Extension of PageRank and application to social networks

The Dang Huynh

Université Pierre et Marie Curie
École doctorale
Alcatel-Lucent Bell Labs France

Extension de PageRank et Application aux Réseaux Sociaux

Par The Dang Huynh

Thèse de doctorat d'Informatique

Présentée et soutenue publiquement le 01/06/2015

Devant le jury composé de

Rapporteur :	M. Konstantin Avratchenkov, Directeur de recherche, INRIA
Rapporteur :	M. Jean-Loup Guillaume, Professeur, L3i
Examinateur :	Mlle. Nidhi Hegde, Chercheur, ALBLF
Co-directeur:	M. Fabien Mathieu, Chercheur, ALBLF
Examinateur :	M. Sébastien Tixeuil, Professeur, LIP6
Directeur de thèse :	M. Laurent Viennot, Directeur de recherche, INRIA

Acknowledgements

This work would not have been possible without the advice and support of many people, and I will always keep all of that in my heart with a deep appreciation.

First and foremost, enormous gratitude is due to *Dr. Fabien Mathieu* who has been there as my supervisor for almost four years. His supervision, support and guidance are immeasurable during my Master and PhD. I am highly indebted and thoroughly grateful to him for providing me with material and knowledge that I could not possibly have discovered on my own, for his kind words and suggestions, and also for introducing me to *Dr. Laurent Viennot* and *Dr. Dohy Hong*.

Particular thanks must also be recorded to *Laurent Viennot* for his valuable comments, discussions and motivations.

I would like to thank *Dohy Hong*, from whom I have learned a lot and who put me on the track of D-Iteration.

I would also like to say thank to *Dr. Chung Shue Chen* for his guidance in an interesting topic outside the mainstream of my PhD.

I must thank reviewers, *Dr. Konstantin Avratchenkov* and *Professor Jean-Loup Guillaume*, for their time revising my thesis. Without them, these chapters below can not be as readable as they are.

Special thanks go to the members of my jury who kindly accepted to judge my work, *Dr. Konstantin Avratchenkov*, *Professor Jean-Loup Guillaume*, *Dr. Nidhi Hegde*, *Dr. Fabien Mathieu*, *Professor Sébastien Tixeuil* and *Dr. Laurent Viennot*.

I would appreciate *Maura Covaci* and *Gérard Burnside* for correcting English and French grammatical errors in the thesis.

I would also thank *Madame Félix-Noël Gudrun* for sincerely hosting me in the last three years.

I would like to express my gratitude to all colleagues of the Department of Mathematics of Complex Dynamic Network of which I am proud to be a part. Here, I have a great leader *Philippe*, a humorous *Gérard*, a friendly *Dimitrios*, an amiable *Alonso* and many other people that I can not list all, *Walid*, *Ludovic*, *Nidhi*, *Amira*, *Loreta*, *Lamine*, etc. I would also like to thank my friends at LINCS for their supports, specially *Yuting* and *Rim* with whom I am always in a good mood after talking.

Most importantly, none of this would have been possible without the love and patience of my family. My parents to whom this dissertation is dedicated to, have been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to my family. Also many thanks to my friends who have helped me stay sane through these challenging years.

Abstract

In this thesis, we investigate the performance and applications of a new method that can accelerate the computation of the PageRank vector. The method, called D-Iteration (DI), has been introduced by Dohy Hong. It is based on the decomposition of the matrix vector that can be seen as a fluid diffusion model and is potentially adapted to asynchronous implementation. We recall the theoretical results about the convergence of the algorithm and we show through numerical analysis on a real Web graph that DI can improve the computation efficiency compared to other classical algorithms like Power Iteration (PI), Gauss-Seidel (GS) or Adaptive On-Line Page Importance Computation (OPIC). Besides, performance comparison between DI and other algorithms is further studied through experiments on real graphs and in various metrics such as iteration and operational cost. In the second part of the thesis, we study one of the main challenges for large networks data mining that deals with high dynamics of huge datasets: not only are these datasets difficult to gather, but they tend to become obsolete very quickly. We are also interested in the evolution at large time scale of any large corpus available online. Our primary focus will be the Web, but our approach, called LiveRank, encompasses any online data with similar linkage enabling crawling, like P2P networks or online social networks. We thus focus on batch crawling, where starting from a completely out-dated snapshot of a large graph like the Web, we want to identify a significant fraction of the nodes that are still alive now.

Contents

List of Tables

List of Figures

PUBLICATIONS

- **Localization Method for Device-to-Device through User Movement**, *T.D. Huynh, C.S. Chen, S.W. Ho*, Workshop on Advances in Network Localization and Navigation, IEEE International Conference on Communications (ICC'15), London, UK, June 2015.

- **METHOD, SYSTEM AND COMPUTER-READABLE MEDIUM TO DETERMINE THE POSITION OF AN APPARATUS (Method for accurate positioning under only one and mobile reference)**, *C.S. Chen and T.D. Huynh*, Filing No. 15305765.8, Europe, May 2015.

- **LiveRank: How to Refresh Old Datasets**, *T.D. Huynh, F. Mathieu, L. Viennot*, Internet Mathematics journal, April 2015, **Submitted**.

- **Analyzing methods computing PageRank vector of large matrix**, *T.D. Huynh, D. Hong, G. Burnside, F. Mathieu*, Preprint, https://hal.archives-ouvertes.fr/hal-01109536/document, January 2015.

- **D-Iteration: diffusion approach for solving PageRank**, *D. Hong, T.D. Huynh, F. Mathieu*, Preprint, https://hal.archives-ouvertes.fr/hal-01109532/document, January 2015.

- **Exploiting user movement for position detection**, *T.D. Huynh, C.S. Chen, S.W. Ho*, The 12^{th} Annual IEEE Consumer Communications and Networking Conference (CCNC'15), Las Vegas, Nevada, USA, January 2015.

- **LiveRank: How to refresh old crawls**, *T.D. Huynh, F. Mathieu, L. Viennot*, 11^{th} Workshop on Algorithms and Models for the Web Graph (WAW'2014), Beijing, China, December 2014.

- **LiveRank: Comment faire du neuf avec du vieux**, *T.D. Huynh, F. Mathieu, L. Viennot*, The 16^{th} Francophone conference on Telecommunication algorithms (AlgoTel'14), Le-Bois-Plage-en-Ré, France, June 2014.

- **Contenu généré par les utilisateurs: une étude sur DailyMotion**, *Yannick Carlinet, T.D. Huynh, B. Kauffmann, F. Mathieu, L. Noirie, and*

S. Tixeuil, The 5^{th} Francophone conference on Telecommunication algorithms (AlgoTel'13), Pornic, France, May 2013.

- **Four Months in DailyMotion: Dissecting User Video Requests**, *Yannick Carlinet, T.D. Huynh, B. Kauffmann, F. Mathieu, L. Noirie, and S. Tixeuil*, In Proceedings of the Third International Workshop on Traffic Analysis and Classification (TRAC'2012), Limassol, Cyprus, August 2012.

Résumé en français

Contexte

Internet a connu une croissance rapide au cours de la dernière décennie, en particulier le World Wide Web. Selon une étude de Netcraft [3], il y a plus d'un milliard de sites Web à la fin d'octobre 2014. Une autre étude [1] montre un nombre légèrement différent, environ 1,1 milliards de sites Web actifs pour la même période. La figure 1 indique l'évolution du World Wide Web pendant les 14 dernières années, à partir de 2000 à 2014.

Cette thèse parlera principalement du PageRank, un algorithme de classement introduit par Page et al. (cf [4, 5]) et utilisé par le moteur de recherche Google comme un des facteurs les plus importants pour classifier les pages web. Notez que le terme *site Web* fait référence à un nom de domaine, et que le terme de *page Web* est une page qui se trouve sur ce domaine. Nous devons distinguer clairement ces deux termes parce que, dans le contexte de l'algorithme de PageRank, il existe plusieurs méthodes [6, 7, 8, 9] qui estiment la valeur de PageRank d'une page web en fonction du PageRank de tout le site, spécialement dans le contexte de calcul parallèle où le calcul et le stockage sont souvent au niveau du site web.

Les moteurs de recherche comme *Google*, *Bing* et *Yahoo* classifient les documents au niveau de chaque page Web. Avec un milliard de sites Web sur Internet, selon [10] Google a environ 50 milliards de pages Web, ce qui revient à dire que chaque site a en moyenne 50 pages web indexées par Google. Toutefois, la taille du WWW pourrait être encore plus considérable en raison du fait que la grande majorité des pages Web n'est obtenue qu'en interrogeant des serveurs Web. D'autre part, l'émergence de réseaux sociaux, tels que *Twitter* [11], *Facebook* [12] et *Google+* [13],

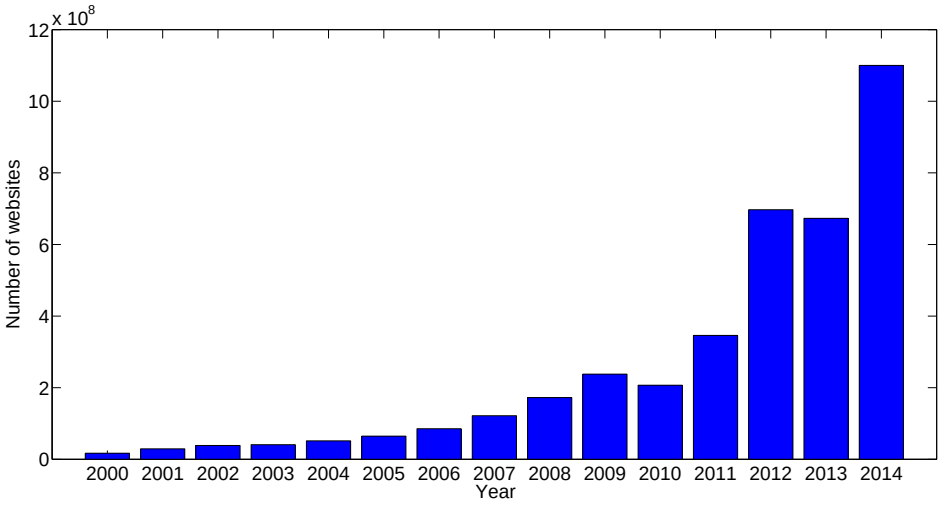


Figure 1: Nombre de sites web sur Internet. Source: InternetLiveStats.com [1]

enrichit le World Wide Web de manière significative grâce à sa croissance rapide. La figure 2 montre l'évolution d'utilisateurs actifs mensuels de Twitter et Facebook du premier trimestre 2010 (Q1'10) au troisième trimestre 2014 (Q3'14). Comme nous pouvons voir, les sites Web ont augmenté de façon exponentielle, contrairement à la croissance linéaire du nombre d'utilisateurs des réseaux sociaux. L'explication intuitive peut provenir du fait que les sites Web n'ont aucune limite à grossir et les moteurs de recherche plus modernes (plus intelligents) peuvent indexer plus de pages Web basées sur les requêtes. Pourtant, les membres de réseaux sociaux sont limités par la taille de la population humaine qui augmente linéairement.

Problème du classement

Le classement des objets est une des questions importantes et typiques dans notre vie quotidienne. De nombreuses applications ont besoin de classifier des objets en fonction de certains critères, parfois simple comme de classifier les étudiants dans une classe en fonction de relevé de notes ou plus compliqué comme le classement des universités. Classifier des objets consiste à les ordonner selon certains critères exigés par une application spécifique.

Avec la popularisation de l'Internet, un problème typique qui a émergé ces deux dernières décennies est le classement des résultats renvoyés par les moteurs de

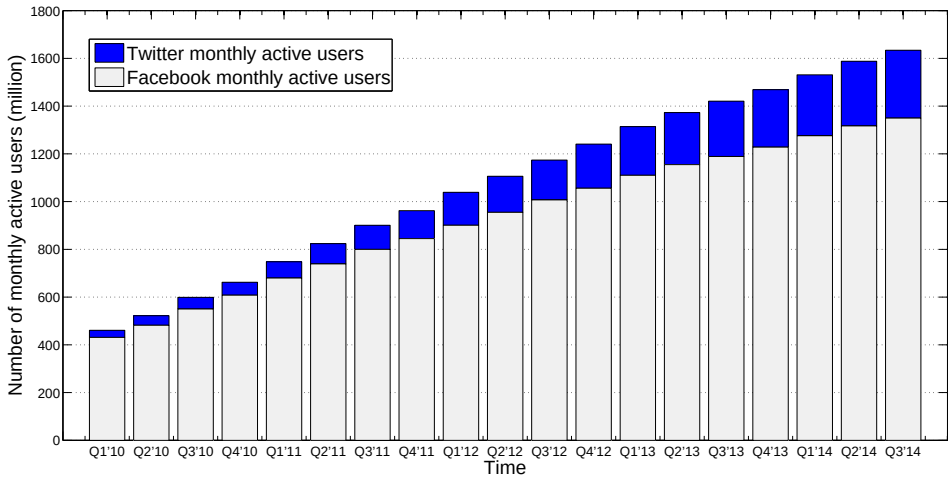


Figure 2: Nombre d'utilisateurs actifs mensuels de *Facebook* et de *Twitter*. Source: Statista.com [2]

recherche. Dans les moteurs de recherche classiques (comme *Google*, *Yahoo* ou *Bing*), l'importance d'une page web est la base pour le classement. Cette valeur est calculée sur la base de l'analyse des hyper-liens entre les pages Web. Avec un ensemble de documents $V = \{v_1, ..., v_n\}$, quand il y a une requête q d'un utilisateur arrivant, le moteur de recherche cherche des documents dans V correspondant à la requête q, puis trie les documents dans l'ordre décroissant de leur pertinence pour la requête. Ce processus peut être réalisé grâce à une fonction de classement qui permet de calculer la similarité $sim(q, v_i)$ entre la requête q et un document $v_i \in V$. La fonction de classement peut être considéré comme le noyau qui détermine essentiellement la qualité du moteur de recherche.

À fin des années 1990, Cohen [14] a dit qu'il y aurait de nombreuses applications nécessitant un classement d'objets plutôt qu'un rangement en bonnes catégories. Toutes les applications dont les résultats renvoyés aux utilisateurs comme une liste d'objets triés les aident à trouver rapidement les objets les plus pertinents de ce qu'ils recherchent. Ça a été confirmé par beaucoup d'applications que nous avons aujourd'hui. Cela montre que le classement fait partie des problèmes importants et significatifs.

Un nouveau concept de classement a été récemment introduit pour les objets du Web, en particulier les pages web. La valeur d'importance est appelée la valeur

de page rank et le PageRank [4, 5] est considéré comme un algorithme de classement du Web le plus connu aujourd'hui. Essentiellement, il calcule la valeur de page rank basé sur une analyse des liens entre les pages Web d'un graphe du web.

Au cours des dernières années, l'extraction de données dans les réseaux sociaux en ligne est devenu un sujet de recherche à la mode [15, 16, 17]. Les ressources et le contenu des réseaux sociaux générés par les utilisateurs reflètent une vie sociale riche et l'esprit de la société humaine. Par conséquent, le classement des objets dans le réseau social (personnes, contenu, etc.) tend à devenir de plus en plus important.

Réseaux sociaux

La tendance de communication de ce début de 21^e siècle est associé au terme "réseau social" où les gens peuvent trouver et partager des informations rapidement et efficacement. Étant donné un nom ou une adresse mail, les gens sont capable de se trouver. Toutes les informations sur les réseaux sociaux peuvent se propager rapidement grâce à des connexions entre tous les membres (voir Figure 3). Le réseau social est un endroit connectant des gens ayant des intérêts similaires sur l'Internet, indépendamment de l'espace et du temps à travers différents services. Les réseaux sociaux sont devenus de plus en plus populaires et bien évidemment l'émergence de Facebook, Twitter, Youtube, etc., a considérablement changé la façon dont nous nous communiquons.

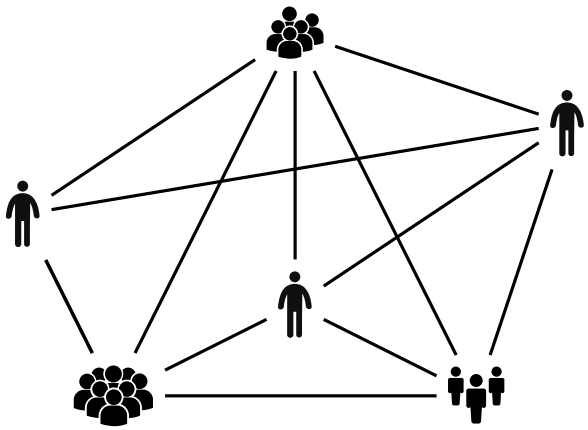


Figure 3: Les connections simplifiées du réseau social.

Le domaine de *Social Network Analysis* (SNA) consiste à étudier les relations, les connexions, les communications et les comportements des différents groupes sociaux. SNA est dérivé à la fois de la sociologie, de l'analyse de réseau et de la théorie de graphes. Les scientifiques utilisent l'informatique pour étudier les réseaux sociaux, le trafic de communication sur Internet et également la diffusion de l'information. Les résultats de SNA aident à expliquer les comportements sociaux.

Avec le SNA vient le problème de détermination des nœuds clés, aussi connu comme les facteurs critiques dans les réseaux sociaux. Ils sont des éléments (utilisateurs, articles, ...) qui sont considérés cruciaux par rapport à certains critères. On peut dire que dans les graphes sociaux les éléments clés sont des nœuds ayant la capacité de contrôler le flux d'information, les plus importants et les plus influents par rapport à d'autres nœuds dans le réseau.

Pour déterminer les nœuds clés dans les graphes de réseaux sociaux, on pourrait se baser sur la *mesure de centralité*. *"En théorie de graphe et analyse de réseau, la centralité se réfère aux indicateurs qui identifient les sommets les plus importants dans un réseau"* [18]. Selon les propriétés des graphes et les problèmes à résoudre, les méthodologies pour trouver les nœuds clés sont différents. En général, on distingue quatre types principaux de la mesure de centralité :

- *Centralité de degré (degree centrality)* : défini par le nombre de liens d'un nœud avec d'autres nœuds. Dans les réseaux sociaux, cette valeur d'un utilisateur est principalement déterminée par le nombre d'autres utilisateurs avec lesquels il partage des connexions dans un sens ou l'autre.

- *Centralité de proximité (closeness centrality)* : montre comment un nœud est proche d'autres nœuds. Le nœud central a des distances plus courtes aux autres, et la distance entre une paire de nœuds est définie par la longueur du *chemin le plus court* entre eux. Dans les réseaux sociaux, plus petite est la distance qu'un utilisateur a besoin pour atteindre les autres, plus cet individu est important. Dans le contexte de propagation de l'information, cette mesure pourrait donner une vue fondamental sur la façon de diffuser le contenu le plus rapidement possible étant donnée une seule ou peu de sources d'information. Inversement, ce sont les nœuds qui peuvent recevoir des informations venant d'autres nœuds plus rapidement et avec moins d'effort.

- *Centralité de milieu (betweenness centrality)* : d'un nœud est défini par combien de chemins les plus courts entre toutes les paires de nœuds le traversent. Cela mesure la capacité d'un élément du réseau à se connecter à d'autres éléments. Cette mesure est importante pour étudier la façon dont un réseau est connecté à l'intérieur et comment maintenir ou renforcer la structure du réseau. Si un nœud ayant une haute centralité est enlevé du réseau, beaucoup de paires de nœuds risquent de ne plus pouvoir communiquer entre eux.

- *Centralité de vecteur propre (eigenvector centrality)* : mesure l'importance des nœuds dans un rseau, partant du principe qu'une connexion sortant d'un nœud plus important contribue plus de score qu'une connexion équivalente sortant d'un nœud moins important. Cette déclaration peut être interprétée de différentes manières. Dans un graphe du web, un nœud est considéré comme important s'il est pointé par (beaucoup) d'autres nœuds importants. De même, dans le contexte de réseau social, un utilisateur est considéré influent s'il est connecté avec (beaucoup) d'autres utilisateurs influents. L'algorithme de PageRank est une variante de la centralité de vecteur propre et cette thèse donc portera principalement sur cette mesure de centralité.

Il y a beaucoup de façons de construire un modèle de classement pour les réseaux sociaux. Elles consistent à répondre à deux questions : que trie-t-on et comment le trier. Cependant, la construction d'un tel modèle de manière satisfaisante n'est pas une tâche facile.

Défis du classement de réseau social

Ordonner des objets dans les réseaux sociaux est difficile à cause de différentes raisons.

Tout d'abord, les graphes de réseaux sociaux sont *hétérogènes*. Au contraire des graphes du web *homogènes* qui consistent simplement en pages Web, fichiers et liens entre eux, les graphes sociaux contiennent une large variété d'objets tels que les personnes, les évènements, les messages... Chaque réseau social a ses propres objets et des liens spécifiques. Par exemple Twitter a *utilisateurs*, *tweets*, *retweets* comme objets et *following/follower*, *tweet*, *retweet*, *mentionner*, *préférer* sous forme

de liens. D'autre part, Facebook a *utilisateurs*, *messages*, *commentaires* comme objets et *amis*, *commentaires*, *aimer*, *partager*, *mentionner* comme liens. Les graphes sociaux sont hétérogènes en termes des objets ainsi que des liens (voir Figure 4). Cette hétérogénéité rend la question de *comment ordonner* beaucoup plus difficile, par exemple *devrions-nous ordonner un utilisateur de Twitter en fonction de ses followers ou du nombre de fois que d'autres utilisateurs partagent ses tweets? Ou les deux?*

Deuxièmement, les graphes sociaux évoluent beaucoup plus vite que les graphes du Web. Leur dynamique peut être observée par les apparitions rapides de nouveaux objets et les disparitions de relations anciennes ou d'éléments obsolètes. Supposons que nous voulons calculer les valeurs de PageRank de nœuds d'un graphe social, il n'est pas pratique d'utiliser des méthodes de calcul classiques parce qu'elles doivent prendre un certain temps pour s'adapter aux changements de la structure du graphe, et ensuite recalculer les valeurs de PageRank presque du début. C'est la raison pour laquelle nous allons introduire *D-itération*, un algorithme qui permet d'actualiser le vecteur PageRank par rapport à l'évolution du graphe.

Troisièmement, même si nous avons déjà un modèle de classement, ce n'est pas évident de savoir comment évaluer sa validité. Le classement du Web a aussi ce problème parce que la base de validité est difficile à définir. Ce défi concerne plus ou moins la question de *quoi ordonner*. Si l'on veut chercher les utilisateurs les plus influents dans le réseau, on pourrait se retrouver coincé de la vérification de la validité. Dans ce cas, un facteur humain peut aider, en introduisant un risque de biais non-négligable mais au prix de beaucoup de temps et d'efforts. Pour éviter ce problème, nous choisissons d'ordonner des objets selon des critères vérifiables comme le dynamisme des utilisateurs dans les graphes sociaux qui peut être correctement dérivé de l'enregistrement de leurs activités accessibles.

LiveRank

Beaucoup d'exemplaires de grands graphes datant d'une dizaine d'années ou plus sont disponibles aujourd'hui. Reconstruire ce qui reste de ces archives pourraient conduire à des études intéressantes de l'évolution à long terme de ces graphes. Pour les grandes archives où l'on s'intéresse à un type particulier de pages Web, recrawler

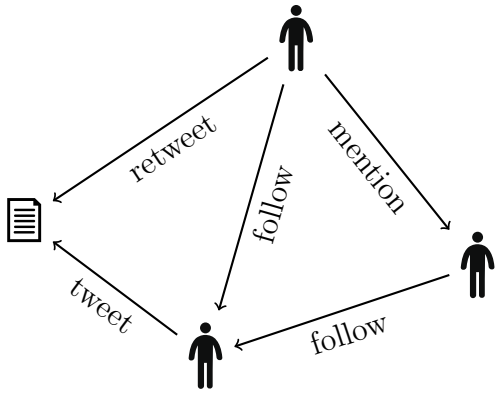


Figure 4: Hétérogeneité du graphe de réseau social de Twitter: différents types d'objets (personnes, tweets) et liens (follow, tweet, retweet, mention)

.

un grand ensemble de pages peut être prohibitif. Nous proposons d'identifier le plus rapidement possible une fraction significative des pages encore en vie. Une sélection supplémentaire peut alors être faite pour identifier un ensemble de pages appropriés pour l'etude, afin de les explorer. Ces techniques seraient particulièrement intéressantes lorsque le test de la viabilité d'une page est beaucoup plus léger que le téléchargement complet. C'est par exemple le cas pour le Web avec des requêtes HEAD par rapport à des requêtes GET.

De plus, certains graphes tendent à être plus difficiles à crawler avec le temps. Par exemple, Twitter restreint sa capacité à être exploré. Effectuer une analyse complète était possible il y a quelques années [56], mais il pourrait être prohibitif de nos jours à cause des restrictions de volume introduites dans l'interface API. De nouvelles techniques doivent donc être développées pour identifier efficacement les comptes actifs efficacement dans de tels contextes.

Étant donné un "vieux" graphe, notre objectif est d'identifier une fraction significative des éléments qui sont encore en vie ou actifs à l'instant présent. Nous considerons le coût comme le nombre de récupérations nécessaires pour atteindre cet objectif. Une mesure typique de coût sera le nombre moyen de récupérations pour obtenir un élément actif. La stratégie consiste à produire un ordre pour récupérer les pages. Nous appelons *LiveRank* un ordre tel que les éléments qui sont encore en vie ont tendance à apparaître en premier. Nous considérons le problème de trouver un LiveRank efficace dans trois contextes : statique quand il est cal-

culé uniquement à partire du vieux graphe et des liens enregistrés à l'époque, sur la base d'échantillonnage quand un échantillon sur le graphe actuel est testé dans une première phase permettant d'ajuster l'ordre selon la viabilité des éléments de l'échantillon, ou enfin dynamique quand il s'est progressivement calculé au cours de la récupération des pages.

Nous proposons divers algorithmes de LiveRank en fonction de la structure du graphe à un instant donné. Nous les évaluons sur deux graphes de Web réels (10-20 millions de nœuds) et sur un graphe de Twitter (40 millions de nœuds). Nous donnons plusieurs algorithmes sur la base de la structure du graphe. Un bon algorithme consiste à combiner une petite phase d'échantillonnage avec une propagation de l'information d'activité partielle obtenue sur le graphe restant. Il permet de recueillir de 15% à 75% des nœuds actifs avec un coût qui reste en dessous d'un facteur 2 de la solution optimale.

Système de localisation

Les systèmes de positionnement sont cruciaux pour la société numérique d'aujourd'hui. Ils aident à localiser des objets ou des personnes portant les objets, et à fournir l'information géographique pour faciliter de nombreuses activités humaines. Par exemple, les systèmes de navigation de véhicule sont indispensables pour les conducteurs dans les grandes villes. Certains services basés sur la localisation sont déployés dans les centres commerciaux afin que les clients puissent obtenir un guidage tout en marchant dans un environnement complexe et puissent recevoir des promotions ou publicités des magasins. Le marché des services de localisation intérieure et extérieure a connu une croissance rapide dans la dernière décennie. Cependant, la question majeure du système de localisation intérieure est affaire de compromis entre le coût de la mise en œuvre et la précision du système. Un système moins cher qui n'exige que quelques appareils sera moins précis et risquera d'introduire des erreurs de localisation.

Le système de localisation global (GPS) est très populaire et largement utilisé pour la localisation extérieure. Lorsque la vue-directe (line-of-sight) aux quatre satellites GPS est disponible, le lieu (latitude, longitude et altitude) et des informations de synchronisation peuvent être obtenues. Cependant, la qualité de lo-

calisation est sensible aux conditions météorologiques, par exemple quand la vue du ciel est limitée à cause du brouillard, de la pluie, de nuages, etc., ou bloquée par de grands bâtiments dans les zones urbaines. Ces problèmes peuvent dégrader considérablement la précision du GPS. Comme on peut s'y attendre, le GPS ne fonctionne pas bien pour une utilisation à l'intérieur en raison de l'absence de la vue-directe avec les satellites. Il existe également des systèmes de localisation cellulaires [69] qui sont construits en mesurant la puissance du signal de trois ou plusieurs stations de base pour localiser l'utilisateur. Cependant, ces solutions ne fonctionnent pas non plus très bien à l'intérieur.

Divers systèmes de localisation à l'intérieur ont été développés, par exemple [70, 71, 72]. Ils peuvent être classés à base de réseau ou de non-réseau. L'approche à base de réseau, qui prend avantage de l'infrastructure du réseau existant tels que les réseaux locaux sans fil (WLAN), sans exiger de nouvelles infrastructures, peut maintenir le coût de déploiement bas. L'approche à base de non-réseau utilise une infrastructure de localisation dédiée et peut avoir une grande fiabilité, mais à un coût supplémentaire. Par exemple, les solutions actuelles d'ultra-sons et d'infrarouges ont un coût de déploiement élevé. On peut aussi considérer une solution simple basée sur la détection de proximité comme iBeacon [73] qui n'est capable de dire que si un appareil est à proximité d'un point de référence. Certains systèmes utilisent la lumière visible pour construire un système de localisation précise [74, 75]. Un bon système de localisation devrait être bon marché en offrant une précision acceptable.

La construction d'un système de localisation simple et efficace est toujours un défi. Techniquement, cela dépend du nombre de points de référence (PR) que nous pouvons avoir, des technologies utilisées (par exemple, base-RF, ultra-sons, infrarouge, etc.), ainsi que de la nature de l'environnement. Dans cette étude, nous proposons une méthode de localisation basée sur la géométrie qui peut déterminer la position de l'utilisateur en utilisant un PR unique et en exploitant son mouvement, par exemple en marchant ou en agitant son appareil, et quelques informations simples. Comme la solution ne nécessite qu'un seul PR et qu'elle peut donner un bon résultat même dans un environnement bruté, notre approche apporte des avantages compétitifs par rapport à d'autres méthodes. La méthode est intéressante et peut avoir un fort potentiel pour améliorer la technologie d'aujourd'hui ou des solutions existantes.

Structure du mémoire et contributions

Cette thèse vise à étudier l'état de l'art du PageRank, puis nous présenterons l'algorithme *D-Itération*, initialement introduit par Dohy Hong [19], qui supporte le calcul du vecteur de PageRank de manière distribuée. Nous ne nous limitons pas au graphe du web, et étendons la méthode à d'autres types de graphes, par exemple des graphes de réseau social et d'appels téléphoniques, dans lesquels le PageRank classique ne peut pas donner un classement approprié.

En gardant cela en tête, nous introduisons également une extension du PageRank, appelée *LiveRank* qui peut aider par exemple à classifier les utilisateurs du graphe de *Twitter* par rapport à leur dynamisme, ou à détecter rapidement les pages vivantes d'un graphe du web ancien. Cette extension peut être utile pour des crawleurs de web, ou plus généralement dans le contexte de la recherche d'information.

La thèse est organisée comme suit: le chapitre 2 introduit l'algorithme de PageRank et à quoi il ressemble lorsqu'il est appliqué à un graphe du Web ou un graphe de log téléphonique.

Le chapitre 3 présente D-Itération (DI), un nouvel algorithme qui a été proposé par Dohy Hong [19], pour calculer le vecteur de PageRank. Le chapitre inclut la définition de DI et les principaux résultats théoriques (l'exactitude, la convergence, et la condition d'arrêt...). L'algorithme montre son potentiel grâce à des expériences sur des données réelles en comparaison avec d'autres algorithmes classiques.

Le chapitre 4 montre des évaluations de performance de DI et d'autres algorithmes, comme like Power Iteration (PI), Jacobi, Gauss-Seidel (GS), Adaptive On-Line Page Importance Computation (OPIC) et Generalized Minimal Residual (GMRES), sur un petit graphe du web. Nous comparons les méthodes en termes d'itérations et de coût d'exploitation. Nous proposons aussi une stratégie pour approximer le vecteur de PageRank d'un graphe dont toutes les colonnes (ou lignes) ne sont pas données.

Le chapitre 5 présente des évaluations de performance de DI sur différents critères: le nombre d'itérations, le coût d'exploitation, etc., en comparaison avec d'autres algorithmes. Les paramètres de comparaison sont: itérations, opérations élémentaires et messages de requête. Pour les deux derniers critères, nous proposons

une variante de DI qui obtient la meilleure performance par rapport aux autres méthodes d'itération classique.

Le chapitre 6 introduit LiveRank, un ordre de classement de nœuds dans un graphe tels que les nœuds vivants devraient apparaître en premier. Nous proposons divers algorithmes de LiveRank basés sur des structures de graphes, ensuite les évaluons sur deux captures de Web et sur une capture de Twitter. Nous avons publi les résultats dans [22, 23].

Chapitre 7 conclut la thèse et souligne des directions de recherche dans le futur.

Le mémoire contient également quelques annexes dont certains sont liées au PageRank et d'autres pointent vers des sujets supplémentaires sur lesquels j'ai travaillés pendant les trois dernières années.

L'Annexe A présente un exemple de comment l'algorithme GMRES fonctionne.

L'Annexe B donne une preuve détaillée sur la relation de l'algorithme de Gauss-Seidel et l'une des variantes de D-Itération.

À part du PageRank, je participais à une démonstration de la théorie de jeu dans le cadre des Open Days d'Alcatel-Lucent Bell Labs. Dans l'Annexe C, je présente deux prototypes graphiques j'ai construites pour cet évenement. Le premier prototype concerne la distribution des ressources limitées sur plusieurs objets pour gagner plus d'entre eux. Le second prototype est une extension du premier dans laquelle nous considérons une quarantaine de pays européens comme des objets à gagner, avec l'introduction de différentes stratégies de jeu, par exemple, Tit-for-tat, ressources inégales. Les deux prototypes sont disponibles au téléchargement.

Dans l'Annexe D, nous proposons une nouvelle méthode appelée Two-Step Movement (2SM) pour estimer la position d'un appareil mobile (MT). Il ne nécessite qu'un seul point de référence (RP) en combinaison avec l'exploitation de l'information utile générée par le changement de position du MT ou le mouvement de l'utilisateur. Nous combinons par ailleurs la méthode 2SM avec la technique multi-sampling pour améliorer la performance de localisation. On peut alors réduire le nombre de points de référence nécessaires et réduire ainsi le coût de construction du système de localisation. En plus, une généralisation de 2SM (appelée G2SM) au contexte de communication "machine à machine" est également décrite qui permet au point de

référence unique de se déplacer ou d'être un autre appareil mobile. Nous avons publié les résultats correspondants dans [24, 25].

Chapter 1

Introduction

1.1 Context

The Internet has been growing rapidly over the last decade, particularly the World Wide Web. According to the Netcraft's survey [3], there were more than 1 billion websites by the end of October 2014. Another survey done in [1] shows a slightly different number, about 1.1 billion websites active by the same period. Figure 1.1 indicates the evolution of World Wide Web over the last 14 years, starting from 2000 to 2014.

This thesis will be mainly talking about PageRank, a ranking algorithm first introduced by Page et al. (cf [4, 5]) and used by the Google Internet search engine as one of the most important factors to rank web pages. Note that the term website refers to a domain name whereas web page is a page that stems from that domain name [1]. Search engines like *Google*, *Bing* or *Yahoo* rank documents at web page level. Given 1 billion websites on the Internet, according to [10] Google handles approximately 50 billions web page indices , roughly saying that each website has in average 50 web pages indexed by Google. However, the size of WWW could turn out to be much more enormous due to the fact that a vast majority of web pages can be reached only by querying web servers. On the other hand, the emergence

[1] We should clearly distinguish these two terms because in the context of PageRank algorithm, there are several methods [6, 7, 8, 9] which estimate the PageRank value of a web page as a function of PageRank of the whole website, specially in parallel context where computation and storage scheme at each server is often at website level.

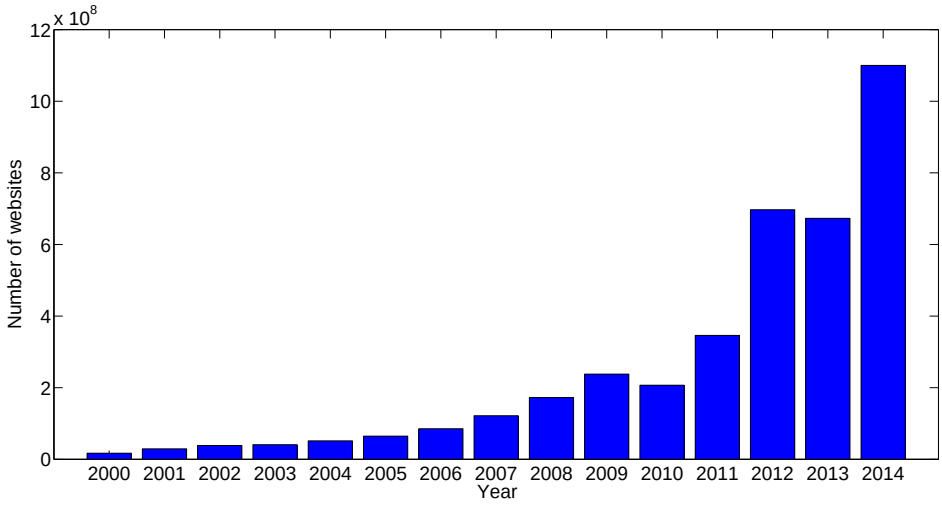


Figure 1.1: Number of websites on the Internet over years. Source: InternetLiveStats.com [1]

of social networks, such as *Twitter* [11], *Facebook* [12] and *Google+* [13], enriches the World Wide Web significantly thanks to its rapid growth. Figure 1.2 shows the evolution of monthly active users of Twitter and Facebook from the first quater 2010 (Q1'10) to the third quater 2014 (Q3'14). As we can see, the websites have been increasing exponentially, in contrast to linear growth of the social networks. Intuitive explanation may come from the fact that websites have no limit of growing and more modern (smarter) search engines can index more query-based web pages mentioned above. Meanwhile, social network members are bounded by human population which increases linearly.

1.1.1 Ranking problems

Ranking objects is one of the important and typical issues in our daily life. Many applications need to rank objects according to certain criteria, as simple as ranking students in a class according to average grades, or more complicated as ranking universities. Ranking objects means to arrange them in accordance with some criteria depending on the specific application.

