A Note on Models, Algorithms, and Data Structures for Dynamic Communication Networks

Afonso Ferreira, Laurent Viennot

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A note on models, algorithms, and data structures for dynamic communication networks

Afonso Ferreira and Laurent Viennot

N° 4403

March 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4403--FR+ENG

A note on models, algorithms, and data structures for dynamic communication networks

Afonso Ferreira* and Laurent Viennot†

Thème 1 — Réseaux et systèmes
Projets Hipercom/ARC Soleil Levant et Mascotte

Rapport de recherche n° 4403 — March 2002 — 8 pages

Abstract: New technologies and the deployment of mobile and nomadic services are driving the emergence of complex communications networks, that have a highly dynamic behavior. Modeling such dynamics, and designing algorithms that take it into account, received considerable attention recently. In this note, we discuss a formal generalization of dynamic graphs, the evolving graphs, which aims at harnessing the complexity of an evolving setting as yielded by dynamic communication networks. We argue that evolving graphs are of great help when dealing with fixed-schedule networks. Moreover, we show how to exploit our model with networks where short time prediction is available.

Key-words: Mobile networks, models, algorithms, dynamic networks, dynamic graphs, evolving graphs, shortest paths

* CNRS — I3S & INRIA Sophia-Antipolis, INRIA Sophia-Antipolis, 2004, route des Lucioles, BP 93, F-06902 Sophia-Antipolis Cedex, France, Afonso.Ferreira@sophia.inria.fr
† INRIA Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France, Laurent.Viennot@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Modèles, algotithmes et structures de données pour les réseaux de communication dynamiques

Résumé : Les nouvelles technologies et le déploiement de services mobiles et nomades impliquent l'émergence de réseaux de communication complexes au fort comportement dynamique. Modéliser cette dynamique et concevoir des algorithmes qui la prennent en compte a été largement étudié récemment. Dans cette note, nous proposons une formalisation des graphes dynamiques, les graphes évolutifs, dans le but d'adresser la complexité due au contexte évolutif. Cette modélisation s'avère utile dans le contexte de réseaux dont on connaît l'évolution à l'avance. Elle peut même s'étendre au cas de réseaux où une prévision à court terme est possible.

Mots-clés : Réseaux mobiles, modèles, algorithmes, réseaux dynamiques, graphes dynamiques, graphes évolutifs, plus courts chemins

1 Introduction

The advent of new technological communication networks, such as the Internet and ad-hoc radio networks, highly motivates the study of several facets of the dynamic behavior of such networks. The underlying mobility of users and/or relays is just one of the factors that contribute to their dynamics. Others include varying link congestion, node and link faults, and components addition and deletion.

The related domain of dynamic graphs has been extensively studied in the past decade (see [3] and references therein). In this case, some property (e.g. a minimum spanning tree) of an input graph is computed, and algorithms are given in order to maintain such a property after the graph is modified (typically an addition/deletion of a vertex/edge). One could say that the discrete step is one modification undergone by the graph.

Recently, the field of communications in dynamic networks, i.e., networks with evolving characteristics, received considerable attention [4, 5, 6, 1]. From a theoretical standpoint, an excellent state of the art on models and techniques for communications in such networks can be found in [8]. We refer the interested reader to its extensive bibliography, where issues concerning connectivity, routing, and admission control are addressed.

In this paper, we try to introduce a framework that goes beyond the classical routing scheme where a node can send a packet to another at time t if there exists a path linking them at that time. Indeed, we try to take into account the evolution of the connections in the network during the forwarding of the packet. The route is thus calculated in an evolving context. We formalize this by defining evolving graphs and giving algorithms for computing shortest paths in this context.

Our work in this area focuses on the design of models and algorithmic techniques that can harness the complexity of an evolving setting as yielded by dynamic communication networks. In this note, we give in the next section a simple but powerful model that captures most of the characteristics of such networks. Then, we exemplify in Section 3 its use by the computation of shortest paths under several settings. Optimization of the data-structure for representing evolving graphs is then studied in Section 4. Our model can be directly applied to fixed-schedule networks where the evolution of the network is known in advance. However, Section 5 shows that it can also apply to networks where a short time prediction can be made. We close this note with ways for further research.

