**MajecSTIC 2009**
**Avignon, France, du 16 au 18 novembre 2009**

# Heuristiques pour le routage multicritère

Alia BELLABAS[1], Miklos Molnar[1], Samer Lahoud[2].

1 : INSA-Rennes/IRISA.
2 : Université Rennes1/IRISA.

Contact :{alia.bellabas, miklos.molnar, samer.lahoud}@irisa.fr

**Résumé**
Les applications dans les réseaux actuels deviennent de plus en plus gourmandes en ressources et exigeantes en qualité de service. Ainsi, le routage doit satisfaire plusieurs contraintes telles que le délai, la bande passante ou la gigue. Il s'agit alors d'appliquer un routage multicritère. Plusieurs solutions algorithmiques existent dans la littérature. L'un des algorithmes les plus performants est SAMCRA (Self Adaptive Multiple Constraints Routing Algorithm) proposé par Van Mieghem et al. en 2001. SAMCRA est un algorithme multicritère unicast exact mais de grande complexité. Dans notre étude, nous cherchons à remplacer SAMCRA par un algorithme moins couteux qui calcule des $k$ plus courts chemins. Les simulations montrent que l'application d'un tel algorithme réduit de manière significative la complexité du problème de routage multicritère, tout en donnant des solutions satisfaisantes.

**Abstract**
The applications in current networks require more and more quality of services. Hence, the routing algorithms have to satisfy several constraints such as the delay, the bandwidth or the jitter. Therefore, multicriteria routing algorithms are needed. Since it is an NP-Hard problem, we propose heuristics that quickly calculate paths that satisfy Quality of service QoS constraints between a source node and a destination node. Several solutions exist in the literature; the most efficient one is SAMCRA (Self Adaptive Multiple Constraints Routing Algorithm) which was proposed by Van Mieghem and al. in 2001. SAMCRA is an exact unicast algorithm for multiple constraints which has a high complexity. In our study, we propose to replace SAMCRA by an optimized $k$ shortest paths algorithm. The simulation results show that applying such algorithm reduces significantly the complexity of the multicriteria routing algorithm, and gives satisfying solutions.

**Mots-clés :** Réseau, routage, qualité de service, multicritère.

**Keywords:** Network, routing, quality of service, multicriteria.

## 1. Introduction

Le routage multicritère consiste à transmettre des données d'une source vers une destination en tenant compte de plusieurs contraintes imposées (souvent par les applications). Durant les dernières années, ce type de routage a pris de l'importance vu l'émergence d'applications qui nécessitent des garanties sur un ensemble de paramètres de qualité de service tels que le délai, le coût, la bande passante, le taux de perte, la gigue, etc. Ces nouveaux défis ont donné lieu au problème de routage multicritère. Les algorithmes proposés pour résoudre le problème du routage multicritère considèrent principalement deux métriques: le coût et le délai de transmission [1][2]. Deux approches existent pour ces algorithmes. Une première approche traite le routage comme un problème d'optimisation

mono-objectif qui minimise le coût sous une contrainte de délai. Une seconde approche utilise une formulation multi-objectif : pour résoudre cette classe de problèmes, des travaux utilisent des méta-heuristiques telles que les algorithmes génétiques [2], la recherche tabou [4] ou les colonies de fourmis [4]. D'autres travaux proposent des algorithmes multi-objectifs exacts parmi lesquels on trouve l'algorithme SAMCRA (Self Adaptive Multiple Constraints Algorithm) qui a été proposé par Van Mieghem et al. dans [7]. SAMCRA est un algorithme de routage unicast multicritère optimal. Cependant, l'inconvénient majeur de SAMCRA est sa complexité [6]. Dans cet article, nous cherchons à remplacer SAMCRA pour résoudre le problème de routage multicritère par un algorithme de faible complexité qui donne des solutions satisfaisantes.
Dans la suite, nous formulons le problème du routage unicast multicritère. Dans la section 3, nous décrivons l'algorithme SAMCRA. Dans la section 5, nous donnons un aperçu des différents algorithmes de calcul des $k$ plus courts chemins existants dans la littérature. Ensuite, nous détaillons le fonctionnement de l'algorithme de Yen. Dans la section 6, nous présentons dans le détail notre contribution, et donnons les résultats des simulations.