In the era of the Internet, a typical problem emerging in the last decades is the ranking of results returned by search engines. In conventional search engines (like

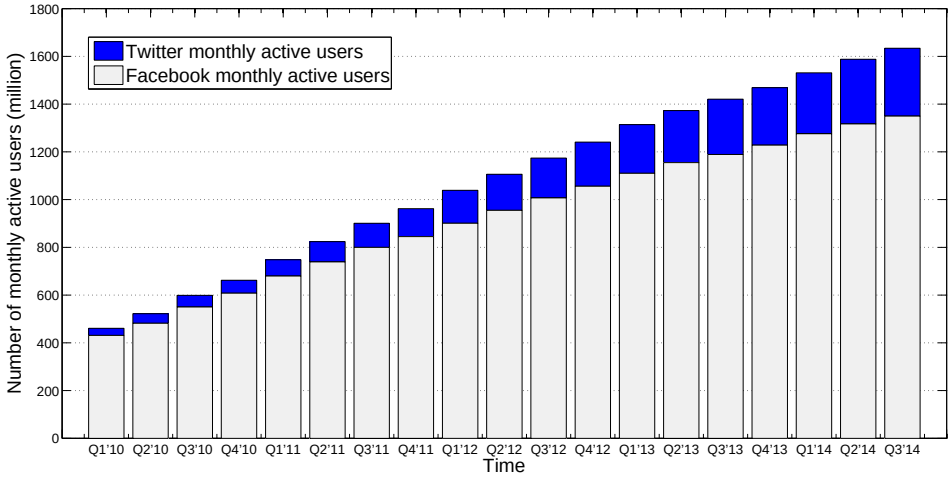


Figure 1.2: Number of monthly active users of *Facebook* and *Twitter*. Source: Statista.com [2]

Google, *Yahoo* or *Bing*), the importance of a web page is the basis for ranking. This value is determined based on the analysis of graph links between web pages. With a set of documents $V = \{v_1, ..., v_n\}$, when there is a user's query q arriving, the search engine looks for documents in V matching the query q, then sorts the documents according to their relevance to the query in descending order. This process can be done thanks to a *ranking function* which allows us to compute the similarity $s(q, v_i)$ between the query q and a document $i \in V$. Obviously, the ranking function can be seen as the core and significantly determines the quality of the search engine.

In the late 1990s, Cohen [14] made the comment that there would be many applications requiring arrangement of objects rather than classifying them. All applications whose results returned to users as a list of sorted objects helps them quickly reach those most relevant to what they look for. This comment has been confirmed by an uncountable number of such applications we meet nowadays. That said, ranking is one of the important and meaningful problems.

A new concept of ranking was recently introduced to orient rating objects on the Web, specifically web pages. The pages need to be arranged in descending importance. Value of importance is called page rank value and the PageRank [4, 5] is considered as the most successful and well-known web ranking algorithm. Essentially,

it calculates page rank value based on an analysis of the links between web pages in a web graph.

In recent years, data mining in online social networks has become a trendy research topic [15, 16, 17]. Resources and content of social networks are generated by users, reflecting an increasingly rich social life and spirit of human society. Therefore, ranking objects in social networks (people, content, etc) has also become important.

1.1.2 Social networks

Communication trends of 21^{st} century are associated with the phrase "Social network" where people can find and share information rapidly and effectively. Given a name or an email address, people are able to find each other. All information on the social networks can be quickly spread thanks to connections among members (see Figure 1.3). The social network is a place connecting members with similar interests on the Internet irrespective of space and time through different services. Social networks have become more and more popular and obviously the advent of Facebook, Twitter, Youtube, etc, has significantly changed the way we communicate.

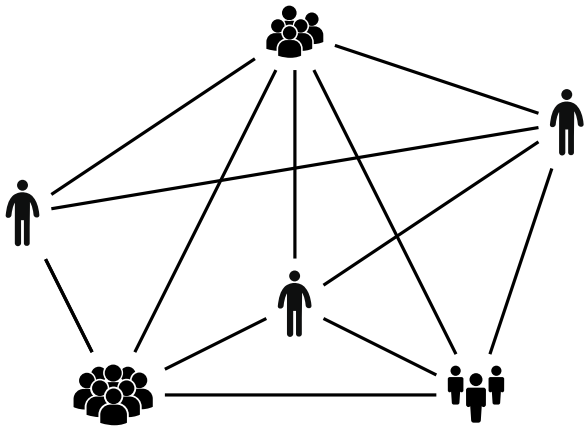


Figure 1.3: Simplified social network connections.

Social Network Analysis (SNA) involves studying relationships, connections, communications and behaviour patterns of different social groups. SNA is derived from sociology, network analysis and graph theory. Scientists use computers to study social networks, communication traffic on the Internet and also information dissemination. Results of SNA help to explain social behaviours.

Part of the problem of the SNA is determining the key nodes, also known as the critical factors in social networks. They are elements (users, items, ...) which are considered crucial with respect to certain criteria. It can be said that in social graphs the key elements are nodes who have the ability to control the flow of information, they are the most prominent and influential compared to other nodes in the network society.

To determine the key nodes in social network graphs, one could use *centrality* measurement. *"In graph theory and network analysis, centrality refers to indicators which identify the most important vertices within a network"* [18]. Depending on properties of social graphs and problems to be solved, methodologies of finding the key nodes are different. However in general, there are four main types of centrality measurement:

- *Degree centrality*: is defined as the number of links a node has with other nodes. In social networks, this value of a user is mostly determined by how many other users he has connections with and/or have connections with him.

- *Closeness centrality*: shows how close a node is to other nodes. The more central node has lower distances to others, and the distance between a pair of nodes is defined by the length of their *shortest path*. In social networks, the least number of steps a user can reach others, the more important he is. In the context of information propagation, this measurement could give a basic view on how to spread content as quickly as possible given only one single or a few information sources. Reversely, they are the nodes that can receive information from other nodes fastest and with least effort.

- *Betweenness centrality*: of a node is defined by how many shortest paths between all pairs of nodes passing through it. This measures the ability of a network element to connect other elements. This measure is crucial to study how well a network is internally connected and how to maintain or strengthen the network structure. If a node having high betweenness centrality is removed from the network, it turns out that many pairs of nodes may not communicate any more.

- *Eigenvector centrality*: measures the importance of nodes in a network, saying that a connection from a higher-score node results in a higher score contribution

than an equivalent connection from a lower-score node [2]. PageRank algorithm is a variant of the eigenvector centrality and this thesis therefore will be mainly talking about this centrality measure.

There are many ways to build a ranking model for social networks. It consists in answering two questions: what to rank and how to rank. However, constructing a good model is not such an easy task.

1.1.3 Challenges of social network ranking

Ranking objects in social networks is challenging due to various reasons.

Firstly, social network graphs are *heterogeneous*. Unlike *homogeneous* web graphs which simply consist of web pages, files and hyperlinks between them, social graphs contain a variety of objects such as people, news, posts... Each social network has its own specific objects and links. For instance Twitter has *users*, *tweets*, *retweets* as objects and *following/follower relationship*, *tweet*, *retweet*, *mention*, *favorite* as links. On the other hand, Facebook has *users*, *posts*, *comments* as objects and *friendship*, *comment*, *like*, *share* as links. Social graphs are heterogeneous in terms of both objects and links (see Figure 1.4). This heterogeneity makes the question of *how to rank* much more difficult, for instance *should we rank a Twitter user based on his followers or on the number of times other users retweet his tweets? Or both?*

Secondly, social graphs evolve much faster than web graphs. Their dynamics can be observed by the rapid appearances of new objects and disappearances of old relationships or out-dated elements. Assume we want to compute PageRank values of nodes in the graphs, it is not convenient to use classical computation methods because they must take some time to adapt to the graph structure changes, then recompute the PageRank values almost from scratch. That is why we will later introduce *D-Iteration*, an algorithm that can continuously update the PageRank vector along with the graph evolution.

Thirdly, even when we have already a raking model, it is not obvious how to evaluate its validity. Web ranking has also encountered this issue since the

[2]This statement can be interpreted in different ways. In a web graph, a node is considered important if it is pointed to by (many) other important nodes. Similarly, in the context of social network, a user is influential if he is connected to by (many) other influential users.

ground truth is hard to define. This challenge more or less relates to the question of *what to rank*. If one wants to find the most influential users in the graph, then he might get stuck at validity verification. In this case, the human factor may help, i.e., a human will check the validity based on the outcomes of the ranking algorithm, and of course it takes much time and effort. To avoid this problem, we choose to rank objects according to some verifiable criteria like activeness of users in social graphs which can be correctly derived by their accessible activity records.

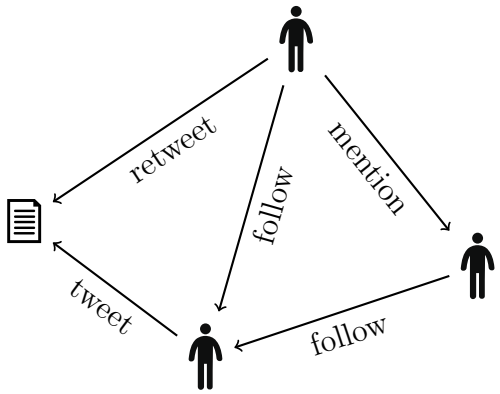


Figure 1.4: Heterogeneity of Twitter social network graph: different types of objects (users, tweets) and links (follow, tweet, retweet, mention).

1.2 Roadmap and contributions

This thesis aims at investigating the state of the art of PageRank, then we will present *D-Iteration* algorithm, firstly introduced by Dohy Hong [19], that fully supports PageRank vector computation in a distributed way. The scope of this thesis is not limited to the web graph but stretched to other types of graphs, e.g., social network or call-log graphs in which the classical PageRank may not give an appropriate ranking order. Keeping that in mind, we also introduce an extension of PageRank, called *LiveRank* which can help for example to rank users in *Twitter* graph with respect to their activeness, or to quickly detect alive pages in a web graph that makes a lot of sense in web crawling applications, or generally in the context of information retrieval.

The thesis is organized as follows. Chapter 2 introduces the PageRank algorithm and how it looks like when applied on a web graph or a call-log graph.

Chapter 3 introduces D-Iteration (DI), a new algorithm for computing PageRank that has been proposed by Dohy Hong [19]. The chapter is organized as a self-contained summary of DI's definition and main theoretical results (correctness, convergence bounds, stopping condition...). The algorithm shows its potential through experiments on real data in comparison with other classical algorithms.

Chapter 4 shows the performance evaluations between DI and other algorithms, like Power Iteration (PI), Jacobi, Gauss-Seidel (GS), Adaptive On-Line Page Importance Computation (OPIC) and Generalized Minimal Residual (GMRES), on a small web graph. We compare the methods in terms of iterations and operational cost. We also propose a strategy to approximate the PageRank vector of a graph of which all columns (or rows) are not given.

Chapter 5 provides further performance evaluation of DI on different metrics: iteration, operational cost, etc, in comparison with the other algorithms like PI, GS and OPIC. The comparison metrics are iterations, elementary operations and request messages. For each of the later two criteria, we proposed a DI variant that gives better performance compared to the other classical iteration methods. Additionally, we show the application of PageRank to Twitter user ranking and compare its efficiency with the classical indegree ranking.

Chapter 6 talks about LiveRank, a ranking order of nodes in a graph such that alive nodes should appear first. We propose various LiveRank algorithms based on graph structures. We evaluate them on two Web snapshots and on a Twitter snapshot. We propose several propositions based on the graph structure of the snapshot. A rather simple combination of a small sampling phase and the propagation of the partial activity information are obtained in the remaining graph of the snapshot through a modified PageRank algorithm. We published the results in [22, 23].

Chapter 7 concludes the thesis and points out future research directions.

The thesis also contains a few appendices, some related to the main subject and others pointing to additional work I made during the last three years.

Appendix A shows through a didactic example how the GMRES algorithm works.

AppendixB gives a detailed proof of a relationship between the Gauss-Seidel algorithm and one of D-Iteration's variant.

Unrelated to PageRank issues, I participated to a demonstration of game theoretical results for Alcatel-Lucent Bell Labs' Open Days. In Appendix C, I present two graphical demonstrations I built for that occasion. The first demo is about distributing limited resources over several objects to win most of them. The second demo is an extension of the first one that we consider forty European countries as objects to win in Internet Service Provider (ISP) market, along with the introduction of various game strategies to play with, e.g., Tit-for-Tat, unequal resources. The two demos are available for download.

In Appendix D, we propose a new method called Two-Step Movement (2SM) to estimate the position of a Mobile Terminal (MT). It requires only one reference point (RP) by exploiting useful information given by the position change of the MT or user movement. Also, we combine the 2SM method with multi-sampling technique to improve the positioning performance. One can therefore reduce the number of RPs required and lower the system cost. Furthermore, a Generalization of the Two-Step Movement (G2SM) to Device-To-Device context is also described as it allows the unique Reference Point to move or to be another mobile device. We published the results in [24, 25].

Chapter 2

PageRank

Issues concerning systems of linear equations have attracted many research efforts. A system of linear equations can be written under the form of the equation $Ax = b$ where A is a $n \times n$ matrix, b is a constant vector of size n and unknown vector x. Different methods exist for solving this equation, such as Jacobi, Gauss-Seidel, etc. They vary in memory requirement, computation cost and convergence speed. Further, those algorithms can deal with different problems which spread throughout a large research area. One of their well-known applications is solving the PageRank equation.

PageRank is a link analysis algorithm that has been initially introduced in [4] and used as an important factor among thousands of others by the Google Internet search engine to rank web pages. It assigns a numerical value to each element of a hyper-linked set of nodes, such as the World Wide Web. The algorithm may be applied to any collection of entities (nodes) that are linked through directional relationships. The numerical value assigned to each node is called PageRank and is associated to an eigenvector problem of which we are interested in the computation issue. Although PageRank may today only be a small part of Google's ranking algorithm (the complete algorithm is obviously kept secret, but it seems to take into account hundreds of parameters, most of them have been related to the user's profile), but it stays appealing, especially in the research community, as it balances simplicity of definition, ranking efficiency and computational challenges. These challenges include the growing size of the dataset (Web graphs can have tens of billions of nodes) and the dynamics of the structure that requires frequent updates.

In this chapter, our main contributions is conducting a survey on the PageRank algorithm, including an introduction of different techniques solving the PageRank equation like Power Iteration, Gauss-Seidel, etc. Next, we visualized a web graph and a call-log graph in descending PageRank order to better understand the graph structures. The structures will be explained using the diffusion approach which will also be presented in this chapter. We also conducted a survey on some basic web graph compression techniques which exploit locality of the graph structures.

This chapter is organized as follows. Section 2.1 presents the PageRank model. Section 2.2 introduces several methods to solve the PageRank equation. Section 2.3 shows the definitions of collection and diffusion approaches in the PageRank context. Section 2.4 and Section 2.5 show how a web graph and a call-log graph look like respectively under canonic and PageRank point of view. Section 2.6 talks about some existing web graph storage methods. Finally, Section 2.7 concludes the chapter.

2.1 Model

The informal definition of PageRank is rather simple: it is an importance vector over Web pages such that important pages are referenced by important pages [26].

More formally, let $G = (V, E)$ be a weighted oriented graph. The size of G is $n = |V|$ and $w_{ij} > 0$ is the weight of edge $(i, j) \in E$. G represents a set of nodes and their (weighted, oriented) relationships. In [26], G was a Web graph, V representing web pages and E hyperlinks, but the principle applies to most structured sets, e.g., social network graphs.

Let P be a $n \times n$ diffusion matrix defined by:

$$P_{ij} = \begin{cases} \dfrac{w_{ji}}{\sum_{(j,k)\in E} w_{jk}} \text{ if } (j,i) \in E, \\ 0 \text{ otherwise.} \end{cases} \tag{2.1}$$

P is a left substochastic matrix, column j summing to 1 if node j has an outgoing edge at least, 0 if j is a dangling node. Note that:

- For unweighted graphs, the expression of P is simpler: for $(j, i) \in E$, we just have $P_{i,j} = 1/\text{out}(j)$, where $\text{out}(j)$ is the out-degree of j.

- Some variants of PageRank require P to be stochastic. For these variants, one usually pads the null columns of P with $\frac{1}{n}$ (*dangling nodes completion*).

P represents how importance flows from one node to another. When it is stochastic, it represents the Markov chain over V implied by the edges E. In that case, the PageRank can be defined as a stationary state of the Markov chain, that is a distribution x over V that verifies

$$x = Px. \tag{2.2}$$

Thanks to Perron-Frobenius theorem, we know that x is unique if G is strongly connected [27].

In practice, the following variant is used instead of (2.2):

$$x = dPx + (1 - d)Z, \tag{2.3}$$

where $0 < d < 1$ is called *damping* (often set to 0.85), and Z a *default distribution* over V (often set to the uniform distribution).

If P is stochastic, the solution x of (2.3) is a distribution, which corresponds to the Markov chain defined by: with probability d, use the Markov chain induced by P; otherwise jump to a random node according to distribution Z.

Introducing parameters d and Z has many advantages:

- It guarantees the existence and uniqueness of a solution for any substochastic matrix P, without any assumption on G.

- It speeds up the PageRank computation (cf below).

- Parameter d introduces some locality: influence of a node at distance k is reduced by a factor d^k. This strengthens the impact of the local structure and mitigates the possibility of malicious PageRank alterations through techniques like links farm [28].

- Parameter Z allows to customize the PageRank. For instance, one can concentrate the default importance on pages known to talk about some given topic to create a topic-sensitive PageRank [29].

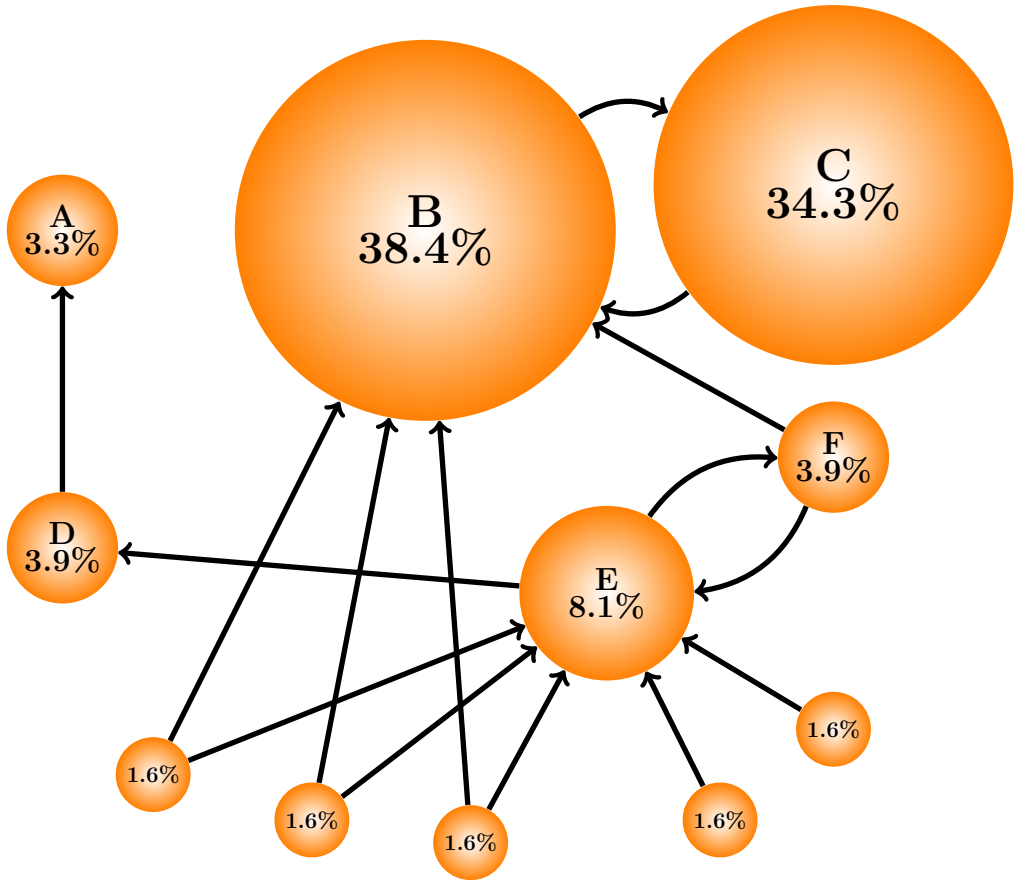


Figure 2.1: Example of PageRank values of nodes in a graph.

In the rest of the thesis, unless stated otherwise, we focus on solving (2.3). Writing the solution is straightforward:

$$x = (1-d)(I-dP)^{-1}Z, \text{ where } I \text{ is the identity matrix.} \tag{2.4}$$

However, such a direct approach cannot be used due to the size of P that forbids an explicit computation of $(I-dP)^{-1}$. Instead, one can use different iterative methods (see [30] and Section 2.2).

Figure 2.1 shows an example of *normalized* PageRank values of nodes in a graph with the vector Z uniform on V. That means, if the matrix P is substochastic, L1-norm of the vector x, denoted by $\|x\|_1 = \sum_i x_i$, is less than one. In this case if one wants a distribution for output, we pad the null columns of P with $\frac{1}{n}$ (known as *dangling node completion*) before the computation.

The complexity of computing the PageRank of a graph rapidly increases with the number of nodes, as it is equivalent to computing an eigenvector on some huge, matrix, and efficient and accurately computing eigenvalues and eigenvectors of arbitrary matrices is in general a difficult problem. In the particular case of the PageRank equation, several specific solutions were proposed and analysed [30, 28] including the power method [31] with adaptation [32] or extrapolation [33, 34], or the adaptive on-line method [35], etc.

For performance evaluations in later chapters, we now briefly introduce these methods.

2.2 PageRank computation techniques

2.2.1 Power Iteration

The simplest approach is **Power Iteration (PI)**, also called **Jacobi**: starting from an initial guess vector x_0, the stationary PageRank vector is iteratively computed using (2.3):

$$x_{k+1} = dPx_k + (1-d)Z, \tag{2.5}$$

until the change between two consecutive vectors is negligible. During an iteration round, entries $x_{k+1}(i)$ are computed from $i = 1$ to $i = n$ using:

$$x_{k+1}(i) = d\sum_j P_{i,j}x_k(j) + (1-d)Z(i). \tag{2.6}$$

It is straightforward that the error decays by a factor at least d at each iteration (hence one of the interests of introducing a damping factor). PI requires to maintain two vectors x_k and x_{k+1}. The Power Iteration is shown in Algorithm 1.

2.2.2 Gauss-Seidel

The **Gauss-Seidel (GS)** applied to PageRank consists in using the updated entries of x_k as they are computed:

$$x_{k+1}(i) = d\left(\sum_{j<i} P_{i,j}x_{k+1}(j) + \sum_{j\geq i} P_{i,j}x_k(j)\right) + (1-d)Z(i). \tag{2.7}$$

Algorithm 1 Power Iteration: $x = dPx + (1 - d)Z$

```
1: for i = 1 : n do
2:     x_old(i) = 1/n;                          ▷ Initializing uniform vector x_0
3: end for
4: while (true) do
5:     for i = 1 : n do
6:         sum = 0;
7:         for j = 1 : n do
8:             sum+= d × P_{i,j} × x_old(j);
9:         end for
10:        x_new(i) = sum + (1 − d) × Z(i);
11:    end for
12:    for i = 1 : n do
13:        x_old(i) = x_new(i);
14:    end for
15:    Check the convergence, continue if necessary;
16: end while
```

Thanks to the immediate update, one needs to maintain only one vector and the convergence is faster, typically by a factor 2 asymptotically. The main downside of the update mechanism is the necessity to access the entries in a round-robin fashion, which can cause problems in a distributed scenario. The Gauss-Seidel is presented in Algorithm 2.

Algorithm 2 Gauss-Seidel: $x = dPx + (1 - d)Z$

```
1: for i = 1 : n do
2:     x(i) = 1/n;                              ▷ Initializing uniform vector x_0
3: end for
4: while (true) do
5:     for i = 1 : n do
6:         sum = 0;
7:         for j = 1 : n do
8:             sum+= d × P_{i,j} × x(j);
9:         end for
10:        x(i) = sum + (1 − d) × Z(i);
11:    end for
12:    Check the convergence, continue if necessary;
13: end while
```

2.2.3 Successive Over-Relaxation

The Gauss-Seidel belongs to a larger class of methods called **Successive Overrelaxation (SOR)**, but other SOR variants are seldom used for Web PageRank computations [36]. However, for performance evaluation purposes we introduce here the SOR algorithm. The PageRank vector x_{k+1} is computed as follows:

$$x_{k+1}(i) = d\left[\omega\left(\sum_{j<i} P_{i,j}x_{k+1}(j) + \sum_{j\geq i} P_{i,j}x_k(j)\right) + (1-\omega)x_k(i)\right] + (1-d)Z(i). \tag{2.8}$$

Intuitively, if $\omega > 1$ the algorithm may converge faster thanks to the boost induced by (2.7). However, this convergence is not guaranteed in general, i.e., the algorithm may diverge.

2.2.4 Generalized Minimal Residual

The **Generalized Minimal Residual (GMRES)** is an algorithm used to find the unknown x in the equation $Ax = b$ on the Krylov subspace $\mathcal{K}(A, r_0)$ where $r_0 = b - Ax_0$ and x_0 is the initial vector. In principle, GMRES finds the vector x by using the Krylov subspace $\mathcal{K}(A, r_0) = \{r_0, Ar_0, A^2r_0, ...\}$.

One can use GMRES to compute the PageRank vector after transforming the equation 2.3 to the form $Ax = b$ [1].

One important aspect of GMRES is the *restart* parameter. In the PageRank context, the number of nodes may go up to billions and the PageRank vector itself takes GBs, not to mention the whole web graph. By adjusting the *restart* value, one can restrict the Krylov subspace size so that it fits system memory. After some iterations equal to the *restart* value, the algorithm erases the current Krylov subspace, keeps the last result vector, then uses it as the initial guess vector for the next GMRES repetition. Its disadvantage is again the convergence speed due to the lack of information from previous result vectors in the subspace. To better

[1] We rewrite the PageRank equation $x = dPx + (1-d)Z$ as $(I - dP)x = (1-d)Z$ where I is an *identity matrix* (a square matrix with ones on the main diagonal and zeros elsewhere). The GMRES algorithm will then be applied to compute PageRank vector x in the quation $Ax = b$ such that $A = I - dP$ and $b = (1-d)Z$.

understand how GMRES works step-by-step, reader can refer to Appendix A where there is a numerical example of the algorithm.

2.2.5 Online Page Importance Computation

The **Online Page Importance Computation (OPIC)** was proposed in [35]. Its core idea: most PageRank algorithms implicitly use a *pull* approach, where the state of a node is updated according to the states of its incoming neighbors. By contrast, OPIC proposes a *push* approach, where the state of a node is read and used to update the states of its outgoing neighbors. In details, OPIC focuses on solving (2.2) for a modified graph $G' = (V \cup z, E \cup J)$, where z is a virtual *zap* node and $J = (V \times z) \cup (z \times V)$ is all possible edges between V and z, plus edge (z, z). This was introduced to make P stochastic and irreducible, allowing (2.2) to admit a unique solution.

The OPIC algorithm works as follows: initially, each node receives some amount of *fluid* (a non-negative number) and a null history record. A scheduler, which can be associated to a web crawler, iterates among the nodes. When a node i is selected, its fluid $F(i)$ is, in order,

- credited to its history: $H(i) = H(i) + F(i)$;
- equally pushed to its neighbors: for all j that are outgoing neighbors of i, $F(j) = F(j) + \frac{F(i)}{\text{out}(i)}$;
- cleared: $F(i) = \frac{F(i)}{\text{out}(i)}$ if i has a loop, $F(i) = 0$ otherwise.

It has been shown that as long as the scheduler is fair (i.e. any given node is selected infinitely often) then the history vector converges, up to normalization, to the desired solution [35]. Algorithm 3 describes how OPIC works. At a given moment, the importance of a node i is proportional to the fluid going through it and is equal to $(F(i) + H(i))/(\mathcal{G} + 1)$ where $\mathcal{G}$ is the total fluid diffused on the entire graph.

The main advantage of OPIC is its flexibility. In particular, it is easy to adapt and incorporate to a continuous, possibly distributed, Web crawler, allowing to get a dynamic, lightweight, PageRank importance estimation. One drawback is that it is not designed to work with (2.3).

Algorithm 3 OPIC algorithm

```
1: for i = 1 : n do
2:     H(i) = 0;                          ▷ Initializing history vector
3:     F(i) = 1/n;                        ▷ Initializing fluid vector
4: end for
5: G = 0;                                 ▷ Accumulated fluid diffused
6: while (true) do
7:     Choose some node i;
8:     H(i)+ = F(i);
9:     for all child node j of i do
10:        F(j)+= F(i)/out(i);
11:    end for
12:    G+= F(i);
13:    F(i) = 0;
14: end while
```

2.3 Collection and diffusion approach

To solve the PageRank equation, most of the methods (e.g., Jacobi, Gauss-Seidel) exploit the matrix-vector multiplication. It means, the PageRank score of a node is calculated by iteratively *collecting (pulling)* the scores from its incoming neighbours. Few others compute PageRank by continuously *diffusing (pushing)* its score to outgoing neighbours (e.g., OPIC).

We take a look of how collection and diffusion approach work. Given a directed three-nodes graph in Figure 2.2, collection methods make use of incoming links of nodes as in Figure 2.3 (or rows of the corresponding transition matrix) whereas diffusion method exploits outgoing links as in Figure 2.4 (or columns of the matrix). If the iteration is based on vector level update (such as Jacobi or Power iteration), the collection and diffusion approaches become equivalent (full cycle operations on all nodes).

Somehow, these two types of operations can be seen as dual operations, but with different consequences.

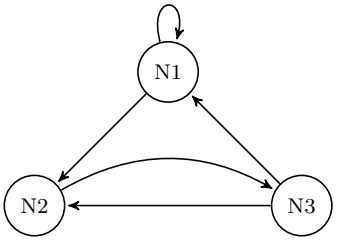
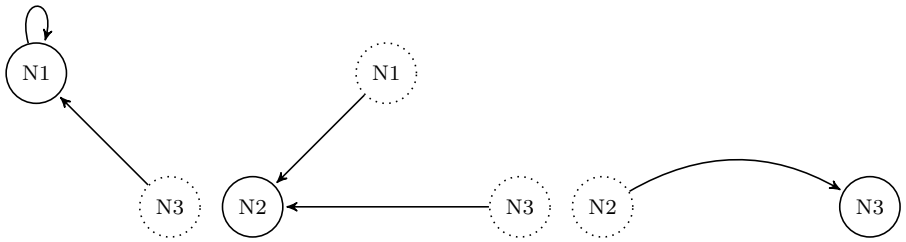
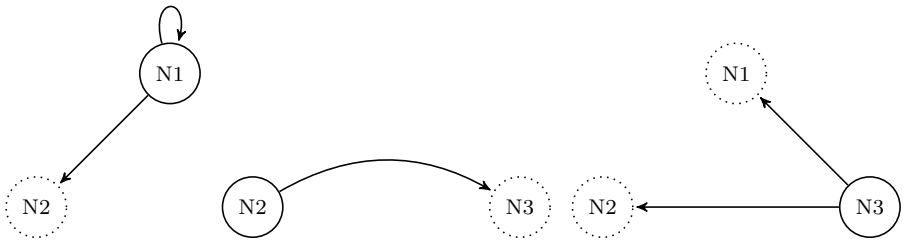


Figure 2.2: A directed graph



(a) Node 1 (b) Node 2 (c) Node 3

Figure 2.3: Collection model: N1 is updated from N1 and N3; N2 from N1 and N3; N3 from N2.



(a) Node 1 (b) Node 2 (c) Node 3

Figure 2.4: Diffusion model: N1 is used to update N1 and N2; N2 to N3; N3 to N1 and N2.

2.4 PageRank of Web graph

In this part of the chapter we describe how web pages are connected and what a web graph looks like under the PageRank point of view. The locality of web graph is well-know [37, 8]. It can be intuitively understood by observing the fact that web pages of the same website often point to each other rather than to those of other

websites. If nodes in a web graph are numbered in a lexicographic order, i.e., pages of the same website are marked by consecutive series of numbers, the web graph will look like Figure 2.5 where each point represents a hyper-link from a *Source* page to a *Destination* page. The dataset `uk-2007` [2], collected by UbiCrawler [38] and compressed using techniques in [39, 40], holds 1M nodes of `.uk` domain. There are two remarks:

- The diagonal with a high density of points confirms the locality characteristic.
- On the diagonal, there are dense squares nested in other less-dense squares. All of them may belong to a main website (domain). The dense squares are sub-websites (sub-domains) and the less-dense would be main website, say *company.com* containing various sub-websites *serviceA.company.com*, *serviceB.compagny.com*, etc.

2.5 PageRank of Call-log graph

To have a broader view of how the PageRank algorithm works, it is worth to test it on different types of graphs. The dataset used in this section comes from phone calls in a prepaid telephon service on April 20, 2012. In its simple version, each line of log represents a call and contains four types of information:

- *caller:* group of user making a call.
- *callee:* group of user receiving a call.
- timestamp of a call when it was made.
- duration of a call (in seconds).

We are only interested in *caller* and *callee* information to visualize the transition graph. In the log file, each caller/callee identification represents a group of user having the same prefix (e.g., company, association, ...) and is anonymized under the form of a string of digits. The entire call log was collected in one single day. Suppose

[2] `http://law.di.unimi.it/webdata/uk-2007-05@1000000/`. This is an extraction of the dataset `uk-2007-05` (containing 106M nodes) using Breadth-First Search (BFS) starting at a random node.

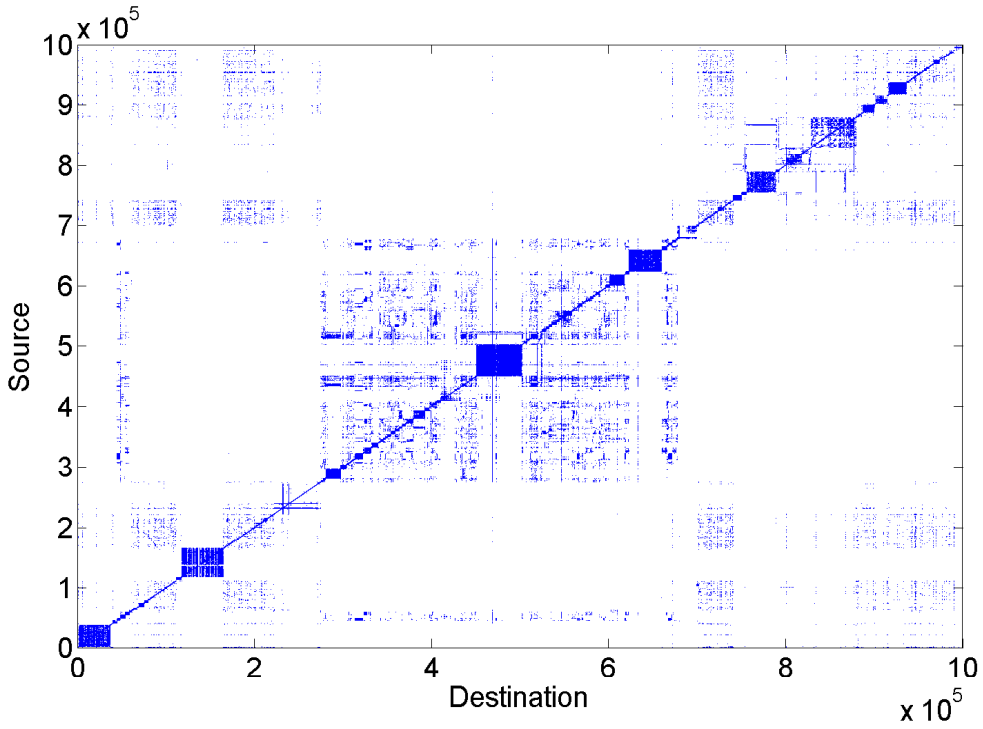


Figure 2.5: Web graph `uk-2007` visualization with sorted node identity

that each node corresponds to a user group in the call log, there is a link between a pair {caller, callee} if the caller has called the callee. Table 2.1 shows some properties of the graph. Note that the *degree* [3] shows the real number of calls a user makes or receives, i.e., the graph is weighted. Figure 2.6 shows what the graph looks like if nodes are sorted by their identification number. We cannot really see any structure like web graph, except the diagonal which represents the self-calls (or internal calls) within each group. This is the motivation of why we want to observe the graph in a PageRank-sorted order.

With the goal of measuring user groups' importance, a corresponding transition matrix P is then constructed, on which we can apply the PageRank algorithm (with damping factor $d = 0.85$).

In the call-log graph, edges are weighted according to how many calls were made between each corresponding pair of caller and callee. Hence, the transition matrix P is filled as follows:

[3]The term *degree* we use in this study (call-log dataset) is weighted, i.e., it takes into account the number of in-coming or out-going calls of a user.