2 On models

Papers in this area often define a network as a graph, along with a fault probability, p, for each edge. At each time step, each edge is kept independently at random with probability p. Problems studied under such a random-fault model are, for instance, fault-tolerant routing and the computation of large fault-free connected components [8].

A good example of stochastic models for dynamic networks appears in papers presenting research on the graph of the Web, like in [5]. A recent effort in the direction of taking into account the dynamics of networks was shown in [10], where routing in a moving-robots network was discussed.

In this paper, we try to formally define a model capturing most of the characteristics of dynamic networks, in which to study graph-theoretic properties and graph algorithms.

Definition 1 *A dynamic network $\mathcal{N}$ shall be considered as a simple time-dependent discrete dynamical system running from time $t_1 = 1$ until time $t_\mathcal{T} = \mathcal{T}$. It is modeled as a directed graph $G(V, E)$, along with a* presence *matrix $P_E[(u, v), i]$, indicating whether (u, v) is present at time step t_i, for each arc (u, v), and another* presence *matrix $P_V[u, i]$, indicating whether u is present at time step t_i, for each vertex u. The network at time t_i, denoted $G_i(V_i, E_i)$ is represented by a subgraph of G, denoted $G_i(V_i, E_i)$, and obtained by taking those vertices and arcs for which their corresponding $P[i]$'s indicate they are to be present. We call these graphs* evolving. *Let $|V| = N$ and $|E| = M$.*

Notice that in the random-fault model, the image often taken is that of a given network where nodes and communication links may disappear because of faults, while in the graph-of-the-web model, one is interested in predicting its topology. In the *evolving graphs* model above, it is made clear that between two subsequent time steps, *any* changes may happen, with the possible creation and/or deletion of any number of vertices and arcs. The model, as it is described above, imposes a fixed schedule for the topology of the subgraphs. Therefore, it is interesting to notice that evolving graphs could also be used to model the behavior of fixed-schedule dynamic transport networks, like railway networks.

We further remark that an evolving graph could also be defined just as a sequence of subgraphs, whose description becomes available at the different time steps. In such a dynamic setting, the topology schedule

would still be fixed, but unknown beforehand, thus preventing any kind of pre-processing. In the following we will exemplify their use, under the above definition, on the modeling of dynamic communication networks, through a simple and clear case study.

3 Computing shortest paths in evolving graphs

To exemplify the use of this model, we show how to compute a shortest path in $\mathcal{N}$, under the hypothesis that the matrices P are boolean. Thus, an arc $(u,v) \in G$ is such that $(u,v) \in G_i$ if and only if $P[(u,v),i] = 1$, and respectively for the vertices. Given a source vertex $A \in G$, we shall show how to compute all shortest paths from A.

First, remind that the usual Dijkstra's algorithm [2] proceeds by building a set S of *closed* vertices, for which the shortest paths have already been computed, then choosing a vertex u not in S whose shortest path estimate, $d(u)$, is minimum, and adding u to S, ie, closing u. At this point, all arcs from u to $V - S$ are *opened*, ie, they are examined and the respective shortest path estimate, d, is updated for all end-points. In order to have quick access to the best shortest path estimate, the algorithm keeps a min-heap priority queue Q with all vertices in $V - S$, with key d. Note that d is initialized to ∞ for all vertices but for A, which has $d = 0$.

The main problem to implement such an algorithm in an evolving setting is how to keep the correct values of the shortest path estimate at the time we will need them. Indeed, we need to know the vertex with least estimated distance to the set S only at *the appropriate time step.*

Furthermore, since we are dealing with communication networks, several hypotheses may be made with respect to the actual communication of information. A basic one, valid throughout this note is that we shall consider packet networks. Hence, transmitting one piece of information means transmitting one packet over one arc. Other hypotheses will be specified case by case. We assume the input graph is given as linked adjacency lists, with the sorted schedule attached to each neighbor, indicating the time steps where that arc is alive. The head of each list is a vertex with its own sorted schedule list attached. Notice that this pre-processing of the presence matrices takes time $O((M+N)\mathcal{T})$.