## 2. Formulation du problème

Soit le réseau modélisé par un graphe valué $G(N, E)$, où $N$ est l'ensemble des nœuds de $G$ et $E$ l'ensemble de ses liens. Soit une source $s$ et une destination $d$. Soit $m$ le nombre de métriques à considérer. Pour le problème de routage multicritère, les contraintes sur les métriques sont données par le vecteur $\overrightarrow{L} = (L_1, L_2, \dots, L_m)$. Chaque lien $e_j$ est valué par un vecteur :

$$\overrightarrow{w}(e) = (w_1^e, w_2^e, \dots, w_m^e)$$

Les métriques dans les réseaux peuvent être additives comme le délai et le coût, ou concaves comme la bande passante, ou encore multiplicatives comme le taux de perte. Pour les métriques concaves, il suffit de supprimer tous les liens qui violent les contraintes, ensuite trouver dans le graphe résiduel un chemin entre $s$ et $d$. D'autre part, les métriques multiplicatives peuvent être ramenées à des métriques additives en appliquant le logarithme. Ainsi, les métriques les plus générales et les plus difficiles à satisfaire sont les métriques additives. Dans la suite nous considérons uniquement les métriques additives.
Le poids d'un chemin $p$ pour une métrique $i$ est égal à la somme des poids de ses liens pour cette métrique: $l_i(p) = \sum_{e \in p} w_i^e$

Le problème du routage multicritère a été formulé de deux façons différentes [6].

**Définition 1 : problème MCP**
Le problème de calcul du chemin multicritère (Multi-Constraint Path problem MCP) *(P)* consiste à trouver un chemin dit faisable $p$ qui satisfait toutes les contraintes $L_i$ :

*(P)* : $l_i(p) \leq \overrightarrow{L_i}$ pour tout $1 \leq i \leq m$ (1)

**Définition 2 : problème MCOP**
En définissant une fonction longueur $l$, par exemple: $l(p) = \sum_{i=1}^{m} l_i(p)$, le problème MCOP (Multi-Constraint Optimal Path problem) *(P*)* cherche à déterminer dans l'ensemble *chemins(s,d)* des chemins entre la source *s* et la destination *d,* le chemin faisable avec la plus petite longueur $l$ :

*(P*)*: $min_{p \in chemins(s,d)} l(p)$ (2)

## 3. L'algorithme SAMCRA

L'algorithme SAMCRA [7] est un algorithme exact de routage unicast multicritère qui résout le problème MCOP en se basant sur deux principes : la longueur non linéaire et la dominance de chemins.

**Définition 3 : longueur non-linéaire**
SAMCRA utilise une fonction de longueur non-linéaire $l$ pour le calcul de chemins : pour un chemin $P$ qui contient $k$ liens: $P = \{e_1, e_2, \dots, e_k\}$, la longueur non-linéaire de $P$ est :

$$l(p) = max_{1 \le i \le m} \left( \frac{\sum_{j=1}^{k} w_i^{e_j}}{L_i} \right)$$

**Définition 4 : dominance de chemins**
Soient deux chemins $P_1$ et $P_2$. On dit que $P_1$ domine $P_2$ et on le note $P_1 <^d P_2$ si et seulement si : $l_i(P_1) \le l_i(P_2)$ pour chaque métrique $i$, $l_j(P_1) < l_j(P_2)$ pour au moins une métrique $j$.