Property	**Value**
Number of nodes (n)	233,308
Number of links	540,399
Maximum indegree	319
Maximum outdegree	244
Average degree	2.32
Dangling nodes	201,346 (86%)
Zero-indegree nodes	6310 (2.7%)
Self-loop nodes	2631 (1.13%)

Table 2.1: Call-log graph statistics

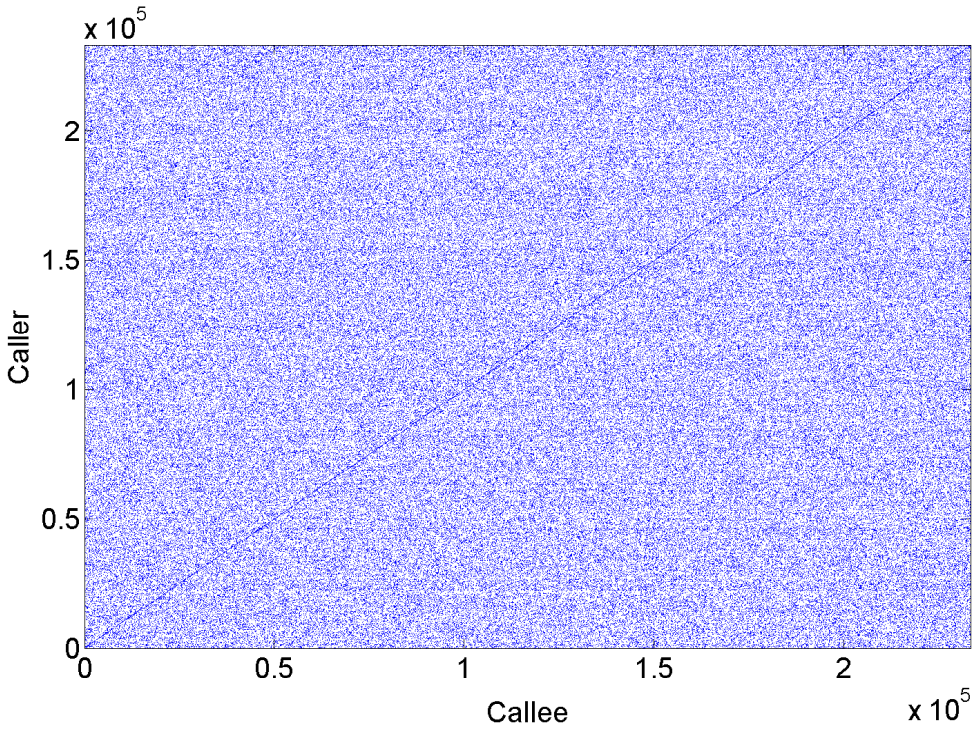


Figure 2.6: Call-log graph visualization with normal order.

$$p_{ij} = \begin{cases} calls(j \to i)/outdeg(j) & \text{if } j \text{ ever makes a call to } i \\ 0 & \text{otherwise.} \end{cases}$$

where $calls(j \to i)$ is the number of calls from j to i and $outdeg(j) = \sum_k calls(j \to k)$ is the total number of calls j has made.

The PageRank vector of the sub-stochastic matrix P is computed. Callers and callees are sorted in descending PageRank order and their connections are visualized in Figure 2.7a. In the plot, nodes are arranged from the left/lower part (higher PageRank) to the right/upper part (lower PageRank). There are some remarks:

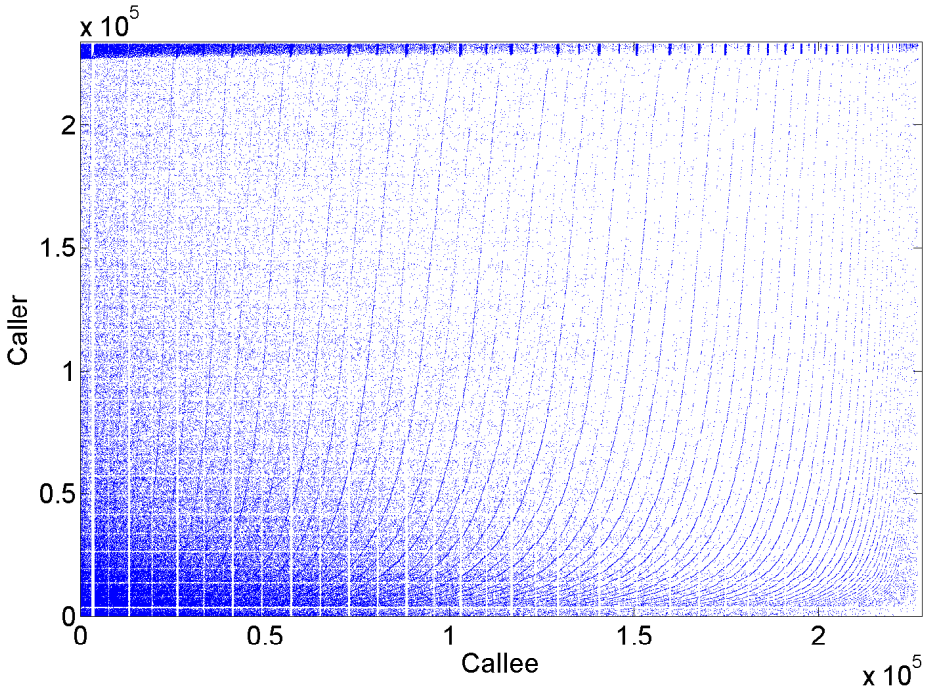
- Each identification stands for a group of users and naturally people of the same group often make internal calls. This fact results in the appearance of the diagonal which indicates self-calls (or internal calls), like observed in Figure 2.6.

- Strong connection among top users due to the fact that important users receive calls from other important users.

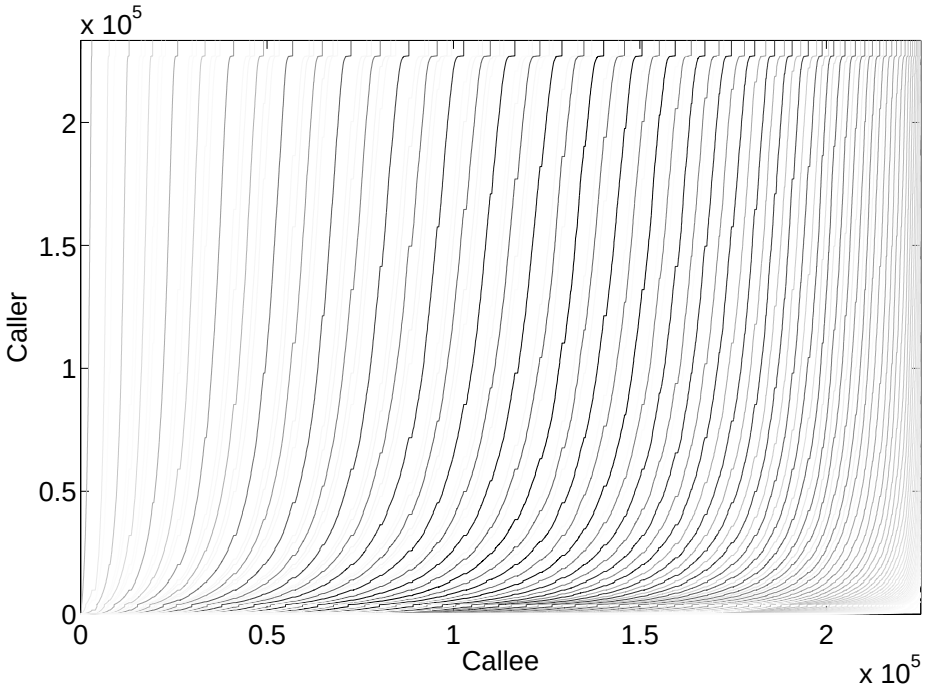
- There exist many blank lines throughout the figure. They represent users called by many other users but make few (or no) calls. Those users users may be call-centers which often receive lots of calls from clients.

There are curly traces appearing across the figure, separated by uniform gap distances. Those lines stem from callees who receive calls from unique callers. Under the diffusion point of view, the importance score of the callee is expressible as a function of that of the caller. The position of the curly lines varies according to the out-degree of the single caller. In detail, given damping factor d, let PageRank score of the caller be $pr(caller)$ and its out-degree be $outdeg(caller)$ (the total number of calls it made), because the callee has only the single caller as parent.

Besides, the curly lines move upwards to the top of the figure and form clouds of points which are distinguished by an almost uniform gap. Those points correspond to unimportant callers who have no incoming call. Their initial importance score is set to $pr(caller) = (1 - d)/n$ before the diffusion process starts. All connections of single callers are reconstructed in Figure 2.7b. In fact, there is a function g that can

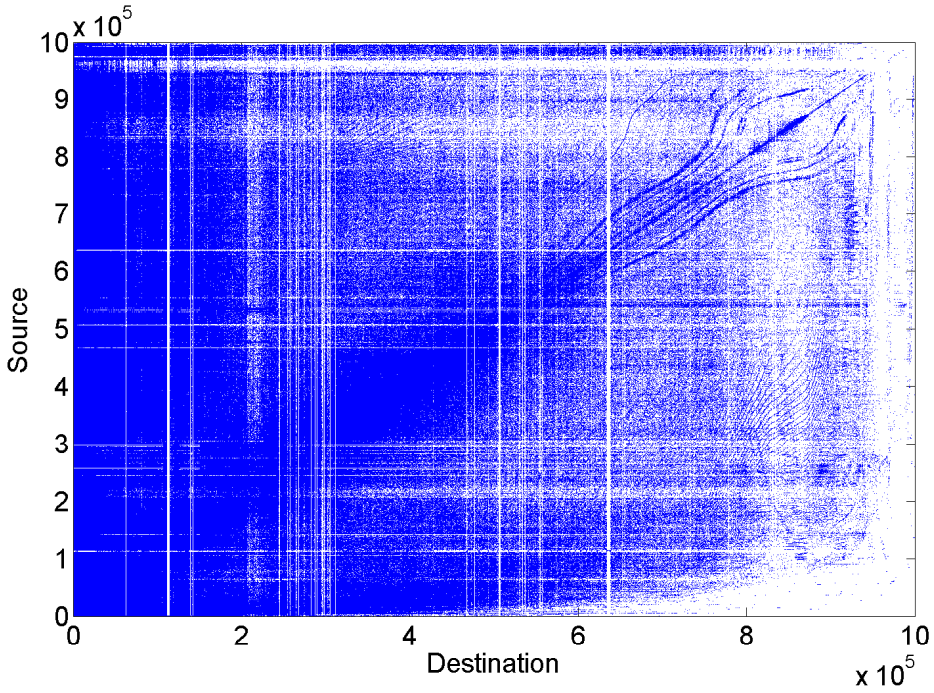


(a) Call-log graph: visualization with descending PageRank order. Nodes are arranged from the left/lower part (higher PageRank) to the right/upper part (lower PageRank).

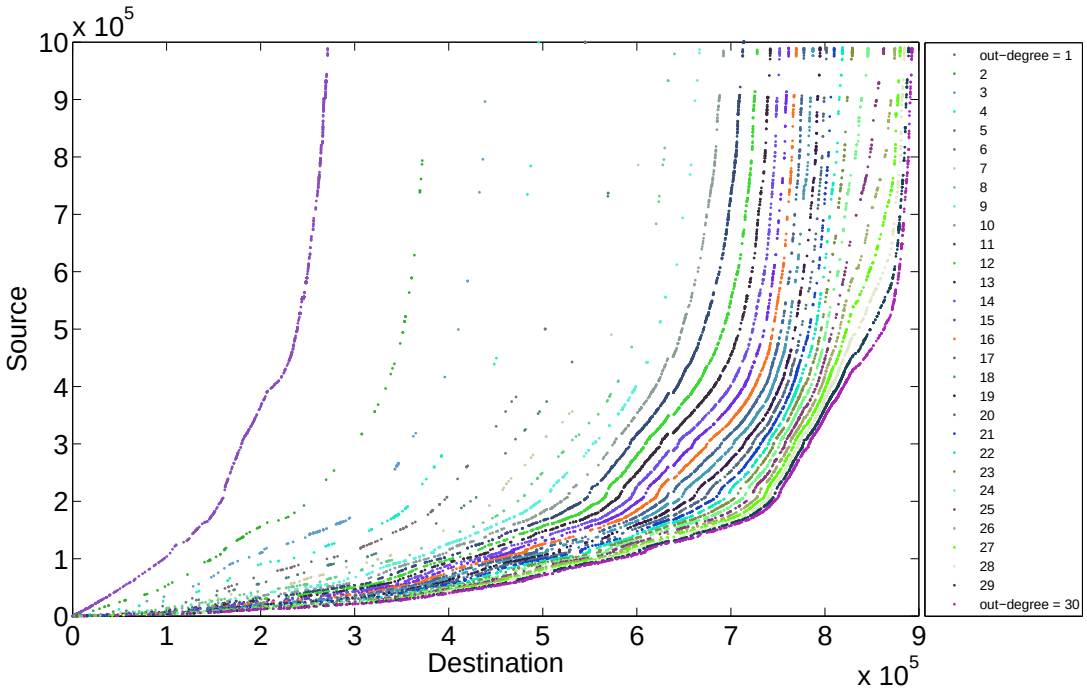


(b) Call-log graph: visualization of nodes (descending PageRank order) having single parent. The color is porportional to density, i.e., less ratio is less visible. Each coordinate (x, y) means callee x only receives calls from a unique caller y but not from others.

Figure 2.7: Call-log graph



(a) Web graph `uk-2007`: Visualization with descending PageRank order. Nodes are arranged from the left/lower part (higher PageRank) to the right/upper part (lower PageRank).



(b) Web graph `uk-2007`: visualization of nodes (descending PageRank order) having single parent whose out-degree increases from left (1) to right (30). Each coordinate (x, y) means page x is pointed to by a unique page y but not from others.

Figure 2.8: Web graph `uk-2007`

transform the rank of a node x, denoted by $R(x)$, to the PageRank $PR(x)$:

$$g : R \in [1, n] \rightarrow PR \in [0, 1]\,. \tag{2.9}$$

and its inverse function g^{-1} exists as well. If a callee j receives k calls from a unique caller i who has made K calls in total ($k \leq K$), the PageRank of user j is

$$PR(j) = d \times \frac{k}{K} PR(i) + \frac{1-d}{n}. \tag{2.10}$$

and his rank is

$$R(j) = g^{-1}\left(d \times \frac{k}{K} g\left(R(i)\right) + \frac{(1-d)}{n}\right). \tag{2.11}$$

As a consequence, the curly lines are computed thanks to a function $f(k, K)$ such that

$$f(k, K) : x \rightarrow g^{-1}\left(d \times \frac{k}{K} g\left(x\right) + \frac{(1-d)}{n}\right). \tag{2.12}$$

Moreover, the blank lines can be explained by observing the uniform gaps. The difference in PageRank score of nodes residing in the middle between the two adjacent vertical (or horizontal) lines depends mainly on the $outdeg(caller)$. For example, between the k^{th} and $(k+1)^{th}$ blank lines (from left to right), the difference is $d \times (1-d)/(nk(k+1))$.

In case of the web graph `uk-2007`, there are also series of obvious curly lines located at the lower-right part of the Figure 2.8a. They characterize all incoming connections of nodes having single parent, i.e., nodes with only one incoming link. We apply the same method to the web graph to extract the curly lines from it. The result is shown in Figure 2.8b. As expected, what we obtain confirms the explanation of the curly lines. They characterize all incoming connections of nodes having a single parent, i.e., web pages pointed to by only one other page and the lines appear from left to right according to the out-degree of the single parent.

2.6 Web graph storage

Web graphs and social network graphs are gigantic. Suppose we need to compute the PageRank vector of a fraction of the World Wide Web graph which contains one billion nodes, and also assume that each PageRank value associated

to a node is 8-bytes *double* type. The PageRank vector itself takes 7.45 GBs and of course the storage space for such a graph is even much larger. The necessity of having an efficient graph storage scheme becomes very important if one wants to deal with enormous data. Fortunately, graphs containing directed relationship between nodes are sparse, i.e., one node often connects to fewer other nodes. Based on this characteristics, many methods were invented. Paolo et al. [41] shows several compression techniques There is always a trade-off between the size of compressed graph and the complexity of compression/decompression process. We show below some basic web graph storage techniques:

- **Square matrix**: the classical web graph storage method uses directly a square matrix of size $n \times n$ where n is the number of pages. Each entry value of a web graph matrix P is defined by $p_{ji} = w_{ij}$ such that w_{ij} is the link weight from node i to j. Storing a web graph under square matrix form requires memory $\mathcal{O}(n^2)$ (n^2 bits if it represents an unweighted graph) if n grows large.

$$\left(\begin{array}{cccc|ccc|ccc}
0 & 1 & 1 & 0 & & & 1 & & & \\
1 & \ddots & \ddots & 1 & & \mathbf{0} & & & \mathbf{0} & \\
1 & \ddots & 1 & 0 & & & & & & \\
\hline
 & & & & 1 & 0 & 1 & & & \\
 & \mathbf{0} & & & 1 & \ddots & 1 & 1 & \mathbf{0} & \\
 & & 1 & & 1 & 1 & 0 & & & \\
\hline
 & & & & & & 1 & 1 & 1 & 1 \\
 & \mathbf{0} & & & & \mathbf{0} & & \ddots & \ddots & 0 \\
 & & & & & & & 1 & 0 & 1
\end{array}\right)$$

- **Adjacency list** (see Table 2.2): this method is often preferable over the square matrix thanks to its simplicity and better memory usage for sparse graph. Each node, numbered according to the lexicographic ordering of the URL, would be attached to a list of neighbours that it points to. We also note that the average out-degree k is much smaller than n. Thus the memory required is $\mathcal{O}(nk)$. Compared to the square matrix, memory space gains around $(n^2 - nk)$ if the same value data type is used. Table 2.2 shows an example.

Node	Outdegree	Neighbors
...	...	...
24	5	22, 23, 24, 26, 28
25	3	22, 23, 25
26	0	
...	...	...

Table 2.2: An adjacency list of neighbors attached to a node.

- **Gap-based neighbors** [41] (see Table 2.3): one can easily observe that the web graph's out-going links are mostly local. Web pages from the same website (or the same domain name) often point to each other. This method is based on a adjacency list and exploits locality characteristic. By keeping only the difference between node identifications, we can keep the neighbor list small. Let $out(x)$ be the list of outgoing neighbours of node x. Then, the list $out(x) = (s_1, s_2, ..., s_k)$ will be replaced by a list of gaps $\texttt{out}(x) = (s_1 - x, s_2 - s_1 - 1, ..., s_k - s_{k-1} - 1)$ or in general:

$$gap = \begin{cases} s_k - x & \text{if } k = 1 \\ s_k - s_{k-1} - 1 & \text{if } k > 1 \end{cases}$$

Note that the list $(s_1, s_2, ..., s_k)$ should be sorted in ascending order (to optimize the gaps) so that $s_k - s_{k-1} \geq 1$. In that case, the gaps are equal to $s_k - s_{k-1} - 1$ starting from $k > 1$ (to minimize the value of each gap, i.e., one unit less), but not at the first gap $s_1 - x$. Otherwise, for the first k. However, negative gap results in the use of, for example, *integer (int)* instead of *unsigned integer (uint)*, thus less efficient if one prefers to work with *uint*. In order to deal with this, a transformation function can be defined as follows:

$$gap = \begin{cases} 2 \times gap & \text{if } gap \geq 0 \\ 2 \times |gap| - 1 & \text{if } gap < 0 \end{cases}$$

Additionally, if the list $(s_1, s_2, ..., s_k)$ is sorted in ascending order, only the first gap $s_1 - x < 0$. Table 2.3 converts the adjacency list of Table 2.2 to a gap-based list. Note that the memory benefit between the adjacency list and the gap-based neighbours appears thanks to the use of a smaller value type which

sufficiently fits the gap, e.g., use 2-bytes *short* type instead of 4-bytes *int* type. For the bigger gaps which do not follow the locality, we can store them apart with bigger value types under the form of an array next to the local small gaps.

Node	Outdegree	Neighbors
...	...	...
24	5	3, 0, 0, 1, 1
25	3	5, 0, 1
26	0	
...	...	...

Table 2.3: Gap-based list exploiting locality of web graph.

- **Reference list** [41] (see Table 2.4): one can further optimize web graph representation by again using locality. There are probably nodes which share almost the same neighbour list. Instead of having its own list, a node can refer to another whose neighbor list looks similar to its, then add remaining different nodes. The reference value of a node is the distance between itself to the referred (and precedent) node or zero if it does not refer to any other. Table 2.4 details the reference list of the adjacency list of Table 2.2. It shows to which node the current node refers and a binary string corresponds to elements copied from the referred node. The method then adds extra nodes which are different from the two neighbouring lists.

This section aims to give readers an overview of web graph compression so that the presented techniques are very basic. For more advanced methods like *reference block* or *interval exploitation*, one may read [41] for a complete reference. Many

Node	Outdegree	Reference	Copy list	Extra node
...	...	...	...	...
24	5	0		22, 23, 24, 26, 28
25	3	1	11000	25
26	0			
...	...	...	...	...

Table 2.4: Reference list exploiting locality of web graph.

graphs [4] were compressed using these techniques and also most of the datasets that we use in this thesis. In general, the complete compression model can be used to compress different types of graphs, such as web graphs or social network graphs. Of course, the compression ratio depends on the nature of the graph and inversely proportional to the compression time. For example, the web graph `uk-2005` [5] (40 millions nodes, 936 millions links) can achieve a ratio 1.463 bits/link whereas the social network graph `twitter-2010` [6] (41 millions nodes, 1.5 billion links) needs 13.897 bits/link.

2.7 Conclusion

In this chapter, we gave the introduction of the PageRank algorithm and how it is formulated mathematically. Several techniques to find the PageRank vector were introduced. We then visualized the transition matrix under the PageRank point of view which unveils some specific structures of the web graph. We looked at what could be observed by computing PageRank on another type of graph (call-logs). Some strange graph structures were explained using the definition of the diffusion approach. Efficient compression methods were also exposed as a solution to store enormous web graph. The principle goal of this chapter was to give readers a panoramic and preliminary view of PageRank, which will facilitate the reading of the later chapters.

[4] http://law.di.unimi.it/datasets.php
[5] http://law.di.unimi.it/webdata/uk-2005/
[6] http://law.di.unimi.it/webdata/twitter-2010/

Chapter 3

D-Iteration: diffusion approach to solve PageRank

Several techniques to solve the PageRank equation were introduced in Chapter 2; Power Iteration (PI), Gauss-Seidel (GS) and Online Page Importance Computation (OPIC). In this chapter we present a new method that can accelerate the computation of the PageRank importance vector. The method, called D-Iteration (DI) and firstly introduced by Dohy Hong [19], is based on the decomposition of the matrix-vector product that can be seen as a fluid diffusion model and is potentially adapted to asynchronous implementation. We give theoretical results about the convergence of the algorithm and we show through experimentations on a real Web graph that DI can improve the computation efficiency compared to other classical algorithms like PI, GS or OPIC. D-Iteration aims at solving the PageRank equation with an efficiency similar to Gauss-Seidel while keeping the scheduling flexibility offered by OPIC. This results in a fluid diffusion approach similar to OPIC with some damping added to the mix.

The main contribution of this chapter is doing a survey on D-Iteration, based on [19]. We give the definition of DI and explain how the algorithm can compute the PageRank vector. We also point out the advantages of DI such as convergence speed thanks to implicit completion due to *dangling nodes* [1], freedom in choosing nodes to iterate and adaptation to the graph evolutions. The results of this chapter are put in [20].

[1] Nodes without out-going links, or zero-outdegree nodes.

This chapter is organized as follows. Section 3.1 presents the definition the D-Iteration, followed by Section 3.2 explaining the convergence of the algorithm. Section 3.3 shows how to complete the PageRank vector with the presence of dangling nodes. Section 3.4 talks about DI schedulers. Section 3.5 demonstrates the adaptability of DI to the graph evolutions. Finally, Section 3.6 concludes the chapter.

3.1 Definition

In the following, we assume that a deterministic or a random diffusion sequence $\mathcal{I} = \{i_1, i_2, ..., i_k, ...\}$ with $i_k \in \{1, .., n\}$ is given. $\mathcal{I}$ is not obliged to be fixed in advance but can be adjusted on the fly as long as the *fairness* [2] property stands. Note that the fairness assumption is sufficient but not necessary and will be discussed in detail later.

Like OPIC, we have to deal with two variable vectors at the same time: a *fluid vector* F, initially equal to $(1-d)Z$ and a *history vector* H, initially null for all nodes. When a node is selected, its current fluid value is added to its history, then a *fraction* d of its fluid is equally pushed to its neighbors and its fluid value is cleared.

Formally, the fluid vector F associated to the scheduler $\mathcal{I}$ is iteratively updated using:

$$F_0 = (1-d)Z, \tag{3.1}$$

$$F_k = F_{k-1} + dF_{k-1}(i_k)Pe_{i_k} - F_{k-1}(i_k)e_{i_k}, \tag{3.2}$$

where e_{i_k} is the standard basis vector corresponding to i_k.

The second term in (3.2) represents the damped diffusion and the third term clears the local fluid (up to loops). Similarly to OPIC, an iteration reads one value, $F_{k-1}(i_k)$ and updates i_k and its outgoing neighbors. Note that F is always non-negative.

We also formally define the history vector H:

[2] The scheduler $\mathcal{I}$ is fair if each value of i_k appears in $\mathcal{I}$ infinitely often.

$$H_0 = \vec{0}, \tag{3.3}$$

$$H_k = H_{k-1} + F_{k-1}(i_k)e_{i_k}. \tag{3.4}$$

By construction, H_k is non-decreasing with k.

3.2 Convergence

The following Theorem states the convergence of the D-Iteration algorithm:

Theorem 1. *For any fair sequence $\mathcal{I}$ (all nodes are chosen infinitely often) and any positive damping factor $d < 1$, the history vector H_k converges to the unique vector x such that $x = dPx + (1-d)Z$:*

$$\lim_{k\to\infty} H_k = (1-d)(I-dP)^{-1}Z.$$

Moreover, we have

$$|x - H_k| \leq \frac{|F_k|}{1-d}, \text{ where } |\cdot| \text{ is the } L_1 \text{ norm.} \tag{3.5}$$

Proof. We first prove the equality:

$$H_k + F_k = F_0 + dPH_k. \tag{3.6}$$

This is straightforward by induction: (3.6) is true for $k = 0$; assuming it is true for $k-1$, we have

$$\begin{aligned} H_k + F_k &= H_{k-1} + F_{k-1}(i_k)e_{i_k} + F_{k-1} + dF_{k-1}(i_k)Pe_{i_k} - F_{k-1}(i_k)e_{i_k} \\ &= F_0 + dP(H_{k-1} + F_{k-1}(i_k)e_{i_k}) = F_0 + dPH_k. \end{aligned}$$

From the equation 3.6, we have:

$$\begin{aligned} H_k &= (I-dP)^{-1}(F_0 - F_k) \\ &= x - \textstyle\sum_{i=0}^{\infty} d^i P^i F_k \end{aligned}$$

Noticing that P is substochastic, we get

$$|x - H_k| = |\sum_{i=0}^{\infty} d^i P^i F_k| \leq \sum_{i=0}^{\infty} d^i |F_k| = \frac{|F_k|}{1-d}.$$

All we need is to show that $|F_k|$ tends to 0. Notice that the total available fluid is non-increasing. That being said, the fluid of a given node is non-decreasing until it is scheduled, and when it is, a quantity $(1-d)$ of it is "lost" due to the damping (or more if it is a dangling node). Given these two observations, let us consider a time k and another time $k' > k$ such that all nodes have been scheduled at least once between k and k' (this is always feasible thanks to the fairness assumption). For each node i, its fluid $F_t(i)$ at the time t of its first scheduling after k is greater than its fluid $F_k(i)$ at time k and $|F_{t+1}| \leq |F_t| - (1-d)F_k(i)$, so we have $|F_{k'}| \leq |F_k| - (1-d)F_k = d|F_k|$. This concludes the proof. □

3.3 Implicit completion

Equation (3.5) gives a first stopping condition of the algorithm. If one aims at $|x - H_n| < \epsilon$, the condition becomes $|F_n| \leq (1-d)\epsilon$ which stands for a non-normalized version. The Equation (3.5) further becomes an equality if P is stochastic (for example thanks to dangling node completion), in which case we have $|P^i F_k| = |F_k|$.

It is more efficient to perform the computation on a non-completed matrix (every time a dangling node is selected, all non-null entries of Z are updated if P is completed). The question is: can we control the convergence to the solution of the completed matrix while running the algorithm on the original one?

To address this problem precisely, we count the total amount of fluid that has left the system when a diffusion was applied on a dangling node. We call this quantity l_k (up to step k of the DI). To compensate this loss and emulate completion, a quantity $dl_k Z$ should have been added to the initial fluid, leading to $(1-d+dl_k)Z$ instead of $(1-d)Z$. But then the fluid $dl_k Z$ would have produced after k steps a leak $(dl_k^2/(1-d))Z$ on dangling nodes, which needs to be compensated...

In the end, the correction that is required to compensate the effect of dangling nodes on the residual fluid $|F_k|$ consists in replacing the initial condition $|F_0| = (1-d)$ by $|F_0'|$ such that:

$$\begin{aligned} |F_0'| &= (1-d) + dl_k + dl_k\frac{dl_k}{1-d} + dl_k\left(\frac{dl_k}{1-d}\right)^2 + ... \\ &= (1-d) + dl_k\sum_{n=0}^{\infty}\left(\frac{dl_k}{1-d}\right)^n \\ &= \frac{(1-d)^2}{1-d-dl_k}. \end{aligned}$$

As $|F_0'|/|F_0| = (1-d)/(1-d-dl_k)$, H_k needs to be renormalized (multiplication) by $(1-d)/(1-d-dl_k)$ so that the exact L_1 distance between x and the normalized H is equal to:

$$|x - \frac{1-d}{1-d-dl_k}H_k| = \frac{|F_k|}{1-d-dl_k}.$$

To summarize, we can run the algorithm on the original matrix using $\frac{|F_k|}{1-d-dl_k}$ as a stopping condition that guarantees the precision of the normalized result.

3.4 Schedulers

The actual performance of DI is directly related to the scheduler used. A simple scheduler, which we call `DI-cyc`, is a Round-Robin (RR) one, where a given permutation of nodes is repeated as long as necessary.

Theorem 2. *For any Round-Robin scheduler $\mathcal{I}$, we have:*

$$|x - H_k| \leq d^{\lfloor \frac{k}{n} \rfloor}. \tag{3.7}$$

The proof is a direct application of the proof of Theorem 1 considering successive sequences of n steps.

Theorem 2 ensures that D-Iteration performs at least as well as the PI method: in both cases, after a round where all nodes have been processed once, the error is reduced by at least d.

While the bound can be tight for specific situations (for instance a clockwise scheduler applied to a counterclockwise-oriented cycle), it is conservative in the sense that it ignores that some of the fluid can be pushed multiple times during a sequence of n steps. For that reason, and keeping in mind that D-Iteration is a *push* version of

the Gauss-Seidel method, we expect that `DI-cyc` will perform more like Gauss-Seidel in practice.

However, one strength of D-Iteration is the freedom in the choice of a scheduler. As the convergence is controlled by the remaining fluid $|F_k|$, a good scheduler is one that makes the fluid vanish as quickly as possible. This disappearance is caused by two main parameters: the damping factor d and dangling nodes (cf above). Noting that at time k a quantity $(1-d)F_{k-1}(i_k)$ vanishes through damping, a natural greedy strategy would consist in selecting at each step $i_k = \text{argmax}_{i=1,\dots,n} F_{k-1}(i)$.

The main drawback of such a strategy is the expensive searching cost. To address that issue, Dohy Hong introduced in [19] a simple heuristic, called`DI-argmax`, which works as follows: we use a RR scheduler, but at each iteration, we run through the scheduler until we find a node that possesses a fluid greater or equal to the average fluid in the whole graph. The advantage of this method is that nodes with relatively low fluids will be skipped, avoiding unprofitable update operations, with a searching cost lower than picking up the best node at each iteration.

Theorem 3. *Using `DI-argmax`, we have:*

$$|x - H_k| \leq \left(1 - \frac{1-d}{n}\right)^k \approx e^{-(1-d)\frac{k}{n}}. \tag{3.8}$$

The proof is immediate, as by construction we have $|F_k| \leq (1 - \frac{1-d}{n})|F_{k-1}|$.

Note that the Theorem proves the convergence of `DI-argmax`, which is not a fair scheduler: for instance, after some time, it will ignore the transient nodes of the graph, which eventually have no fluid left. We conjecture that (3.8) is not tight (tightness would require to always choose an average node) and that the actual convergence should be faster.

3.5 Update equation

The existing iterative methods (such as Gauss-Seidel iteration, Power Iteration, ...) can naturally adapt the iteration when G (and thus P) is changed because they are generally independent of the initial condition (for any starting vector, the iterative scheme converges to the unique limit). The simplest way to adjust is to

compute the PageRank of the new graph using the previous computation as starting vector.

This technique cannot be used in the case of DI so we need to provide an adapted result.

Theorem 4. *Assume that after k_0 diffusions, the DI algorithm has computed the values (H_{k_0}, F_{k_0}) for some matrix P, and consider a new matrix P' (typically an update of P). One can compute the unique solution of the equation $x' = dP'x' + (1-d)Z$ by running a new D-Iteration with starting parameters*

$$F'_0 = F_{k_0} + d(P' - P)H_{k_0}, \tag{3.9}$$

$$H'_0 = H_{k_0}. \tag{3.10}$$

A few remarks on Theorem 4:

- It implies that one can continue the diffusion process when P is regularly updated: we just need to inject in the system a fluid quantity equal to $d(P' - P)H_{k_0}$ and then change to the new matrix P', keeping the same history.
- The precision of the result directly relates to the quantity of fluid left. Here the precision induced by $F_{k_0} + d(P' - P)H_{k_0}$ seems rather minimal, as the original fluid is only altered by the difference between the two matrices. In particular, if the difference $P - P'$ is small, the update should be quickly absorbed.
- For the sake of clarity, we assumed that the set of nodes is the same between P and P', but the result can be extended to cope with variations of V.

Proof. Call H'_∞ the asymptotic result of the new D-Iteration. We first use (3.6) on the reduced history $H'_k - H_{k_0}$ (Equation (3.6) requires that the history is initially empty):

$$(H'_k - H_{k_0}) + F'_k = F'_0 + dP'(H'_k - H_{k_0}).$$

Letting k go to ∞ leads to

$$\begin{aligned}(H'_\infty - H_{k_0}) &= F'_0 + dP'(H'_\infty - H_{k_0}) \\ &= F_{k_0} + d(P' - P)H_{k_0} + dP'(H'_\infty - H_{k_0}) \\ &= dP'H'_\infty + F_{k_0} - dPH_{k_0},\end{aligned}$$

which can be written

$$H'_\infty = dP'H'_\infty + H_{k_0} + F_{k_0} - dPH_{k_0}.$$

Equation (3.6) ($H_{k_0} + F_{k_0} = F_0 + dPH_{k_0}$.) concludes the proof.

□

3.6 Conclusion

In this chapter, we summarized the theoretical results of D-Iteration, an algorithm based on a diffusion approach, to solve the PageRank equation. These results include properties concerning the correctness (convergence), the precision measurement and update equations. This chapter is the basis for chapter 4 and chapter 5 where we will show the potential of DI through experiments on real data in comparison with other classical pull (Power Iteration, Gauss-Seidel, etc) and push (OPIC) methods.

Chapter 4

Preliminary performance evaluations on a small graph

To better understand the convergence of the algorithms exposed in Section 2.2, we test them on a real web graph of 1 million nodes. The evaluation will be based on two main criteria: iterations and elementary operation cost. We will notice that some algorithms perform well on one criterion but not on the other. In contrast, there is a D-Iteration variant that converges quickly with respect to a certain measurement.

On the other hand, the computation of PageRank vector turns out to be even more challenging if the graph is not fully visible. Therefore, this chapter will also investigate how to approximate the PageRank vector of a matrix of which all columns (or rows) are not given. In other words, the transition matrix is now partially hidden. This study makes sense in a distributed environment where the graph is not stored on the computation machine but on another. That means, the graph can only be gradually unveiled by exchanging requests between machines.

In this chapter, the main contribution is studying the performances of Jacobi, Gauss-Seidel, SOR, GMRES and OPIC, then compare them with DI. We explain where the difference comes from and in which case an algorithm can be a good choice. Besides, we propose a method to approximate the PageRank vector of a graph partially hidden. Our method consists in combining a random strategy and a maximal one that experimentally achieves a gain factor of ten compared to using only one of them. The results are presented in [21].

The chapter is organized as follows. Section 4.1 presents the data set we use and the choice of PageRank parameters. Section 4.2 introduces the two measurement criteria. Section 4.3 then shows and explains experiment results. Section 4.4 presents the method to approximate the PageRank vector of a graph partially hidden. Section 4.5 summarizes the chapter.

4.1 Dataset and settings

We use the same web graph as in Chapter 2: a small graph available on [42] named `uk-2007-05@1000000` [1] in DELIS project [2] [43]. This is an extraction of the dataset `uk-2007-05` (containing 106M nodes) using Breadth-First Search (BFS) starting at a random node. Each node is a web page of *.uk* domain and edges are hyperlinks between them.

We recall the transition matrix P (see Section 2.1) in the PageRank equation $x = dPx + (1-d)x$ as follows:

$$P_{i,j} = \begin{cases} 1/outdeg(j) & \text{if hyperlink } j \to i \text{ exists,} \\ 0 & \text{otherwise.} \end{cases}$$

where $P_{i,j}$ is the entry at i^{th} row and j^{th} column of matrix P; $outdeg(j)$ is out-degree of node j. All the algorithms are tuned to solve the equation (2.3) using $d = 0.85$ and $Z \equiv \frac{1}{n}$. Remind that DI does not need to pad null columns of P with $\frac{1}{n}$ thanks to the implicit completion (see Section 3.3). We use the GMRES function in Matlab library [3] to solve the PageRank equation.

In the case of OPIC, remind that the original version does not take into account damping factor. In order be consistent with other algorithms, we emulate (2.3) by running OPIC on the stochastic matrix P' defined by:

$$P'_{i,j} = \begin{cases} \frac{d}{outdeg(j)} + \frac{1-d}{n} & \text{if hyperlink } j \to i \text{ exists,} \\ \frac{1}{n} & \text{if } j \text{ is a dangling node,} \\ \frac{1-d}{n} & \text{otherwise.} \end{cases} \tag{4.1}$$

[1] `http://law.di.unimi.it/webdata/uk-2007-05@1000000/`
[2] `http://delis.upb.de/`
[3] `http://fr.mathworks.com/help/matlab/ref/gmres.html`

Note that even if we do not write P' as an explicit full matrix, this emulation makes each diffusion rather costly as all entries need to be updated at each elementary step. It is only introduced to allow comparison with other methods, assuming all diffusions have the same cost, and should not be used in practice.

For DI, we used the two exposed variants, `DI-cyc` and `DI-argmax`. The same schedulers were used for OPIC, called `OPIC-cyc` and `OPIC-argmax`. Remember that the fluid amount F is constant in OPIC, so the threshold that triggers diffusion in `OPIC-argmax` is constant (it is the average fluid).

