0, \dots, P - 1\}$ .

$$\sum_{t \in \{0, \dots, T - 1\}} \sum_{\substack{j \in \{P, \dots, P + N - 1\} \\ |(s_q, j) \in A}} x_{s_q, j, q, t} = 1 \quad \forall q \in \{1, \dots, Q\} \quad (2)$$

**3. Packet destination:** For all  $q \in \{1, \dots, Q\}$ , packet  $q$  has to reach IP  $d_q \in \{0, \dots, P - 1\}$ .

$$\sum_{t \in \{0, \dots, T - 1\}} \sum_{\substack{i \in \{P, \dots, P + N - 1\} \\ |(i, d_q) \in A}} x_{i, d_q, q, t} = 1 \quad \forall q \in \{1, \dots, Q\} \quad (3)$$

**4. Packets conservation:** For all instant time  $t \in \{0, \dots, T-1\}$ , for all  $q \in \{1, \dots, Q\}$  and for all  $i \in \{P, \dots, P+N-1\}$ , if packet  $q$  reaches router  $i$  at time  $t$ , then it has to leave that router at time  $t+1$ .

$$\sum_{j \in V \setminus \{(i,j) \in A\}} x_{j,i,q,t} - \sum_{i \in V \setminus \{(i,j) \in A\}} x_{j,i,q,(t+1)} \bmod T = 0 \quad \forall q \in \{1, \dots, Q\} \quad (4)$$

$$\forall t \in \{0, \dots, T-1\}, \quad \forall i \in \{P, \dots, P+N-1\}$$

**5. Origin IPs are not reentrant:** For all instant time  $t \in \{0, \dots, T-1\}$ , for all  $q \in \{1, \dots, Q\}$ , packet  $q$  cannot enter IP  $s_q \in \{0, \dots, P-1\}$ .

$$\sum_{i \in \{P, \dots, P+N-1\} \setminus \{(i,s_q)\}} x_{i,s_q,q,t} = 0 \quad \forall q \in \{1, \dots, Q\} \quad (5)$$

$$\forall t \in \{0, \dots, T-1\}$$

**6. Destination IPs are not reentrant:** For all slot time  $t \in \{0, \dots, T-1\}$ , for all  $q \in \{1, \dots, Q\}$ , packet  $q$  cannot leave IP  $d_q \in \{0, \dots, P-1\}$ .

$$\sum_{j \in \{P, \dots, P+N-1\} \setminus \{(d_q,j) \in A\}} x_{i,j,q,t} = 0 \quad \forall q \in \{1, \dots, Q\} \quad (6)$$

$$\forall t \in \{0, \dots, T-1\}$$

**7. Origin IPs generate at most one packet at a time:** For all time slot  $t \in \{0, \dots, T-1\}$ , for all IP  $i \in \{0, \dots, P-1\}$ , the sum of the packets  $q$  sent out of  $i$  and such that  $s_q = i$  must be less than one.

$$\sum_{\substack{q \in \{1, \dots, Q\} \\ |s_q=i}} \sum_{\substack{j \in \{P, P+N-1\} \\ |(i,j) \in A}} x_{i,j,q,t} \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \quad (7)$$

$$\forall t \in \{0, \dots, p-1\}$$

**8. Destination IPs consume at most one packet at a time:** For all time slot  $t \in \{0, \dots, T-1\}$ , for all IP  $j \in \{0, \dots, P-1\}$ , the sum of the packets  $q$  sent to IP  $j$  and such that  $d_q = j$  must be less than one.

$$\sum_{\substack{q \in \{1, \dots, Q\} \\ |d_q=j}} \sum_{\substack{i \in \{P, P+N-1\} \\ |(i,j) \in A}} x_{i,j,q,t} \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \quad (8)$$

$$\forall i \in \{P, \dots, P+N-1\}$$

**9. Message constraints:** A message is modeled as a sequence of packets that must have the same route in the NoC. More formally, message  $k \in \{1, \dots, K\}$  is an ordered set of packets  $\{\omega_{k,p}, \dots, \omega_{k,l(k)}\}$  where  $l(k)$  is the length of message  $k$  (i.e., the number of packets in the message).  $\omega_{k,p} \in \{1, \dots, Q\}$  is the first packet and  $\omega_{k,l(k)}$  is the last packet of message  $k$ . In any consistent instance, all the packets of a message must have the same origin and destination router, i.e., they must satisfy:  $s_q = s_q$  and  $d_q = d_q \quad \forall k \in \{1, \dots, K\}, \forall_{q,q'} \in \{\omega_{k,1}, \dots, \omega_{k,l(k)}\}$ .

The message constraints are enforced as follows:

$$x_{i,j,\omega_{k,q},t} = x_{i,j,\omega_{k,q+1},(t+1) \bmod T} \quad \forall q \in \{1, \dots, l(k)-1\}, \quad (9)$$

$$\forall t \in \{0, \dots, T-1\}, \quad \forall (i,j) \in A$$

These constraints say that if packet  $\omega_{k,q}$  occupies arc  $(i,j)$  at time  $t$ , then packet  $\omega_{k,q+1}$  must occupy this arc at time  $(t+1) \bmod T$ . The equality also enforces that if packet  $\omega_{k,q}$  does not occupy arc  $(i,j)$  at time  $t$ , then packet  $\omega_{k,q+1}$  cannot occupy this arc at time  $(t+1) \bmod T$ . Thus, all the packets of a message move along the same route within a time shift.

**10. Packet destination:** The destination  $d_q$  of packet  $q$  is a terminal node of the path of the packet  $q$ . For all time slot  $t \in \{0, \dots, T-1\}$ , for all  $q \in \{1, \dots, Q\}$ , the packet  $q$  cannot go to another destination other than its destination  $d_q$ .

$$\sum_{\substack{j \in \{0, \dots, P-1\} \setminus \{d_q\} \\ (i,j) \in A}} x_{i,j,q,t} = 0 \quad \forall q \in \{1, \dots, Q\}, \quad (9)$$

$$\forall t \in \{0, \dots, T-1\}, \quad \forall i \in \{P, \dots, P+N-1\}$$

**11. Objective:** Minimize the total length of all paths for all packets

$$\text{Min} \sum_{(i,j) \in A} \sum_{q \in Q} \sum_{t \in T} x_{i,j,q,t}$$

When compared with the CKPP problem definition which is a slightly more general formulation, Constraints 7 and 8 are specifically added to take into account the hardware conditions of emission and reception. These conditions state that each IP node implements a TDMA (Time Division Multiple Access) table that can only send one single packet at each time slot occurrence. A router node, at the difference of an IP node, allows packets to cross at a given time slot following the different directions stated by the communication graph. An IP node will always be connected to a single router and its emission capability restricted by the TDMA mechanism such that no more than one packet will be emitted or received at each time slot.