Scenario 1 *Packet transmission time is normalized so as to coincide with the duration of a time step* $\Delta(t) = t_{i+1} - t_i$*, which is constant for all* i*. There is information conservation, ie, if a vertex disappears and then reappears, it still has the received informations.*

This is the easiest case, because all the weights on the arcs can be seen as being unitary. Notwithstanding, in opposition to Dijkstra's algorithm, we may have to open arcs from arbitrary closed nodes in case they appear in an upcoming G_i.

Variations of the modified Dijkstra's algorithm described in [10] are described below.

Algorithm 1

1. *Make all* $d(v) = \infty$*, but for* $d(A) = 0$*. Initialize a min-heap* Q *with a record* $(A, d(A))$ *in the root. Put in* Q *a dummy record* $(dummy, d(dummy) = \infty)$.
2. $i \leftarrow 0$.
3. *While* $d(root(Q)) \neq \infty$ *do*
   (a) *While* $d(root(Q)) = i$ *do*
      i. *Extract* x*, the vertex at* $root(Q)$.
      ii. *Delete* $root(Q)$.
      iii. *Traverse the adjacency list of* x*, computing* $f(v)$ *the first valid schedule time for each open neighbor* v *(ie, the first schedule time which is largest than* i*).*
      iv. *Update all* $d(v)$ *with* $f(v)$*. If* $d(v)$ *was* ∞*, then insert* v *in* Q.
      v. *Update* Q.
      vi. *Close* x *and insert it in the shortest paths tree.*
   (b) $i \leftarrow i + 1$.

The shortest path is found by traversing the shortest paths tree back from a destination to A. In case two successive labels differ by more than 1, this implies that the shortest path yields a forced stay of the information in that vertex for a number of steps, until the connection is established to its successor in the tree.

Analysis. We can see that, starting from A, the algorithm examines all its out-neighbors ($\Gamma^+(A)$), and for each one there is a table look-up to find the first valid schedule time, plus a heap update. Therefore, for each closed vertex, the algorithm performs $O(\log \mathcal{T} + \log N)$ operations. Hence, the total number of operations is at most $O(\sum_{v \in V}[\Gamma^+(v)(\log \mathcal{T} + \log N)]) = O(M(\log \mathcal{T} + \log N))$.

The space used is proportional to the size of the adjacency linked lists, plus the size of the arcs schedule lists. Therefore, the total size of the lists is $O(M + M\mathcal{T}) = O(M\mathcal{T})$ in the worst case. □

As we said, the hypothesis above represents the easiest setting. We will now try and relax it point by point.

Scenario 2 *One time step* $\Delta(t)$ **allows for the traversal of** k **arcs in the graphs**.

Actually, relaxing the traversal time constraint in this sense does not cause a big problem. It is like each column of the arc presence matrix had been copied k times. A simple counting strategy could be used to implement it.

Analysis. The same as above, but for the parameter k. □

Scenario 3 Crossing an arc takes arbitrary time, *given as a positive cost* $c_{(u,v)}$ *associated to them.* **The subgraphs** G_i **are only allowed to grow** *(ie, no arc disappears).*

Under the assumption that no arc disappears [9], then the algorithm above correctly computes all shortest paths, since a vertex x is closed at time step t_k if and only if $d(x) \leq t_k$, ensuring that no shorter path can be found for it. The only modification would be when updating the values of all $d(v)$, $v \in \Gamma^+(x)$, in order to take $c_{(x,v)}$ into account.

Analysis. Same as for Scenario 1. □

Now, we look at the interesting case where the communication links may be arbitrarily up or down, in the fixed topology schedule.