Pour un couple de source-destination *(s,d),* SAMCRA retourne le meilleur chemin en terme de longueur non linéaire qui satisfait toutes les contraintes, si un tel chemin existe. L'algorithme commence par explorer les voisins de la source s, ensuite passe vers le voisin avec la plus petite longueur non-linéaire, et explore ses voisins. Ainsi, SAMCRA explore les voisins de tous les nœuds en partant de la source s, et au fur et à mesure élague les chemins dominés. SAMCRA s'arrête quand la destination *d* est sélectionnée comme étant le nœud qui sauvegarde le plus court chemin non dominé. Ainsi, tous les chemins améliorant la longueur du meilleur chemin trouvé jusqu'à présent sont déjà parcourus.

## 4. Résumé de la contribution

Il a été prouvé dans [16] que le routage multicritère est un problème NP-difficile. SAMCRA est un algorithme exact de routage multicritère unicast, et d'une grande complexité [6][1]. Le routage multicritère ne nécessite pas forcement la recherche de la solution optimale, mais une solution faisable. De ce fait, nous proposons des heuristiques qui calculent les chemins faisables plus rapidement en partant du chemin le plus court au moins court. L'idée d'appliquer un tel algorithme vient du fait que les chemins les plus courts peuvent s'avérer faisables. Dans notre étude, nous proposons la modification d'un algorithme connu (algorithme de Yen cf. Section 5.2) pour calculer les chemins entre deux nœuds en partant du plus court au moins court jusqu'à l'obtention d'un chemin qui satisfait toutes les contraintes. Ces heuristiques résolvent le problème MCP mais ne garantissent pas de trouver des solutions optimales. Cependant, les solutions trouvées sont satisfaisantes et les résultats des simulations présentées dans la section 7 confirment l'idée avancée.

## 5. Les algorithmes de calcul des $k$ plus court chemins

### 5.1. Travaux existants

Les plus courts chemins (en nombre de saut par exemple) sont souvent utilisés pour le routage. Lorsque le plus court chemin ne satisfait pas les contraintes de qualité de service, il devient nécessaire de calculer un ensemble de $k$ plus courts chemins entre une source et une destination. Le calcul des $k$ plus courts chemins qui peuvent contenir des boucles est un problème à complexité réduite : un algorithme de complexité $O(|E| + klog|N|)$ est connu depuis 1975 [8]. Cet algorithme a été amélioré par Eppstein [9] pour arriver à une complexité de $O(|E| + |N|\log\ |N| + k)$. Dans les applications réelles, le plus défiant consiste à trouver les $k$ plus courts chemins qui ne contiennent pas de boucles. Le problème est étudié à l'origine par Hoffman et Paveley [10]. Pour les graphes non orientés, l'algorithme le plus efficace est celui proposé par Katoh et al. [11] qui a pour complexité $O\big(k(|E| + |N|\log\ |N|)\big)$. Quant au cas le plus général, l'algorithme le plus connu est celui de Yen [12], généralisé par Lawler [13] et qui a pour complexité $O\big(k|N|(|E| + |N|\log\ |N|)\big)$.

### 5.2. L'algorithme de Yen

L'algorithme de Yen appartient à la classe des algorithmes de déviation. La déviation d'un chemin par rapport à un ensemble de chemins est expliquée dans l'exemple de la Figure *1.*

[1] [6] : dans cet article Van Mieghem et Kuipers étudient la complexité du problème de routage multicritère et l'application de SAMCRA à ce problème.

Soient trois chemins $P_1, P_2$ *et* $P_3$. Le chemin $P_1$ dévie de $P_2$ au nœud 1, et de $P_3$ au nœud 3. Ainsi, on dit que $P_1$ dévie de l'ensemble des chemins $\{P_2, P_3\}$ au nœud 3.