## Iterated Local Search Approaches

### Iterated local search main loop

We choose one of the simplest ways to address the problem by means of heuristic methods, that is, by a simple iterated local search scheme that exploits randomized choices [25]. The main loop simply iterates a search many times by restarting the algorithm from a new random starting point. We identify two levels of operations. The first level puts the emphasis on the fast construction of partial solutions. It manages the message emission dates and then, applies a set of construction operators. Starting from such a solution, the second level applies local modifications on the path set in order to "improve" or "repair" the solution. Improvement is based on a neighborhood operator that explores a smaller region around the current solution. The construction search is followed by an improvement search as often in heuristic methods. Hence, the iterated approach respectively, alternates large and small moves in the search space. Large construction moves are for diversity generation, whereas small improvement moves for intensification of the search in a particular region. The pseudo-code of the main loop is presented in Algorithm 1. The two internal calls "constructSolution" and "improveSolution" respectively correspond to the construction trial followed by the improvement trial. These two procedures are detailed in the next sections. In each case, possibly non admissible solutions can be generated. That is why solutions must be ranked according to the hierarchical two objectives of respectively the number of built paths and the paths length. Maximizing the number of built paths is the first objective, whereas minimizing the paths length is the second objective. The "selectBest" procedure performs such a ranking.

### Algorithm 1: Iterated local search method

Output: BestSolution

begin

```

S ← initialize(); // Initialize data structures and emission dates
count ← 0;
while count < maxCount do
    count ← count + 1;
    S ← constructSolution(S);
    S ← improveSolution(S);
    BestSolution ← selectBest(S, BestSolution);
return BestSolution;
    
```

## Construction loop

Here, we present the structure of the construction procedure referred to as *constructSolution* in the main loop of Algorithm 1. The procedure is detailed in Algorithm 2. Its role is to quickly generate new candidate solutions, admissible or not, that present the maximum number of paths built. To do so, the procedure iterates a basic construction process *maxConstruct* times. The first type of operations is the management of the message departure dates. Once the emission dates are set, a greedy parallel construction operator tries to quickly and simultaneously generate the *K* conflict-free paths. Then, a modified Dijkstra algorithm, which is more time consuming, only tries to rebuild the residual "blocked" paths one by one. The *selectBest* procedure performs a ranking as explained in the previous section. The main operators are detailed below.

Emission date management. As mentioned in the problem definition, the construction of admissible source/target paths mainly depends on the message emission dates that are specified in each

---

### Algorithm 2: constructSolution

---

Output:  $S \in \text{Solution}$ , *maxConstruct*

```
begin
  count  $\leftarrow$  0;
  while count < maxConstruct do
    count  $\leftarrow$  count + 1;
     $S \leftarrow$  permuteDepartureDates(S);
     $S \leftarrow$  translateDepartureDates(S);
    Initialize the TEG; // all arcs are set "free" at each time-slot
    Remove the all paths from S;
     $S \leftarrow$  constructSolutionPar(S); // Greedy parallel construction
     $S \leftarrow$  constructSolutionWithModifiedDijkstra(S); // Construct
    the remaining paths one-by-one in TEG
     $\text{BestSolution} \leftarrow$  selectBest(S, BestSolution);
  return BestSolution;
```

---

TDMA table of each source node in the interval  $[0, T - 1]$ . Assuming that the emission dates are first set at initialization, it is necessary to make evolve such dates while constructions are reiterated. Two operators are proposed in order to manage emission dates. They are respectively named *permuteDepartureDates* and *translateDepartureDates* in Algorithm 2. The first operator is applied with probability 0.5. In each operator, a given TDMA table (i.e., a source node) is first chosen randomly. The first operator applies a random permutation between two emission dates of that TDMA table. The second operator applies a one time unit forward translation to the TDMA emission dates.

**Greedy parallel construction.** This procedure, called *constructSolutionPar* in Algorithm 2, greedily builds the origin/destination paths step by step and in a parallel way. Before this procedure can proceed, a set of routing tables must first be computed at the whole initialization. They embed the costs of the shortest paths from each location to all destination nodes in the original NoC graph. These tables are computed only once at the initialization of the main algorithm. The origin/destination paths are built step by step, i.e., time slot by time slot, and jointly. An image of the behavior of the algorithm is that of several waves that propagate in all directions at constant speed, from the source node until it reaches the destination node or not. Each iteration step corresponds to a given time-slot, and to the introduction of the next vertex into each path. The next router is chosen according to the minimization of its distance to the destination

node, and by verifying in the TEG that the corresponding arc is free for the message packets. When several choices are available, the next vertex is chosen randomly. Then, the arc status structure is updated in the TEG. The process continues until all messages have reached their destination node, or no vertex insertion is possible. Then, a path is said "constructed" or "blocked". The temporal complexity is  $O(N \times T^2)$  for planar graphs, or graphs with constant degree, and  $O(N^2 \times T^2)$  for general graphs, with *N* the number of nodes and *T* the emission cycle. This is a rough estimation. A node can be visited as soon as there is an available time-slot in some incident arc, then it can be visited at most  $O(N \times T)$  for general graphs and messages of size 1. The number of time-slot examinations at each node depends on the degree of the node and message size, then it is at most  $O(N \times T)$  for general graphs and a message of size *T*.

Dijkstra algorithm in TEG. We have designed a breath-first-search algorithm able to construct a one-to-one shortest path in an occupied time-expanded graph. Here, the TEG may contain "free" or "occupied" arcs as part of the input. The procedure is called *constructSolutionWithModifiedDijkstra* in Algorithm 2. The term Dijkstra is used in reference to the function of the algorithm, that produces a shorter path, and to the fact that it is a simplification of the classical Dijkstra algorithm with unit costs associated to the arcs. The algorithm is extended to take into account the cyclic nature of the problem and multi-packet traveling. Each iteration of the algorithm corresponds to a single time-slot increment, so that the path length increases of one time-unit. Thus, all the successors of a visited node are necessarily in minimum distance from the origin and will constitute the nodes to visit in the next iteration. A main simplification over the classical Dijkstra, is that we do not have to manage an unordered list of successors and select lower cost one. All successors are at lower cost necessarily and must be examined. As well as in Dijkstra's algorithm, multiple predecessor node choices are possible. In this case, only one of the possible predecessors is selected randomly. Then, only one single path is computed based on these random choices since the embedding metaheuristic only necessitates one random single path at each call of the Dijkstra algorithm. Randomness must allow diversity of the paths found due to multiple calls. The temporal complexity is  $O(N \times T^2)$  for planar graphs, and  $O(N^2 \times T^2)$  for general graphs. A node of the communication graph cannot be visited more than *T* times, otherwise the visit will constitute a loop in the TEG where nodes are duplicated *T* times. The number of time-slot examinations at each node depends on the graph degree and message size, then, it is at most  $O(N \times T)$  for general graphs and a message of size *T*.