Scenario 4 *Crossing an arc takes arbitrary time, given as a positive cost* $c_{(u,v)}$ *associated to them.* **Arcs may disappear.**

The problem now is that an arc may disappear before the information had enough time to cross it. One solution for this problem is to first get for each arc (u, v) a new schedule list given by pairs (b, l) indicating beginning and duration times for each of its appearances. Then, compare this pair with its cost $c_{(u,v)}$, and only keep those pairs for which $l \geq c_{(u,v)}$. Finally, run Algorithm 1 with this new arcs schedule list. Again, another modification would be when updating the values of all $d(v)$, $v \in \Gamma^+(x)$, in order to take $c_{(x,v)}$ into account.

Analysis. Same as for Scenario 1. □

Scenario 5 There is no longer information conservation, *ie, if a vertex disappears and then reappears, it may have lost the received informations.*

In this case, a node may have to be reached several times before we find out that it was part of a shortest path. Hence, we should use the vertex presence matrix in order to get the vertex schedule list for each vertex. Recall that Algorithm 1 first extracts the current root of the heap Q – say vertex x –, then traverses its adjacency list, and finally closes it. In order to cope with the loss of information, during the adjacency list traversal, $f(v)$ should only be computed for those neighbors either **(1)** open with a first valid schedule transmission time smaller than the last time step x will be still alive in the present time interval, or **(2)** closed but with the last time step v will still be alive smaller than $d(x)$.

Notice that this version of Algorithm 1 will not produce a shortest path tree, since a node may have to be reached several times in order to ensure information delivery. Therefore, this version of Algorithm 1 builds a "directed tree", ie a digraph with no cycles with monotonically increasing arcs. In order to compute the shortest paths, we take the inverse path in this digraph from any destination back to A. In each vertex we take the inverse arc with the highest label which is smaller than the current distance to A, and mark the arc as done.

Analysis. It is easy to see that the cost of the whole procedure now depends on the maximum number of steps a node may reappear in the network, say K_m. Therefore, this version of Algorithm 1 performs at most $O(K_m M(\log \mathcal{T} + \log N))$ operations.

□

In the next section, we show how a different data structure allows for a more efficient algorithm for the scenarios above, and even for a scenario where time is not required to be discrete.

4 Optimizing the evolving graphs data structure

To get a better performance, the following data structure may be used to represent an evolving graph, instead of a graph and a presence matrix.

Definition 2 *A dynamic network $\mathcal{N}$ may be modeled by a set of vertices V and a set $\mathcal{E}$ of edge events. An edge event $(u, v, t, state)$ represents either a link creation (say, when state = up) or deletion (when state = down) from node u to node v at time t. Let $|\mathcal{E}| = \mathcal{M}$.*

Note that this data structure can be significantly smaller than the adjacency matrix. Its size is proportional to the networks dynamics (if there are few edge events, the data structure remains small). Moreover, the subgraphs G_i may be sparse but their union G may be very dense. In any case, the number of edge events is bounded by the size of the presence matrix $M\mathcal{T}$, but it is highly improbable that this upper bound is reached since it would imply that all edge states change at each time slot.

Notice that this definition is equivalent to Definiton 1. Edge events occur when $P_E[(u,v),i] \neq P_E[(u,v),i-1]$. The edge event set can thus be easily computed in linear time from the presence matrix. Conversely, the presence matrix may be computed from the edge event set in linear time, with respect to size of the result when the edge events are sorted lexicographically ($(u,v,t,state) < (u',v',t',state')$ if $u < u'$ or $u = u'$ and $t < t'$). When the edge events are not given in that order they may be sorted in $O(\mathcal{M} \log \mathcal{M})$ time. Alternatively, bucket sorting on times and then on sources allow to them lexicographically in $O(N + \mathcal{T} + \mathcal{M})$ time.