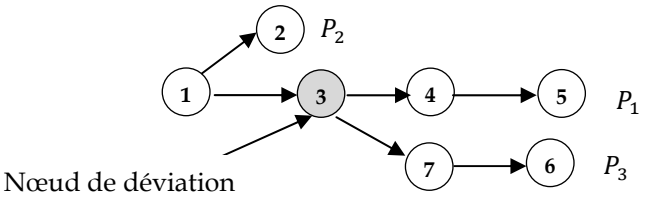


**Figure 1.** *Déviation de chemins*

Le pseudo-code de l'algorithme de Yen et les notations sont présentés dans la Figure 2. L'algorithme commence par calculer, pour un couple source-destination choisi, le plus court chemin *P** (ligne 1). L'algorithme effectue *k* itérations. A la *i*ème itération, on récupère le plus court chemin stocké dans la liste des candidats *D*, qui est le *i*ème plus court chemin. Ensuite, on calcule le nœud de déviation de ce chemin par rapport à l'ensemble des *i-1* chemins de *X*. Pour éviter de parcourir des chemins déjà calculés, l'algorithme supprime des nœuds et des liens (lignes 9 et 10). Dans le graphe résiduel, le plus court chemin *P'* entre le nœud de déviation et la destination est calculé. *P'* est ensuite concaténé au sous chemin entre la source et le nœud de déviation du *i*ème plus court chemin, puis sauvegardé dans l'ensemble des candidats *D*. A la ligne 14, on restaure tous les nœuds et liens supprimés, et on passe au successeur du nœud de déviation dans le *i*ème plus court chemin (ligne 15).

**Algorithme de Yen(*G,s,t,k*)**

1. *P**←Dijkstra (*s,t*) // le plus court chemin entre *s* et *t* calculé avec Dijkstra
2. *D*←{*P**} //ensemble des chemins candidats
3. *X*←{} // ensemble des chemins calculés
4. pour *i* allant de *1* à *k*
5. *P*←le plus court chemin dans *D*
6. *v*←le nœud de déviation entre *P* et les chemins dans *X*
7. *X*←*X*+{*P*}
8. Tant que *v*!=*t*
9. supprimer tous les nœuds de *P* de *s* à *v* et leurs liens associés
10. supprimer tous les arcs sortants de *v* qui appartiennent aux chemins dans *X*
11. *P'*←Dijkstra (*v,t*) //le plus court chemin entre *v* et *t* calculé avec Dijkstra
12. concaténer *P'* avec le sous chemin de *P* entre *s* et *v*
13. *D*←*D*+{*P'*}
14. restaurer tous les nœuds et liens supprimés
15. *V*←successeur(*v.P*)

**Figure 2.** *Pseudo-code de Yen*

## 6. Algorithme et approches proposés

Dans cet article, nous proposons de calculer le premier chemin faisable par une modification de l'algorithme de Yen pour le calcul des *k* plus courts chemins. Cet algorithme a une complexité nettement plus réduite que celle de SAMCRA. Cependant, le calcul des *k* plus courts chemins utilise une seule métrique. Pour pouvoir l'adapter, nous ramenons le problème multicritère à un problème monocritère. Pour cela, nous proposons deux approches :

- **l'approche nombre de sauts :** dans cette approche, l'algorithme cherche le premier chemin faisable prenant en compte le nombre de sauts
- **l'approche linéarisation des métriques :** l'une des approches les plus courantes pour la linéarisation des métriques est celle proposée par Jaffe [14]. Cette approche considère le

poids d'un lien donné par : $w(e) = \sum_{i=1,m} \alpha_i w_i^e$, où $w_i^e$ est le poids de $e$ pour la métrique $i$. Pour calculer les paramètres $\alpha_i$, nous calculons les $m$ plus courts chemins $P_1, P_2, \dots, P_m$ correspondants aux $m$ métriques. Le paramètre $\alpha_i$ serait égal à : $\alpha_i = \frac{\sum_{e \in P_i} w_i^e}{L_i}$

Par ailleurs, l'algorithme de Yen cherche, pour un couple source-destination donné, et pour un entier $k$ donné, les $k$ plus courts chemins. Dans cet article, l'algorithme proposé cherche le premier chemin faisable, ou s'arrête à un nombre de chemins maximal *k_max* donné comme paramètre d'entrée.