The y-axis in all figures shows either L1-norm or L2-norm [4] of the distance indicating how far the current vector x_k is to the PageRank vector x_∞. The ground truth vector x_∞ is pre-computed using DI with a precision 10^{-9}, i.e., until the remaining fluid (residual) $\|F_k\|_1 < 10^{-9}$. Remind that with DI we can explicitly know the distance to the PageRank vector thanks to $\|F_k\|_1$ (see Section 3.2) while we cannot with other algorithms (only an upper bound for the error is provided).

4.2 Comparison criteria

The performance of the algorithms is evaluated with respect to the two following criteria:

- **Iterations**: conventionally, performance of iterative methods is measured by the number of iterations required to reach certain precision. One iteration of, for instance, Jacobi method, GS or SOR corresponds to one matrix-vector multiplication.

- **Elementary operations**: among different reasons that can cause bottlenecks like slow memory access or synchronization delay in distributed computation, we consider here the number of elementary operations that the algorithms need to perform the calculation tasks. We define the cost of one *addition* (T_a) and *multiplication* (T_m). To numerically estimate values of T_a and T_m, we made a small program running several times 10^{10} each operation, take the average of its `cpu_time` (CPU clock in a second) and `real_time` (real running time), then

[4] L1-norm and L2-norm of a vector u are defined as $\|u\|_1 = \sum_{i=1}^{n} |u_i|$ and $\|u\|_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$ respectively.

normalize it. Not surprisingly we obtain the same value as [44] that $T_a = T_m$, and we consider this value as one unit cost.

4.3 Experiments

4.3.1 Jacobi, GS, SOR and GMRES

These four algorithms were described in Section 2.2, we recall briefly how they can be used to compute the PageRank vector x iteratively:

Jacobi / Power Iteration:

$$x_{k+1}(i) = d \sum_j P_{i,j} x_k(j) + (1-d)Z(i).$$

Gauss-Seidel (GS):

$$x_{k+1}(i) = d \left(\sum_{j<i} P_{i,j} x_{k+1}(j) + \sum_{j \geq i} P_{i,j} x_k(j) \right) + (1-d)Z(i).$$

Successive Over-Relaxation (SOR):

$$x_{k+1}(i) = d \left[\omega \left(\sum_{j<i} P_{i,j} x_{k+1}(j) + \sum_{j \geq i} P_{i,j} x_k(j) \right) + (1-\omega) x_k(i) \right] + (1-d)Z(i).$$

Generalized Minimal Residual (GMRES): exploits the Krylov subspace to find the PageRank vector thanks to an orthogonalization of all vectors in the subspace.

We begin with the comparison of the two classical iterative methods, Jacobi and GS. Let us consider a graph of size $n = 10^4$ extracted from the first 10^4 nodes of `uk-2007-05@1000000`. Figure 4.1 shows that Jacobi takes 72 iterations to reach the residual 10^{-6} while GS only requires 39. The main difference between the two methods is that with Jacobi, vector x_{k+1} (vector x at $(k+1)^{th}$ iteration) is calculated by using only elements of vector x_k whereas GS exploits right away all elements $x_{k+1}(j)$ (element j of the vector x at $(k+1)^{th}$ iteration) to compute $x_{k+1}(i)$ for $j < i$. Thanks to that, GS has a better performance than Jacobi, not only w.r.t. convergence speed but also memory usage (at $(k+1)^{th}$ iteration GS keeps only vector x_{k+1} whereas Jacobi needs both x_{k+1} and x_k). However, the fact that GS updates "in real time" its working PageRank vector could be perceived as a downside in asynchronous computation because it loses the freedom of choosing nodes to iterate.

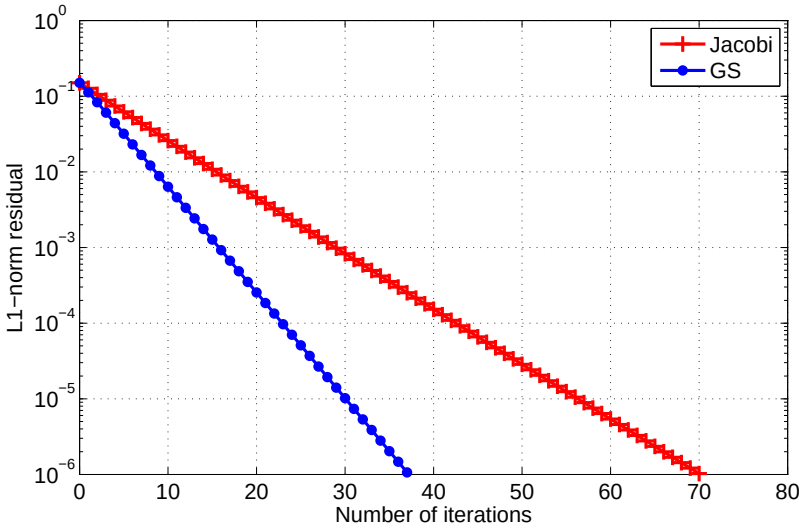


Figure 4.1: $n = 10^4$ nodes, GS and Jacobi

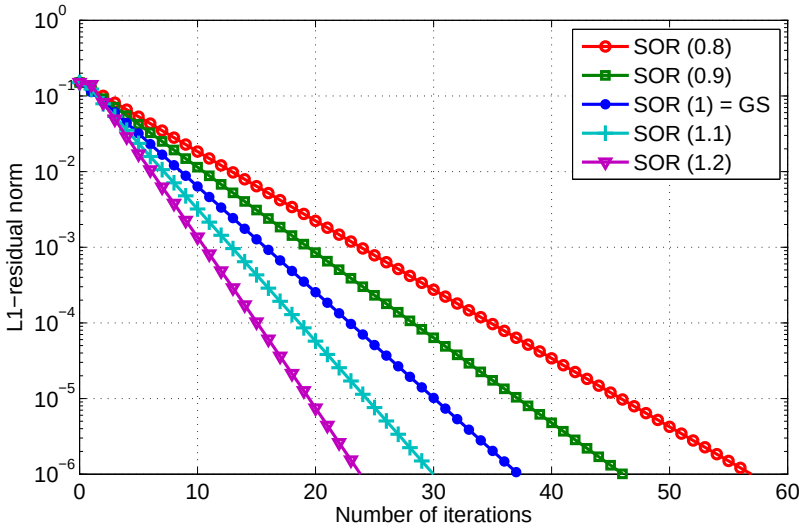
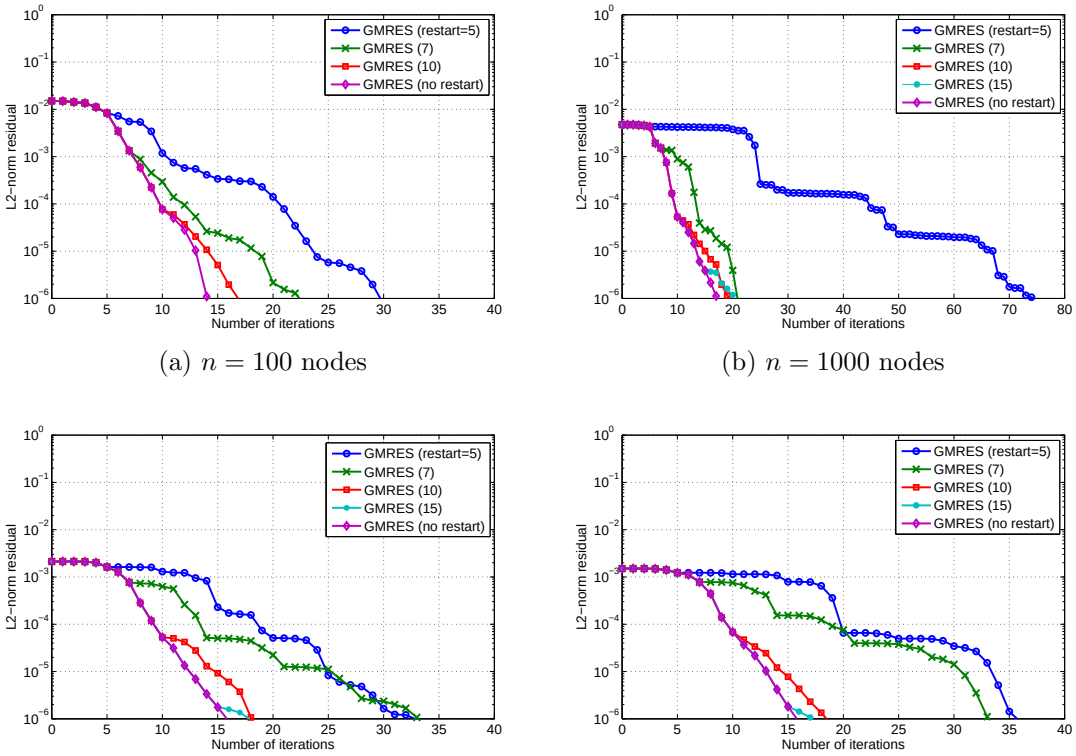


Figure 4.2: $n = 10^4$ nodes, SOR algorithm varying ω from 0.8 to 1.2



(a) $n = 100$ nodes

(b) $n = 1000$ nodes

(c) $n = 5000$ nodes

(d) $n = 10000$ nodes

Figure 4.3: GMRES with different *restart* values comparison.

Next we analyze the SOR algorithm. SOR is a variant of GS, characterized by an additional relaxation parameter ω. In detail, SOR improves the convergence rate by adding ($\omega < 1$) or removing ($\omega > 1$) a fraction of the vector x_k. According to [45], SOR diverges if $\omega \leq 0$ or $\omega \geq 2$. It has also been proven converged if the matrix A is *symmetric positive definite* [5] and $0 < \omega < 2$. Adjusting the value of ω can be used to speed up the convergence. For example, if we set $1 < \omega < 2$, the part of $(1 - \omega)x_k$ is negative and the GS-like part receives extra weight which could result in a faster convergence as shown in Figure 4.2. SOR becomes GS if $\omega = 1$. In our experiment, SOR gets worse when $\omega > 1.2$.

The number of elementary operations required by Jacobi, GS and SOR is linearly dependent on the number of iterations so that we do not plot a figure for this criterion.

[5] A symmetric $n \times n$ real matrix M is said to be positive definite if if $z^T M z > 0$ for all for all nonzero complex vectors z

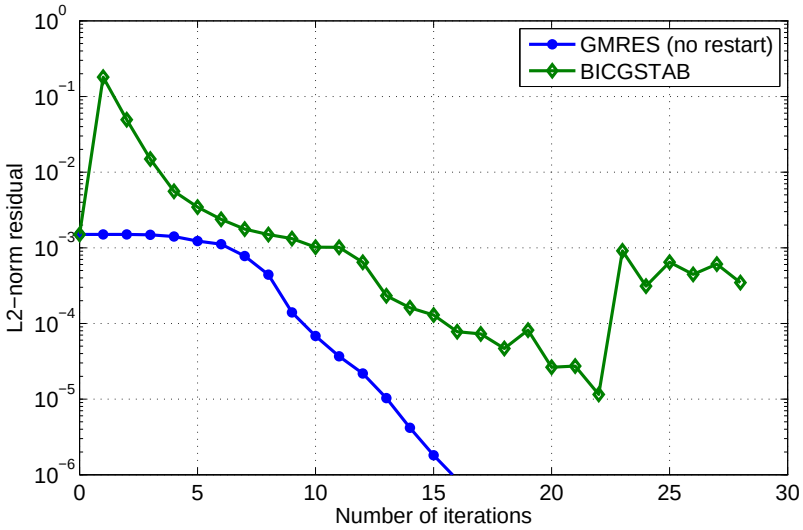


Figure 4.4: $n = 10^4$ nodes, GMRES and BICGSTAB

Regarding GMRES, given an unlimited memory, one would run the algorithm without *restart* value, i.e., $restart = \infty$ called `GMRES (no restart)`, the algorithm would converge more quickly at the expense of a (much) larger memory due to an expansion of the Krylov subspace. In other words given a *restart* value the subspace contains $\{r_0, Ar_0, ..., A^{restart-1}r_0\}$.

Figure 4.3 shows the results while varying matrix size from $n = 10^2$ to $n = 10^4$. Each GMRES curve applying a *restart* value, i.e., $0 < restart < \infty$, is stemmed from the root curve ($restart = \infty$) at its corresponding *restart* value, and converges from then on more slowly than the root curve. In Figure 4.3b, GMRES with $restart = 5$ requires up to 75 iterations compared to 17 iteration of that with $restart = \infty$ to both converge at 10^{-6}, and a bad initial guess vector after each turn of *restart* iterations may cause this slowness. Besides the basic cost (matrix-vector multiplication cost), GMRES introduces some extra costs coming from:

- Arnoldi process to orthogonalize current vector with all previous vectors which are stored in Hessenberg matrix at each iteration.
- Residual minimization based on projection method after each turn of *restart* iterations.

Thus, we will see that this method is not performing quite well in terms of the number of elementary operations when *restart* value is large.

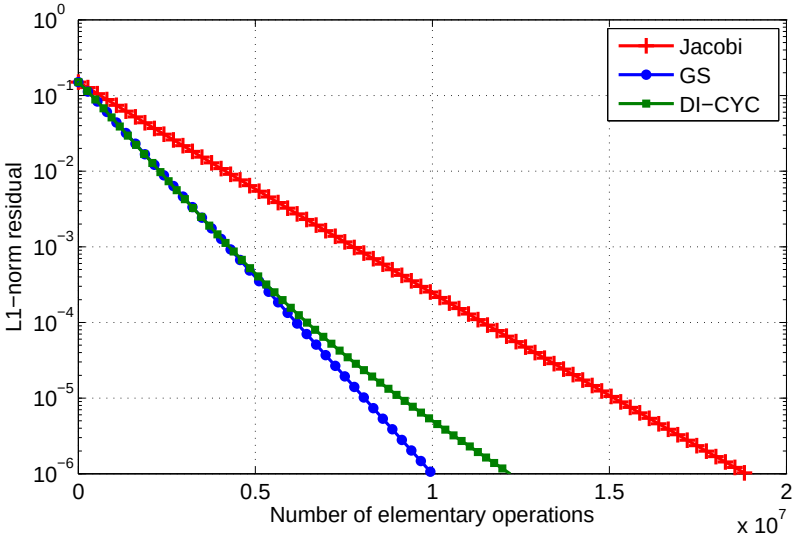


Figure 4.5: $n = 10^4$ nodes; GS, Jacobi and DI comparison

In addition to GMRES, we also study Biconjugate Gradient Stabilized [46] (BICGSTAB) which also exploits Krylov subspace to produce the result vector x. However, in case of PageRank matrix, BICGSTAB diverges as shown in Figure 4.4.

4.3.2 D-Iteration

Classical iterative methods, like Jacobi or GS, perform matrix row operations. Each node gathers "score" from incoming links (row-based collection). With DI, in contrast, each node distributes its "score" (or "fluid") to its neighbours following the outgoing links (column-based diffusion). DI is well-adapted to asynchronous computation thanks to the freedom of diffusion sequence. Figure 4.5 shows a quick comparison between Jacobi, GS and `DI-cyc`. We can see that `DI-cyc` behaves almost similarly to GS, but starts to converge more slowly at the middle of the curve. We will prove later in Appendix B that the convergences of GS and `DI-cyc` are exactly the same if diagonal (or self-loop link) elimination is applied. The graph here contains those links so that after the first tens of iterations converging almost at the same rate, the `DI-cyc` requires more time to diffuse fluid trapped inside the loops, and it leads to a slower convergence.

We show the convergence of DI and OPIC in Figure 4.6. As one can observe, given the same number of iterations, `OPIC-argmax` converges more quickly than `OPIC-cyc` which always stays monotone. An iteration of `DI-argmax` is one

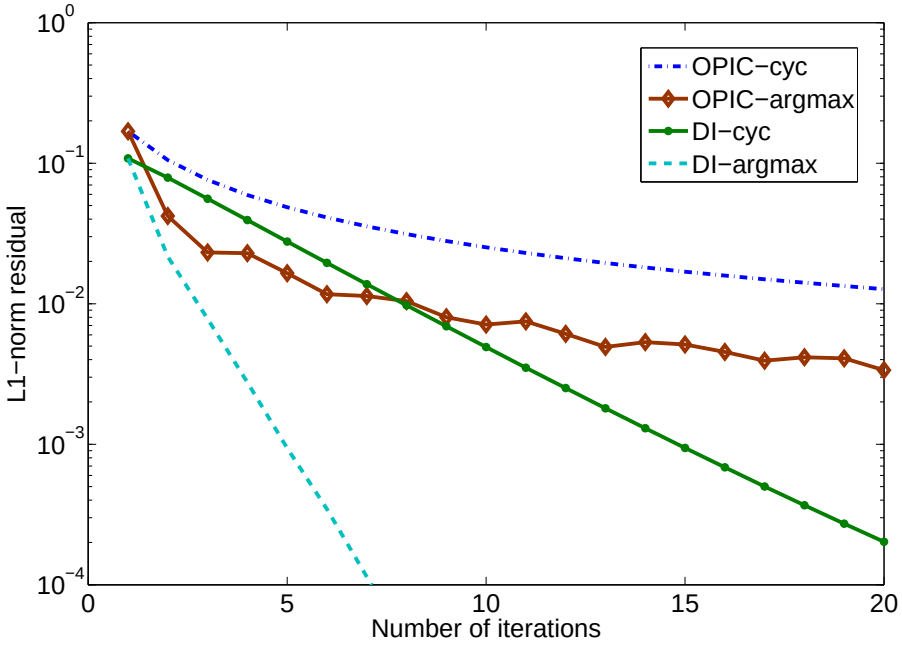
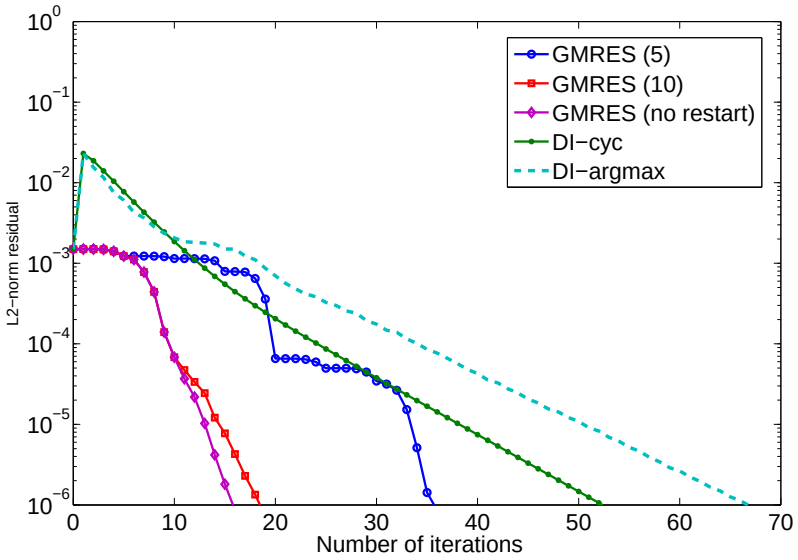


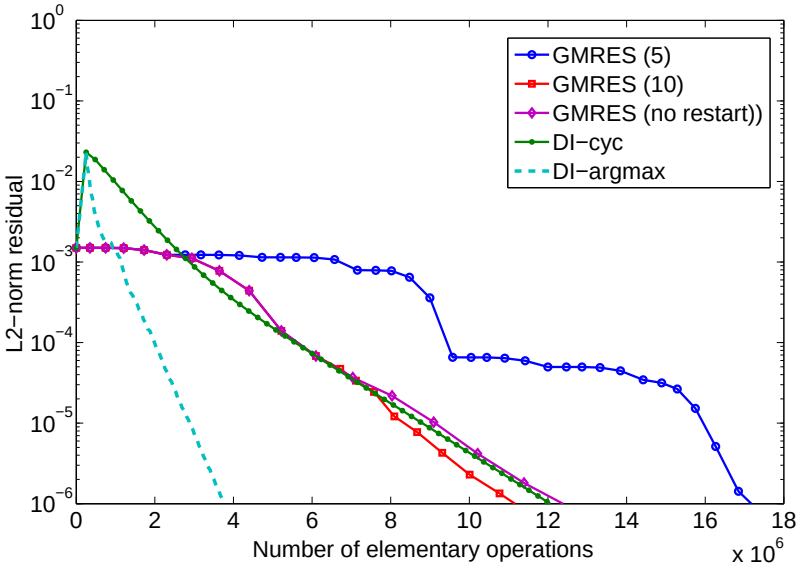
Figure 4.6: $n = 10^6$ nodes, DI and OPIC

pass through all nodes in the graph regardless of either diffusing fluid at nodes or not, because it takes time to check whether a node satisfies the diffusion condition. The `DI-argmax` is better than `DI-cyc` and OPIC as it reaches the precision 10^{-3} after several rounds. This can be intuitively understood because only nodes having sufficiently large fluid are diffused. It then results in a less effort wasted on minor nodes and thus a performance boost. OPIC is suitable for a less-precise computation since it converges slowly over time. Its main advantage compared to DI is the automatic adaptation to the graph evolutions while DI needs to be tuned as mentioned in Section 3.5.

To be comparable with GMRES (the GMRES function in Matlab library that we use returns L2-norm of the distance), in Figure 4.7a we use the L2-norm residual $\|F_k\|_2$. The peak at the second iteration of DI is due to the definition of L2-norm of F: because F_0 is uniform, after the first diffusion iteration, due to damping factor we have $\|F_1\|_1 < \|F_0\|_1$ but as the entries of F_1 are non-uniform, we get $\|F_1\|_2 > \|F_0\|_2$. The figure shows that GMRES converges faster than DI in terms of iterations. However, DI consumes much less elementary operations than GMRES as seen in Figure 4.7b. Remind that the *restart* value determines the Krylov subspace



(a) Number of iterations



(b) Number of elementary operations

Figure 4.7: Dataset `uk-2007`: $n = 10^4$ nodes, DI and GMRES comparison

size. If it increases, the Krylov subspace will grow larger and the cost spending on some tasks such as vector orthogonalization will become remarkably heavy.

4.4 Intermezzo: PageRank approximation with partial information

Before we move on to larger graphs, we propose one last experiment on the dataset `uk-2007-05@1000000`. In this section, we will investigate how to approximate true PageRank vector of a matrix of which all columns (or rows) are not given. In other words, the transition matrix is now partially hidden.

4.4.1 Context

The PageRank vector computed in the previous chapters is the principle eigenvector of the (possibly modified) transition matrix P of a graph $\mathcal{G}(V, E)$ such that $x = Px$. In this section, we consider that $\mathcal{G}$ is not stored on the same machine used to compute the PageRank vector. Now the graph is not visible and can only be gradually revealed by sending requests to a storage server (see Figure 4.8). At a time, the PageRank computation machine can request information of a full i^{th} column of the matrix (list of outgoing neighbours of node i) or information of a full i^{th} row of the matrix (list of incoming neighbours of node i). For the sake of simplicity, we assume that the machine always requests either column or row consistently during the whole computation process, but not the mix of them. To further fix the assumption, we suppose that the machine always asks for column information (outgoing neighbours) of a node. The problem that we study in this chapter is finding a good strategy for requesting columns, i.e., order of nodes requested, such that these following two statements are satisfied:

- the number of requests exchanged between the machine and the server is minimized, i.e., the computation cost (time) is kept minimum, assuming that communication is the main bottleneck.

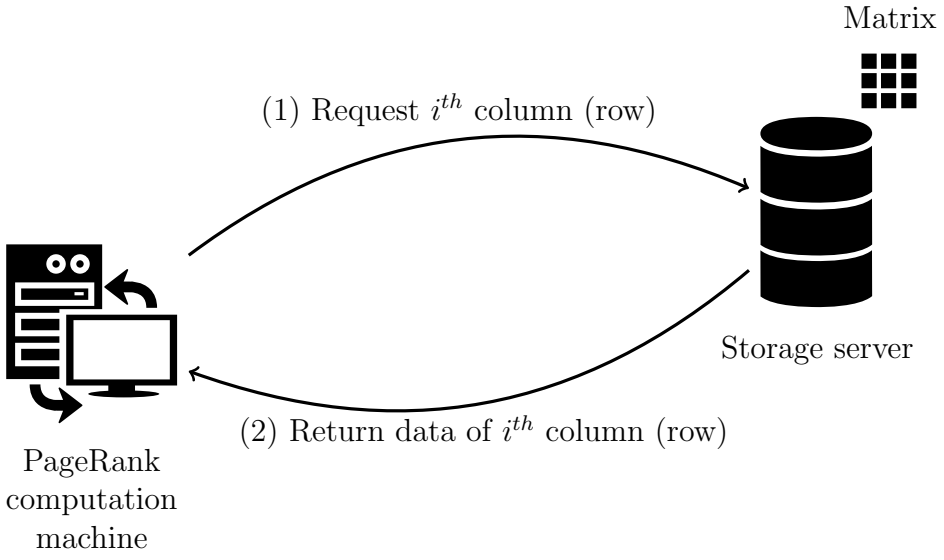


Figure 4.8: Context of PageRank approximation where storage server and computation machine are apart. Information of column (or row) matrix is transferred upon request from storage server to computation machine.

- the approximate PageRank vector (or top values of the approximate vector) calculated based on partial information is as close as possible to the true PageRank values computed as if the full graph (matrix) is known.

In order to achieve that, previous studies suggest to request new nodes in a Bread First Search (BFS) manner [47] or high-ranking nodes [48]. However, we show that a mix strategy could give a much better result.

4.4.2 PageRank approximation

To begin with, we have to differentiate two problems: (i) how to compute PageRank vector of P in case of lacking information and (ii) what is a good strategy to request column of P to have a good approximation. It is natural to say that, the vector can be calculated merely based on the visible part (information of nodes requested) and we consider $\vec{0}$ (vector of all zeros) at all columns for the unknown part, i.e., those nodes are supposed to be dangling nodes (zero outdegree). However, what is a good request order is not obvious to see. Figure 4.10 illustrates an example of sequential request to a full graph (cf Figure 4.9). Initially at time t_0, the machine does not have any information about edges in the graph, thus all nodes have equal

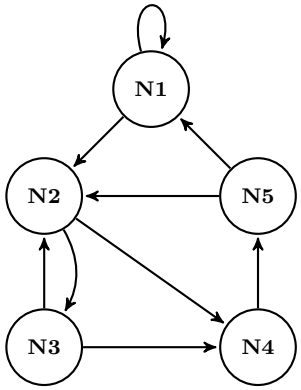


Figure 4.9: Full matrix at storage server

PageRank value that is $1/n$. However, this PageRank vector is far from that of the full graph due to a huge lack of information. Then, at time t_k, it asks for outgoing neighbour list of node k and modified its perspective on the graph, then recompute the PageRank vector according to the updated perspective. Obviously, the more nodes are requested, the closer the perspective is to the real graph, and therefore the more precise the PageRank approximation is.

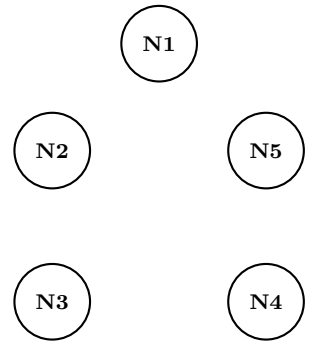
4.4.3 Experiment

In the following, we assume that the order of requests is $\mathcal{I} = \{i_1, i_2, ..., i_k, ...\}$ with $i_k \in \{1, .., n\}$. We can think of two basic choices:

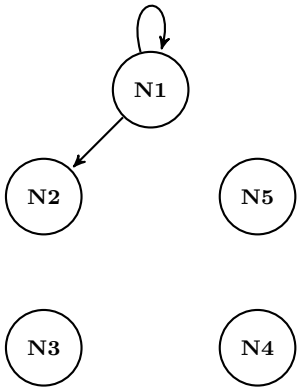
- `Random`: node i_k is randomly requested among nodes which have not been chosen yet.
- `Max`: node i_k is requested if it has the highest PageRank value (and not yet chosen), computed based on the information given by previous chosen nodes $i_1, ..., i_{k-1}$.

We propose an additional strategy:

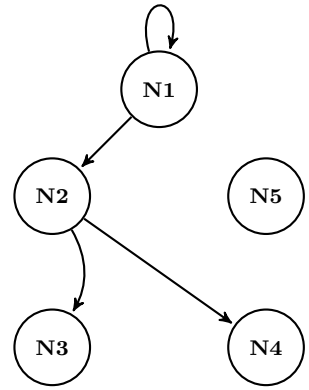
- `Max + Random`: the `Random` and the `Max` strategy are used alternatively. If at time t_n, `Random` strategy is applied to request node i_n, so at time t_{n+1}, `Max` strategy is applied to request node i_{n+1}, and vice-versa. The frequency at which each strategy is used is an important factor to optimize the algorithm.



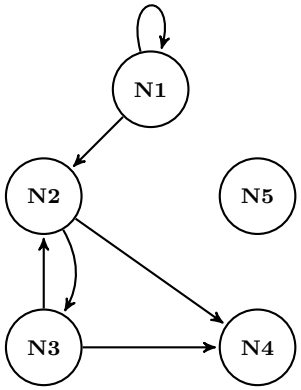
(a) Time t_0, graph without edge



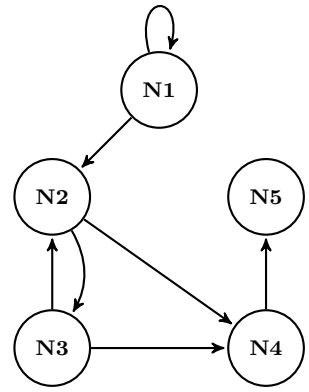
(b) Time t_1, request **N1**



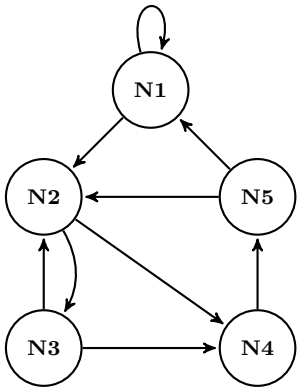
(c) Time t_2, request **N2**



(d) Time t_3, request **N3**



(e) Time t_4, request **N4**



(f) Time t_5, request **N5**

Figure 4.10: The graph is gradually reconstructed over time at the computation machine. Requests for outgoing neighbour list are made sequentially from **N1**,...,**N5**.

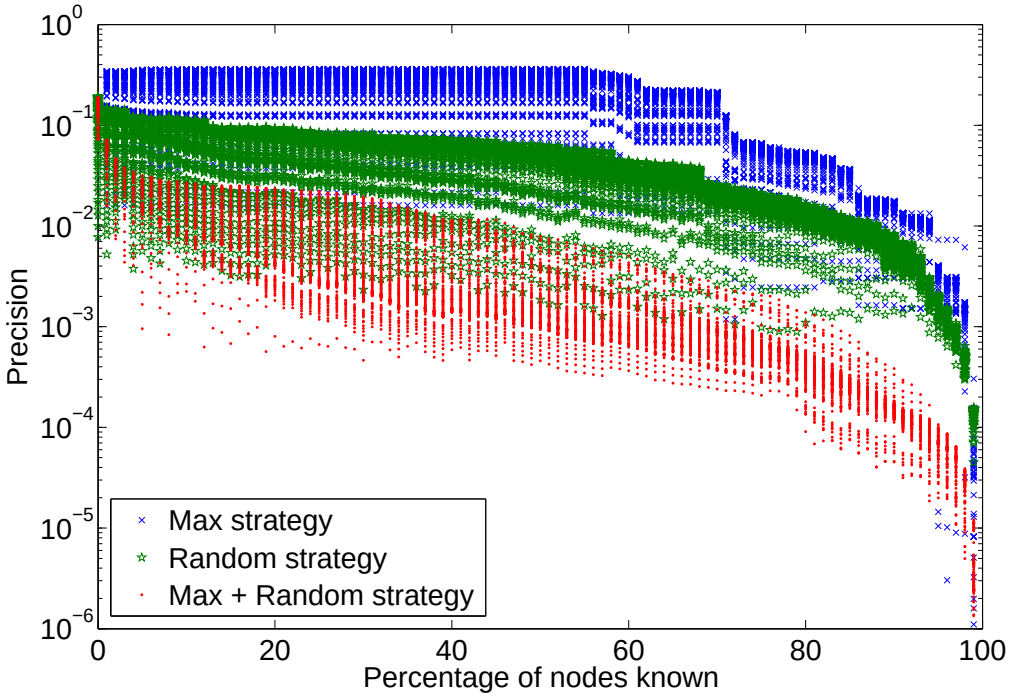


Figure 4.11: Web graph `uk-2007`: PageRank approximation with partial information

This is subject to future research. In this preliminary work, we consider equal probability between the two strategies, i.e., $\frac{1}{2}$ `Random` + $\frac{1}{2}$ `Max`.

Figure 4.11 shows the result on the `uk-2007-05@1000000` dataset. The y-axis represents the L1-norm of the difference between the approximate PageRank vector and the true vector of the graph, computed using DI with precision 10^{-9}. The x-axis indicates the percentage of nodes requested with respect to the total number of nodes in the graph. The request sequence follows the three strategies mentioned above: `Random`, `Max` and `Max + Random`. For each additional percent of new nodes unveiled, we recompute the vector and compare it with the true one. The more nodes we crawled, the better the performance indicated by a lower (better) precision. As expected, all strategies converge as the percentage reaches 100%: at that time, the "approximate vector" is exactly the true PageRank vector of the graph. In the figure, we show the comparison results of various sets of vector entries randomly extracted from the approximate vector and the true PageRank vector.

A remarkable point is, the `Max` strategy does not perform well among the three strategies. This can be explained as follows. Since we only choose the node having highest PageRank according to the updated perspective, that node is probably connected to those which were already requested because it receives more PageRank score than others that are isolated (or those apart from the visible part of the graph). Therefore, it takes a long time for this strategy to find high PageRank nodes that are far from the initial node.

Unlike `Max`, the `Random` strategy can avoid being locally trapped because nodes are randomly requested. Hence, it behaves better than the `Max` strategy. However, the combined strategy of `Max + Random` outperforms the other two by a gain factor of 10, thanks to the fact that it benefits the advantages of both strategies.

4.5 Conclusion

In this chapter, we evaluated the performances of some algorithms (Jacobi, Gauss-Seidel, SOR, GMRES and OPIC) on a real web graph, then compare them with the two variants of D-Iteration (`DI-cyc` and `DI-argmax`). We saw that GMRES requires less iterations to converge but the additional cost caused by vector orthogonalization makes it become heavy. On the other hand, thanks to a smart scheduler, the `DI-argmax` only diffuses at important nodes (containing much fluid) so that it converges faster than other methods in terms of elementary operation cost which relates directly to the computation time. Additionally, we proposed a method to approximate the PageRank vector of a graph partially hidden. Note that we used a $\frac{1}{2}$ `Random` + $\frac{1}{2}$ `Max` strategy and how to weight the two choices is still an open question.

Chapter 5

Experiments on large graphs

After first experiments with small graph in the previous chapter, we now consider bigger datasets. Computation on large graphs in general is challenging due to resource constraints such as memory usage and computation time. The first part of this chapter focuses on evaluating the performance of D-Iteration (DI) specifically in solving PageRank equation and compare it on a larger scale with other algorithms seen before like Jacobi, Gauss-Seidel (GS), Online Page Importance Computation (OPIC), etc. Each of them has its own pros and cons. Obviously, finding the best algorithm for all cases is impossible. Therefore, the main target of this chapter is instead to find a good candidate, given a specific case which depends on the nature of the graph and how the resource constraints are defined.

We will also discuss the application of PageRank to Twitter social graph. Ranking users in such a network is interesting but quite challenging as mentioned in Section 1.1.3. Some results will be shown to compare the outcomes of ranking according to PageRank and to indegree.

In this chapter, our main contributions consists in evaluating the performance of D-Iteration in solving PageRank equation. The benchmarks are carried out on graphs varying in size (number of vertices and edges) and in type (undirected/directed social networks and web graphs). Besides, we present the application of PageRank to rank Twitter users and compare its efficiency with the classical ranking according to the number of followers (indegree). The results of this chapter are partly presented in [21].

The chapter is organized as follows. Section 5.1 introduces the large datasets used. Section 5.2 shows the comparisons criteria and evaluates the performance of the algorithms. Section 5.3 talks about applying PageRank to rank Twitter users. Section 5.4 concludes the chapter.

One other related experiment on large graph, "LiveRank", will be described in the next chapter to limit the size of this chapter.

5.1 Datasets and settings

In order to compare the convergence speed of DI with other algorithms exposed in Section 2.2: Power Iteration, Gauss-Seidel, Jacobi and OPIC, we use the six following datasets [1]:

- `it-2004`: contains 41 million nodes and over 1.1 billion links, representing part of Italian web graph (*.it* domain). Link from node x to node y indicates an hyperlink from page x to page y. We use the original version.

- `uk-2005`: roughly 39.5 million nodes and 940 million links of British web graph (*.uk* domain). This crawl has been done by UbiCrawl [38] which aims at enlarging number of hosts rather than number of pages crawled at each host. We make use of the original graph.

- `uk-06-07`: this dataset is a time-aware graph [49] generated by combining twelve monthly snapshots (from May 2006 to May 2007) of the `.uk` domain. It contains 133 million nodes and 5.5 billion links. This is a crawl done for DELIS project [43].

- `orkut-2007`: this dataset is a snapshot of *IMC 2007 Data Sets*[2]. Orkut is a social network being launched and operated by Google. Unlike Twitter, a relationship between two users requires acceptance from both of them (one sends request and one confirms). As a consequence, its links are undirected. However, about 10% of links miss their opposite in the original graph due to crawling issues, as suggested in [50] we completed the graph so that it is

[1] The first five datasets are available at [42] and the last one (`twitter-2012`) at `http://www-sop.inria.fr/members/Arnaud.Legout/Projects/sotweet.html`

[2] `http://socialnetworks.mpi-sws.org/data-imc2007.html`

symmetric, and an undirected link from x to y depicts a mutual friendship between x and y. After completion, this dataset contains 3 million nodes and over 234 million directed links (representing 117 million friendships).

- `twitter-2010`: contains about 42 million nodes and 1.5 billion links. Nodes are Twitter users and link from x to y means user y follows user x. In other words, links indicate direction of tweet propagation. The dataset was presented in [51]. In PageRank context, a web page is important if it is pointed to by many other important pages. We assume one applies the same philosophy to Twitter user ranking, i.e., an important user is pointed to by many other important users. Obviously, link direction in user ranking graph is inverse with respect to tweet propagation graph. So instead of using the original `twitter-2010` graph, we exploit its transposed version.