The procedure for computing the presence matrix follows. Define $G = (V, E)$ as the graph with all edges (u, v) where some edge event is recorded from u to v. Then create the corresponding presence matrix and initialize all entries with false. Scanning all the edge events in chronological order allows to fill the matrix. While reading all edge events for (u, v) (which appear in a row), record the time t_c of the last link creation. When the first link deletion is encountered with time t_d, all the entries $P_E[(u,v),i], P_E[(u,v),i+1], \ldots, P_E[(u,v),j]$ such that $t_c <= t_i, t_{i+1}, \ldots, t_j < t_d$ are set to true. (If no link deletion is encountered for link (u, v), the end of edge events for (u, v) is considered as link deletion with time $t_d = \infty$.)

Using the edge event set, it is not difficult to determine how long a link is going to last after its creation. Hence, we compute a *edge event list* $\mathcal{L}(x)$ of pointers to all link creations from any node x. Using this information we can modify Algorithm 1 to allow shortest paths computation using this new data structure. Step 3.a.iii now becomes:

iii Traverse $\mathcal{L}(x)$ and for all link creations (x, v, t, up) with time $t > t_i$, set $f(v) = t$ if it was not set.

This modified verstion of Algorithm 1 starts from A, examines all its out link creations $\mathcal{L}(A)$, and for each one updates the heap. Hence the total number of operations is at most $O(\sum_{v \in V} |\mathcal{L}(x)| \log N) = O(\mathcal{M} \log N)$. The space used is linear $O(N + \mathcal{M})$.

Another nice feature of this data structure is that it allows a scenario with continuous time, where a current time t_{now} is stored instead of i.

Scenario 6 *Scenario 4 and the time is not slotted.*

Algorithm 2

1. *Make all $d(v) = \infty$, but for $d(A) = 0$. Initialize a min-heap Q with a record $(A, d(A))$ in the root. Put in Q a dummy record $(dummy, d(dummy) = \infty)$.*
2. *$t_{now} \leftarrow 0$.*
3. *While $d(root(Q)) \neq \infty$ do*
   (a) *Extract x, the vertex at root(Q).*
   (b) *$t_{now} \leftarrow d(x)$*
   (c) *Delete root(Q).*
   (d) *Traverse $\mathcal{L}(x)$. For each link creation (x, v, t, up) with time $t > t_{now}$, update $d(v)$ with $t_{now} + c(x, v)$ if the computed link duration for the link creation is greater than $c(x, v)$ and $d(v)$ was greater than $t_{now} + c(x, v)$. If $d(v)$ was ∞, then insert v in Q.*
   (e) *Update Q.*

(f) Close x and insert it in the shortest paths tree.

Analysis. It is easy to see that Algorithm 2 is compliant with Scenario 6. The time complexity is again $O(\mathcal{M} \log N)$, as above. □

To cope with both Scenario 5 and Scenario 6, an efficient strategy can be achieved. First of all, dummy entries can be inserted in the edge event lists to mark node failure events (when a node is switched off) and node reset events (when the node is switched on again). A node failure implies that edges alive at that time fail and are then created again when the node recovers. We suppose that the corresponding entries are present in the edge event set. If we had to add them this would imply adding an edge creation for each edge broken by the node failure when the node recovers (edge duration of the breaking edges can be updated without adding an edge deletion event). A trivial but pessimistic upper bound for the number of edge creations due to node recover is $O(K_m \mathcal{M})$ when all edges experience a node reset.

The algorithm is now modified as follows. For a fixed node u the list of edge events from u is thus splitted in k parts, the first part corresponding to edge events after the first reset, the second part corresponding to edge events after the second reset and so on. Considering these edge event lists as k edge event lists of k virtual nodes $u_1, \ldots, u_k$ associated to u solves the problem. Moreover, we can again explicit a shortest path tree (where several nodes may correspond to same node after different reset events) to compute shortest paths. If $\mathcal{K}$ denotes the total number of node resets, then the algorithms performs $O(\mathcal{K} + \mathcal{M} \log \mathcal{K})$ operations. Notice that the size of the part of the input describing when node resets occur is precisely $O(\mathcal{K})$.

Sections 2, 3 and 4 deal with fixed-schedule networks. We now show how this framework may be extented when short time prediction is available.