Le méta-code de l'approche de linéarisation des métriques est donné dans la Figure *3*. L'algorithme commence par calculer, pour un couple source-destination, les plus courts chemins correspondants à chaque métrique. A la ligne 2, les paramètres $\alpha_i$ associés au poids de chaque métrique sont calculés. A la ligne 3, on associe à chaque lien $e$ un poids $w'$ qui est égal à la pondération des poids pour chaque métrique de ce lien avec les paramètres $\alpha_i$. Les lignes 5 à 12 illustrent le procédé pour calculer le prochain plus court chemin. A la ligne 6, le *i*ème plus court chemin est calculé en appliquant le même principe que celui d'un algorithme de Yen (Figure 2). A la ligne 8, l'algorithme vérifie la faisabilité du chemin calculé après restitution des poids initiaux de ses liens. Si ce chemin est faisable l'algorithme s'arrête, sinon les poids pondérés sont remis sur les liens du graphe et le prochain plus court chemin sera calculé.

**Approche_linéarisation(graphe G, nœud s, nœud t, vecteur L, entier k_max)**

1. Calculer $P_1^*, P_2^*, \dots P_m^*$ les $m$ plus courts chemins entre $s$ et $t$ pour les $m$ métriques
2. $\alpha_i = \frac{\sum_{e \in P_i^*} w_i^e}{L_i}$ : le rapport entre la longueur linéaire de $P_i^*$ et la contrainte $L_i$
3. $w'(e) = \sum_{i=1}^{m} \alpha_i w_i$ pour chaque lien e du graphe $G$.
4. i←0, solution_trouvée←faux
5. Tant que ((solution_trouvée≠ vrai) et (i< k_max))
6. Calculer le *i*ème plus court chemin $P_i$ entre s et t
7. Restituer les poids initiaux des liens de $G$
8. Si $P_i$ est faisable { solution_trouvée← vrai, retourner$P_i$ }
9. Sinon {i←i+1}
10. Remettre les poids pondérés sur les liens de $G$

**Figure 3.** *Méta-code de l'approche de linéarisation*

## *7.* Evaluation des performances

Dans nos simulations, nous utilisons une topologie qui représente un réseau d'opérateur de 50 nœuds et 88 liens [15]. Pour chaque lien, nous générons deux poids non corrélés tirés aléatoirement selon une loi uniforme dans l'intervalle [1,100]. Nous effectuons ce tirage 100 fois. Pour chaque tirage de poids, nous choisissons aléatoirement 100 couples source-destination, et pour chaque couple source-destination, nous appliquons les trois algorithmes : SAMCRA, l'approche nombre de sauts, et l'approche linéarisation des métriques. Le vecteur des contraintes $L$ est tiré uniformément, et prend ses valeurs dans un intervalle qui est fixé au préalable pour chaque distribution de poids, et pour chaque couple source-destination. La Figure 4 illustre le tirage des contraintes $L_i$. Soient $P_1$ et $P_2$ les chemins qui minimisent respectivement les métriques $w_1$ et $w_2$. Le rectangle gris représente l'espace des contraintes dites critiques. Nous utilisons 15 intervalles de tirage des contraintes qui parcourent la diagonale du plus strict (rectangle A), au plus lâche (rectangle B).

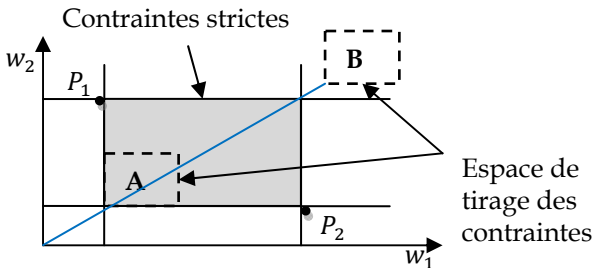


**Figure 4.** *Tirage des contraintes*

Pour les deux approches, nous fixons le nombre maximal de chemins à calculer à trois ($k \leq 3$). Pour évaluer les performances des trois algorithmes, nous nous intéressons à six critères de comparaison :

- Taux de succès : nombre de requêtes satisfaites divisé par le nombre de requêtes générées
- Complexité absolue : nombre moyen de *visites des nœuds*[2] pour les requêtes générées
- Complexité relative : nombre moyen de visites des nœuds pour les requêtes satisfaites
- Longueur non linéaire : la longueur utilisée par SAMCRA (voir section 3)
- Longueur moyenne : la moyenne des sommes des poids pour chaque métrique
- Le nombre de chemins calculés avant de trouver une solution faisable : ce critère est uniquement significatif pour les approches basées sur le calcul des $k$ plus courts chemins.