- `twitter-2012`: a complete snapshot of Twitter crawled in 2012 by Gabielkov *et al.* [52] and available for academic studies. The graph contains 400 millions nodes and 23 billions edges. It is given under the form of a very large [3] adjacency list: each user having at least one follower corresponds to one line showing the user identifier, the number of followers and the list of followers. All users who are completely isolated, i.e., zero indegree and zero outdegree, are not referenced (and that results in 399 millions instead of 505 millions [52]).

The properties of the graphs are summarized in Table 5.1.

Dataset name	N	L	max_in	max_out	L/N	D/N	E/N	O/N
it-2004	41,291,594	1,150,725,436	1,326,745	9,964	27.87	0.1276	0.0001	0.36
uk-2005	39,459,925	936,364,282	1,776,852	5,213	23.73	0.1100	0.0010	0.38
uk-2006-2007	133,633,040	5,507,679,822	6,366,525	22,429	41.22	0.09	0.0537	0.24
orkut-2007	3,071,378	234,370,166	33,313	33,313	76.28	0.0001	0.0001	0
twitter-2010	41,652,230	1,468,365,182	2,997,469	770,155	35.25	0.1439	0.0382	0
twitter-2012	398,846,191	23,137,510,395	24,635,412	734,806	58.01	0.0718	0.2697	0

Table 5.1: Datasets statistics

- **N**: number of nodes in the graph, i.e., number of columns/rows in matrix,

- **L**: number of links, i.e., number of non-null entries in transition matrix P,

[3] 100 GBs of compressed text format

- **max_in**: maximum in-degree of a node, i.e., row with maximum non-null entries in the transition matrix,
- **max_out**: maximum out-degree of a node, i.e., column with maximum non-null entries in the matrix,
- **D**: number of dangling nodes, i.e., nodes with no outgoing link or columns with only null entries,
- **E**: number of *in-tendril* nodes; initially the in-tendril nodes are nodes without incoming links (zero-indegree nodes) and they are recursively defined: if a node has all incoming links from in-tendril nodes, it can be called as transitory node and also counted as in-tendril node. The reason we introduce this notion is that those nodes converge in finite steps with DI, so DI will perform especially well on graphs with a high number of those.
- **O**: number of self-loop nodes.

The PageRank settings (construction of matrix P, $d = 0.85$ and $Z \equiv \frac{1}{n}$) are as in Section 4.1

5.2 Performance evaluation

5.2.1 Comparison criteria

Besides the two metrics mentioned in Section 4.2 (iterations and elementary operations), we also benchmark the performance based on *request messages*. This metric is often used in parallel computations where the matrix is stored on a separated machine. Each time a node is iterated on, the computation machine will send a request message to get a row value (Jacobi, GS) or column value (DI) to the central database. This should be considered as a communication cost in a distributed system. As a direct consequence, regardless of how many non-null entries there are in a row/column, the cost to get information of a row/column is identical, *i.e.* the cost of one message sent.

Keeping in mind that updating values can be costly in very large datasets due to parallelism, a good DI scheduler should try to achieve a trade-off between

the amount of fluid (convergence) and the outdegree (update-cost) of a node. If one wants to optimize the diffusion achieved per update, it seems natural to try and select the nodes with the highest fluid/outdegree value. Following that idea, Dohy Hong introduced in [19] a DI variant, called DI-argmax/outdeg, which works as follows: at k^{th} iteration, diffuse $F_k[i]$ iff $F_k[i]/outdeg[i] > l$ where $l = \sum_i F_{k-1}[i]/|E|$ ($|E|$ is the total number of links in the graph).

5.2.2 Web graphs

The two next sections will show the performance evaluations of the algorithms on the datasets.

it-2004 and uk-2005

Minimizing elementary operation cost means optimizing link utilization which aims at a prudent choice of whether or not DI should perform diffusion on a link. Simply put, if fluid passing through each outgoing link (coming from the same node) is greater than that of the average of the whole graph, `DI-argmax/outdeg` will diffuse that node; otherwise, it does not.

The other strategy, `DI-argmax`, targets at minimizing the number of requests sent in a parallel deployment. Each request of the client asks the central database for an entire column information representing outgoing neighbours list of a node. The strategy consists in optimizing the amount of fluid to diffuse at each node. If this quantity is sufficiently large, i.e., greater than the average of the whole graph, it is worth a diffusion. Another intuition is that each request sent means some fluid disappeared. The larger the volume vanishes, the quicker the convergence.

One of the important differences between a social network graph and a web graph is the existence of self-loop nodes. They should not appear in social network friendship context because a user cannot make friend with himself. In contrast, it is quite common to have such nodes in various sorts of web graph mostly because of anchors and permanent navigation links. In case of DI, self-loop nodes slow down the convergence speed since the nodes always receive some fluid coming back to themselves after diffusion. To avoid this problem, we modify the diffusion condition as in Algorithm 4. Each time a self-loop node is chosen, we emulate what would

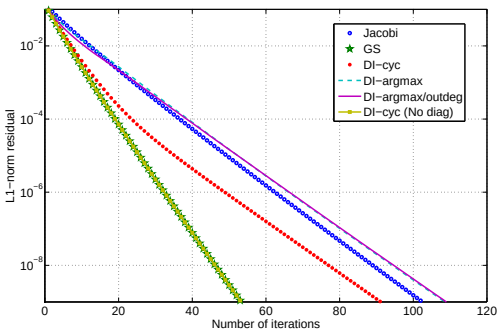
happen if that node was selected an infinite number of times until all of its fluid is gone. In that case, the fluid is accumulated over time before being completely diffused through the outgoing links. The additional cost of setting the new fluid amount (see Algorithm 4: line 11) is equal to $2T_m + T_a$ operations. This cost then can be further reduced to only T_m if we pre-compute once the value $1/(1 - dP_{i_k,i_k})$ and store it at each self-loop node. Remind that although GS and the modified version of `DI-cyc` (*i.e.* `DI-cyc` using the new diffusion condition, called `DI-cyc (No diag)`) are computed differently, they are exactly equivalent (w.r.t number of iterations and precision), this will be explained in Appendix B. In general, if one applies *diagonal elimination* [4] on a graph having self-loop nodes, it will result in the same behaviour between GS and `DI-cyc`. One can also observe this phenomenon in the social network graphs where GS, `DI-cyc` and `DI-cyc (No diag)` behave in the same way due to the absence of loops (**O**=0).

Algorithm 4 D-Iteration with modified diffusion condition: $x = dPx + (1 - d)Z$.

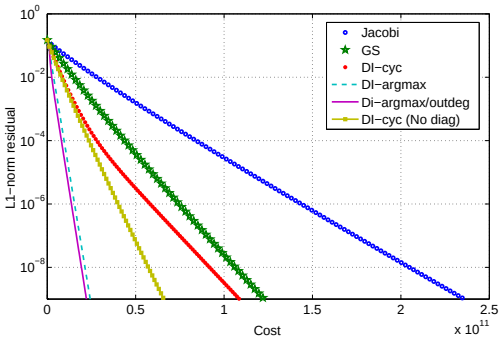
1: **for** $i = 1 : n$ **do**
2: $H(i) = 0$; ▷ Initialize result vector
3: $F(i) = (1 - d)Z(i)$; ▷ Initialize diffusion vector
4: **end for**
5: $k = 1$;
6: **while** $(\|F\| > Target\_Error)$ **do**
7: Choose i_k;
8: $sent = F(i_k)$;
9: $F(i_k) = 0$;
10: **if** $P_{i_k,i_k} \neq 0$ **then** ▷ check if i_k is a self-loop node
11: $sent = sent \times (1/(1 - dP_{i_k,i_k}))$;
12: **end if**
13: **for** all child node j of i_k $(j \neq i_k)$ **do**
14: $F(j) += sent \times dP_{j,i_k}$;
15: **end for**
16: $H(i_k) += sent$;
17: $k++$;
18: **end while**

`It-2004` is built on the web graph of Italian network. The highest indegree node is pointed to by 1.3 million nodes and the highest outdegree one points to 10

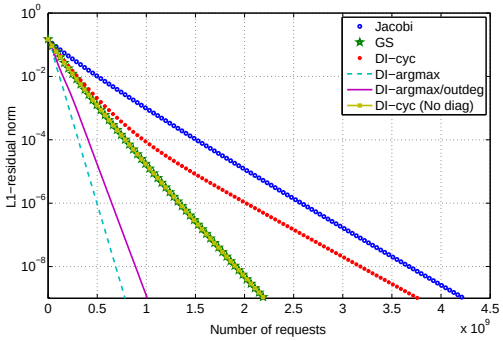
[4] Diagonal elimination is the process of removing self-loop links of nodes in a graph and adjusting their corresponding incoming link weights (details in Appendix B).



(a) Iterations



(b) Number of elementary operations



(c) Number of requests sent

Figure 5.1: `it-2004`

thousands others. On average, each page has 28 outgoing links whereas 12% of them are dangling nodes. A very small are in-tendrils (**E/N**=0.01%) and a noticeable number of nodes (36%) have loops.

Figure 5.1 shows some benchmarking results. GS and `DI-cyc (No diag)` outperform other algorithms w.r.t. iterations in Figure 5.1a. We deliberately plotted `DI-cyc` with diagonal links to illustrate its impact on convergence speed: `DI-cyc` starts to converge more slowly from 10^{-2} precision point. `DI-argmax` and `DI-argmax/outdeg` clearly dominate `DI-cyc (No diag)` and thus GS in the two remaining criteria. In Figure 5.1b, `DI-cyc (No diag)` brings a remarkable improvement to `DI-cyc` (a gain factor of 4) in terms of operations and `DI-argmax` achieves a gain factor greater than 2 (w.r.t. request messages) as depicted in Figure 5.1c. But the general result still holds: `DI-argmax` works well with the request messages while `DI-argmax/outdeg` is optimized for elementary operation cost.

To verify our results, we perform the same tests on `uk-2005`, a web graph constructed from *.uk* domain with similar properties than `it-2004`. They behave alike on all three criteria as shown in Figure 5.2.

The advantage of `DI-argmax/outdeg` is in terms of elementary operations. Each node diffusion corresponds to a sequence of multiplications and additions. More precisely, let c_i and r_i be the number of non-null entries of the i^{th} column and the i^{th} row of the transition matrix P of the graph. Recall that T_a and T_m are the operational costs of an addition and a multiplication respectively, and that $T_a = T_m$ (see Section 4.2). In the PageRank context, outgoing links from the same node receive a uniformly distributed fluid portion from that node. Suppose that the current result vector is a full vector (without any null entry), we have

$$\begin{aligned} D_i &= T_m + T_a \times c_i. \\ C_i &= (T_m + T_a) \times r_i. \end{aligned}$$

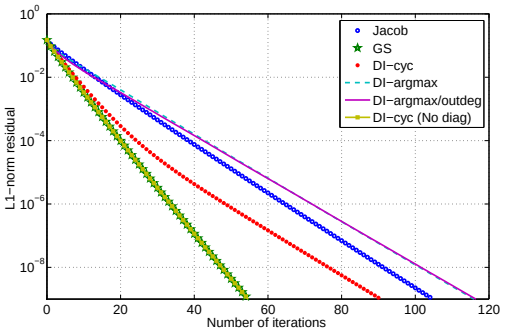
where D_i and C_i are diffusion cost (of DI) and collection cost (of Jacobi and GS) at node i. In general, the costs of one full iteration of DI (`DI-cyc`) and Jacobi/GS could be written

$$\begin{aligned} Cost_{DI-CYC} &= \sum_i D_i &= T_m \times N + T_a \times L. \\ Cost_{Jacobi/GS} &= \sum_i C_i &= (T_m + T_a) \times L. \end{aligned}$$

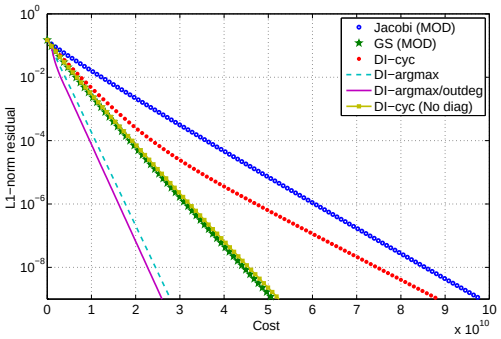
where N is the number of nodes and L is the number of links of the graph. Consequently, with the same number of iterations, `DI-cyc` consumes less elementary operations than GS and Jacobi. However, one can think of an improvement of GS and Jacobi , by storing and making use of the value $x^i/outdeg(i)$ instead of x^i (i^{th} element of vector x), for all components of vector x as follows:

`Jacobi (MOD):`

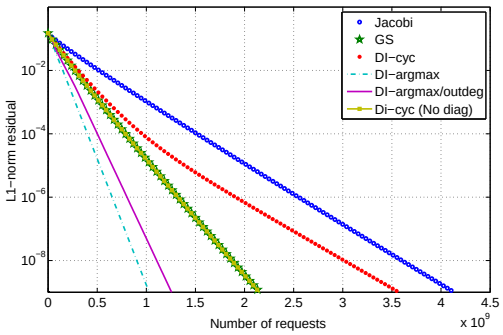
$$\frac{x_{k+1}(i)}{outdeg(i)} = d \sum_j \frac{x_k(j)}{outdeg(j)} + (1-d)Z(i). \tag{5.1}$$



(a) Iterations



(b) Number of elementary operations



(c) Number of requests sent

Figure 5.2: `uk-2005`

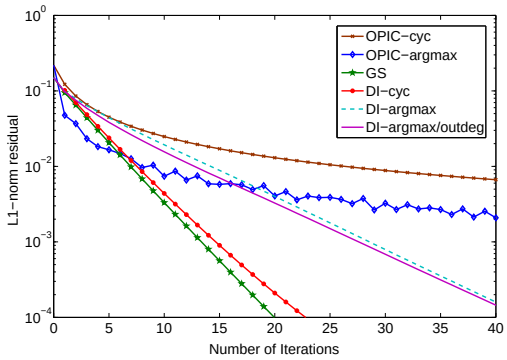
`GS (MOD)`:

$$\frac{x_{k+1}(i)}{outdeg(i)} = d\left(\sum_{j<i}\frac{x_{k+1}(j)}{outdeg(j)} + \sum_{j\geq i}\frac{x_k(j)}{outdeg(j)}\right) + (1-d)Z(i). \tag{5.2}$$

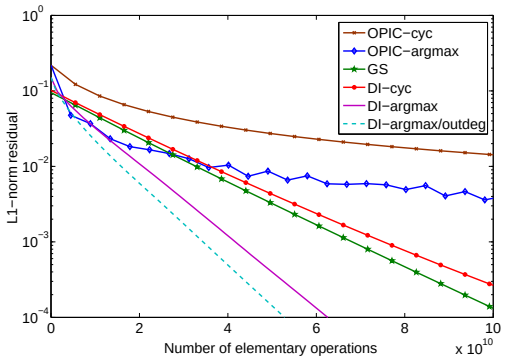
Additional costs caused by this modification, e.g., reconstruction of the final vector x, can be considered negligible so that the cost of `Jacobi (MOD)` and `GS (MOD)` decreases roughly by a half (thanks to $T_m = T_a$). We give the result in Figure 5.2b with the analysis of `uk-2005` dataset. From now on, unless we mention explicitly on the plots the algorithms `Jacobi (MOD)` and `GS (MOD)`, we use the classical Jacobi and GS.

Web graph: uk-2006-2007

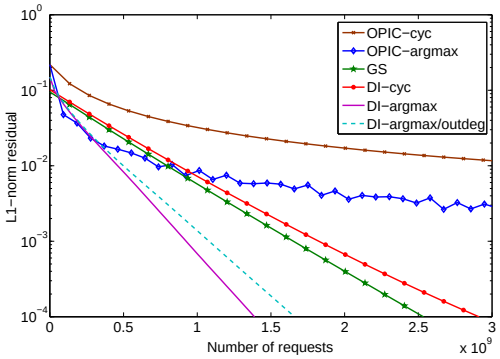
The main goal of this experiment is comparing the performances of OPIC (`OPIC-cyc` and `OPIC-argmax`) with GS and DI (`DI-cyc`, `DI-argmax` and



(a) Iterations



(b) Number of elementary operations



(c) Number of requests sent

Figure 5.3: `uk-2006-2007`

`DI-argmax/outdeg`). The evaluation results on web graph `uk-2006-2007` are shown in Figure 5.3.

The `OPIC-argmax` method performs well during the first few steps in all comparison criteria: iterations (see Figure 5.3a), elementary operations (see Figure 5.3b) and request messages (see Figure 5.3c). It has a clear advantage over `OPIC-cyc`. However, the convergence slows down really hard after. Note that OPIC remains interesting as its primary goal was to provide lightweight PageRank estimates. The results only state that OPIC should not be used for precise PageRank estimations.

GS converges fastest in terms of iterations and has the second best convergence speed, better than OPIC but worse than DI, in the other two metrics. It still is a good candidate if one needs a simple and efficient way to compute a precise PageRank.

`DI-cyc` performs very similarly to Gauss-Seidel (although a little slower). This is in line with our interpretation that `DI-cyc` is a kind of *push* version of

Gauss-Seidel. On the other hand, `DI-argmax` clearly outperforms the other methods in request messages metric and `DI-argmax/outdeg` in computational cost.

5.2.3 Social network graphs

In addition to web graphs, we test the algorithms on social network graphs. The properties of the two types of graphs are different from each other, for example in the degree distribution or the fraction of self-loop nodes, etc.

Social network undirected graph: orkut-2007

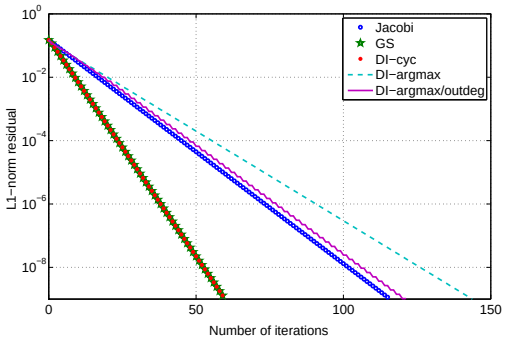
`Orkut-2007` is constructed based on mutual relationship between users so that each link is supposed to have its opposite. Remind that about 10% of links of the original dataset missed their opposite and we had to complete them to have a symmetric graph. This graph is clearly characterized by some specific properties, *e.g.* maximum indegree and outdegree are equal. Orkut social network graph does not have self-loop node (like Twitter) and in our graph each node has on average 76 links. A negligible number of nodes (0.01%) are isolated nodes (for example new users) with no connection to the rest of the graph.

An important remark is that for undirected graphs, without the damping factor d ($d = 0$), the PageRank of a node is proportional to its degree. However, if the damping factor is used, this property no longer holds [53].

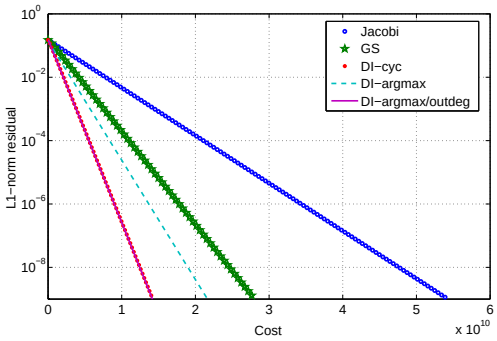
Convergence rate measured in iterations is quite similar between GS and `DI-cyc` as in Figure 5.4a. The explanation comes from social network context where there is no self-loop node. In Figure 5.4b and Figure 5.4c, `DI-cyc` and `DI-argmax/outdeg` perform quite similarly and we suspect that even `DI-argmax/outdeg` needs more iterations to converge, the nodes are activated almost at the same frequency due to the symmetric graph structure without loops. However, a concrete explanation requires more investigations. Figure 5.4c confirms that `DI-argmax` is still a good candidate in terms of number of messages.

Social network directed graph: twitter-2010

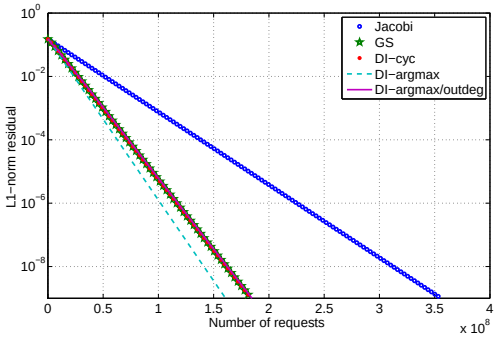
In the `twitter-2010` dataset, the node having maximum indegree is labelled `23934132` standing for Twitter User ID `19058681`. This user is Ashton Kutcher,



(a) Iterations



(b) Number of elementary operations



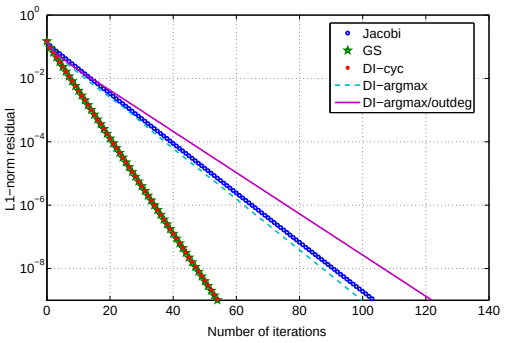
(c) Number of requests sent

Figure 5.4: `orkut-2007`

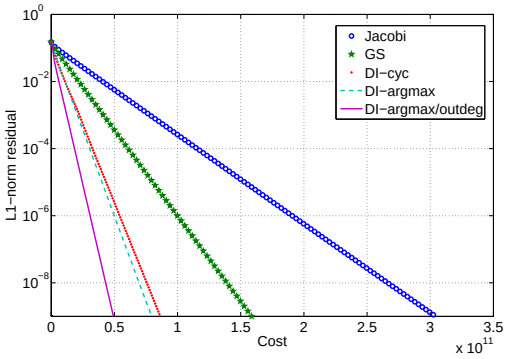
an American actor and producer. He had 2,997,469 followers when the crawl was done in 2010. On the other hand, the node with maximum outdegree (770,155) is `21513299` which corresponds to Twitter User ID `28744551`.

In DI, diffusion at dangling nodes gives faster convergence because the fluid amount vanishes without being transferred to other nodes. 14% nodes of Twitter dataset are dangling nodes (**D**). They can be seen as users following no one and probably be accounts created only for pushing information, e.g., institutional accounts. The dataset also has no self-loop node (**O**) because a Twitter user cannot follow himself.

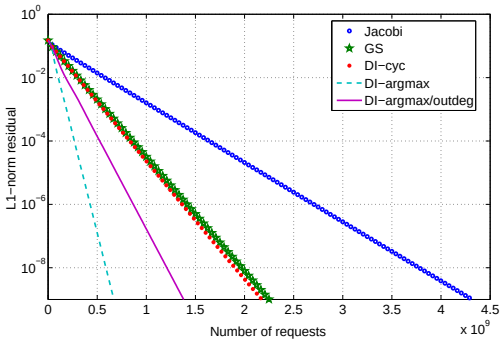
Figures 5.5 details how the algorithms perform. In Figure 5.5a, GS and `DI-cyc` converge within 55 iterations to precision 10^{-9} whereas `DI-argmax`, Jacobi and `DI-argmax/outdeg` require 101, 104, and 122 iterations respectively. As expected with this criterion, GS behaves twice better than Jacobi thanks to the use of the last updated result vector. We have no gain in the number of iterations from



(a) Iterations



(b) Number of elementary operations



(c) Number of requests sent

Figure 5.5: `twitter-2010`

`DI-argmax` and `DI-argmax/outdeg` because they always check if fluid at a node satisfies certain condition before diffusing so they perform less operations per iteration. The number of diffused nodes at each iteration is then less than n, it ends up with a larger number of iterations. However, the gain of `DI-argmax/outdeg` is remarkable compared to GS (and of course Jacobi) in Figure 5.5b, about a factor of 3. Figure 5.5c indicates a gain factor of 3 between `DI-argmax` and GS.

5.3 PageRank for Twitter user ranking

The Twitter social network [5] allows to disseminate information in the form of short messages (*tweets*) that are transmitted mainly through subscription relation-

[5] https://twitter.com/

ships: a message posted by a user will be sent to all subscribers (*followers*) who will each in turn decide to transmit the message to their subscribers.

Given a large amount of information generated (500 million tweets per day on average in 2014), it is important to have efficient search tools, such as a popularity indicator to distinguish the most interesting users . A commonly used metric is the number of subscribers. However because of its simplicity, it generates many abuses, such as the ability to buy large quantities of subscribers to give an illusion of reputation [54].

A natural alternative is using PageRank to modulate recursively the importance of users. The idea of applying PageRank to rank Twitter users is not new, but to our knowledge there has been no evaluation of the algorithm in large scale.

5.3.1 Model

The Twitter subscription graph indicates the follower/followee relationships. It is a directed graph $G = (V, E)$, where V is the set of Twitter users and E is the set of relationships. The presence of a pair $(i, j) \in E$ means that i follows j. This is inspired by the standard representation of Web graphs hyperlinks models and therefore the number of followers of a user is its indegree.

We use the graph `twitter-2012` (see Section 5.1) for the experiment. Because identities of users in the graph are anoymized, it is impossible to verify the validity of the PageRank by checking the actual user profiles. However, what we do is comparing the ranking given by PageRank and the one given by indegree which represents somehow the popularity of users.

To be precise, given two ranking order R_1 and R_2, we define $C_k(R_1, R_2)$ as the overlap of the first k items in R_1 and R_2, or more formally:

$$C_k(R_1, R_2) = 100\frac{|\{i \text{ such that } R_1(i) \leq k \text{ and } R_2(i) \leq k\}|}{k}. \tag{5.3}$$

where $R(i)$ is the rank of item i in the ranking order R.

We use D-Iteration to compute the PageRank of the Twitter graph. Due to the lack of studies on how to choose suitable parameters for Twitter case, we conserve the damping $d = 0.85$ and $Z \equiv \frac{1}{n}$.

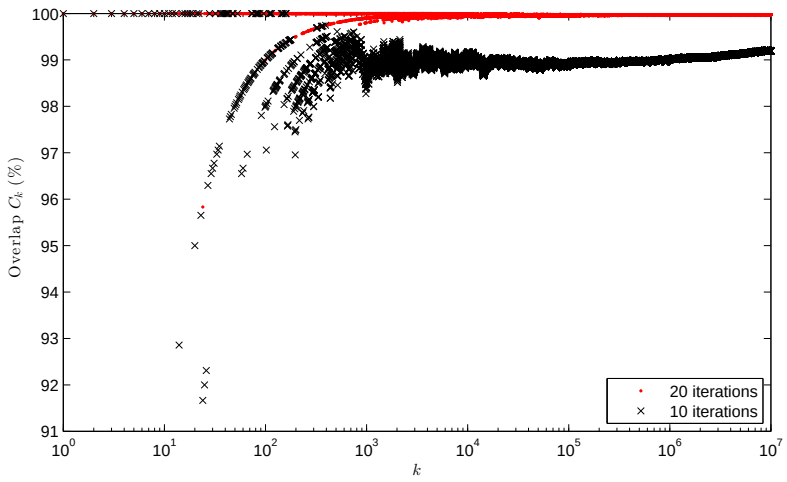


Figure 5.6: k-overlapping between PageRank after first few iterations and the real PageRank.

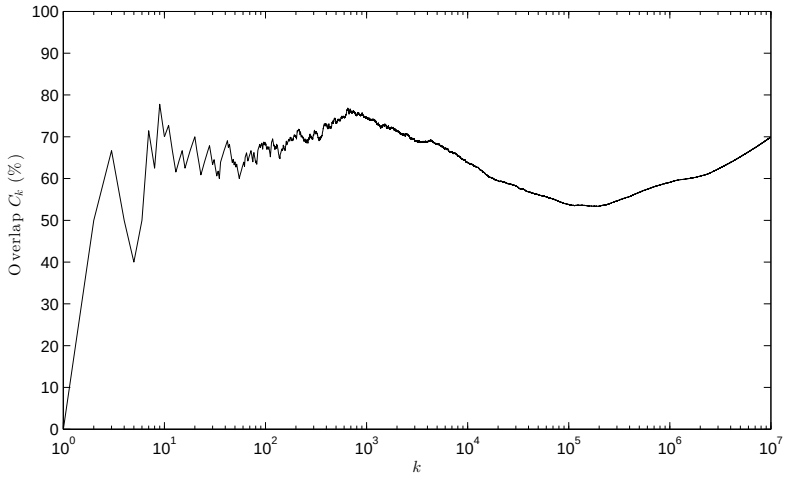


Figure 5.7: k-overlapping between PageRank and indegree.

5.3.2 Result

If we are only interested in the most popular users, a few iterations are sufficient to point them out. This is shown in Figure 5.6: after 10 iterations, the overlap with the real PageRank is more than 90% over the first thousand users, and 99% between 10^3 and 10^7. To 20 iterations, the classification of the first 10^7 users almost converges. So for the study of the most popular users, we can set the stopping condition at 20 iterations.

PageRank can be seen as a refinement of indegree. So one might think that the difference between the two rankings is to be relatively small. As shown in Figure

5.7, it is not the case: the overlap is relatively low over the range of interest, seldom greater than 70%. The PageRank thus provides additional information that is not contained in the indegree, which is already interesting in itself.

Another interesting point is the presence of a fairly steady slope between $k \approx 1000$ (users with one million or more followers) and $k \approx 100,000$ (10,000 or more followers), which leads to a local minimum of 54%. The greatest divergence action on this range means that among users who mainly around 10,000 subscribers and more (up to 1,000,000), we find relatively more users receiving little PageRank from their followers, either because the followers have low popularity or because they follow more users than the average.

It is difficult to give a concrete explanation about this phenomenon due to the anonymity of the graph, but we can think of the two following hypothesis: firstly, it is possible that the most popular users in terms of followers (indegree) are mostly followed by many passive users who produce very few contents or even not at all; another possibility is that the depression corresponds to users who have illegally bought followers to boost their illusive popularity [55]. The second hypothesis is reinforced by the depression zone, meaning the weak PageRank comes from the fact that most of follower sellers provide buyers with low-quality accounts that do not have any follower in general, i.e., very low PageRank and are also sold to many other buyers. As a consequence, the PageRank given by those accounts are very low.

5.4 Conclusion

In this chapter we presented some experimental benchmarking results of the algorithms: Power Iteration, Jacobi, GS, OPIC and DI to solve PageRank equation. The comparison metrics are the number of iterations, elementary operations and request messages. For each of the later two criteria, there is a DI variant that gives better performance compared to the other classical iteration methods. One advantage of DI is the capability to be adapted to distributed computation thanks to its fully asynchronous nature. Evaluating the actual performance gain of the DI variants in a real distributed environment requires more studies (clever partitioning scheme, current state of the art, etc.) and are subject to future work. Additionally, we also see the results of applying PageRank to rank Twitter users. It reveals

interesting information that the algorithm can provide in comparison with the mere indegree-based ranking.

Chapter 6

LiveRank

One of the main challenges for large networks data mining is dealing with the high dynamics of huge datasets: not only are these datasets difficult to gather, but they tend to become obsolete very quickly. We are interested in the evolution at large time scale of any large corpus available online. Our primary focus will be the Web, but our approach encompasses any online data with similar linkage enabling crawling, like P2P networks or online social networks. We thus focus on batch crawling, where starting from a completely out-dated snapshot of a large graph like the Web, we want to identify a significant fraction of the nodes that are still alive now.

In this chapter, our main contribution is proposing a PageRank-based algorithm helping to build a crawler that can efficiently crawl alive nodes of an old graph. The advantage of our method is the ability to work in a distributed way. It can also be exploited in different types of graphs, for example to guess active users in social networks like Twitter. The results of this chapter were published in [22, 23].

This chapter is organized as follows. Section 6.1 introduces context and motivation. Section 6.2 talks about the state of the art of previous works. We propose in Section 6.3 a simple metric to evaluate the quality of a LiveRank and we propose several classes of possible solutions, which we call LiveRanks. To confirm this intuition, we use three datasets from the Web and from Twitter, for which we have extracted the activity of items. They allow to verify activity correlations and to test the proposed LiveRanks. The datasets and how activities were obtained are

described in Section 6.4. In Section 6.5, we benchmark our LiveRanks against the datasets and discuss the results. Finally, Section 6.6 concludes the chapter.

6.1 Introduction

Many old snapshots of large graphs are available today. Reconstructing roughly what remains from such archives could result to interesting studies of the long term evolution of these graphs. For large archives where one is interested in a particular type of pages, recrawling the full set of pages can be prohibitive. We propose to identify as quickly as possible a significant fraction of the pages still alive. Further selection can then be made to identify a set of pages suitable for the study and then to crawl them. Such techniques would be especially interesting when testing the liveness of an item is much lighter than downloading it completely. This is for instance the case for the Web with HEAD queries compared to GET queries. If a large amount of work has been devoted to maintaining a fresh set of crawled pages, little attention has been paid to the coverage of a partial recrawling a fairly old snapshot.

Second, some graphs tend to be harder to crawl with time. For instance, Twitter has continuously restricted its capacity to be crawled. Performing a full scan was possible a few years ago [56], but it can be prohibitively long nowadays. New techniques must thus be developed for identifying efficiently active accounts in such settings.

Given an old snapshot, our goal is to identify a significant fraction of the items that are still alive or active now. The cost we incur is the number of fetches that are necessary to attain this a goal. A typical cost measure will be the average number of fetches per active item identified. The strategy for achieving this goal consists in producing an ordering for fetching the pages. We call *LiveRank* an ordering such that the items that are still alive tend to appear first. We consider the problem of finding an efficient LiveRank in three settings: static when it is computed solely from the snapshot and the link relations recorded at that time, sampling-based when a sampling is performed in a first phase allowing to adjust the ordering according to the liveness of sampled items, or finally dynamic when it is incrementally computed as pages are fetched.

We propose various LiveRank algorithms based on the graph structure of the snapshot. We evaluate them on two Web snapshots (from 10 to 20 millions nodes) and on a Twitter snapshot (40 million nodes). We propose several propositions based on the graph structure of the snapshot. A rather simple combination of a small sampling phase and the propagation of the partial activity information is obtained in the remaining graph of the snapshot through a modified PageRank algorithm. It allows to gather from 15% to 75% of the active nodes with a cost that remains within a factor of 2 from the optimal ideal solution.

6.2 Related work

The process of crawling the Web has been extensively studied. A survey is given by Olston and Najork [57]. Batch crawling where the process starts from a given set of pages and terminates at some point is classically opposed to incremental crawling where pages are continuously fetched. In incremental crawling, one of the main tuning is to balance the fetch of new and old pages: the former increase coverage while the latter increase freshness. Both types may allow to discover new links towards unknown new pages (old pages can change). Cho and Garcia-Molina have extensively studied the problem of incremental crawling. See for example [58] for one of the first formalization of freshness and a thorough study on refreshing policies. They show the counter-intuitive result that adapting the frequency of crawl proportionally to the frequency of change works poorly with respect to the overall freshness of the fetched copy of the Web. Variations of freshness have been introduced. For instance, information longevity [59] considers the evolution of fragments of the content of a page. Closer to our problem, Cho and Ntoulas [60] introduce the problem of using sampling to estimate the frequency of change per site and then to fetch a set of pages such that the overall change ratio of the set is maximized. Their technique consists in estimating the frequency of page change per site and to crawl first sites with high frequency change. Tan et al. [61] improve slightly over this technique by clusterizing the pages according to several features: not only their site (and other features read from the URL) but also content based features and linkage features (including pagerank and incoming degree). A change ratio per cluster is then estimated through sampling and clusters are downloaded in descending order

of the estimated values. More recently, Radinsky and Bennett [62] investigate a similar approach using learning techniques and avoiding the use of sampling. However, these approaches mainly focus on highly dynamic pages and use various information about pages whereas we are interested in stable pages and we use only the graph structure. With a slightly different objective, Dasgupta et al. [63] investigate how to discover new pages while minimizing the average number of fetches per new page found. Their work advocates for a greedy cover heuristic when a small fraction of the new pages has to be discovered quickly. On the opposite, they recommend a heuristic based on out-degrees which performs better for gathering a large fraction of the new pages. The methods are directed toward dynamicity again but the framework is closer to ours since we naturally use a similar cost of average number of fetches per interesting page found. To compare with such previous work, we could derive a method based on [60] that we call active-sites first: estimate through a sampling phase the fraction of active pages per web site and then crawl each site completely by decreasing order of liveness (sites with a higher proportion of alive pages first). This performs similarly to our techniques for gathering more than 80% of the overall active pages, but not for smaller fractions (less than 40%) as shown in Section 6.5.

On the other hand such recrawling policies have much less been studied for other sources of online data such as social networks. Indeed it is possible to similarly crawl the Twitter network by fetching information about user accounts that are linked by the follower-followee relations. However, crawling is much more restricted as all the data is possessed by a single company. This makes our approach even more relevant in such contexts where gathering a large amount can be extensively long.

Interestingly, Kwak et al. [15] show, among various observation, a correlation between number of followers and pagerank. On the other hand the activity of a user in number of tweets seems to be more correlated to his number of followees that his number of followers. First reported Twitter crawls include [64, 65, 15]. Recently, Gabiekov *et al.* [66, 56] have presented a preliminary study on a complete picture of Twitter social graph. The authors themselves claim that such extensive crawling could not be possible now anymore as Twitter has restricted its white list of IP authorized to query its API at high rate.

6.3 Model

Let $G = (V, E)$ be a graph obtained from a past crawl of a structured network. By structured network, we mean something like:

- A Web graph, V representing the crawled pages and E the hyperlinks. For i, j in V, (i, j) is in E if, and only if, there is an hyperlink to j in i. For Web graphs, edges are always directed.

- A social network, V representing the users and E social relationships between them. For social networks, edges can be undirected (symmetric relationships like friendship) or directed (asymmetric relationship like follower/followee).