5 Short time prediction mobile ad hoc networks

The above framework is mainly meaningful in predictive networks where the evolution of the connection graph is known in advance (such as in satellite networks or robot networks with known movements fixed-schedule dynamic transport networks). However it may also be useful in more general contexts such as mobile ad hoc networks where some short time prediction is possible. This may happen, for instance, with GPS capable nodes. With the knowledge of the position and the speed of the nodes, a short time prediction may be achieved.

Alternatively, if nodes receive power level information, they may be able to predict a link breakage if the power level is decreasing. Link creation could also be anticipated if a neighbor is accepted only above a certain threshold as used classically in link hysteresis strategies [7]. Depending on how fast the power level f a future neighbor is increasing, a node may estimate the link creation time with that node.

Suppose now that the evolution of connections may be predicted until some time step T_p. If a packet may be routed to the destination within that time, then our shortest path algorithm allows to compute such a route. Notice that this is different from classical routing since, at computation time, there may not exist a path connecting the source and the destination, but the packet can still eventually be forwarded to its final destination (depending on predicted link creation).

To formalize when an evolving graph framework is applicable in short time prediction networks, we can give the following definitions.

Definition 3 *The* prediction period T_p *of a dynamic network is the maximum period such that at any time t, the evolution of the connections in the network can be predicted until $t + T_p$.*

Definition 4 *The* time connectivity T_c *of an evolving graph is the minimum period of time such that at any time t, any node v may be reached from any node u before $t + T_c$.*

We then have the obvious property that packet forwarding may be achieved in this dynamic context whenever $T_p \geq T_c$. Notice that T_c depends mainly on the connectivity and the mobility in the network. It can be seen as a temporal analogue of the diameter of a graph for an evolving graph.

Indeed in many practical situations, it may happen that only a probability of link breakage or link creation within a certain period of time may be estimated. A probabilistic version of our model is reserved for future work. Here are some other open issues.

6 Open problems and perspectives

Several open problems in the area of dynamic communication networks are given in [8]. With respect to evolving graphs, we think that many interesting ways of further research remain wide open. In this note we presented algorithms in the setting where there is prior knowledge about the network dynamics schedule. Even in this simple setting, how would more intricate algorithms behave, like multi-packet transmissions or flow algorithms? What happens when the presence matrices represent a probabilistic process? Not mentioning all the questions arising in the case where a distributed setting, even partial, is assumed.

References

[1] *Handbook of Wireless Networks and Mobile Computing*. John Wiley and Sons, February 2002.

[2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[3] C. Demetrescu, D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: An experimental study. In *Proceedings of the 4th Workshop on Algorithm Engineering (WAE) 2000*, volume 1982 of *LNCS*, pages 218–229, Saarbrücken, Germany, September 2001.

[4] T. Goff, N. Abu-Ghazaleh, D. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In *Proceedings of the seventh International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 43–52, Rome, Italy, July 2001. ACM.

[5] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the IEEE Symposium on Foundations of Computer Science,*, 2000.

[6] X. Lin, M. Lakshdisi, and I. Stojmenovic. Location based localize alternate, disjoint, multi-path and component routing schemes for wireless networks. In *Proceedings of the 2001 ACM International Symposium on Mobile ad hoc Networking and Computing (MobiHoc'01)*, pages 287–290, October 2001.

[7] B. Narendran, P. Agrawal, and D.K. Anvekar. Minimizing cellular handover failures without channel utilization loss. In *Proceedings of IEEE GLOBECOM'94*, pages 1679–1685, San Francisco, Decembre 1994.

[8] C. Scheideler. Models and techniques for communication in dynamic networks. In *Proceedings of STACS 2002*, volume 2285 of *LNCS*, Juan-les-Pins, France, March 2002.

[9] V. Stix. Finding all maximal cliques in evolving graphs. Technical Report TR2001-05, Department of Statistics and Decision Support Systems, University of Vienna, February 2001.

[10] L. Viennot. Routage entre robots dont les déplacements sont connus – Un exemple de graphe dynamique. Réunion TAROT, ENST, Paris, Novembre 2001.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399