Dans la Figure 6, les solutions trouvées par nos deux approches nombre de sauts et linéarisation des métriques sont moins bonnes que celles trouvées par SAMCRA, l'écart est en moyenne de 6,67% et 2.39% respectivement. Cependant, nous remarquons dans la Figure 5 que pour les longueurs moyennes, les résultats de SAMCRA s'avèrent moins bons que ceux trouvés par l'approche linéarisation des métriques (1.59%), et très proche de ceux de l'approche nombre de sauts. En effet, la pondération des poids pour cette approche sur un lien implique le calcul de chemins basé sur les deux métriques à la fois contrairement à SAMCRA qui ne considère que la plus grande longueur, donc ne prend pas en considération la variance entre les valeurs des deux métriques. De plus, les solutions trouvées en considérant les deux longueurs sont très proches pour les trois algorithmes quand les contraintes sont strictes. En effet, les contraintes strictes réduisent le nombre de solutions faisables, alors les solutions trouvées par nos deux approches sont proches des solutions optimales. Pour les contraintes moins strictes, SAMCRA trouve de meilleures solutions que les deux autres approches. Notons que, pour ne pas biaiser les résultats, seules les solutions trouvées par les trois algorithmes sont considérées.

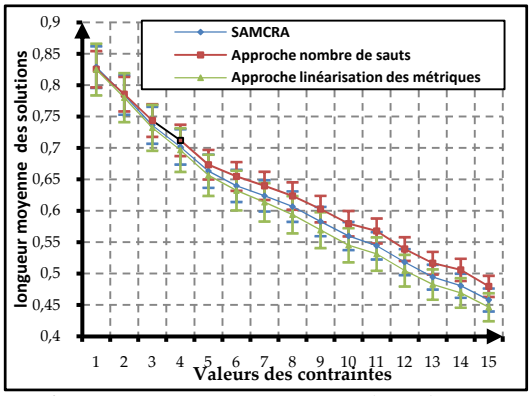


**Figure 5.** *Longueur moyenne des solutions*

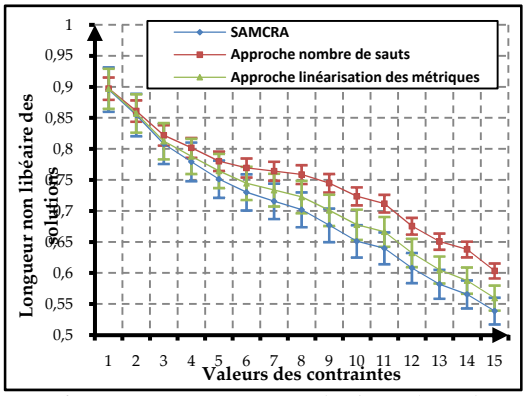


**Figure 6.** *Longueur non linéaire des solutions*

[2] Une visite de nœud représente une opération fondamentale.