Another important difference with a standard Dijkstra algorithm arises from the cyclic and multipacket nature of the shorter path problem. Here, arcs occupation must be dynamically updated into the TEG as the path is currently built. This is necessary in order to avoid self-conflicts for a given multi-packet message, since a header packet could hit queue packets at the same emission cycle or at different emission cycles. The situation can occur if the length of a path is larger than the period *T*, but it can also occur with a message with length sufficiently large such that the header packet could hit its queue packets according to a loop in the path, regarding the NoC communication graph. This dynamic change of TEG occupation status has a direct impact on the conditions of optimality. In the case of a single packet message, the algorithm guarantees optimality in time polynomial with the TEG size, that is, in pseudo-polynomial time over the CKPP instance size. This naturally follows from the breath-first-search algorithm behavior. However, in the case of a multi-packet message and due to dynamic updates of arc occupation,

optimality is no longer guaranteed. It arises that further arc/vertex choices should depend on the previous choices made. This is an indication of a more complex and specific problem. We actually do not prove some NPhardness for that specific subproblem, nor prove that a polynomial solution exist. At the moment of writing, we let the complexity analysis of that specific subproblem open. However, the less the number of packets, the more the probability to get an optimal one-to-one path. The shorter will be the path, the less will be the probability of conflict. Some possible solution that could be envisaged to guaranty optimality could be to delegate the single shorter path computation to the ILP method, once adapted to a given occupied TEG as an input. This could be envisaged in further work that would combine exact and heuristic methods.

### Local search improvement algorithms

We now detail the improvement procedure, called *improveSolution* in Algorithm 1. While the construction procedure generates diversified solutions starting from random initializations, the improvement procedure allows to perform more or less smaller steps in a smallest region of the solution space in order to transform non admissible solutions to admissible ones. The general scheme of an improvement procedure consists of embedding a neighborhood operator within some search strategies. We first detail the neighborhood operator, then we propose three versions for the search strategy.

The pseudo-code of the neighborhood operator, called *generateNeighbor*, is given in Algorithm 3. Because of the many paths constructions that need to be performed, we choose to operate at the level of paths deconstruction and reconstruction. The neighborhood operator simply suppresses a subset of the solution paths, with the *removeMessages* procedure, and hence tries to construct the solution again from that partial solution. To do so, it uses the greedy parallel construction procedure followed by the one-to-one Dijkstra algorithm, that were specified for the construction phase (cf. previous section). The number of removed paths stands for the neighborhood size, that can be more or less large. It is intended that the new paths, together with the previously "blocked" paths, would have a new chance to being completely reconstructed by the operator. Since many random choices are possible during path construction, the neighborhood can be seen as large, and the method interpreted as a "reparation" procedure, following the ruin and recreate principle as formulated by Schrimpf et al. [26].

---

#### Algorithm 3: generateNeighbor

**Input:**  $S \in \text{Solution}$ , nbMsg  $\in \text{Integer}$

**Output:** NewSolution

```

begin
  // Remove the nbMsg paths from S and update the TEG
  NewSolution ← removeMessages(S, nbMsg);
  NewSolution ← constructSolutionPar(NewSolution); // Greedy
  parallel
    construction
  NewSolution ← constructSolutionWithModifiedDijkstra(NewSolution); // Construct the remaining paths one-by-one in TEG
  return NewSolution;

```

---

Three versions of a search strategy are considered that embed the neighborhood operator. They are respectively a basic random walk search, that we call iterated random search (IRS), as specified in the pseudo-code of Algorithm 4, and two local search schemes, that are, a

greedy descent local search, also called first improvement local search (ILS-FI), specified in Algorithm 5, and a steepest descent local search, also called best improvement local search (ILS-BI), specified in Algorithm 6. The iterated random search simply makes evolve the current solution by performing a given number of successive neighborhood moves, and then selects the best solution encountered. The method is very simple since very few operations are performed at each iteration step. This can be seen on Algorithm 4. On the contrary, the two local search schemes involve two internal loops each. As shown in Algorithms 5 and 6, the outer loop controls the depth of the search.

---

#### Algorithm 4: iteratedRandomSearch

**Input:**  $S \in \text{Solution}$ , *maxImprove*, *K* number of messages

**Output:** BestSolution

```

begin
  count ← 0;
  while count < maxImprove do
    count ← count + 1;
    S ← generateNeighbor(S, rand(1, K));
    BestSolution ← selectBest(S, BestSolution);
  return BestSolution;

```

---



---

#### Algorithm 5: localSearch first-improvement (ILS-FI)

**Input:**  $S \in \text{Solution}$ , *K* number of messages

**Output:** BestSolution

```

begin
  BestSolution ← S;
  improvementFound ← true;
  while improvementFound /* depth of the search */ do
    count ← 0;
    improvementFound ← false;
    while
      count < neighborhoodSampleSize and not improvementFound
      /* pivot rule */ do
      count ← count + 1;
      S' ← generateNeighbor(S, rand(1, K)) /* neighborhood move */;
      if isBest(S', BestSolution) then
        BestSolution ← S';
        improvementFound ← true;
    S ← BestSolution;
  return BestSolution;

```

---



---

#### Algorithm 6: localSearch best-improvement (ILS-BI)

**Input:**  $S \in \text{Solution}$ , *K* number of messages

**Output:** BestSolution

```

begin
  BestSolution ← S;
  improvementFound ← true;
  while improvementFound /* depth of the search */ do
    count ← 0;
    improvementFound ← false;
    while count < neighborhoodSampleSize /* pivot rule */ do
      count ← count + 1;
      S' ← generateNeighbor(S, rand(1, K)) /* neighborhood move */;
      if isBest(S', BestSolution) then
        BestSolution ← S';
        improvementFound ← true;
    S ← BestSolution;
  return BestSolution;