Let n denote the size of V. At present time, only a subset of G is still active. The meaning of *active* depends of the context and needs to be defined: for instance, alive pages for Web graphs, or non-idle users for social networks, etc. We call a the function that tells if nodes are active or not: $a(X)$ denotes the active nodes from $X \subset V$, while $\bar{a}(X)$ stands for $X \setminus a(X)$. Let n_a be $|a(V)|$.

The problem we need to solve can be expressed as: how to crawl a maximum number of pages from $a(V)$ with a minimal crawling cost. In particular, one would like to avoid crawling too much pages from $\bar{a}(V)$. If a was known, the task would be easy, but testing the activity of a node obviously requires to crawl it. This is the rationale for the notion of LiveRank.

6.3.1 Performance metric

As any ordering can be seen as a LiveRank, we need some performance metrics to define good LiveRanks that succeed in ranking the pages from $a(V)$ first. Following [63], we define the LiveRank cost as the average number of node retrievals necessary to obtain an active node when the process is stopped as soon as $\alpha a(V)$ active nodes have been retrieved for a given desired fraction $0 < \alpha \leq 1$.

Formally, let $\mathcal{L}_i$ represent the i first pages returned by a LiveRank $\mathcal{L}$, and let $i(\mathcal{L}, \alpha)$ be the smallest integer such that $\frac{|a(\mathcal{L}_i)|}{n_a} \geq \alpha$. The refresh cost is then defined by:

$$\text{cost}(\mathcal{L}, \alpha) = \frac{i(\mathcal{L}, \alpha)}{\alpha n_a}.$$

Remarks on the cost function By construction, the cost function of a LiveRank is always at least 1. An ideal LiveRank $\mathcal{I}$ would perfectly separate $a(V)$ from rest of the nodes, so its cost function would simply be $C = 1$. In absence of some clairvoyant knowledge, $\mathcal{I}$ cannot be obtained before all nodes have been tested, which is exactly what we would like to avoid. The cost function allows to capture this dilemma.

Note that keeping a low cost becomes very difficult as α gets close to 1: without some oracle, being able to capture *almost* all active nodes is almost as difficult as capturing all actives nodes. For that reason, one expects that when α gets close to 1, the set of nodes any real LiveRank will need to crawl will tend to V, leading to an asymptotical cost $\frac{n}{n_a}$. This will be verified in Section 6.5.

Lastly, one may have noticed that the cost function uses $n_a = |a(V)|$ and therefore requires a full knowledge of the activity. This is not an issue as the proposed cost is an external metric used to evaluate the LiveRanks.

We now present the different LiveRanks that we will consider in this chapter. We broadly classify them in three classes: static, sample-based and dynamic.

6.3.2 Static LiveRanks

Static LiveRanks are computed offline using solely the information from G. That makes them very basic, but also very easy to be used in a distributed way: given p crawlers of similar capacities, if $\mathcal{L} = (l_1, \ldots, l_n)$, simply assign the task of testing node l_i to crawler $i \mod p$.

Random permutation (R) is proposed here to serve both as a reference and as a building block for more advanced LiveRanks. R completely ignores any information from G, so its cost should be in average $\frac{n}{|a(V)|}$, with a variance that tends to 0 as α tends to 1. We expect good LiveRanks to have a cost function significantly lower than cost(R).

Decreasing Indegree ordering (I) is a simple LiveRank that we expect to behave better than a random permutation. The intuition is that older nodes should have more incoming edges (in terms of correlation), so high degree nodes should already be older at the time G was crawled. In web graphs and social networks, old nodes may last longer than younger ones. Sorting by degree is the easiest way to exploit that correlation.

PageRank ordering (P) pushes forward the *indegree* idea. The intuition is that pages from the snapshot that are still active are likely to point toward pages that are still alive also, even considering only the old links. This suggests to use a PageRank-like importance ranking [26]. A PageRank usually needs two parameters: d, a damping factor, and X, a zap distribution on V (see the Appendix for details). In absence of further knowledge, we propose to choose $d = .85$ (typical value for Web graphs) and X uniform on V.

Note that it is very subjective to evaluate PageRank as an importance ranking, as importance should be ultimately validated by humans. On the other hand, the quality of PageRank as a static LiveRank is straightforward to verify, for instance using our cost metric.

The validity of the assumptions we made for justifying the choice of I and P (existence of correlations) will be verified in Section 6.4.3.

6.3.3 Sample-based LiveRanks

Using a LiveRank consists in crawling V in the prescribed order. During the crawl, the activity function a becomes partly available, and it is natural to reinject the obtained information to produce a new LiveRank.

To keep things simple, we first consider in this chapter a two-steps sample-based approach: we first fix a testing threshold z and test z items following a static LiveRank (like R, I or P). The set Z of nodes tested is called indeffirently the sample set or the training set. We thus obtain for Z the knowledge of $a(Z)$ and $\bar{a}(Z)$, which allows us to to recompute the LiveRank of the remaining untested pages.

Because the sampling uses a static LiveRank, and the adjusted new LiveRank is static as well, sample-based LiveRanks are still easy to use in a distributed way as the crawlers only need to receive crawl instructions on two occasions.

Notice that in the case where the sampling LiveRank is a random permutation, $|a(Z)|\frac{n}{z}$ can be used as an estimate for n_a. This can for instance be used to decide when to stop crawling if we desire to identify αn_a active nodes in $a(V)$.

Simple adaptive LiveRank P_a When a page is alive, we can assume it increases the chance that pages it points to in G are also alive, and that life is transmitted somehow through hyperlinks. Following this idea, a possible adaptive LiveRank

consists in taking for X the uniform distribution on $a(Z)$. This diffusion from such an initial set can be seen as a kind of breadth-first traversal starting from $a(Z)$, but with PageRank flavour, by weighting according to the structure of G. Readers may refer to the Appendix for a more detailed description of adaptive LiveRank computation.

Active-site first LiveRank ASF (from [60]) To have a point of comparison with previsous work, we propose the following variant of the Dasgupta et al. [60] strategy for finding pages that have changed in a recrawl. Their algorithm is based on sampling for estimating page change rate for each web site and then to crawl sites by decreasing change rate. Note that it is appropriate when a site structure can be read from the node identifiers. This is the case while inspecting the URLs of a Web snapshot, but no such structure exists in Twitter. Active-site first (ASF) consists in partitioning Z according to web sites. We thus obtain a collection $Z_1, \ldots, Z_p$ of sets. For each set Z_j corresponding to some site i, we obtain an estimation $|a(Z_j)|/|Z_j|$ of its activity (i.e. the fraction of active pages in the site). We then sort the remaining URLs by decreasing site activity.

Double adaptive LiveRank $P_a^{+/-}$ The simple adaptive LiveRank does not use the information given by $\bar{a}(Z)$. One way to do this is to calculate an "anti"-PageRank based on $\bar{a}(Z)$ instead of $a(Z)$. This ranking would represents a kind of diffusion of death, the underlying hypothesis being that dead pages may point to pages that tend to be dead. As a result, we obtain a new LiveRank by combining these two PageRanks. After having tested several possible combinations not discussed in this chapter, we empirically chose to weight each node by the ratio of the two sample-based PageRank, after having set all null entries of the anti-PageRank equal to the minimal non-null entry.

6.3.4 Dynamic LiveRanks

Instead of using the acquired information just one time after the sampling, Dynamic LiveRanks are continuously computed and updated on the fly along the entire crawling process. On the one hand, this gives them real-time knowledge of a, but on the other hand, as the dynamic LiveRank may evolve all the time, they can create synchronization issues when used by distributed crawlers.

Status	Description	Number of pages	Percentage
Code HTTP 404	Page not found	6 467 219	34,92%
No answer	Host not found	4 470 845	24,14%
Code HTTP 301	Redirection	3 455 923	18,66%
Target 301	Target of redirection	20 414	0,11%
Code HTTP 200	Page exists	2 365 201	12,77%
True 200	**Page really exists**	1 164 998	**6,29%**
Others (403,...)	Other error	1 761 298	9,51%
Total	Graph size	18 520 486	100%

Table 6.1: Status of web pages in `uk-2002`, crawled in December 2013.

Like for sample-based LiveRanks, dynamics LiveRank use a training set Z of z pages from a static LiveRank. This allows to bootstrap the adjustment by giving a non-empty knowledge of a, and prevents the LiveRank from focusing on only a small subset of V.

Breadth-First Search (BFS) With BFS, one may reconsider the diffusion model in adaptive LiveRanks at one-hop distance. A BFS queue is initialized with the (uncrawled) training set Z. The next node to be crawled is popped from the queue following First-In-First-Out (FIFO) rule. If the selected node appears to be alive, all of its uncrawled outgoing neighbours are pushed into the end of the queue. When the queue is empty, we always pick the unvisited node with highest PageRank[1].

Alive indegree (AI) The BFS does not give any priority to the popping order from queue except FIFO. We now propose AI which provides a more reasonable node selection scheme. For AI, each node in the graph is associated with a *live score* value indicating how many alive nodes point to it. These values are set to zeros at the beginning and always kept up-to-date. AI is initialized by testing Z: each node in $a(Z)$ will increment the associated values of its out-going neighbours by one. After Z is tested, the next node to be crawled is simply the one with highest live score (in case of equality, to keep things consistent, we pick the node with highest PageRank). Whenever a new alive node is found, we update the live scores of its untested neighbours.

[1]We tested several other natural options and observed no significant impact.

6.4 Datasets

We chose to evaluate the proposed LiveRanks on existing datasets of the Web and Twitter available on the WebGraph platform[2]. In this Section, we present the datasets, describe how we obtained the activity function a and observe the correlations between a, indegree and PageRank.

6.4.1 Webgraph Dataset

We focused on snapshots of the British domain `.uk`.

uk-2002 dataset The main dataset we will use is the web graph `uk-2002`[3] crawled by UbiCrawler [38]. This snapshot was crawled in 2002. It contains 18,520,486 pages and 298,113,762 hyperlinks.

The preliminary task is to grab statistically current states of all web pages within the graph to determine a. For each URL of the snapshot, we have performed a GET request and hopefully obtained a corresponding HTTP code. The results are summarized in Table 6.1. Our main findings are:

- One third of the total pages are no longer available today, the server returns error `404`.
- One fourth have DNS problem (which probably means the web site is also dead).
- For one fifth of the cases, the server sends back the redirection message `301`. Most redirections for pages of an old site lead to the root of a new site. If we look at the proportion of distinct pages alive at the end of redirections, it is as low as 0.1%.
- Less than 13% of pages return the code `200` (success), and we found that half of them display some text mentioning that the page was not found. To handle this issue, we have fully crawled all the pages with code `200` and filtered out pages whose title or content have either `Page Not Found` or `Error 404`. Finally, the alive set contains 1,164,998 pages, accounting for 6.4% of nodes of the entire graph.

[2] http://webgraph.di.unimi.it/
[3] http://law.di.unimi.it/webdata/uk-2002/

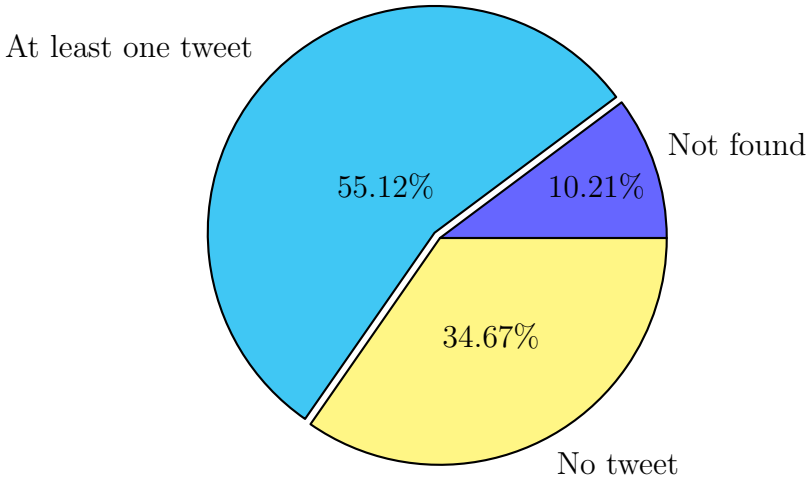


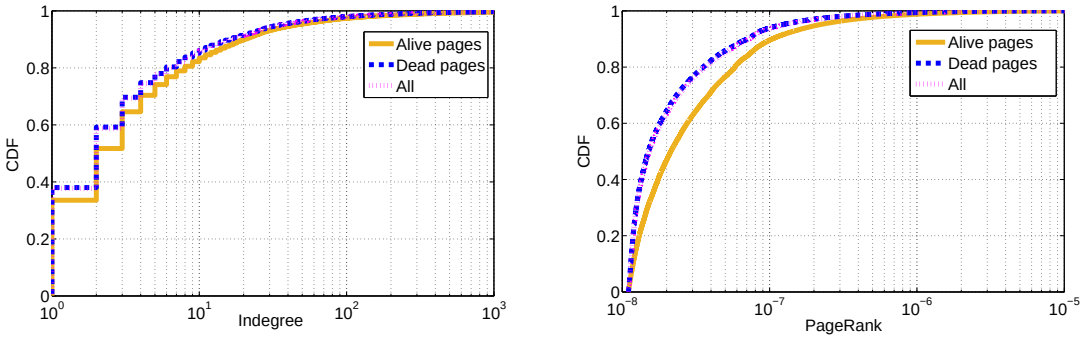
Figure 6.1: Statistics of the `twitter-2010` dataset

uk-2006 dataset We chose `uk-2002` because of its adversarial settings (old snapshot with relatively few alive pages), but it is also important to evaluate the impact of LiveRanks on shorter time scales. In absence of fresh enough available datasets, we used the DELIS dataset [49], a series of twelve continuous snapshots[4] starting from 06/2006 to 05/2007 (one-month intervals). We set G to the first snapshot (06/2006). It contains 31,316,403 nodes and 813,807,972 hyperlinks. We then used the last snapshot (05/2007), and considered the intersection between the two snapshots to be the active set $a(V)$. With this methodology, we hope to have a good approximation for a one-year period. As a result, we obtained $n_a = 11,142,177$ "alive" nodes representing 35.56% of the graph.

6.4.2 Twitter Dataset

Lastly, we used the dataset `twitter-2010`[5] first introduced in [15]. The graph contains roughly 42 millions Twitter user accounts and 1.5 billions follower-followee relationships among them. Arcs in the graph are directed from follower to followee: there is an arc from node x to y if user x follows y. This follows the PageRank intuition: we consider that a user is important when she is followed by important users. (Notice that tweets traverse arcs in the reverse direction.)

[4] `http://law.di.unimi.it/webdata/uk-union-2006-06-2007-05/`
[5] `http://law.di.unimi.it/webdata/twitter-2010/`

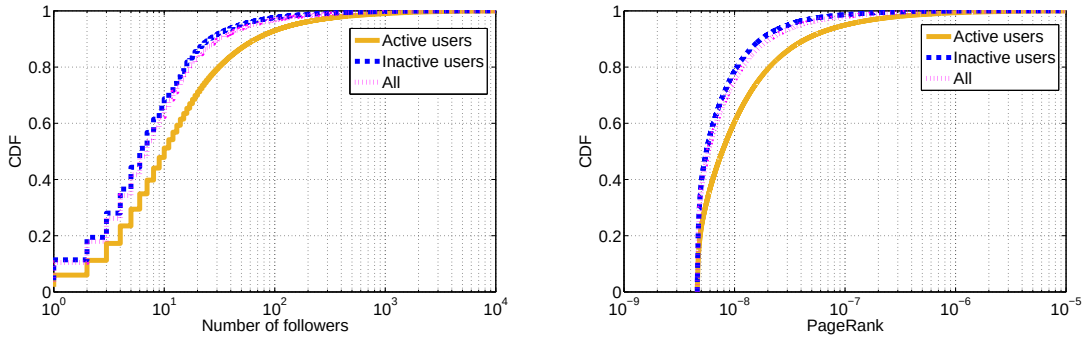


(a) uk-2002: CDF Indegree (b) uk-2002: CDF PageRank

Figure 6.2: Cumulative distribution of nodes according to Indegree/PageRank.

We consider a user as active if he has posted a tweet recently. For that purpose, we can query the Twitter interface to recover the timestamp of the last-tweet of the user associated with a given identifier. Recovering the timestamps of all 41 millions users using Twitter API [67] would be extremely slow: when we made our measurements (05/2014), an authorized Twitter account was limited to 350 API requests/hour so querying all the accounts would have taken 13 years. While this is one of the main reasons for designing good LiveRank, we still need a full crawl to build a ground truth. To overcome this obstacle, we cheated and used a browser-like crawler to recover each user timeline as if a regular browser was connecting to Twitter front servers. This is possible because the timestamp of the last entry can easily be recovered inside the HTML structure of the returned documents. However, such an approach becomes much more difficult for complex queries and might also be detected and prevented by Twitter in the future.

Having tested all nodes, we found three main categories of users corresponding to those who (i) no longer exist, (ii) have no tweet at all and (iii) have tweeted at least once before the crawling time. Figure 6.1 shows the relative proportion of each category. Subsequently for those who have ever tweeted, we crawl and extract the timestamp of their last tweet. Finally, after considering the cumulative distribution of last-tweet timestamps we arbitrarily decided to fix a six months limit. If we consider a user is active if she has tweeted during the last six months, we then obtained a list of 7,300,399 (17.53%) active users, serving as the ground truth for benchmarking purposes.



(a) `twitter-2010`: CDF Indegree (b) `twitter-2010`: CDF PageRank

Figure 6.3: Cumulative distribution of nodes according to Indegree/PageRank.

6.4.3 Correlations

We proposed the LiveRanks I and P on the assumption that the activity of nodes is correlated to the graph structure of the snapshot, so that a node with high in-degree or PageRank has more chances to be active in the future.

To validate this, we plot in Figure 6.2, 6.3 the cumulative distribution of active nodes compared to others according to their in-degree (6.2a and 6.3a) and PageRank value (6.2b and 6.3b) respectively for both datasets `uk-2002` and `twitter-2010`. We can observe that the curve for active nodes is slightly shifted to the right compared to the other curves in each figures: active users tend to have slightly higher in-degree and PageRank than in the overall population. The bias seems relatively small, but we will measure now how this bias impacts the cost function of corresponding LiveRanks.

6.5 LiveRanks evaluation

After having proposed several LiveRanks in Section 6.3 and described our datasets in previous Section, we can now benchmark our proposals.

Regarding the LiveRanks, we focus on two main aspects: (i) efficiency of different LiveRanks classes and (ii) impact of tuning parameters, especially the choice of the number z of pages sampled in sample-based and dynamic settings.

As for the datasets, we evaluate how the performance of LiveRanks varies according to the type of graph structure (Web vs Twitter) and the age of the old

crawl. For instance, Web sites clustering is characteristic of Web graphs: a majority of the links of a Web graph are actually intra-site navigational links, and when a web server or a DNS entry dies, so do all related pages. This is not the case of social graph in which users are independent from each other, and we aim to investigate whether these qualitative differences have an impact on the cost of the LiveRanks.

All our evaluations are based on representations of the cost functions. In each plot, the x-axis indicates the fraction α of active nodes found at some point during the crawl and the y-axis corresponds to the relative cost of the crawl up to that point. A low curve indicates an efficient LiveRank. An ideal LiveRank would recover all active nodes without probing any other useless node, achieving a constant cost of 1. The ideal curve is thus horizontal from $(0, 1)$ to $(1, 1)$. On the other hand, a random LiveRank is quickly constant with an average cost n/n_a. Note that any non-clairvoyant LiveRank will terminate at point $(1, n/n_a)$: in practice, the exact number of active nodes is unknown during the crawl (but well estimated by random sampling as shows the random ordering curve). In practice, We thus cannot now for sure that we have gather all active nodes until we have probed all of them. This is the reason why all practical LiveRank curves begin to increase significantly when approaching 1.

When it is not specified, the training set contains the $z = 100000$ pages of higher (static) PageRank.

6.5.1 Web graph dataset

We gather in five Figures (Figure 6.5, 6.6, 6.7, 6.8 and 6.9) several LiveRank evaluations on the web graph `uk-2002` to serve as reference.

Figure 6.5 compares the static and sample-based LiveRanks. For static LiveRanks, we see that indegree ordering (I) and PageRank (P) significantly outperform random ordering, PageRank being the best of the three: it is twice more efficient than random for small α, and still performs approximately 30% better when up to $\alpha = 0.6$. Second, we can get even much better costs with sample-based approaches. (By default, the sampling set is made of the top z nodes according to PageRank.) The two adaptive LiveRanks P_a allow to improve ordering by a factor of 6 approximately around $\alpha = 0.2$ with a cost of 2.5 fetches per active node found. Notice

that their costs remains steadily low on the range $[0.1, 0.4]$ and even further for the double adaptive version $P_a^{+/-}$.

Figure 6.6 shows the impact of the size z of the sampling set. All curves correspond to the double adaptive LiveRank $P_a^{+/-}$ with varying z. Similar results are obtained with the simple-adaptive version P_a. As the sampling set grows larger, we spend more effort on testing it at the beginning but it results in a significant increment of efficiency in the long run. For this dataset, taking a big training set (z=500 000) allows to reduce the cost of the crawl for $\alpha \geq 0.6$.

Another key aspect of the sampling phase is the choice of the sample set. We can observe in Figure 6.7 that the performance of double adaptive $P_a^{+/-}$ is further improved by using a random sample set rather than selecting it according to the PageRank or by decreasing indegree. The reason is that a random sample avoids a locality effect in the sampling set as high PageRank pages tend to concentrate in some local parts of the graph. Note that double adaptive LiveRank through random sampling is within a factor of 2 from optimal for a large range of α values.

We then compare sample-based approaches to fully dynamic strategies. We see in Figure 6.8 that bread-first search BFS and alive indegree AI perform similarly to double adaptive $P_a^{+/-}$ for low α and can outperform it for large α (especially BFS). BFS begin to significantly outperform double adaptive for $\alpha \geq 0.5$. However, if one needs to gather half of the active pages or less, double adaptive is still the best candidate as it is much simpler to operate, especially with a distributed crawler.

Finally, Figure 6.9 then shows the impact of different sampling sets on BFS and AI. Except for high values of α where a random sampling outperforms other strategies, the type of sampling does not seem to affect the two dynamic LiveRanks as much as for the sample-based approaches.

Additionally, we have repeated the same experiments on the dataset `uk-2006`, where the update interval is only one year. As Figure 6.11 shows, the results are qualitatively quite similar, the main difference being better costs due to a higher proportion of alive pages (less than 1.4 for double adaptive, against 2.8 for the random ordering).

To compare with techniques from previous works for finding web pages that have been updated after a crawl, Figure 6.4 compares double adaptive $P_a^{+/-}$ to active-site first ASF with random sampling (the same number of random pages is tested

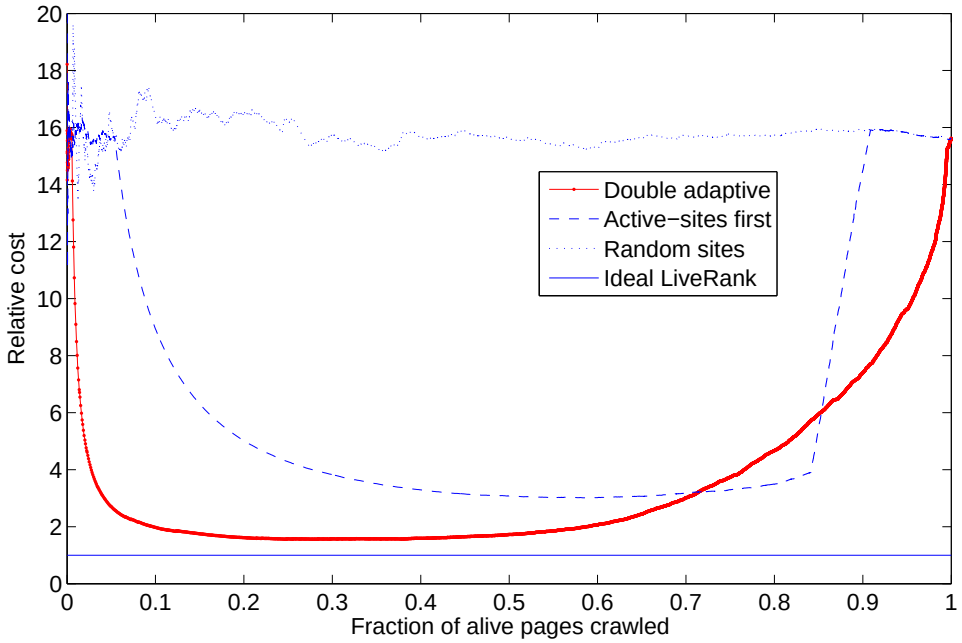


Figure 6.4: Comparison with the cost of an active-site first LiveRank

in each site and the overall number of tests is the same as with double adaptive). We see that crawling according to site activity performs well on the long run with similar performance as double adaptive. However for a smaller fraction of desired coverage (less than 40%), this technique is clearly outperformed by the other (by a factor of 2 for 20% and 4 for 10%).

6.5.2 Twitter graph

As we discussed earlier, the Twitter graph has structural properties distinct from Web graphs. In this part we analyze how these differences may change the performance of LiveRanks. We conduct the same set of experiments on `twitter-2010` as what have done in `uk-2002`.

Figure 6.10a compares the static and sample-based LiveRanks. A first observation is that the double adaptive LiveRank $P_a^{+/-}$ performs very poorly compared to the other LiveRanks, including Indegree I. It indicates that if the intuition of some death propagation was relevant for Web graphs (it was a convenient way to spot dead web sites for instance), this is not the case for Twitter: the fact that followers

become inactive does not seem to have an impact on the activity of the followees. In the end, the simple adaptive LiveRank P_a has the best performance, closely followed by the static LiveRanks P and I. The three of them have a cost function that seem to grow roughly linearly between 2 and 4 as α goes from 0 to 0.6.

In Figure 6.10b, we vary the size of the training set, ranging from z =200 000 to z =1000 000. Results indicate that the cost function is almost not affected by z as long as it is high enough. Compared to the results observed on Web graphs, this means that taking a big training set: (i) will not burden the cost function for small α. This likely comes from the fact that the sampling set is PageRank-based by default, and the static PageRank is already close to the best LiveRank we obtain; (ii) will not improve the performance for large α either, meaning that no significantly useful knowledge is obtained after some threshold. This relative independence with respect to z is another qualitative difference compared to Web graphs,

Figure 6.10c shows the impact of training set types on simple adaptive LiveRank P_a. Unlike Web graphs where random sampling dominates others, in social network the training set filled by PageRank is the best whereas the random seed is worse. This can be interpreted as a result of a weaker structural locality (*i.e.,* no highly correlated clusters like web sites for Web graphs), so that activeness is more concentrated around important Twitter individual users that should be considered as soon as possible.

In Figure 6.10d, we compare the simple adaptive PageRank P_a with the dynamic LiveRanks. All of them are initialized with default values (PageRank sampling of size z =100 000). P_a stays the best option: it is slightly better than AI and much more efficient than BFS. While for Web graphs, dynamics LiveRanks could still be preferred for some settings, it seems that in the context of Twitter it is never the case especially considering their deploiement complexity in a distributed crawler.

Lastly, Figure 6.10e indicates the impact of different training sets on the two dynamic LiveRanks. It confirms that the combination of AI and a PageRank-ordered training set gives the best results for that type of LiveRanks, which is still not enough to compete against P_a.

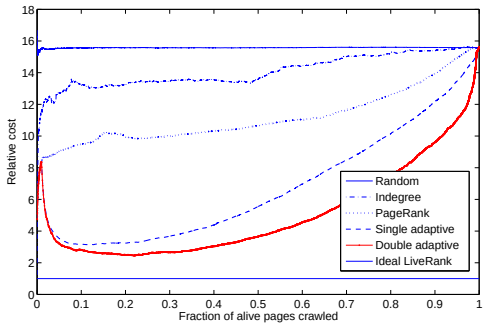


Figure 6.5: Static and sample-based LiveRanks (z=100 000)

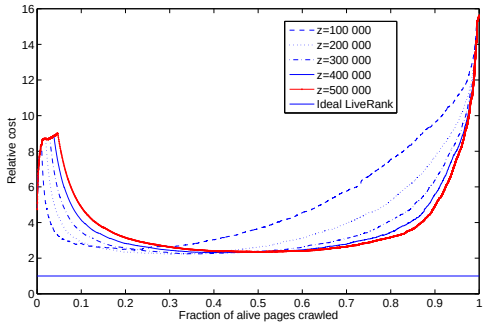


Figure 6.6: Impact of z (Double adaptive LiveRank)

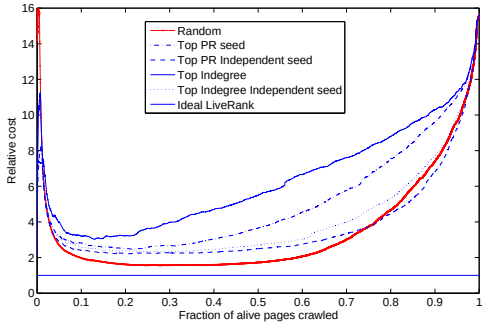


Figure 6.7: Impact of Z (double adaptive with z=100 000)

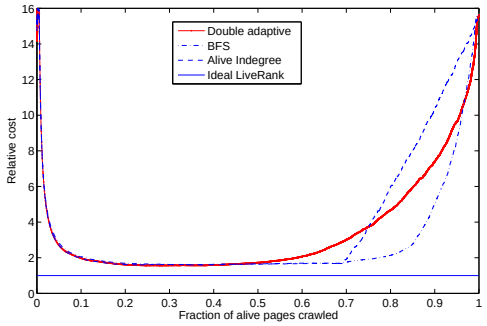


Figure 6.8: Sample-based and dynamic LiveRanks (z=100 000)

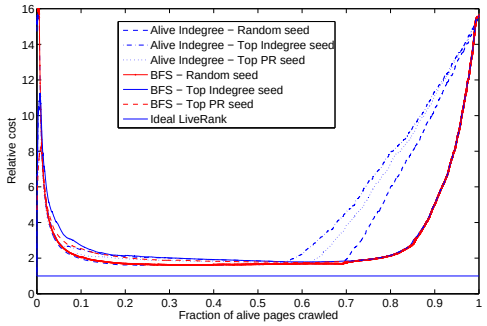


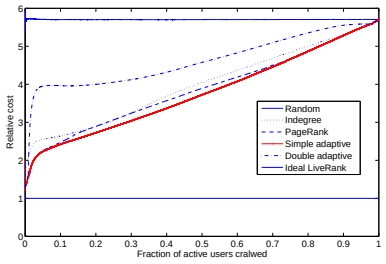
Figure 6.9: Impact of Z on dynamic LiveRanks (z=100 000)

6.6 Conclusion

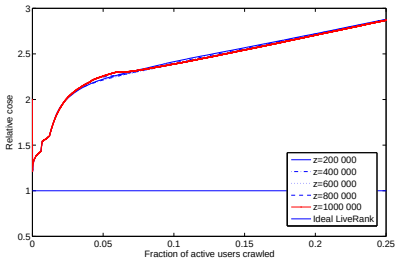
In this chapter we have investigated how to efficiently retrieve large portions of active nodes from an old crawl using orderings we called LiveRanks.

For Web graphs, we observed that PageRank is a good static LiveRank. In a 12 years old crawl, it outperforms a random rank by a factor approximately 1.5 for gathering half of the alive pages with a cost of 10 fetches per alive page. However, we get a significant gain by first testing a small fraction of the pages to adjust the PageRank in a sample-based approach. We then get a cost as low as two fetches per alive page.

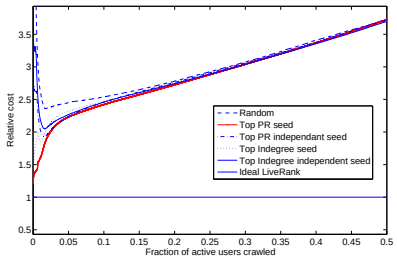
We get somewhat similar results on a Twitter dataset. The main difference is that costs tend to increase linearly with the fraction of alive items recovered with Twitter whereas the cost is rather stable with Web in the range 15%-75%. On this range, the cost increases roughly from 2 to 4 fetches per alive pages for Twitter graph when it remains around 2 for the Web.



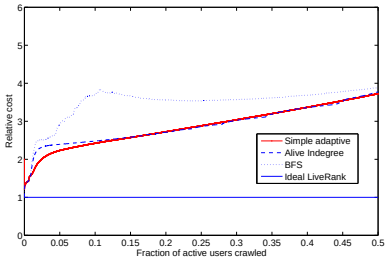
(a) `twitter-2010`: Static and sample-based LiveRanks (z=100 000)



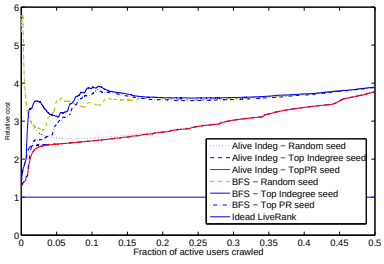
(b) `twitter-2010`: Impact of z (Double adaptive)



(c) `twitter-2010`: Simple adaptive with different training sets (z=100 000)

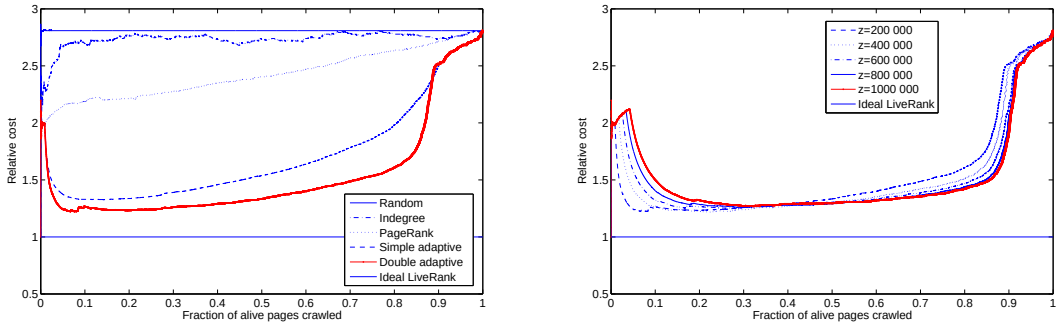


(d) `twitter-2010`: Comparison between sample-based and dynamic LiveRanks (z=100 000)



(e) `twitter-2010`: Impact of training set on dynamic LiveRanks (z=100 000)

Figure 6.10: `twitter-2010` evaluation results



(a) Static and sample-based LiveRanks, (z=200 000)

(b) Impact of z (Double adaptive LiveRank)

Figure 6.11: `uk-2006` main evaluation results

Compared to previous works on identifying modified pages, our technique performs similarly for large desired fraction (around 80%) when compared to the LiveRank algorithm inspired by the technique in [60] that could be adapted to our setting in the Web case. However, for a small fraction (less than 40%) our method outperforms this technique. Interesting future work could reside in using our techniques for the identification of pages that have changed and compare with such techniques in the domain they were conceived for.

Interestingly, we could not get significant gain when using fully dynamic LiveRank. As noted before, each of the two phases of the sample-based approach can be easily parallelized through multiple crawlers whereas this would be much more difficult with a fully dynamic approach. The sample-based method could for example be implemented with in two rounds of a simple map-reduce program whereas the dynamic approach requires continuous exchanges of messages between the crawlers.

Chapter 7

Summary and future work

The rapid growth of World Wide Web and social networks has created a need of having good ranking systems. Although there has been a lot of research efforts and results on this topic, it still has many issues that need to be considered, like ground truth definition or large scale computation, etc.

In this thesis, we investigate the D-iteration, an algorithm proposed by Dohy Hong that can compute the PageRank vector, based on diffusion approach. We summarize the theoretical results concerning the correctness (convergence), the precision measurement and update equation. Our algorithm shows its potential through experiments on real data in comparison with Jacobi, Gauss-Seidel, SOR, OPIC, GMRES, etc.

Based on the properties of DI, there are some interesting issues to be discussed. DI is capable of adapting to asynchronous computation. Recalling that some classical methods are restricted by how nodes are iterated. For instance, Gauss-Seidel updates its vector at element level: it applies right away the vector elements x_k^j to compute x_k^i for $j < i$ in the k^{th} iteration and this procedure impedes the asynchronous deployment. On the contrary, DI has almost no constraint on diffusion sequence except the fairness which is easy to meet. However, DI still requires some further work to be fully efficient in a distributed setting: controlling the fluid spread on several machines in distributed computation is not easy because the issues involve in graph partitioning process (i.e., assigning certain groups of nodes to one or several machines), diffusion strategy and graph storage scheme.

On the other hand, with DI one can control the precision of the approximate PageRank vector while it is impossible with other algorithms. This advantage makes sense in the case where different applications do not demand the same approximation accuracy. For example in many recommendation systems, we are only interested in the top k items instead of the entire ranking vector, and the top k items can probably be detected after a few tens of iterations (see example of Twitter ranking in Figure 5.6 of Section 5.3). Thanks to this observation, one can formulate the number of necessary iterations to find the top k of total n items with a precision p as a function $f(k, n, p)$. This value can be used as the stopping condition of DI.

As for graph dynamics, we only considered in this thesis the evolution of the graph in terms of links modification. In fact, a complete graph update comprises of node/link additions/removals. This will bring a huge impact on the update strategy, for example on how to modify the fluid while having new node arriving. Obviously, the problem is even more difficult to tackle when it is not clear how the graph evolves.

For further developing the existing DI algorithm, we basically plan to focus on how to adapt and implement DI in a distributed manner. Related problems (graph partitioning,...) of course would be also the subject of future investigations.

An approximation of the PageRank vector of a partially hidden graph was discussed as well. In reality, the graph is not always fully accessible (specially in case of distributed computation) so that we have to approximate it based on the visible part. The strategy of how to unveil the graph becomes crucial. It has been shown that if one chooses random nodes and high-rank nodes alternatively, the PageRank vector can be approximated with a precision gain factor of 10 compared to the case where only one of them is chosen infinitely. A potential of future work on this topic is building a dynamic scheduler that, based on the perspective of the current graph observed it can tell what is the good node to request next.