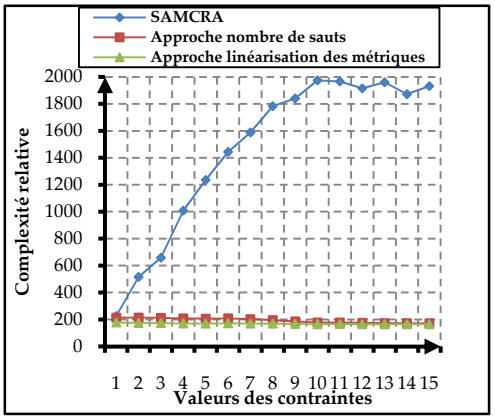


**Figure 7.** *Complexité relative*

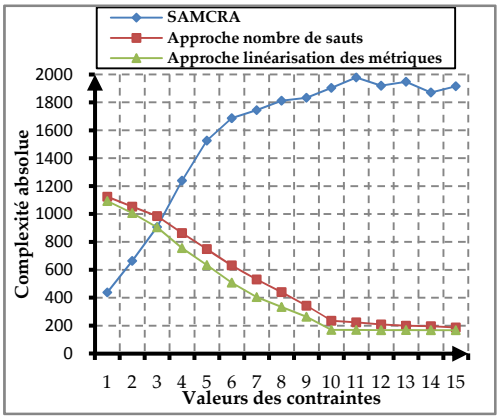


**Figure 8.** *Complexité absolue*

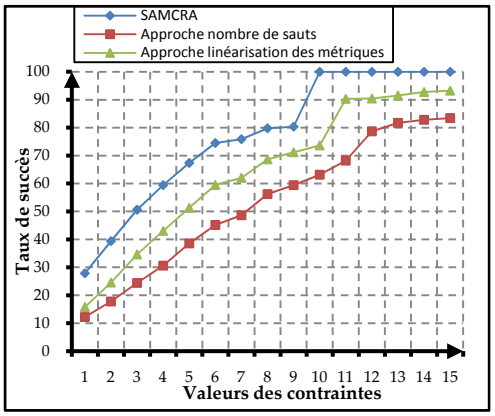


**Figure 9.** *Le taux de succès*

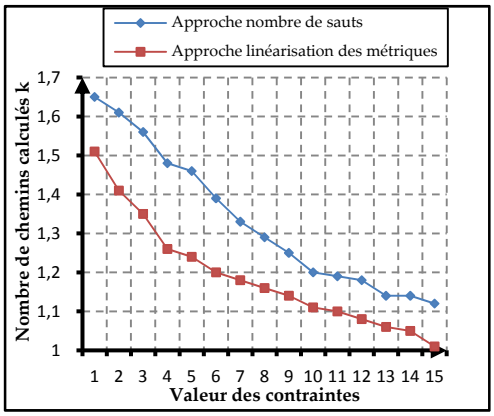


**Figure 10.** *Nombre de chemins calculés*

La complexité relative est calculée seulement dans le cas où une solution est trouvée par les trois algorithmes. Dans la Figure 7, nous remarquons que la complexité de SAMCRA est nettement plus grande que celle des deux autres algorithmes. En effet, plus les contraintes sont lâches, plus SAMCRA a de chemins à explorer avant de retourner la meilleure solution, alors que les deux approches s'arrêtent au premier chemin faisable trouvé. Ceci réduit le temps d'exécution des deux approches basées sur le calcul des $k$ plus courts chemins. Dans la Figure 8, la complexité absolue des deux approches proposées est plus grande que celle de SAMCRA pour les contraintes strictes. En effet, quand les contraintes sont strictes, SAMCRA élague plus rapidement les chemins non faisables, alors que les deux approches explorent le nombre maximal de chemins (fixé à trois) sans trouver de solutions. Plus les contraintes deviennent lâchent, plus les deux approches proposées trouvent des solutions faisables avant de calculer les trois chemins, ce qui réduit leur complexité.

Dans la Figure 9, nous trouvons que le taux de succès des deux approches nombre de sauts et linéarisation des métriques s'approche du taux de succès de SAMCRA en moyenne de 10% et 16% respectivement. Dès que les contraintes deviennent lâches, SAMCRA atteint un taux de succès de 100%, l'approche linéarisation des métriques un taux de succès de 94%, et l'approche nombre de sauts un taux de succès de 84%. Ces taux de succès sont très satisfaisants. La Figure 10 montre que le nombre de chemins calculés avant de trouver une solution faisable est inférieur à deux. Ce nombre diminue avec les contraintes qui deviennent de plus en plus lâches, ceci explique le choix de fixer le nombre maximal de chemins à explorer à trois.