```

---

The algorithms stop when no improvement has been found. Because all algorithms are random, there is no guarantee that a local minimum has been reached even if the algorithm does not make any progress. A given solution constitutes the pivot element around which are performed the neighborhood search. Hence, the inner loop, that is the difference between Algorithms 5 and 6, deals with the pivoting rule, that determines to which better neighboring solution to move to. When compared to the iterated random search above, note that a supplementary variable  $S'$  has to be added to allow testing the improvement move. In the greedy descent local search version, the first encountered best solution becomes the new pivot element. In the steepest descent version, it is the best solution within the whole neighborhood that must be selected as the new pivot element. Here, since the neighborhood is large and the method stochastic, the procedure only examines a random sample of the neighborhood set. The sample size is represented by the *neighborhoodSampleSize* parameter in Algorithms 5 and 6. In experiments, it is set to 100 neighbors to examine. Another important parameter is the neighborhood size itself, which is the maximum number of paths selected for suppression in a single neighborhood move. Its impact on performances will be investigated in experiments. It is set to a maximum of  $nbMsg$  paths suppressed, with  $K$  being the total message number. This means that at each move, the exact number of suppressions is chosen randomly between 1 and  $K$  paths. Note that this parameter is shared by both IRS and ILS methods.

## Experiments

To evaluate the proposed algorithms, five real-life problems and many random artificial derived instances are considered. They are all based on four different NoC topologies of increasing sizes, respectively named N1, N2, N3, N4 in Figures 2-7 in the appendix. The five real-life problem instances were stated by Dafali [7] as representative of concrete real-life applications. The TDMA specifications are detailed in Figures 2-7. Four parameters characterize each instance. Instance N1 is the smallest one, with  $(T,N,P,K) = (6,7,4,12)$ , whereas instance N4 is the largest one with  $(T,N,P,K) = (47,36,35,209)$ . Next, we will attach the  $(T,N,P,K)$  quadruplet to the instance name when necessary. Instances N2 and N3 are of medium sizes. The N3 network topology corresponds to two test cases, named N3A, N3B. In Figure 8, instance N3A has the first three TDMA tables specified beside the components  $IP_0, IP_1, IP_2$ . Instance N3B includes the three TDMA tables specified just below.

The proposed heuristics are developed in C++ and have been run on a PC Intel Core Duo 3.0 GHz computer, where only 1 core was used. All the tests are performed on a basis of 100 runs per problem instance. For each test case, we evaluate the average total length value and the average computation time based on 100 runs. The average computation times are reported in seconds. The length is expressed in time-slots. The algorithms are configured in such a way that they stop their execution once an admissible solution has been found. Hence, they are used as construction methods in this experiment. Here, the hard task is the ability to quickly generate admissible solutions with minimal lengths the number  $N$  of routers, the number  $P$  of IP components, i.e., the source-destination nodes, and the number  $K$ . In local search, the sample size as stated by the *neighborhoodSampleSize* parameter of Algorithm 4 is set to 100. Other algorithm parameters will be specified further in this section. To assess the reliability of the statistical results, 95% confidence intervals for the sample mean, based on standard deviation over the 100 runs, are computed when necessary.

We evaluate the performance of the methods within three sets of experiments. First, we study the impact of two important components, that are, the improvement procedure itself, and the addition of the one-to-one shortest path Dijkstra algorithm in TEG. In a second set of experiments, we present a comparative evaluation of the different methods, including the ILP solution and the iterated search methods, when applied to the five real-life problems. We also study the impact of the neighborhood size on performances. In a third set of experiments, we evaluate the robustness of the approach against many randomly generated traffic configurations, based on the original NoC topologies. We discuss the problem of generating valid instances in relation to a maximum flow problem, and evaluate the algorithm performances as the saturation traffic level, i.e., number of injected packets, increases.

## Impact of the main algorithmic components

In this section, we study the performance of different combinations of the construction/improvement operators and of the Dijkstra procedure. We use the iterated random search (IRS) method specified by Algorithm 1, in which the construction method *constructSolution* corresponds to Algorithm 2 and the improvement method *improveSolution* corresponds to Algorithm 4 to perform the experiments. Adding or not the Dijkstra component, we study the impact of construction and improvement operations when adjusting the *maxCount*, *maxConstruct*, and *maxImprove* algorithm parameters. The goal is to illustrate the importance of each added new algorithm component that improves both computation time and quality of solution substantially at the same time. The algorithm stops once an admissible solution is found or when the maximum number of iterations is reached.

Results are reported in Table 1. The procedure was executed on the N3A(9-15-10-27) instance of Figure 8, for four different combinations of the IRS loops. Results are averaged over 100 runs. Since it may be the case that no admissible solution is produced during a run, we report in a column named "Paths built", the average percentage of built paths related to the message number. We also report the average computation time in seconds in the column named "CPU time", and the number of feasible solutions obtained over the 100 runs in column "Feas. sol.". Numerical results are given within six rows for construction phase only, with construction respectively iterated 1, 1000, and 100000 times. For each configuration of the iteration number, results are reported with or without the Dijkstra component. The last configuration corresponds to the introduction of the improvement loop and reiteration of the construction/improvement phase. For this experiment, the algorithm parameters (*maxCount*, *maxConstruct*, *maxImprove*) will take the values (50, 1000, 1000). The outer loop reiterates 50 times the construction/improvement phases, each one executing 1000 internal iterations.

## Comparative evaluation on the real-life benchmark set

We now compare the different approaches considered in this paper, including the ILP solution, on the five real-life benchmarks. We also evaluate solutions for three different values of the neighborhood size. The three heuristics are specified in Algorithms 1-6 with their default parameters. In the iterated random search (IRS) method, the combination of construction/improvement operations are set to the values (*maxCount*, *maxConstruct*, *maxImprove*) = (50, 1000, 1000). In the iterated local search (ILS) methods, the first or best improvement local search replaces the random walk improvement procedure of the

	Config (it)		Path built (%)	CPU time (s)	Feas. sol. (%)
Construction only	maxCount=1	without Dijkstra procedure	54	0.001	0
		with Dijkstra procedure	81	0.001	0
	maxCount=1000	without Dijkstra procedure	79	0.06	0
		with Dijkstra procedure	96	0.03	9
	maxCount=100000	without Dijkstra procedure	86	6	0
		with Dijkstra procedure	100	6	100
Construction and improvement	maxCount=50 maxConstruct=1000 maxImprove=1000	without Dijkstra procedure	95	9.5	4
		with Dijkstra procedure	100	0.7	100

Table 1: Impact of the main algorithmic components on N3A instance.