An application of PageRank to Twitter social network was carried out to rank Twitter users. The results showed that there is not a strong correlation between PageRank and indegree ranking, i.e., PageRank provides additional information that is not contained in the indegree, which is already interesting in itself. However, the two rankings may be used in slightly different applications. Let us take an example of Twitter. If one wants to optimize the information propagation rate (number of users receiving information) in a short time step, the indegree ranking should be

more interesting than the PageRank which is probably optimized for a long-term propagation.

We also proposed LiveRank, a ranking order that helps to efficiently retrieve large portions of active nodes from an old crawl. Our work establishes the possibility of efficiently recovering a significant portion of the active part of an old snapshot and advocates for an adaptive PageRank with sampling for obtaining an efficient LiveRank. Our work establishes the possibility of efficiently recovering a significant portion of the alive pages of an old snapshot and advocates for the use of an adaptive sample-based PageRank for obtaining an efficient LiveRank.

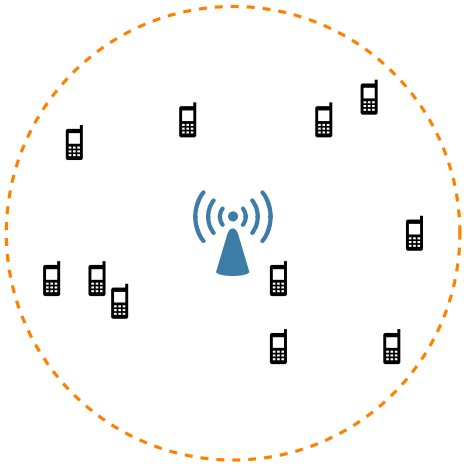


Figure 7.1: Positioning system with 1 RP and many MTs.

Additionally, in Appendix D we proposed a new method called Two-Step Movement (2SM) to estimate the position of MT. It requires only one reference point (RP) by exploiting useful information given by the position change of the MT or user movement. More precisely, the method consists of two basic ideas. The first one is that by using the motion of the device, we can determine its position through one and only one reference point (RP), instead of many (e.g. at least 3 in GPS-like solution). We allow the minimal number of required reference device. The cost of a reference system (e.g. using WiFi access points, RF base stations, Bluetooth beaconing devices) increases proportional to the density of the reference devices. We keep it minimal. The second idea is that by using the geometry through our new algorithm (in solving a couple of quadratic equations), we can determine the position of the device even when the reference is a moving object (surely includes the special case when the reference is static or almost static that is todays common assumption).

The algorithm has four main advantages: (i) implementable to today's smart devices (e.g. smart phones) and wireless networks, (ii) the requirement of infrastructure is low, database (like WiFi fingerprint) is not required, low computation complexity, (iii) it is not limited by the number of devices in use (scalable) and finally (iv) it offers much better performance, which in comparison to iBeacon [1] (with average error = 3 meters), our solution can reduce the positioning error from 2 times (error is reduced to 1 meter) to 10 times (error is reduced to 0.3 meter) in various cases.

It is not obvious to perceive the relation between ranking and localization systems. As an effort to bridge them for future works, we may think of a positioning system using one RP and many MTs in its coverage area shown in Figure 7.1. Suppose that thanks to the 2SM, all MTs can estimate their positions but with different errors, i.e., some of them are localized more precisely than others. Now, the question is how to select one (or a few) MT(s) and to use it as a semi-RP to reduce the estimation error of the other MTs. In such a context, a ranking algorithm managed by the true RP may help. Based on the error observed while solving the quadratic equations of each MT, the RP can assign a reliability degree for each MT and decide which one (probably the highest) can be used as the semi-RP. Hopefully, with the additional information given by the semi-RP (plus the original RP), the position estimation process can be improved.

[1] https://support.apple.com/en-us/HT202880

Appendix A

GMRES: Residual Minimization over Krylov subspace

To provide a complementary information about GMRES introduced in Section 2.2.4, this appendix will give the pseudo-code of the algorithm and an numerical example to show how it works step-by-step.

The pseudo-code of GMRES is shown in Algorithm 5.

Algorithm 5 GMRES algorithm: $Ax = b$

1: Compute $r_0 = b - Ax_0, \beta = \|r_0\|_2$, and $v_1 = r_0/\beta$ ▷ Start with initial guess vector x_0
2: Define the $(m+1) \times m$ matrix $\overline{H}_m = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$. Set $\overline{H}_m = 0$. ▷ Initialize Hessenberg matrix
3: **for** $j = 1 : m$ **do**
4: Compute $w_j = Av_j$
5: **for** $i = 1 : j$ **do**
6: $h_{ij} = (w_j, v_i)$ ▷ Product of w_j and v_i
7: $w_j = w_j - h_{ij}v_i$
8: **end for**
9: $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m = j$ and go to 12.
10: $v_{j+1} = w_j/h_{j+1,j}$
11: **end for**
12: Compute y_m the minimizer of $\|\beta e_1 - \overline{H}_m y\|_2$ and $x_m = x_0 + V_m y_m$. ▷ e_1 is the 1^{st} column of an identity matrix

Given a graph in Figure A.1, we construct the corresponding transition matrix P, then choose the damping d and the zap vector Z as follows

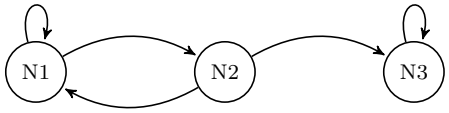


Figure A.1: An example graph.

$$P = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 1 \end{pmatrix}, \qquad d = 0.25, \qquad Z = \begin{pmatrix} 0.5 \\ 0.5 \\ 0 \end{pmatrix}$$

We will find the PageRank vector x of the equation $x = dPx + (1-d)Z$ using GMRES. Thus we need to transform the equation to the form $Ax = b$ such that $A = I - dP$ (I is an identity matrix) and $b = (1-d)Z$. Starting with the initial x_0 as follows

$$A = \begin{pmatrix} 0.75 & -0.25 & 0 \\ -0.25 & 1 & 0 \\ 0 & -0.25 & 0.5 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.25 \\ 0.25 \\ 0 \end{pmatrix}, \qquad x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

we compute r_0:

$$r_0 = b - Ax_0 = \begin{pmatrix} 0.25 \\ 0.25 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.75 & -0.25 & 0 \\ -0.25 & 1 & 0 \\ 0 & -0.25 & 0.5 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.25 \\ 0 \end{pmatrix}.$$

so that $\beta = \|r_0\|_2 = 0.3536$ and $v_1 = \dfrac{r_0}{\beta} = \begin{pmatrix} 0.7071 \\ 0.7071 \\ 0 \end{pmatrix}$ and $\|v_1\|_2 = 1$.

1^{st} iteration:

- First, we compute w_1:

$$w_1 = Av_1 = \begin{pmatrix} 0.3536 \\ 0.5303 \\ -0.1768 \end{pmatrix} \Rightarrow h_{11} = w_1 \cdot v_1 = 0.625.$$

$$w_1 = w_1 - h_{11}v_1 = \begin{pmatrix} -0.0884 \\ 0.0884 \\ -0.1768 \end{pmatrix} \Rightarrow h_{21} = \|w_1\|_2 = 0.2165.$$

• Then v_2:

$$v_2 = \frac{w_1}{h_{21}} = \begin{pmatrix} -0.4082 \\ 0.4082 \\ -0.8165 \end{pmatrix} \Rightarrow \|v_2\|_2 = 1.$$

• We obtain the two matrices $\overline{H}_1$ and V_1:

$$\overline{H}_1 = \begin{pmatrix} 0.625 \\ 0.2165 \end{pmatrix};$$

$$V_1 = (v_1) = \begin{pmatrix} 0.7071 \\ 0.7071 \\ 0 \end{pmatrix};$$

The next step is to compute y_1 which minimizes $\|\beta e_1 - \overline{H}_1 y\|_2$. We apply QR factorization using GramSchmidt process such that the Hessenberg matrix $\overline{H_i}$ is decomposed as follows:

$$\overline{H_1} = Q_1 R_1.$$

where Q_1 is an orthogonal matrix and R_1 is an upper triangular matrix. In the first iteration, the process results in two matrices Q_1 and R_1:

$$\overline{H}_1 = \underbrace{\begin{pmatrix} -0.9449 & -0.3273 \\ -0.3273 & 0.9449 \end{pmatrix}}_{Q_1} \underbrace{\begin{pmatrix} -0.6614 \\ 0 \end{pmatrix}}_{R_1}.$$

To solve the least square problem $\|\beta e_1 - \overline{H}_1 y\|_2$, we rewrite the equation:

$$\begin{aligned} & \overline{H}_1 y = \beta e_1 \\ \Leftrightarrow \quad & \overline{H}_1^T \overline{H}_1 y = \overline{H}_1^T \beta e_1 \\ \Leftrightarrow \quad & R_1^T Q_1^T Q_1 R_1 y = R_1^T Q_1^T \beta e_1 \\ \Leftrightarrow \quad & R_1^T R_1 y = R_1^T Q_1^T \beta e_1 \quad (Q_1 \text{ is orthogonal}) \\ \Leftrightarrow \quad & R_1 y = Q_1^T \beta e_1 \end{aligned}$$

and thus the solution $y_1 = R_1 \backslash Q_1^T \beta e_1$ where the operator '\' is the left division. We have:

$$y_1 = \begin{pmatrix} -0.6614 \\ 0 \end{pmatrix} \backslash \left[\begin{pmatrix} -0.9449 & -0.3273 \\ -0.3273 & 0.9449 \end{pmatrix} \begin{pmatrix} 0.3536 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} 0.5051 \end{pmatrix}$$

and the result x_1 of the first iteration is:

$$x_1 = x_0 + V_1 y_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.7071 \\ 0.7071 \\ 0 \end{pmatrix} \times \begin{pmatrix} 0.5051 \end{pmatrix} = \begin{pmatrix} 0.3572 \\ 0.3572 \\ 0 \end{pmatrix}.$$

and the precision $r_1 = b - Ax_1 = \begin{pmatrix} 0.0714 \\ -00.179 \\ 0.0893 \end{pmatrix} \Rightarrow \|r_1\|_2 = 0.1157$

$\underline{2^{nd} \text{ iteration:}}$ is similar to the previous step. More precisely:

$$w_2 = Av_2 = \begin{pmatrix} -0.4082 \\ 0.5103 \\ -0.5103 \end{pmatrix}.$$

$$w_2 = w_2 - \underbrace{h_{12}}_{proj_{v_1} w_2 = 0.0722} v_1 - \underbrace{h_{22}}_{proj_{v_2} w_2 = 0.7917} v_2 = \begin{pmatrix} -0.1361 \\ 0.1361 \\ 0.1361 \end{pmatrix}.$$

$h_{32} = \|w_2\|_2 = 0.2357.$

$$v_3 = \frac{w_2}{h_{32}} = \begin{pmatrix} -0.5774 \\ 0.5774 \\ -0.0.5774 \end{pmatrix} \Rightarrow \|v_3\|_2 = 1.$$

We obtain the two matrices $\overline{H}_2$ and V_2:

$$V_2 = (v_1, v_2) = \begin{pmatrix} 0.7071 & -0.4082 \\ 0.7071 & 0.4082 \\ 0 & -0.8165 \end{pmatrix}$$

$$\overline{H}_2 = \begin{pmatrix} 0.625 & 0.0722 \\ 0.2165 & 0.7917 \\ 0 & 0.2357 \end{pmatrix} = \underbrace{\begin{pmatrix} -0.9449 & 0.3112 & 0.1015 \\ -0.3273 & -0.8984 & -0.2929 \\ 0 & -0.31 & 0.9507 \end{pmatrix}}_{Q_2} \underbrace{\begin{pmatrix} -0.6614 & -0.3321 \\ 0 & -0.7603 \\ 0 & 0 \end{pmatrix}}_{R_2}.$$

We then have:

$$y_2 = R_2 \backslash [Q_2^T \beta e_1] = \begin{pmatrix} -0.6614 & -0.3321 \\ 0 & -0.7603 \\ 0 & 0 \end{pmatrix} \backslash \left[Q_2^T \begin{pmatrix} 0.3536 \\ 0 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} 0.5778 \\ -0.1447 \end{pmatrix}.$$

and the result x_1 of the second iteration is:

$$x_2 = x_0 + V_2 y_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.7071 & -0.4082 \\ 0.7071 & 0.4082 \\ 0 & -0.8165 \end{pmatrix} \times \begin{pmatrix} 0.5778 \\ -0.1447 \end{pmatrix} = \begin{pmatrix} 0.4677 \\ 0.3495 \\ 0.1182 \end{pmatrix}.$$

and the precision $r_2 = b - Ax_2 = \begin{pmatrix} -0.0134 \\ 0.0174 \\ 0.0283 \end{pmatrix} \Rightarrow \|r_2\|_2 = 0.0358.$

$\underline{k^{th} \text{ iteration:}}$ is similar to the previous iterations. The algorithm continues until finding the solution x_k such that $\|r_k\|_2 < \epsilon$.

Appendix B

Proof of diagonal term elimination with DI

In Section 5.2.2, we saw that `DI-cyc` has exactly the same performance with Gauss-Seidel in case of web graph if we apply diagonal elimination. This Appendix will provide a rigorous proof why we observed such a phenomenon.

D-Iteration is used to solve the PageRank equation $x = dPx + (1-d)Z$ whereas conventionally Gauss-Seidel is used to solve the equation $Ax = b$ as follows

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right). \tag{B.1}$$

Let $\mathcal{A} = dP$, to solve the PageRank equation we transform the GS equation to the form:

$$(I - \mathcal{A})x = b. \tag{B.2}$$

We have:

$$x_i = \sum_{j \neq i} \frac{\mathcal{A}_{ij}}{1 - \mathcal{A}_{ii}} x_j + \frac{b_i}{1 - \mathcal{A}_{ii}}. \tag{B.3}$$

where $\mathcal{A}_{ii} < 1$. Let $\mathcal{A}'$ be a matrix of size $n \times n$ and b' be a vector of size n such that:

$$\begin{aligned} \mathcal{A}'_{ij} &= \tfrac{\mathcal{A}_{ij}}{1-\mathcal{A}_{ii}}. \\ \mathcal{A}'_{ii} &= 0. \\ b'_i &= \tfrac{b_i}{1-\mathcal{A}_{ii}}. \end{aligned}$$

We rewrite the GS equation:

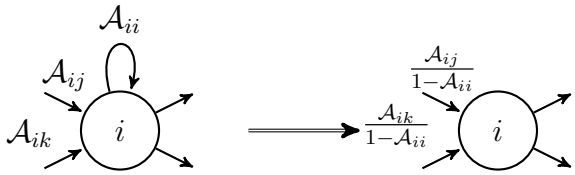
$$(I - \mathcal{A}')x = b'. \tag{B.4}$$

and

$$x_i = \sum_{j \neq i} \mathcal{A}'_{ij} x_j + b'_i. \tag{B.5}$$

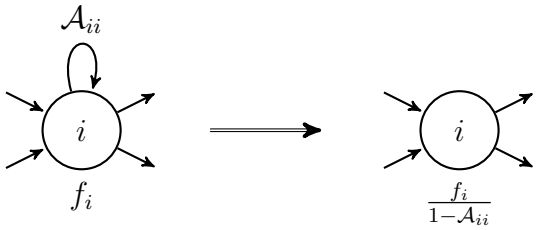
Equation B.3 and B.5 are exactly equivalent. Applying GS on either B.2 or B.4 yields the same result. In other words, the transformation would be obtained by combining two main changes:

- removing self-loop link, updating incoming link weight



- and updating vector b to b' such that $b'_i = b_i/(1 - \mathcal{A}_{ii})$.

In diffusion point of view, diagonal term elimination can be done by infinitely diffusing fluid on self-loop link and then is equal to accumulating fluid at the self-loop node:



With DI, recall the Equation 3.6:

$$H_k + F_k = F_0 + \mathcal{A}H_k.$$

We have:

$$\begin{aligned} H_k &= H_{k-1} + J_{i_k} F_{k-1}. \\ &= (H_{k-1} - J_{i_k}(F_0 - F_{k-1})) + J_{i_k} F_0. \\ &= (H_{k-1} - J_{i_k}(H_{k-1} - \mathcal{A}H_{k-1})) + J_{i_k} F_0 \\ &= (I - J_{i_k}(I - \mathcal{A}))H_{k-1} + J_{i_k} F_0 \end{aligned}$$

where I is the identity matrix, J_m is a matrix with all entries equal to zero except the m-th diagonal term: $(J_m)_{mm} = 1$, F_0 is the initial condition vector and i_k is the k^{th} choice of node for the diffusion. The choice of the sequence $I = \{i_1, i_2, ..., i_n, ...\}$ with $i \in \{1, .., n\}$ is the main optimization factor of the D-iteration.

Therefore, if we apply diagonal term elimination and the same choice sequence of nodes, vector H_k of DI is exactly the same as vector x obtained by GS in Equation B.5, and thus in Equation B.3.

Appendix C

Colonel Blotto games

In the second year of my PhD, I participated in building a Demo for Bell-Labs Future X Days [1]. The demo consists in explaining the basis of Colonel Blotto game (see below) and its applications to solve real-life problem.

According to [68], Colonel Blotto game is a two-person *zero-sum game* [2] in which the two players simultaneously distribute limited resources over several objects, or *battlefields*. The player distributes more resources than the other to a battlefield wins that battlefield, and the gain is equal to the total number of battlefields won.

The Colonel Blotto aims at finding the optimum distribution of soldiers (or resources) over n battlefields knowing that: (i) the party (or player) that has allocated the most soldiers on a battlefield wins that battlefield, but (ii) both parties do not know how many soldiers the other party will allocate to each battlefield, and (iii) both parties try to maximize the number of battlefields they expect to win.

Based on that, we build two demos of Colonel Blotto games [3].

In the first demo (see Figure C.1), we set up two parties, five battlefields and initially give 100 solders to each party. Each battlefield will bring a certain gain value for the party who wins that battlefield. The target of the two parties is to maximize the total gain value they can win.

[1] https://www.bell-labs.com/programs/bell-labs-futurex-days/

[2] Zero-sum game is a situation in which a player's gain (or loss) of utility is exactly equal to the losses (or gains) of the utility of the other player(s)

[3] The source code (16MB) is available at
http://www.mediafire.com/download/05j01bpoi55c0ym/Colonel_Blotto_Demos.zip

The second demo (see Figure C.2) is an extension of the first demo. It comprises of 38 European countries representing 38 battlefields of which the gain values correspond to the number of borders they have in common with other countries. The budget of the two players (for example two enterprises providing internet services) is allowed not to be equal. Besides the classical Colonel Blotto game, players can join the game with some additional strategies like Tit-For-Tat [4], etc.

The two demos are written using PHP, JavaScript, JQuery and Matlab.

[4] http://en.wikipedia.org/wiki/Tit_for_tat

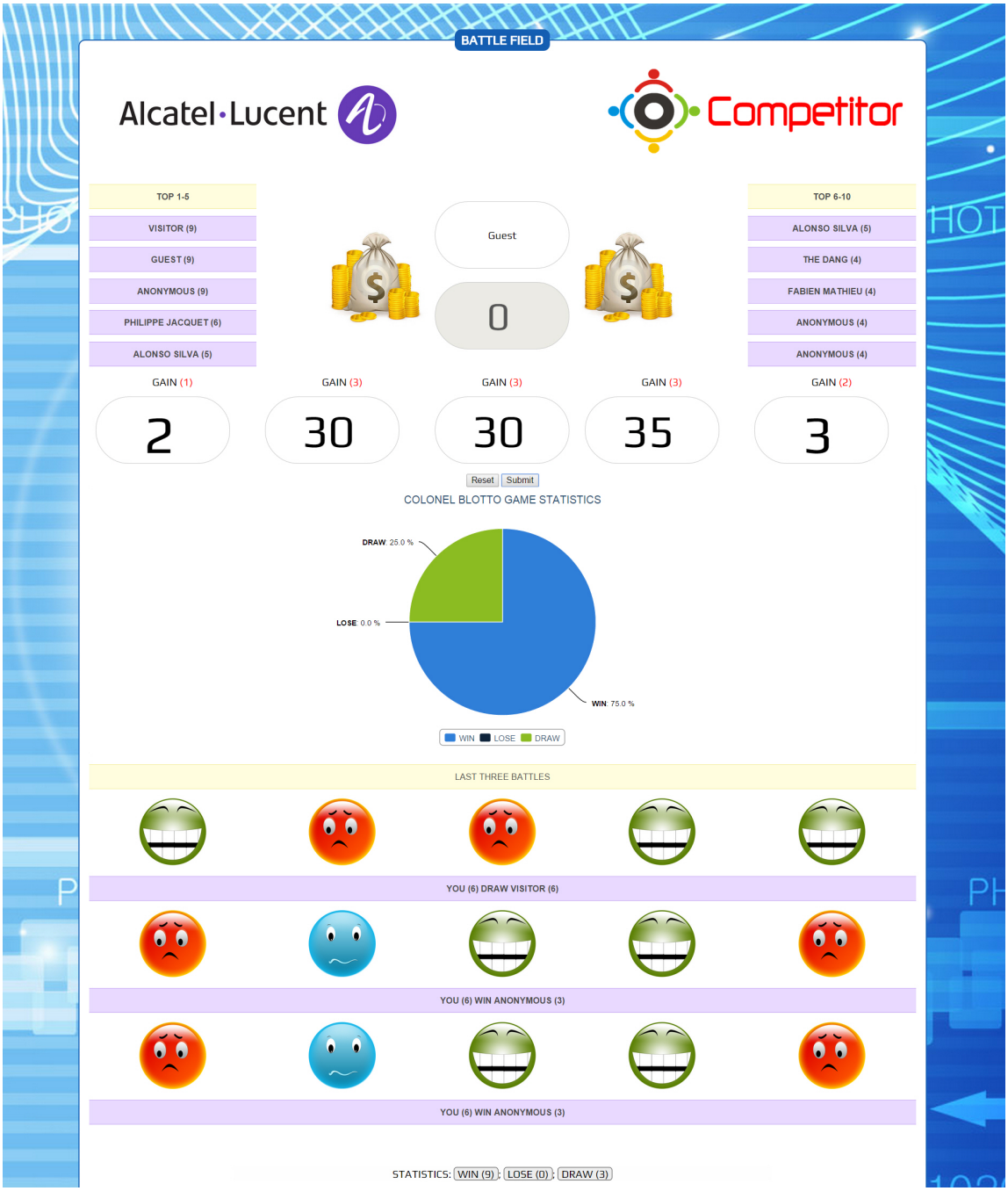


Figure C.1: Demo 1. Colonel Blotto game with 5 battlefields.

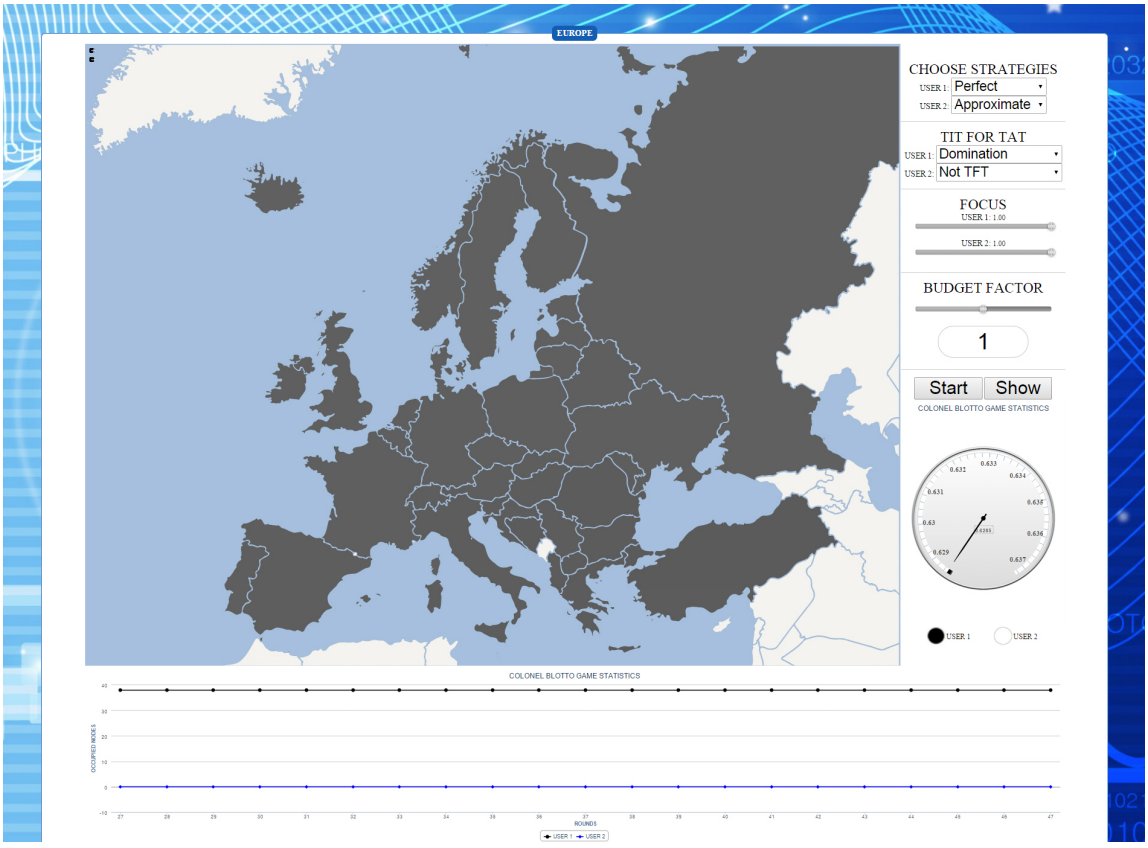


Figure C.2: Demo 2. Colonel Blotto game with 38 battlefields representing 38 European countries.

Appendix D

Exploiting user movement for position detection

During the last year of the PhD, besides the main research topic on PageRank and social networks, I was also interested in positioning systems and localization issues. The main goal of this adventure is firstly to challenge myself and satisfy my curiosity (personal), then to enlarge my research domain (professional).

Positioning systems are crucial to today's digital society. They help to locate objects or people carrying the objects and provide geographic information, thus to facilitate many human activities. For instance, vehicle navigation systems are indispensable for drivers in big cities. Some location-based services are deployed in commercial malls so that customers can get navigation while walking in complex environment and can receive promotion advertisement from shops. The market of indoor and outdoor location-based services has grown rapidly in the last decade.

However, the major issue of indoor localization system is the trade-off between implementation cost and accuracy. A low-cost system which demands only few hardware devices could save the cost but often it turns out to be less reliable.

In this chaper, our main contribution is proposing a new method, called Two-Step Movement (2SM), which requires only one reference point (RP) by exploiting useful information given by the position change of a mobile terminal (MT), or the user movement. This method can minimize the number of reference points required in a localization system or navigation service and reduce system implementation cost. Analytical result shows that the user position can be thus derived and given in

	RPs required	Distance	Additional requirement(s)
Triangulation	3	Yes	No
	2	Yes	Last position tracked or motion prediction
	1	Yes	Angle of arrival (*i.e.,* using array of antennas)
2-Step Movement	1	Yes	Measurement of user movement

Table D.1: Requirement comparison between triangulation and proposed 2-Step Movement (2SM) method.

simple closed-form expression. Finally, simulation is conducted to demonstrate its effectiveness under noisy environment. We published the results of this chapter in [24, 25].

This chapter is organized as follows. Section D.1 introduces the context of the study. Section D.2 talks about existing positioning solutions. Section D.3 describes the system design of the 2SM. Section D.4 evaluates performance of the method. Section D.5 presents how to improve the 2SM by using multi-sampling technique. Section D.6 describes a generalization of the algorithm when the reference point is mobile. Finally, Section D.7 concludes the chapter.

D.1 Introduction

Positioning systems are crucial to today's digital society. They help to locate objects or people carrying the objects and provide geographic information, thus to facilitate many human activities. For instance, vehicle navigation systems are indispensable for drivers in big cities. Some location-based services are deployed in commercial malls so that customers can get navigation while walking in complex environment and can receive promotion advertisement from shops. The market of indoor and outdoor location-based services has grown rapidly in the last decade.

Global positioning system (GPS) is very popular and widely used for user localization. When line-of-sight to at least four GPS satellites is available, location (latitude, longitude, and elevation) and timing information can be obtained. Although GPS is very convenient outdoors, its quality is susceptible to weather conditions, for example when sky view is poor due to fog, rain, cloud, etc., or being blocked by tall buildings in urban areas. These issues can significantly degrade the accuracy. As expected, GPS is not for indoor use due to the lack of line-of-sight. There also exists cellular-based positioning systems [69] which are built on measuring

signal strength from three or more base stations for tracking mobile user's location. However, these solutions also do not work well to indoors.

Various indoor positioning systems have been developed, see e.g., [70, 71, 72]. They can be categorized into network based or non-network based solutions. The network based approach, which takes advantages of existing network infrastructure such as wireless local area networks (WLANs), without demanding new infrastructure, can maintain low deployment cost. The non-network based approach is to use dedicated positioning infrastructure and often can provide higher reliability but at extra cost. For example, ultrasound and infrared based solutions have high deployment cost. One may also consider simple proximity-based solution like iBeacon [73] which however is only able to offer an approximate location. Some systems consider using visible light to construct an indoor positioning system with high accuracy [74, 75]. A good positioning system should be cost-effective and also be able to offer high accuracy.

Constructing an efficient and simple positioning system is always challenging. Technically, it would depend on the number of reference points (RPs) that we can have, on the technologies to be used (e.g., RF-based, ultrasound, infrared, etc.), and also on the characteristics of the environment. In this study, we propose a geometry-based positioning method which can determine user position by only using one RP and exploiting his/her simple movement, for instance walking or waving his/her hand-held device, and some simple information. As the solution requires only one RP and can provide either exact result in noiseless environment or accurate positioning in noisy condition, our approach brings competitive advantages compared to other methods, thanks to its simplicity and effectiveness. Meanwhile, the method is interesting and may have a high potential to improve today's technology or existing solutions.

D.2 Related Work

Indoor positioning problem has attracted research over years [76, 77, 78, 79]. Lots of studies have been done extensively and many possible solutions have been proposed so far. Generally speaking, there are four major approaches to solve this

problem: triangulation, fingerprinting, scene analysis, and proximity. We will discuss them below.

Triangulation is used to estimate the position of a user or mobile terminal (MT) if the geographical coordinates of the RPs are known and assume that the MT is capable of measuring the distance between itself and the RPs. A priori, this method requires three RPs to construct a distinct geometric intersection of three circles, which indicates the position of the MT. Fig. 1(a) illustrates the principle (or see e.g., [70]). Note that not all schemes based on triangulation requires three circles, see e.g., Fig. 1(b) and (c). For instance, given angle-of-arrival (AoA) information, using only one RP is sufficient to locate the MT.

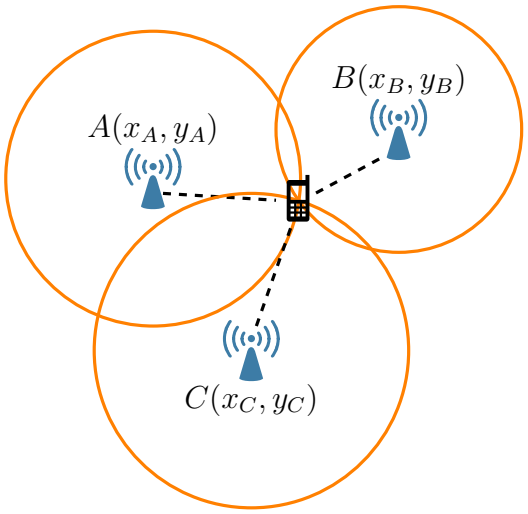
Fingerprinting [80] is to estimate device position by using pre-measured location-related data. This method consists of two phases: an offline training phase and an online position estimation phase. In the offline phase, location-related data is collected at different positions in the area. During the online position determination phase, real-time location-related data is measured and then matched with the set of data gathered during the offline phase to estimate the device's location.

Scene analysis localization method [76] is based on a set of images or scenes received by one or multiple cameras. This approach in principle does not require user (to be tracked) to carry any extra device. However, the solution is usually expensive because it requires one or many cameras to perform tracking and may prone to a high computation cost due to image or video processing.

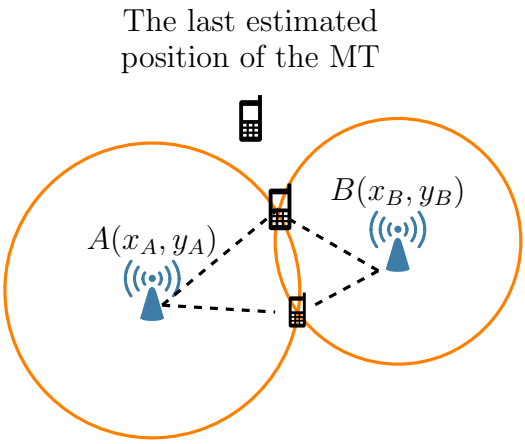
Proximity helps to detect if a MT is nearby or for example in the coverage area of a RP. However, it is hard to provide accurate position with high reliability.

Each of the above method also has some variants or hybrid scheme. Our proposed geometry-based solution is built on triangulation. We will explain and discuss in comparison other methods stemmed from this branch. The cost and accuracy of triangulation method primarily rely on the number of RPs required. Traditionally, one would need at least three RPs to determine the position of the MT.

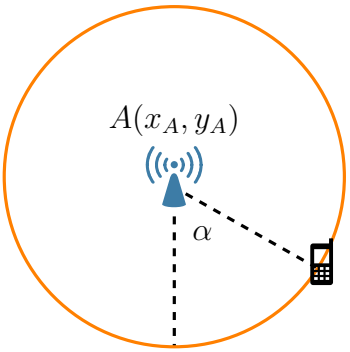
Fig. 1(b) shows a variant of traditional triangulation method, which requires two RPs and the last estimated previous position of the mobile terminal so as to eliminate one of the two intersection points of the two circles constructed by the two RPs. In such case, the location closer to the last estimated position would be



(a) Three RPs



(b) Two RPs, plus the last estimated position tracked



(c) One RP, plus information of angle-of-arrival α

Figure D.1: Positioning techniques using different number of reference points (RPs).

selected. Or, the system has to be able to predict user mobility pattern in order to select one. Note that this method still requires more than one RP. A variant of the above triangulation method is to use only one RP but requires the information of angle-of-arrival (AoA) provided by an array of antennas either implemented in the user terminal (MT) or at the RP [81], see Fig. 1(c). However, such an array of antennas is often costly and cumbersome.

D.3 System design

Here, we propose a new method called "Two-Step Movement (2SM)". It aims to improve the classical triangulation approach and requires only one RP. It consists in making use of movements of the user (i.e., changes in the position of the MT), either active (e.g., a user may wave his/her MT to assist) or natural (e.g., the user is walking or moving), relative to the position of the RP. Therefore, 2SM turns out to have a competitively low deployment cost and without extra or expensive tracking hardware such as antenna array and is able to determine user position in exact closed-form solution. The simplicity and effectiveness would highly facilitate practical indoor positioning systems. Table D.1 gives a comparison of the above methods and outlines their key difference. In our proposed 2SM method, the MT is supposed be able to measure his/her movement using its embedded sensors and applications (Apps), which are common in today's smartphones.

D.3.1 One-Step Movement (1SM)

Our method exploits useful information generated by user movement. For the sake of simplicity, the 2SM is presented as a combination of two One-Step Movements.

One-Step Movement (1SM) makes use of one position change (one move) to identify the two possible locations (position candidates) of the MT. We consider the following system and assumptions:

- The position of the RP is known.
- The MT is capable of measuring the distance between itself and the RP.
- The MT is capable of measuring the distance and the angle (direction) of the movement it has done.

Fig. D.2 illustrates the system design:

- A is the RP and its position (x_A, y_A) is known.
- B is the initial position of the MT that is unknown and we want to compute it, denoted by coordinates (x_B, y_B).

- C is the position of MT right after the first movement, (x_C, y_C), which is also unknown.

- MT is capable of measuring the distance between itself and the RP. That is, the distances AB and AC are given for example by measuring the received signal strength or standard techniques.

- MT is capable of measuring the distance and the angle of the movement it has done, thus BC and the angle $\alpha \in (0, 2\pi]$ (with respect to the positive x-axis) are also measurable.

Theorem 5. *Suppose that $A(x_A, y_A), AB, BC, AC,$ and α are known, the One-Step Movement (1SM) will give two estimated locations, denoted by generic point $B(x_B, y_B)$, whose x and y coordinates satisfy:*

$$x_B \cos\alpha + y_B \sin\alpha = x_A \cos\alpha + y_A \sin\alpha - \frac{(AB^2 + BC^2 - AC^2)}{2BC}. \tag{D.1}$$

Proof. Using Fig. D.2, from the two measured distances AB and AC, the equations of the two circles centered at $A(x_A, y_A)$ on which the MT probably lies are expressible as:

$$\begin{aligned} (x_B - x_A)^2 + (y_B - y_A)^2 &= AB^2 \\ (x_C - x_A)^2 + (y_C - y_A)^2 &= AC^2 \end{aligned} \tag{D.2}$$

where

$$\begin{aligned} x_C &= x_B + BC\cos\alpha, \\ y_C &= y_B + BC\sin\alpha. \end{aligned} \tag{D.3}$$

From (D.2), we have:

$$\begin{aligned} AB^2 - AC^2 = (x_B - x_C)(x_B + x_C - 2x_A) \\ + (y_B - y_C)(y_B + y_C - 2y_A). \end{aligned} \tag{D.4}$$

Substitute x_C and y_C in (D.3) to (D.4), we can have:

$$\begin{aligned} AB^2 - AC^2 = -BC\cos\alpha(2x_B + BC\cos\alpha - 2x_A) \\ - BC\sin\alpha(2y_B + BC\sin\alpha - 2y_A). \end{aligned} \tag{D.5}$$

which can be re-written as:

$$AB^2 + BC^2 - AC^2 = -2BC(x_B \cos\alpha - x_A \cos\alpha + y_B \sin\alpha - y_A \sin\alpha). \quad \text{(D.6)}$$

hence,

$$x_B \cos\alpha + y_B \sin\alpha = x_A \cos\alpha + y_A \sin\alpha - \frac{(AB^2 + BC^2 - AC^2)}{2BC}. \quad \text{(D.7)}$$

□

Eqn. (D.1) can be solved as follows:

- If $\sin\alpha = 0$, thus $\cos\alpha = \pm 1$, (D.1) becomes:

$$x_B = x_A \pm \frac{(AB^2 + BC^2 - AC^2)}{2BC}.$$

It is then straightforward to compute the values of x_B and y_B, by substituting the value of x_B to (D.2).