Les simulations montrent que pour les contraintes lâches, les deux approches proposées retournent des solutions proches de l'optimal et réduisent la complexité jusqu'à 10 fois, ce qui rend leur application pour le routage multicritère unicast très prometteuse.

## 8. **Conclusion**

Pour résoudre le problème du routage multicritère, nous avons proposé un algorithme obtenu par la modification de l'algorithme de Yen, un algorithme efficace pour le calcul des *k* plus courts chemins. Deux approches ont été proposées pour ramener le problème multicritère en monocritère et appliquer l'algorithme proposé : l'approche basée sur le nombre de saut, et celle de la linéarisation des métriques. Les simulations montrent que les deux approches proposées donnent des solutions satisfaisantes avec des taux de succès proches de l'optimal trouvé par l'algorithme exact SAMCRA. De plus, l'algorithme proposé calcule en moyenne 1.43 chemins avant de trouver une solution faisable. Ceci réduit la complexité pour le calcul multicritère de manière significative.

Pour résoudre le problème de routage multicast multicritère, l'algorithme MAMCRA (Multicast Adaptive Multiple Constraints Routing Algorithm) a été proposé dans [5]. MAMCRA utilise comme première étape SAMCRA, et de ce fait hérite de sa complexité. Comme travaux futurs, nous envisageons donc de remplacer SAMCRA dans MAMCRA par les approches proposées, et d'étudier leurs performances. Aussi, il est envisageable d'améliorer ces deux approches en incluant la détection de la non faisabilité des solutions au plus tôt, ce qui nous permet de réduire encore la complexité des algorithmes ici proposés.

**Bibliographie**


1. L. Sahasrabuddhe and B. Mukherjee, "Multicast routing algorithms and protocols: Atutorial," *IEEE Network*, 2000.
2. C. Garrozi, A. F. R. Araujo, « Multiobjective Genetic Algorithm for Multicast Routing», 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, 2006.
3. H. Youssef, A. Al-Mulhem, S. M. Sait, M. A. Tahir, "QoS-driven multicast tree generation using tabu search", Computer communications, (25): 1140-1149, 2002.
4. D. Pinto, B. Baran, "Solving multiobjective Multicast Routing Problem with a new Ant Colony Optimization approach", (LANC'05), Cali, Colombia, October 10-12, 2005.
5. F. Kuipers, P. Van Mieghem, "MAMCRA: a constrained-based multicast routing algorithm", Computer communications, (25): 802-811, 2002.
6. P. Van Mieghem, F. A. Kuipers, "On the complexity of QoS Routing", Computer Communications, 26: 376-387, 2003.
7. P. Van Mieghem, H. De Neve, F. A. Kuipers, "Hop-by-hop Quality of service routing", Computer Networks, 37(3-4): 407-423, October 2001.
8. B. L. Fox. "*k*-th shortest paths and applications to the probabilistic networks". In RSA/TIMS Joint National Mtg., 23: B263, 1975.
9. D. Eppstein. Finding the k shortest paths. SIAM J. Computing, 28(2): 652–673, 1998.
10. W. Hoffman and R. Pavley. A method for the solution of the *nth* best path problem. Journal of the ACM, 6: 506–514, 1959.
11. N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. Networks, 12:411–427, 1982
12. J. Y. Yen. Finding the K shortest loopless paths in a network. Management Science, Vol. 17, pp. 712–716, 1971.
13. E. L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. Management Science, Vol. 18, pp. 401–405, 1972.
14. J. M. Jeffe, "Algorithms for finding paths with multiple constaints", Networks, 4: 95-115, 1984.
15. A. Zimolka, "Design of Survivable Optical Networks by Mathematical Optimization", thesis on mathematic, Berlin University, 2007.
16. Z. Wang, J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Applications", IEEE Journal of Selected Areas in Communications, 14(7): 1228-1234, 1996.