			N1(6-7-4-12)	N2(8-9-7-28)	N3A(9-15-10-27)	N3B(9-15-10-24)	N4(47-36-35-209)	average	deviation
neighborhood large	IRS	time <sup>a</sup>	0.002	0.11	0.70	0.82	8.89	2.1	(±0.14)
		length	53.3	119.5	131.7	119.6	1565	398	(±3.5)
	ILS-FI	time <sup>a</sup>	0.001	0.15	0.69	0.84	7.25	1.79	(±0.1)
		length	53.5	117.8	129.8	118.6	1570	398	(±3.5)
	ILS-BI	time <sup>a</sup>	0.001	0.17	0.66	0.85	7.57	1.85	(±0.13)
		length	53.7	119.0	127.1	118.0	1568	397	(±3)
neighborhood medium	IRS	time <sup>a</sup>	0.003	0.11	0.58	0.68	8.71	2.01	(±0.16)
		length	52.92	121.12	132.05	120.87	1572.80	400	(±3.48)
	ILS-FI	time <sup>a</sup>	0.001	0.18	0.74	0.81	6.97	1.74	(±0.11)
		length	52.86	118.24	128.70	117.37	1568.66	397	(±2.74)
	ILS-BI	time <sup>a</sup>	0.00	0.18	0.74	0.87	7.06	1.77	(±0.10)
		length	54.66	118.48	127.72	117.88	1574.62	399	(±2.91)
neighborhood small	IRS	time <sup>a</sup>	0.002	0.17	0.61	0.61	6.50	1.58	(±0.08)
		length	52.8	118.9	132.9	120.3	1577	400	(±3.89)
	ILS-FI	time <sup>a</sup>	0.001	0.22	1.44	1.81	7.35	2.16	(±0.12)
		length	53.73	119.88	130.85	118.84	1566.04	398	(±3.25)
	ILS-BI	time <sup>a</sup>	0.001	0.20	1.37	1.44	7.45	2.09	(±0.13)
		length	54.57	119.24	129.15	118.80	1567.36	398	(±3.19)
ILP	first-sol	time <sup>a</sup>	5	428	-	-	-	-	-
		length	57	133	-	-	-	-	-
	opt-sol	time <sup>a</sup>	111	1464	-	-	-	-	-
		length	42	76	-	-	-	-	-

Table 2: Comparative results on the whole benchmark set.

<sup>a</sup> Time per run in Intel Core Duo (3.0 GHz) seconds (1 core used), C++ program.

<sup>b</sup> Time per run in Intel core i3-2350M (2.30 GHzx4) seconds, GLPK program.

IRS. This leads to the versions ILS-BI and ILS-FI respectively. The size of the neighborhood sample examined at each pivoting operation, is set to *neighborhoodSampleSize* = 100. The size of the neighborhood itself is set to the maximum *nbMsg* of paths that can be suppressed and rebuilt at each move. The choice is random between 1 and *K*. We first report results for that default configuration, then we study the effect of diminishing the neighborhood size.

Comparative numerical results are reported in Table 2. The first row indicates the instance name and its characteristic parameters (T,N,P,K), respectively period of cyclic emission, router number, IP number, and message number. The other rows present the results for the three heuristics with their neighborhood size and two configurations of the ILP solver. For each method, are reported the computation time and the total length, averaged on a basis of 100 runs per instance. The test cases are ordered by increasing size. The N1 instance is the smallest one, whereas the N4 the larger one.

The last two columns respectively report the average value over the whole benchmark set and the 95% confidence intervals computed based on standard deviation. We can observe that the three heuristics produce solutions with similar length values, then, of similar quality. A slight improvement of the computation time can be observed for ILS methods against IRS. However, there is place for improvement for the ILS methods, since as usual for heuristics [24], complete copy of data structures could be avoided by a better implantation of the local neighborhood operation. The ILP was solved using the GLPK1 integer linear programming solver. Two configurations of the ILP solver are considered and the results reported in the last four rows in Table 2. In the two rows "first-sol", are reported the values corresponding to the first admissible solution obtained during a run. This solution is not optimal. In the two rows "opt-sol", are reported the optimal solutions found. Only the two smallest instances were solved successfully by the GLPK program within a reasonable amount of computation time, that was set to a



IIS-FI							
random instances	MT-real(%)	MT-rand(%)	MT max(%)	time(s)	length	built(%)	admis.(%)
N1-rand(6-7-4-12)	100	100	100	0.002	53	100	100
N2-rand(8-9-7-28)	100	100	100	0.17	117.5	100	100
N3A-rand(9-15-10-27)	90	87.86	90	0.49	126.9	100	100
N3B-rand(9-15-10-24)	90	85.54	89	0.50	117	100	100
N4-rand(47-36-35-209)	26.02	39.97	40	618.45	1405	93.86	0
average	81.20	82.67	83	123.92	363.9	98.77	-
deviation	0.0	(±0.1)	0.0	(±0.8)	(±2.4)	0.0	-

Table 3: Results on random instances.

maximum of 48h for the largest instances. It is worth noting that the ILP model generates a huge number of variables and constraints to be tackled by the solver as the problem size grows. As an example, the N3B instance case roughly produces 31 million variables with 5 hundred thousand constraints. According to the small instance cases N1 and N2 that were solved exactly, the heuristic method generates feasible solutions roughly at least 2000 times faster than the ILP solver. We think that these results highlight the contrast of behaviors when respectively dealing with an exact and heuristic method.

The two default parameter values of the neighborhood sample size and neighborhood size, respectively, were set empirically after a first round of experiments not reported in this paper. It appeared that increasing the sample size made the ILS-BI clearly worst performing, since it examines many more candidate neighborhood solutions at each pivoting operation. About the neighborhood size itself, i.e., the maximum number of paths suppressed/rebuilt at each move, we try now to better evaluate its impact. We diminish the neighborhood size, from actually all paths that can be suppressed (large neighborhood), to half of the paths suppressed (medium neighborhood), and to exactly 2 paths suppressed (small neighborhood). We perform a complete execution of the three heuristics on the whole benchmark set, executing 100 runs per instance, for each of the three neighborhood sizes. Average results are shown in Figure 8 within 95% confidence interval. The two graphics in the figure let us evaluate the trade-off between solution quality (length value) and computation time. The graph on the left shows that length values are rather similar. Since all confidence intervals overlap, the differences in the lengths look not statistically significant. On the contrary, some disparities appear

between computation times, as shown in the right graph in the figure. The first point to note is the similitude of behaviors between the ILS-BI and ILSFI methods as the neighborhood size diminishes. This is not surprising since the methods are similar and examine a relatively small sample in the neighborhood. The second point to note is the contrast of behaviors between ILS and IRS methods. While the former approach performs significantly faster with a large neighborhood size, the situation is inverted with a small neighborhood size since the latter becomes significantly faster than the former. This casts a light on the intensification/diversification tradeoffs into the methods. The ILS searches for improvements around a given pivot element which restricts the search region. But increasing the neighborhood size augments diversification. On the contrary, the IRS performs random steps in random directions starting from a given solution. This naturally increases diversification. But decreasing the neighborhood size restricts the search region. With regards to the trade-off between computation times and lengths, it seems that all configurations yield similar quality results, whereas fastest computation times occur for the IRS with small neighborhood, or ILS methods with medium or large neighborhood. Confidence intervals indicate that these last configurations are the best ones.

The numerical results in Table 2 allow us to compare ILS-FI and ILS-BI regardless of the neighborhood size and considering simultaneously the time and the length, ILS-FI performances are the best on N1, N2 and N4. However, those of ILS-BI are the best on N3A. ILS-FI behaves better on small and large instances while ILS-BI performs better on instances that we could describe as medium. This confirms the best behavior of ILS-FI on many instances. In these

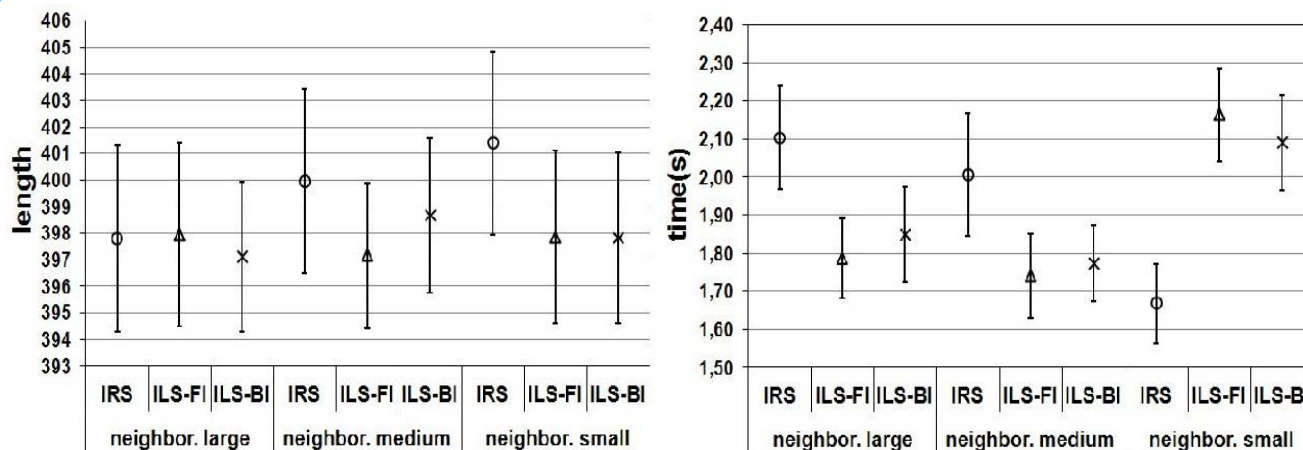


Figure 2: Impact of the neighborhood size on lengths and execution times.

experiments, our main goal was to give an insight of the important contrast in computation time required when respectively dealing with exact and heuristic methods. We should mention that further work could adequately combine the benefit of both methods following the spirit of "matheuristic" approaches that aim at dealing with ILP acceleration according to the quick generation of feasible solutions by heuristic procedures. We expect that such adequate combinations could be studied in the future starting from the preliminary work presented in this paper.

### Evaluation on random instances

We first detail the random instance generator, then we evaluate the local search approach on these randomly generated traffic configurations. The section also illustrates the designer's problem of bandwidth decomposition and bridging the inaccessibility of many real-live problem instances.

### The random instance generator

To evaluate the robustness of the method against a large range of traffic communication configurations, we have developed a generator of random instances. Based upon an application communication graph that specifies the inter-communications links opened between a set of IP components, the goal is to generate the quantity of packets that define the allocated bandwidth between each pair of source-destination nodes. In other words, given a set of  $K$  source-destination nodes  $(i, j)$  of a CKPP instance, the goal is to generate random packet quantities  $x_{ij}$  in a way consistent with the cyclic nature of the problem. Since source-destination messages are emitted based on a temporal cycle of length  $T$ , two types of constraints are to be satisfied. They concern emitters and receivers to avoid trivially infeasible instances respectively. They state that the allotted  $T$  time-slots must be shared by emitters as well as by receivers. More formally, we state the problem of bandwidth decomposition as follows.

Let  $\Pi = \{1, 2, \dots, P\}$  being the set of IPs, which may be either source or destination nodes. We define the application communication graph (ACG) as a matrix  $C = (c_{ij})$ ,  $1 \leq i, j \leq P$ , such that  $c_{ij} \in \{0, 1\}$  whenever  $i \neq j$ , and  $c_{ii} = 0$  otherwise. An entry  $c_{ij}$  is such that  $c_{ij} = 1$  if an open communication link exists from  $i$  to  $j$ ,  $c_{ij} = 0$  otherwise. Given an ACG, the model involves variables  $x_{ij}$  indicating the number of packets associated to each source-destination link  $(i, j)$ . Let  $s$  be a lower bound on the number of packets emitted by a given source. The generation of traffic instances, that we call Traffic Injection Problem (TIP), consists in generating many and diverse random instances for the CKPP.

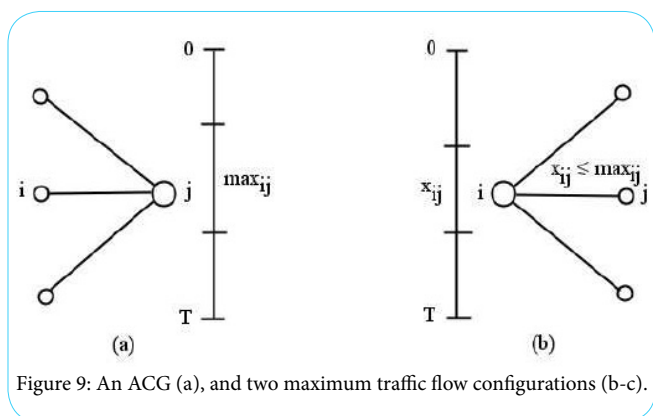


Figure 9: An ACG (a), and two maximum traffic flow configurations (b-c).

We are here concerned with the generation of many flow configurations of relatively good quality, rather than with necessarily generating a single maximum flow configuration. The quantity  $\sum_{ij} c_{ij} x_{ij}$  is the total flow that is injected in the network during the period  $T$ . In order to measure the saturation level of the emitters as in [27], we define the message throughput (MT) as the quantity  $(\sum_{ij} c_{ij} x_{ij} / PT) \times 100$ , that is, the percentage of the emitted packets related to the available time-slots at each source node. Figure 9 illustrates the problem of instance generation on a simple case. In (a) the ACG is shown, whereas in (b) and (c) two optimal configurations of the traffic flow are shown. We should note that, with a threshold of  $s = 0$ , optimal solutions may be unsatisfactory since some IPs may have no packets to communicate, as in (b). The goal is rather to share and decompose the flow between emitters/receivers to allow to communicate, as in (c). It is worth noting that the maximum flow value first depends on the ACG structure, regardless of the NoC topology. As an example, the MT cannot exceed 66,6% in the ACG in Figure 9. This example illustrates how the guaranteed traffic approach constrains the bandwidths to be carefully settled.

To generate many random traffic instances, we proceed in two main steps as shown with Figure 10. First, as illustrated in (a), we randomly partition the receiver bandwidth between its related emitters. The random choices are done according to uniform sampling. The values obtained at each receiver node define the maximum amount of traffic that a related emitter node can send to that receiver. Second, as in (b), we adjust proportionally these maximum bandwidths at the level of each emitter, while trying to maximize the number of packets sent. At each step, a difficulty arises because traffic values are integer values rather than fractional values. Hence, fractional values are floored to their nearest integer, and the lost packets are randomly re-injected one by one to the different bandwidths, taking care of the maximum available time-slots for each emitter/receiver. The random choices also take care of the threshold  $s$ , in such a way that at least  $s$  packets are sent by each emitter node. For the experiments, this threshold is set to  $s = 2$  packets, since messages contain at least a header packet followed by a data packet. Following part of the results in 3, we note that the MT is slightly inversely proportional to the size of  $T$ .

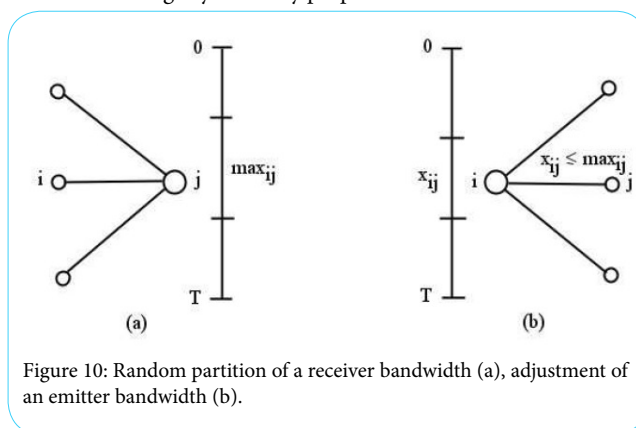


Figure 10: Random partition of a receiver bandwidth (a), adjustment of an emitter bandwidth (b).

### Performances of local search against random instances

We evaluate the performance according to a wide range of random traffic configurations. Starting from the five real-life test cases and their respective communication graphs, we randomly generate new traffic values following the random process specified above. For each test case, we generate 100 new traffic instances with a threshold of two packets per message. Then we run the algorithm on these instances and evaluate the average value over these 100 runs. Results are presented in Table 3.

Only the local search first-improvement is used. Two columns “MT-real” and “MT-rand” respectively report the message throughput of the real-life instances and of their random instances counterpart in average. The column “MT max” gives the maximum MT over the 100 generated instances. In addition to the execution time in seconds and length value, are reported the percentage of built paths and the percentage of admissible solutions found over the 100 runs. On the one hand, we can note that except the N4 test case, all the instances are solved successfully, and in a similar or shorter computation time than for the original real-life instances reported in Table 2. We can also note that, except for the N4 test case, the random MTs are all similar or inferior than those of the real-life instances. On the other hand, we can note that the N4 instance is never solved successfully, since only 94% of the paths are built on average, and no admissible solution was found. This should be explained by the higher random MT (39.97%) which is 50% higher than for the real case instance (26%).

To evaluate to which extent the algorithm performs efficiently as a function of MT, we conduct new experiments. We progressively suppress packets from messages in order to gradually decrease the MT value. The results are shown on the two graphics of Figure 11. On the left part, are respectively shown the percentage of admissible paths and solutions found. On the right part, are indicated the related computation times. We can observe that 100% of the paths built are obtained below an intermediate MT level of roughly 30%. This means that the application can increase the volume of data exchanges from 26% initially to 30%, while preserving the actual NoC topology. As in a bin packing problem, packets are packed in the transmission links until some overload occurs. Hence, this study casts a light on the problem of network dimensioning, as the designer adjusts the packet quantities, while trying at the same time to guarantee contention-free routing, especially for large scale NoC.

### Conclusion

We have presented a combinatorial optimization problem which models conflict-free routing in a network-on-chip with guaranteed traffic. Based on TDMA techniques, the problem can be seen as an extension of a K-shortest paths problem combined with a bin packing problem, where time plays an important role. Messages are emitted in a cyclic way and the problem goal resides in the efficient allocation of the available time-slots for conflict-free routing, while minimizing path lengths. The solution methods are based on a time-expanded

graph structure able to memorize time-slot occupation. An ILP model is stated for the problem, but only very small size instances were solved exactly and successfully in a reasonable amount of time. Three heuristic search methods were presented for attempting to generate feasible solutions fast. They were applied on a set of real-life benchmarks and on many randomly generated traffic configurations. All the heuristics methods were able to find feasible solutions on the real-life benchmarks in few seconds. Experiments show that the addition of a one-to-one path Dijkstra procedure into the local search clearly leads to a great improvement on both length minimization and computation time. Also, the generated random instances helped us determine the limits of the approach as the message throughput increases. The main limit of this GT traffic modeling is the lack of bandwidths flexibility between IPs that are fixed once and for all. It is resulting in an over-dimensioning of network to satisfy the worst case conditions. Making flexible bandwidth between the IPs is a track of future research. The idea would be a dynamic reconfiguration of the routing at execution time. Several TDMA tables would be introduced in emitters. And at the beginning of each cycle, each emitter could change the transmission table. This change should modify bandwidth and communication paths. The challenge would be, at the time of execution, the potential reallocation of vacant time-slots during dynamic and mutually exclusive changes of TDMA from a same emitter, without disturbing or stopping the system.

### Competing Interests

The authors declare that they have no competing interests.

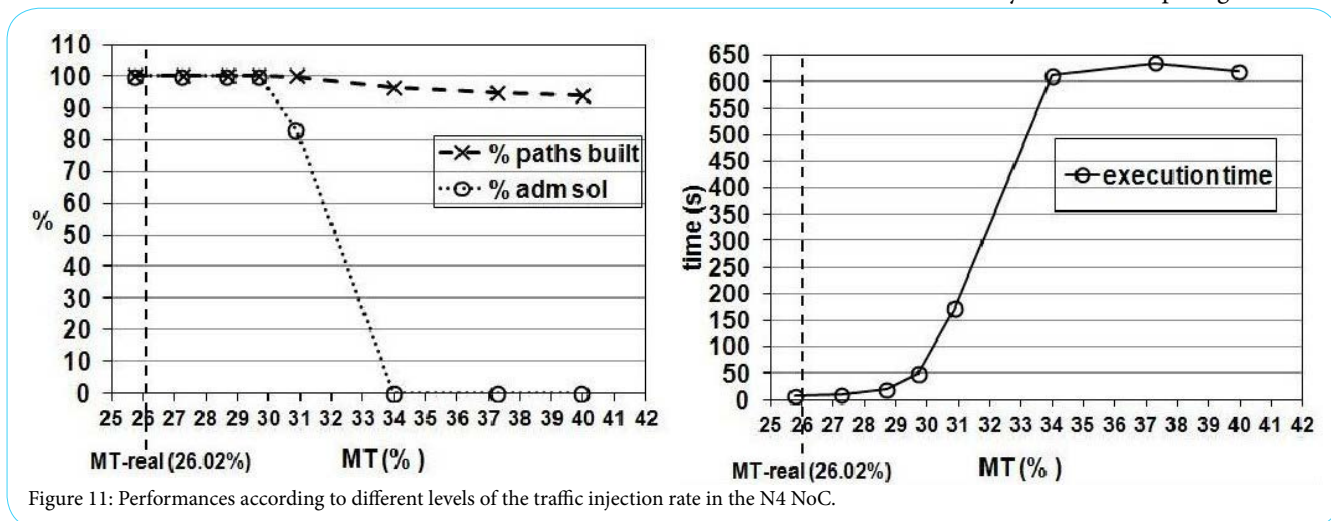


Figure 11: Performances according to different levels of the traffic injection rate in the N4 NoC.

## References

- DallyW, Towles B (2001) Route packets, not wires: On-chip interconnection networks. Proceedings of the 38th Design Automation Conference.
- Benini L, Micheli G (2002) Networks on Chips: A New SoC Paradigm. Computer 35: 70-78.
- Agarwal A, Iskander C, Shankar R (2009) Survey of network on chip (NoC) architectures and contributions. Journal of Engineering, Computing, and Architecture 3: 1-15.
- Goossens K, van Meerbergen J, Peeters A, Wielage R (2002) Networks on Silicon: Combining Best-Effort and Guaranteed Services. Design, Automation and Test in Europe Conference and Exhibition, Proceedings, pp 423-425.
- Cong J, Liu C, Reinman G (2010) ACES: application-specific cycle elimination and splitting for deadlock-free routing on irregular network-on-chip. In: Proceedings of the 47th Design Automation Conference, DAC '10, ACM New York, USA, pp 443-448.
- Evain S, Diguët JP (2007) Efficient space-time noc path allocation based on mutual exclusion and prereservation. In: Proceedings of the 17th ACM Great Lakes symposium on VLSI, Italy.
- Dafali R (2011) Conception des r'eseaux sur puce reconfigurables dynamiquement. Phd dissertation, University of South Brittany, France.
- Millberg M, Nilsson E, Thid R, Jantsch A (2004) Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In: Proceedings of the conference on Design, automation and test in Europe.
- Marescaux T, Brick'e B, Debacker P, Nollet V, Corporaal H (2005) Dynamic time-slot allocation for QoS enabled networks on chip. In: 3rd Workshop on Embedded Systems for Real-Time Multimedia, pp 47-52.
- Hansson A, Goossens K, Radulescu A (2007) A unified approach to constrained mapping and routing on network-on-chip architectures. In: CODES+ISSS'05 Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp 75-80.
- Stefan R, Goossens K (2011) Enhancing the security of time-division-multiplexing networks-on-chip through the use of multipath routing. In: Proceedings of the 4th International Workshop on Network on Chip Architectures, ACM New York.
- Ge F, Wu N (2010) Genetic algorithm based mapping and routing approach for network on chip architectures. Chinese journal of Electronics 19: 91-96.
- Lin CA, Hsin HK, Chang EJ, Wu AYA (2013) ACO-based fault-aware routing algorithm for networkon-chip systems. In: SIPS 2013 Proceedings, pp 342-347.
- Benmessaoud Gabis A, Koudil M (2016) NoC routing protocols-objective-based classification. Journal of Systems Architecture 66: 14-32.
- K'ohler E, Langkau K, Skutella M (2002) Time-expanded graphs for flow-dependent transit times. European Symposium on Algorithms 2461: 599-611.
- Kolliopoulos SG (2007) Edge-disjoint paths and unsplittable flow. In: T. F. Gonzalez (ed) Handbook of Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC.
- Karp RM (1975) On the computational complexity of combinatorial problems. Networks 5: 45-68.
- Lynch JF (1975) The equivalence of theorem proving and the interconnection problem. ACM SIGDA Newsletter 5: 31-36.
- Gonzalez TF, Serena D (2004) Complexity of pairwise shortest path routing in the grid. Theoretical Computer Science 326: 155-185.
- Desrochers M, Soumis F (1988) A generalized permanent labelling algorithm for the shortest path problem with time windows. INFOR 26: 191-212.
- M'ohring RH, K'ohler E, Gawrilow E, Stenzel B (2004) Conflict-free Real-time AGV Routing. Operations Research Proceedings, pp 18-24.
- Ioachim I, G'elinas S, Soumis F, Desrosiers J (1998) A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Networks 31: 193-204.
- Powell WB, Chen ZL (1998) A Generalized Threshold Algorithm for the Shortest Path Problem with Time Windows. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science.
- Ford LR, Fulkerson DR (1958) Constructing maximal dynamic flows from static flows. Operations Research 6: 419-433.
- Johnson DS, McGeoch LA (1997) The Traveling Salesman Problem: A Case Study in Local Optimization.
- Schrimpf G, Schneider K, Stamm-Wilbrandt H, Dueck V (2000) Record Breaking Optimization Results Using the Ruin and Recreate Principle. J of Computational Physics 159: 139-171.
- Pande PP, Grecu C, Ivanov A, Saleh R (2005) Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures. IEEE Transaction on Computers 54: 1025-1040.