- If $\sin\alpha \neq 0$, dividing (D.1) by $\sin\alpha$, we have:

$$y_B = -\cot\alpha x_B + x_A \cot\alpha + y_A - \frac{AB^2 + BC^2 - AC^2}{2BC\sin\alpha}.$$

Let $a = -\cot\alpha$, $b = x_A \cot\alpha + y_A - (AB^2 + BC^2 - AC^2)/(2BC\sin\alpha)$, we see that now y_B is expressible as a function of x_B such that $y_B = ax_B + b$. Substituting y_B to the first equation of (D.2), we have:

$$(x_B - x_A)^2 + (ax_B + b - y_A)^2 = AB^2.$$

Then

$$(1 + a^2)x_B^2 - 2x_B(x_A - a(b - y_A)) + x_A^2 + (b - y_A)^2 - AB^2 = 0. \quad \text{(D.8)}$$

The above quadratic equation (D.8) can be solved easily.

Algorithm 6 shows in detail how to perform 1SM. It outputs two points $B1(x_{B1}, y_{B1})$ and $B2(x_{B2}, y_{B2})$, which are the possible solution of B.

Remark 1. *It is clear that one of the two points, $B1(x_{B1}, y_{B1})$ and $B2(x_{B2}, y_{B2})$, must be the position of the MT (or both of them are, if $B1$ and $B2$ are identical).*

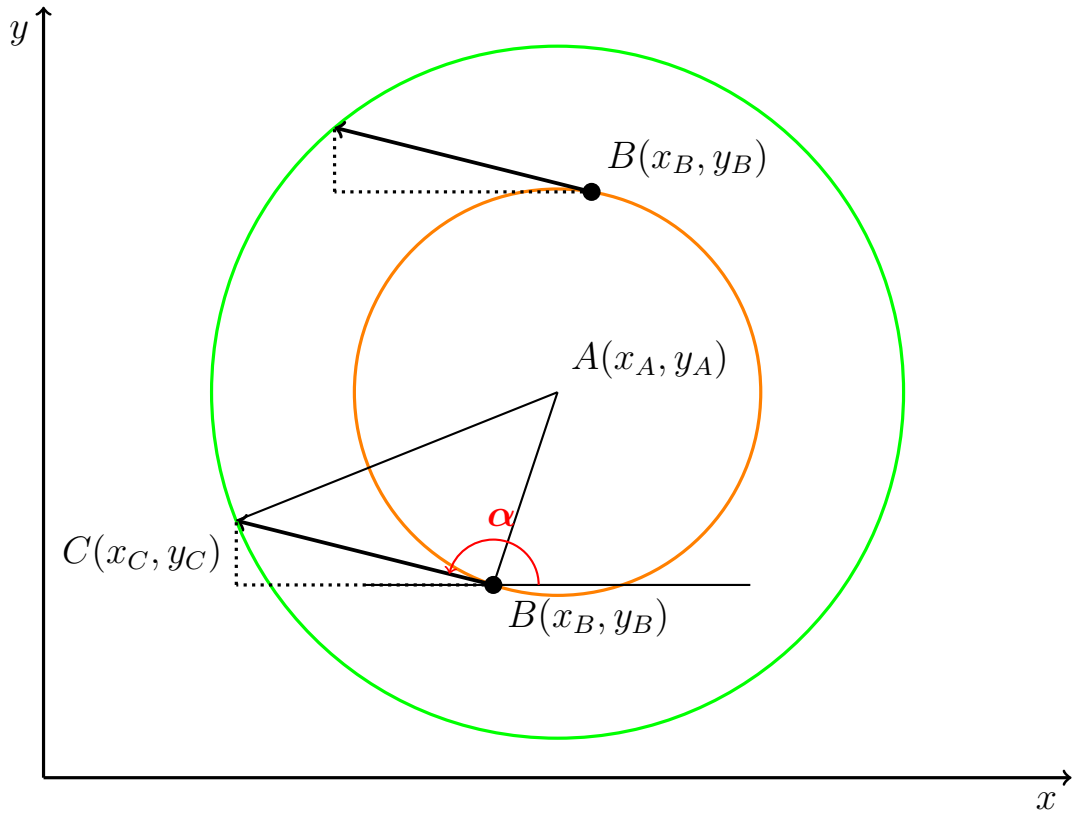


Figure D.2: One-Step Movement (1SM).

D.3.2 Two-Step Movement (2SM)

After the first movement, we have two possible locations of the MT given by 1SM using Algorithm 6, but cannot determine which one is the true location. We need to resolve this ambiguity. It is natural to think about performing an additional movement. The basic idea is simple: a Two-Step Movement (2SM) is a combination of two consecutive 1SM's where each move gives two possible positions (in which one of these two positions must be the true position). It is clear that by comparing the results of two 1SM's, we can determine the location of the MT, given that the results of the two 1SM's are not redundant.

Fig. D.3 depicts how 2SM works. The MT makes the second movement from C to D in the direction of angle β, which is measured from the positive x-axis counter-clockwise. The distance CD and β are known by the MT, whereas the distance AD from the MT to the RP is measured from the received signal strength by standard techniques. The underlying idea is that, we now consider the movement

Algorithm 6 One-Step Movement algorithm

Require: $A(x_A, y_A), AB, AC, BC, \alpha$;
1: **function** OneStep($A(x_A, y_A)$, AB, AC, BC, α)
2: **if** $\sin\alpha == 0$ **then**
3: **if** $\cos\alpha == 1$ **then**
4: $x_B = x_A - (AB^2 + BC^2 - AC^2)/(2BC)$;
5: **else**
6: $x_B = x_A + (AB^2 + BC^2 - AC^2)/(2BC)$;
7: **end if**
8: $y_{B1} = y_A + \sqrt{AB^2 - (x_B - x_A)^2}$;
9: $y_{B2} = y_A - \sqrt{AB^2 - (x_B - x_A)^2}$;
10: **return** $\{B1(x_B, y_{B1}), B2(x_B, y_{B2})\}$;
11: **else**
12: ▷ Pre-compute a, b such that $y_B = ax_B + b$;
13: $a = -\cot\alpha$;
14: $b = x_A \cot\alpha + y_A - (AB^2 + BC^2 - AC^2)/(2BC\sin\alpha)$;
15: ▷ Compute x_B, y_B;
16: $\Delta = (x_A - a(b - y_A))^2 - (1 + a^2)(x_A^2 + (b - y_A)^2 - AB^2)$;
17: $x_{B1} = (x_A - a(b - y_A) + \sqrt{\Delta})/(1 + a^2)$;
18: $y_{B1} = ax_{B1} + b$;
19: $x_{B2} = (x_A - a(b - y_A) - \sqrt{\Delta})/(1 + a^2)$;
20: $y_{B2} = ax_{B2} + b$;
21: **return** $\{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\}$;
22: **end if**
23: **end function**

of 2SM case similarly as that of 1SM case in which the starting point is now B and the ending point is D. We can compute the distance BD and the angle γ analytically (see Algorithm 7: line 6–13) and then use the method of Algorithm 6 to determine B. **Algorithm 7** details how 2SM works. By comparing the results from the two 1SM's computation, we determine the location of the MT.

Remark 2. *Note that the directions of the two movements should not be in parallel, i.e., $\beta \neq \alpha$ and $\beta \neq \alpha \pm \pi$, otherwise the ambiguity cannot be resolved since the system of equations generated by the second movement would be equivalent to that of the first one.*

In practice with estimation error or system imperfection, say noise exists, such that we cannot have a common solution from the two 1SM's computation, i.e., the first movement may give us two possible solutions denoted by $B1(x_{B1}, y_{B1})$ and

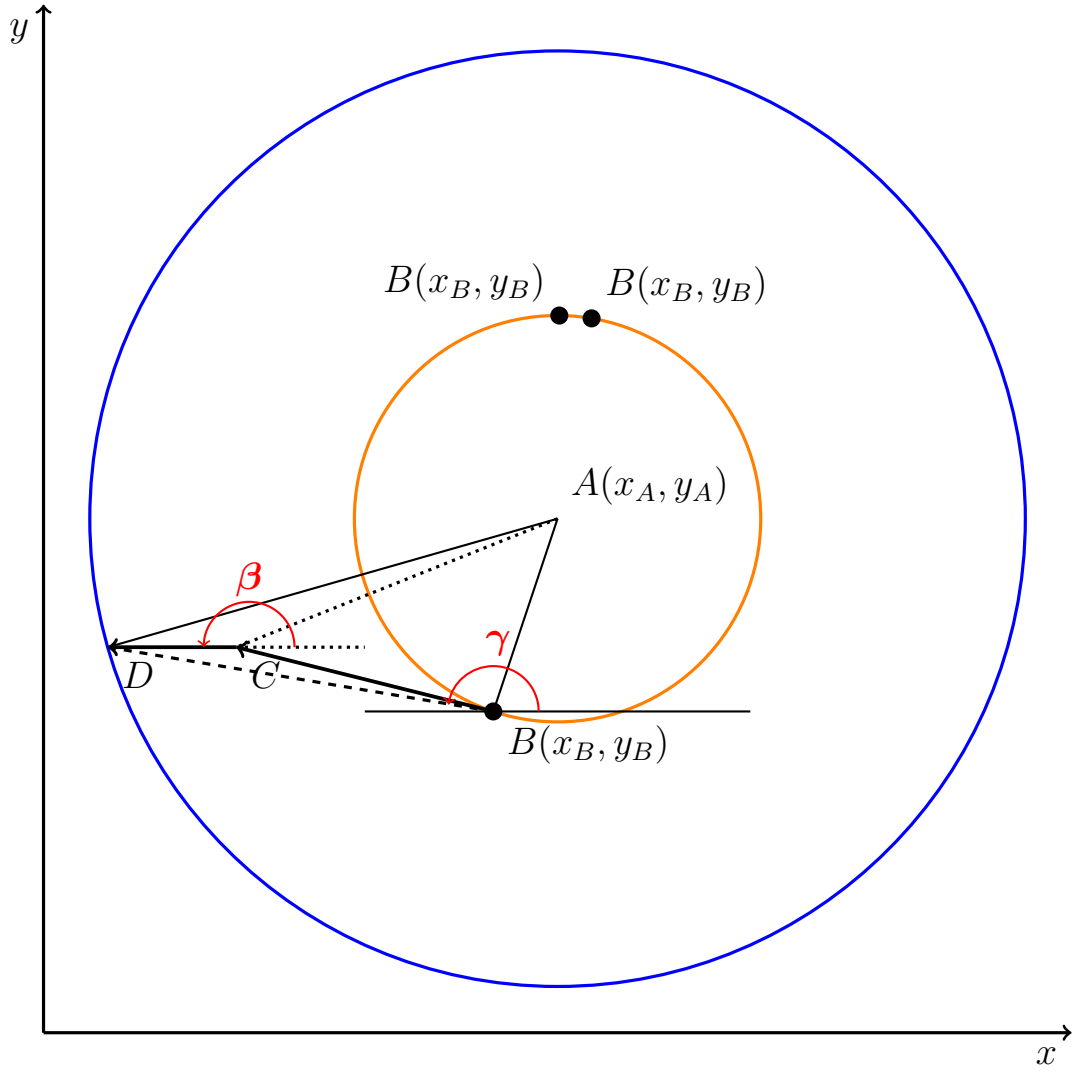


Figure D.3: Two-Step Movement (2SM).

$B2(x_{B2}, y_{B2})$, whereas the second movement may give us another two possible solutions denoted by $B3(x_{B3}, y_{B3})$ and $B4(x_{B4}, y_{B4})$, however $\{B1, B2\}$ and $\{B3, B4\}$ have no common point, as shown in Fig. D.4. To deal with, we can choose the pair of points that have the smallest distance, i.e., solving $\min\{d(P1, P2)|P1 \neq P2\}$, where $P1, P2 \in \{B1, B2, B3, B4\}$ and $d(P1, P2)$ is used to denote the Euclidean distance of points $P1$ and $P2$, and then takes their mean (e.g., the mid-point of $B1$ and $B3$ in Fig. D.4, see below) as the estimate of the MT's position for minimizing the error. In general, one can formulate it as an optimization problem and find the optimal result.

Algorithm 7 Two-Step Movement algorithm

Require: $A(x_A, y_A)$;
function TwoStep($A(x_A, y_A)$)
  MT makes the first movement from B to C; measure AB, AC, BC, α;
  ▷ Compute two location $B1$ and $B2$
  $\{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\}$ = OneStep($A(x_A, y_A), AB, AC, BC, \alpha$);
  MT makes the second movement from C to D; measure CD, AD, β; make sure that $\beta \neq \alpha$ and $\beta \neq \alpha \pm \pi$;
  ▷ The change in x-coordinate after the second move
  $X = BC\cos\alpha + CD\cos\beta$;
  ▷ The change in y-coordinate after the second move
  $Y = BC\sin\alpha + CD\sin\beta$;
  $BD = \sqrt{X^2 + Y^2}$;
  $\cos\gamma = X/BD$;
  $\sin\gamma = Y/BD$;
  Compute $\gamma \in (0; 2\pi]$ from $\cos\gamma$ and $\sin\gamma$;
  ▷ Compute two location $B3$ and $B4$
  $\{B3(x_{B3}, y_{B3}), B4(x_{B4}, y_{B4})\}$ = OneStep($A(x_A, y_A), AB, AD, BD, \gamma$);
  ▷ Determine MT location $B(x_B, y_B)$ from the set of $B1$, $B2$, $B3$ and $B4$
  $B(x_B, y_B) = \{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\} \cap \{B3(x_{B3}, y_{B3}), B4(x_{B4}, y_{B4})\}$;
  return $B(x_B, y_B)$;
end function

D.4 Simulation

Simulation is performed to investigate the performance of the proposed scheme (2SM) under noisy environment. The RP is placed at the center of a room, say $A = (0, 0)$. The user device or MT is randomly distributed in the room at $B(x_B, y_B)$, which is to be determined. For analysis, we discuss the following three scenarios and the performance: the MT (denoted by B, in Fig. D.3) is at a distance of 1, 5, and 10 meters from the RP, respectively, and the direction from B to A is uniformly distributed in $(0, 2\pi]$.

For a given AB, the movement from B to C or from C to D is equal to 0.1, 0.2, and 0.5 times of AB. The directions of the movement, i.e., α and β, are uniformly distributed in $(0, 2\pi]$. Estimation error to the measurement of distances AB, AC, AD, and BC, is considered to be bounded in $[-1\%, 1\%]$, $[-2\%, 2\%]$, and $[-5\%, 5\%]$, for comparison. We use e_d to denote the above bound such that $e_d = 1\%$, 2%, and 5%, respectively. Estimation error to the measurement of angles α and β

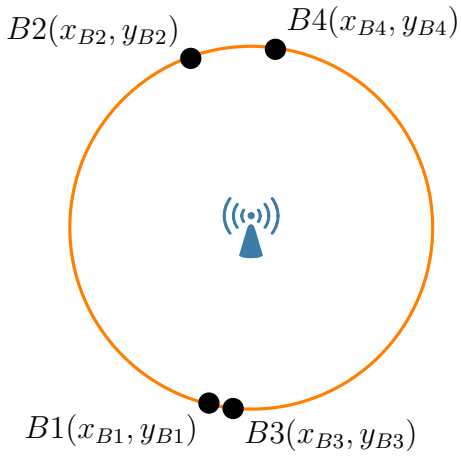


Figure D.4: Ambiguity elimination in case of noise.

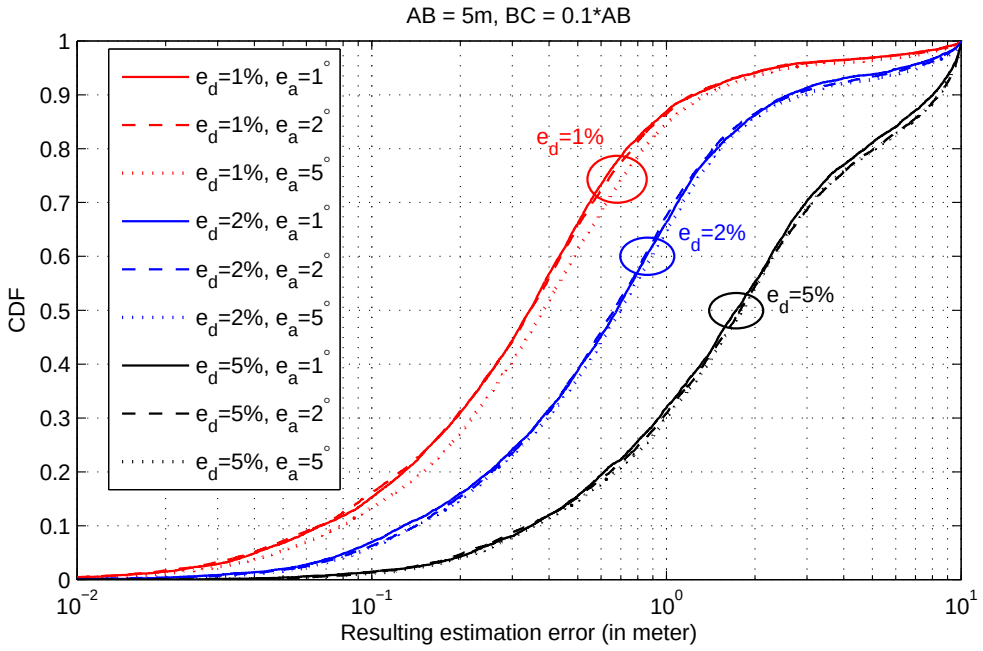


Figure D.5: Resulting error when $AB = 5$ meters, $BC = CD = 0.1 \times AB$ under various (e_d, e_a).

is considered to be bounded in $[-1°, 1°]$, $[-2°, 2°]$, and $[-5°, 5°]$. The bound on the angle measurement error is denoted by e_a such that $e_a = 1$, 2, and 5 degrees. For each (e_d, e_a) setup, the errors are randomly generated to corrupt the proposed algorithm in determining $B(x_B, y_B)$. Results when $AB = 5$ meters are shown in Fig. D.5, D.6, and D.7, respectively. Note that each curve in the figures is obtained by 10,000

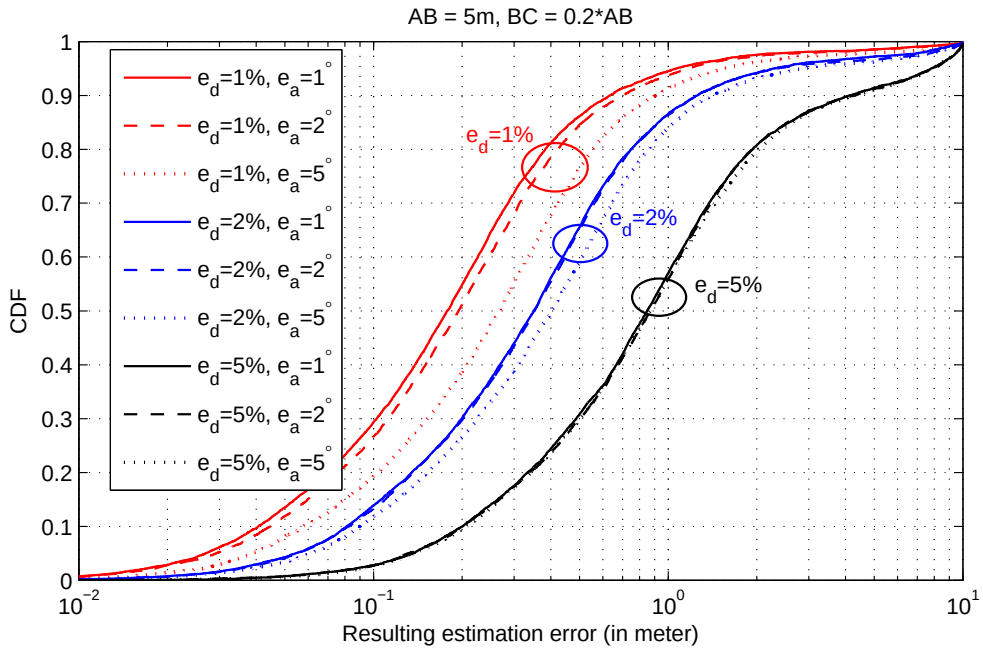


Figure D.6: Resulting error when $AB = 5$ meters, $BC = CD = 0.2 \times AB$ under various (e_d, e_a).

runs. During these runs, we observe that about 10% of the time the system fails to find the MT position (i.e., the quadratic equation (D.8) has no solution since Δ in Algorithm 6 is negative) due to the noises (which would be accumulated to Δ). We find that when $\Delta < 0$, the system is indeed heavily corrupted. We therefore consider them as bad movements and do not use for determining the MT. Note that it would be interesting to derive the position of the MT even when $\Delta < 0$ or see how to extract useful information to optimize results. This is subject to future work.

As shown in Fig. D.5-D.7, the estimation error in determining the position of the MT increases as e_d increases. Note that the estimation error is defined by the distance between the real position of the MT and the result given by Algorithm 7. Clearly, $e_d = 1\%$ (curves in "red") results in smaller estimation error than that $e_d = 3\%$ or $e_d = 5\%$ (curves in "blue" and "black", respectively) makes, given that e_a is the same.

As expected, the estimation error in determining the position of the MT also increases as e_a increases. However, when e_d is relatively large (5%), the impact of

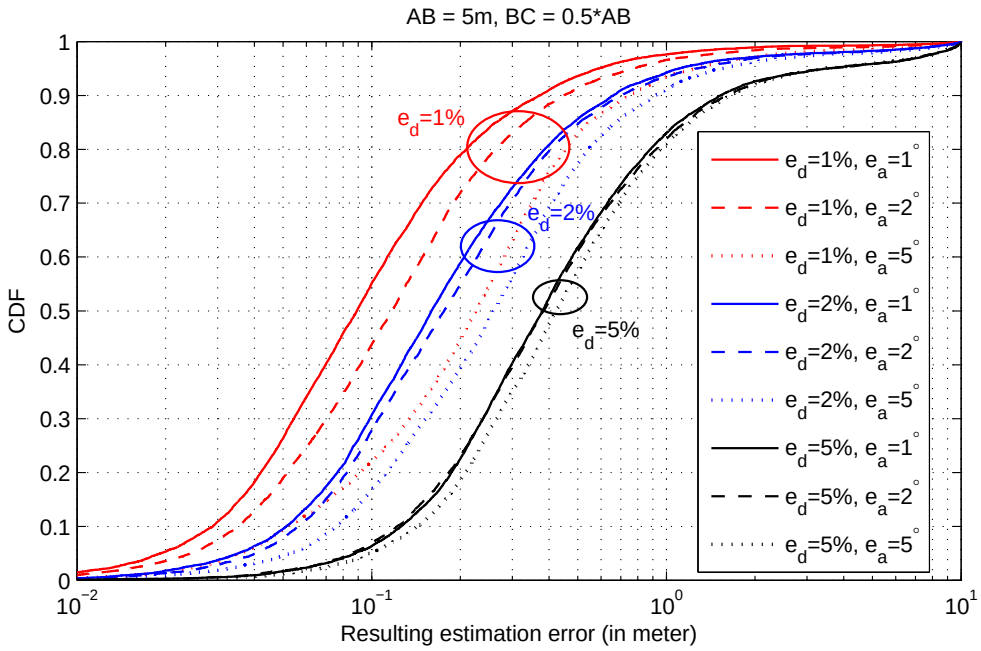


Figure D.7: Resulting error when $AB = 5$ meters, $BC = CD = 0.5 \times AB$ under various (e_d, e_a).

the considered e_a is relatively less significant. This can be clearly shown by Fig. D.6 and D.7. Roughly speaking, e_d is more dominating.

In comparing Fig. D.5-D.7, we observe that when increasing BC and CD from $0.1\times$ to $0.5 \times AB$, the estimation error in determining the position of the MT decreases quite substantially. See in Fig. D.6 and D.7, the curves shift to the left. The distance of the movement is a significant factor. We can improve the system performance by requiring a larger movement distance. However, a larger movement may be less favorable in some usage. In addition, from obtained simulation results (we did not plot all of them here), we see that the improvement is indeed decreasing and starts to get flat at $0.5 \times AB$.

Table D.2 shows the average error in determining the position of the MT under different AB (at 1, 5, and 10 meters, respectively) and various BC, CD, and noise levels (e_d, e_a). Result at $AB = 5$ was plotted in Fig. D.5-D.7. Note that since the curves for $AB = 1$ and 10 have very similar characteristics to those in Fig. D.5-D.7, we do not plot them here. Comparing the results at $AB = 1$, 5, and 10, we see that the magnitude of the error increases roughly proportional to AB, as expected. It is clear that the estimation error is minimized when (e_d, e_a) are small and the

	$e_d = 1\%$ $e_a = 1^\circ$	$e_d = 1\%$ $e_a = 2^\circ$	$e_d = 1\%$ $e_a = 5^\circ$	$e_d = 2\%$ $e_a = 1^\circ$	$e_d = 2\%$ $e_a = 2^\circ$	$e_d = 2\%$ $e_a = 5^\circ$	$e_d = 5\%$ $e_a = 1^\circ$	$e_d = 5\%$ $e_a = 2^\circ$	$e_d = 5\%$ $e_a = 5^\circ$
$AB = 1$ (BC=CD= $0.1AB$)	0.1412	0.1434	0.1581	0.2583	0.2640	0.2708	0.5530	0.5608	0.5691
$AB = 1$ (BC=CD= $0.2AB$)	*0.0808*	*0.0859*	*0.1036*	0.1463	0.1508	0.1631	0.3202	0.3340	0.3417
$AB = 1$ (BC=CD= $0.5AB$)	***0.0484***	*0.0566*	*0.0896*	*0.0753*	*0.0804*	*0.1086*	0.1701	0.1759	0.1868
$AB = 5$ (BC=CD= $0.1AB$)	0.7194	0.7279	0.8027	1.2797	1.3012	1.3668	2.7913	2.9031	2.9226
$AB = 5$ (BC=CD= $0.2AB$)	*0.3957*	*0.4235*	*0.5513*	0.7145	0.7480	0.8246	1.6222	1.6372	1.6587
$AB = 5$ (BC=CD= $0.5AB$)	***0.2193***	*0.2831*	*0.4481*	*0.4136*	*0.4412*	*0.5448*	0.8738	0.8829	0.9134
$AB = 10$ (BC=CD= $0.1AB$)	1.4165	1.4348	1.6130	2.4798	2.6257	2.7011	5.7899	5.8059	5.8929
$AB = 10$ (BC=CD= $0.2AB$)	*0.8006*	*0.8602*	*1.0845*	1.4779	1.1508	1.5112	3.2304	3.3131	3.3873
$AB = 10$ (BC=CD= $0.5AB$)	***0.4987***	*0.5601*	*0.9362*	*0.8180*	*0.8798*	*0.1058*	1.7551	1.7652	1.8750

Table D.2: Average error (in meter) under various AB, BC, CD, and noise levels (e_d, e_a).

movement distance is relatively large. Roughly speaking, at $BC = CD = 0.5 \times AB$, the performance is quite desirable when $e_d \leq 2\%$ and $e_a \leq 5\%$. When the movement distance is at the level of $0.2 \times AB$, the same performance can be achieved when e_d is reduced to $\leq 1\%$. The average error can be limited to within about 10% of AB. In the best case, the average error can be less than 5% of AB.

D.5 Two-Step Movement using Multi-Sampling

To further improve the performance of 2SM, this section demonstrates the use of *multi-sampling* technique so that many measurements will be conducted during the movement of the MT instead of using only one at the end of each movement to combat measurement errors and improve the positioning performance.

D.5.1 System Design

To begin with, we recall our system design (cf Section D.3) and the 2SM positioning method which requires only one RP (for minimal system implementation cost). We determine user position by exploiting its movement, e.g., in walking or by waving his/her hand-held device, as demonstrated in Figure D.8.

D.5.2 Motivation of Multi-Sampling

The precision of the estimation is depending on the accuracy in the measurement phase (e.g., α, AB, AC, BC). In reality, the limitation of hardware technology and presence of noise may severely degrade the quality of the inputs to our algorithm and then leads to poor position estimation. To reduce the impact of noise,

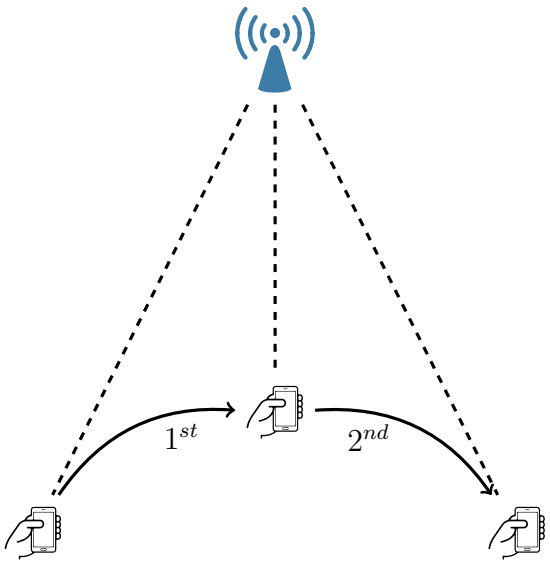
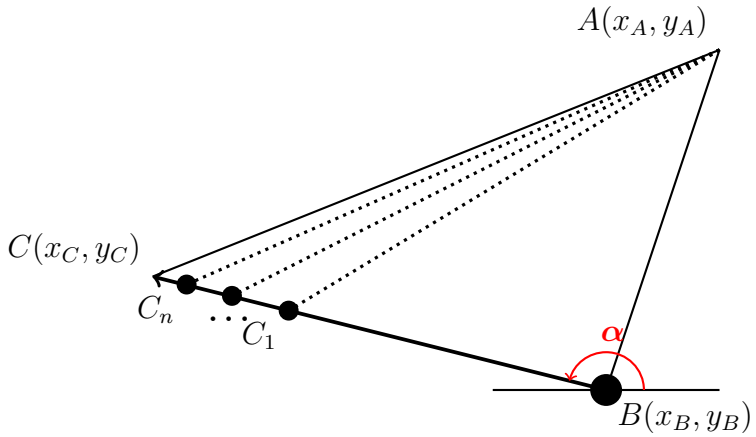


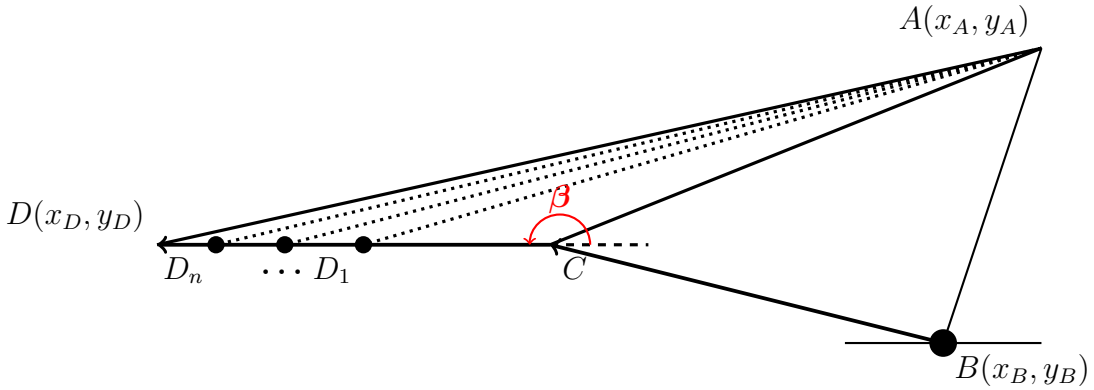
Figure D.8: Two-Step Movement (2SM) application: waving hand to get position.

naturally one can think of making many measurements and then combine them to produce better result. The intuition is that in the simplest case where noise follows a zero-mean distribution, we can expect the output of our algorithm also to have roughly a zero-mean error distribution. Another idea is that one could probably use a set of measurements to infer or pick out a better result.

Recall that in 2SM, all measurements are done only once at the end of each movement step (e.g., at C). Here, we propose a *multi-sampling* 2SM such that many measurements are carried out along the path so that the MT continuously keeps track of the movement and the distance to the RP. In other words, the step BC is considered as a series of small steps and we will use all the data obtained from these steps for positioning (see Fig. D.9). In the first movement from B to C (see Fig. D.9a), measurements are taken at the intermediate points $C1, \ldots, C_n$ and each of them allows to compute two possible positions of B, denoted by $B1$ and $B2$. Obviously, n intermediate measurements give us two sets of $B1$ and $B2$, denoted by S_{B1} and S_{B2} with $|S_{B1}| = |S_{B2}| = n$. One can simply take the middle points of S_{B1} and S_{B2} respectively or formulate an optimization problem to find the best estimates of $\overline{B1}$ and $\overline{B2}$. Similarly, the above process is applied to the second step movement (see Fig. D.9b) such that we can find two estimates, denoted by $\overline{B3}$ and $\overline{B4}$, respectively. In general, we will have four sets of points as shown in Fig. D.10.



(a) The first movement with multi-samples



(b) The second movement with multi-samples

Figure D.9: Two-Step Movement (2SM) with multi-sampling near the end points.

To resolve the ambiguity, we can then for example take the mid-point of the pair which have the minimum Euclidean distance (i.e., the mid-point of $\overline{B1}$ and $\overline{B3}$ in Fig. D.10) as the MT's position or formulate an optimization problem to minimize the error.

D.5.3 Numerical Studies

Simulation is performed to investigate the performance of the above multi-sampling 2SM in comparison to that in D.4, say single-sampling 2SM. The simulation set-up is as follows:

- RP is placed at the center of a room, i.e., $A(0, 0)$.

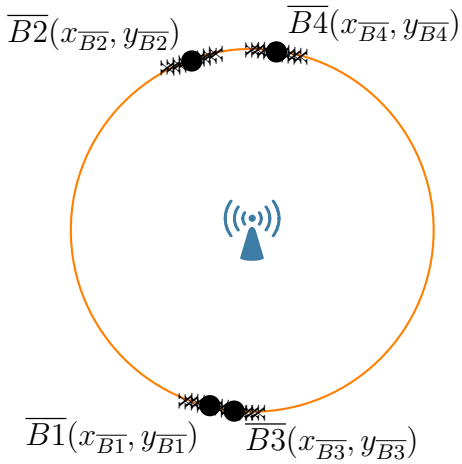


Figure D.10: The possible solutions in the presence of noise.

	$e_d = 1\%$, $e_a = 1°$		$e_d = 2\%$, $e_a = 2°$		$e_d = 5\%$, $e_a = 5°$	
	Single	Multi	Single	Multi	Single	Multi
$AB = 1(BC = CD = 0.1AB)$	0.1412	0.1154	0.2640	0.2164	0.5691	0.4090
$AB = 1(BC = CD = 0.2AB)$	0.0808	0.0666	0.1508	0.1370	0.3417	0.2638
$AB = 1(BC = CD = 0.5AB)$	0.0484	0.0416	0.0804	0.0792	0.1868	0.1541
$AB = 5(BC = CD = 0.1AB)$	0.7194	0.5858	1.3012	1.0269	2.9226	2.1480
$AB = 5(BC = CD = 0.2AB)$	0.3957	0.3623	0.7480	0.6062	1.6587	1.2861
$AB = 5(BC = CD = 0.5AB)$	0.2193	0.2108	0.4412	0.3393	0.9134	0.8067
$AB = 10(BC = CD = 0.1AB)$	1.4165	1.1650	2.6257	1.9102	5.8929	4.0634
$AB = 10(BC = CD = 0.2AB)$	0.8006	0.5777	1.1580	1.1238	3.3873	2.5696
$AB = 10(BC = CD = 0.5AB)$	0.4987	0.4325	0.8798	0.7959	1.8750	1.5831

Table D.3: Average estimation error (in meter) due to the single-sampling and multi-sampling 2SM methods under various AB, BC, CD, and noise levels.

- The initial position of MT is called B and its distance to the RP is set to three values: 1, 5, and 10 meters. Its position on a corresponding circle is randomly generated.

- Measurement error of distance is denoted by e_d and bounded by $[-1\%, 1\%]$, $[-2\%, 2\%]$, and $[-5\%, 5\%]$ with respect to its true value. Meanwhile, measurement error of angle is denoted by e_a and bounded by $[-1°, 1°]$, $[-2°, 2°]$, and $[-5°, 5°]$.

Note that there are several ways to choose the sampling intervals. For example, one may simply use uniform sampling, i.e., BC is divided into n intervals of equal distance such that $BC_1 = C_iC_{i+1}$, where $1 \leq i \leq n-1$. However, we observe

that it is better to do sampling closer to the end point of each movement, i.e., close to C in the first movement (and D in the second movement). Indeed, this coincides the result obtained in single-sampling 2SM that the larger the BC_i in the first move, the more accurately the second move can help to determine the true MT position.

Table D.3 compares the performance of the single-sampling and multi-sampling 2SM. The result is obtained by 10^5 runs. In multi-sampling 2SM, we perform 1000 sampling near the end point of each movement (see Fig. D.9). Table D.3 shows that the multi-sampling has effectively reduced the position estimation error by $15\% - 30\%$. Besides, it is also interesting to see that when the positioning error resulted in the single-sampling is larger, the improvement thanks to multi-sampling is even more significant.

D.6 Generalization of 2SM to Device-to-Device system

In this section, we generalize the Two-Step movement algorithm, called Generalized Two-Step Movement or **G2SM** in short, to Device-to-Device (D2D) environment. The considered system consists of a Mobile Terminal (MT) and a movable Reference Point (RP) (*e.g.,* another mobile device) such that the location of the MT, which always move in the coverage area of the RP, can be estimated with respect to the position of the RP (see Figure D.11). Additionally, the RP is always precisely localized regardless of where it moves to.

D.6.1 System Design and Basic Idea

We consider the following system and generalization:

- The RP is also mobile (movable).
- The MT is capable of measuring the distance between itself and the RP.
- The MT is capable of measuring the distance and the angle (direction) of the movement it has done.

Figure D.12 depicts the following technical details:

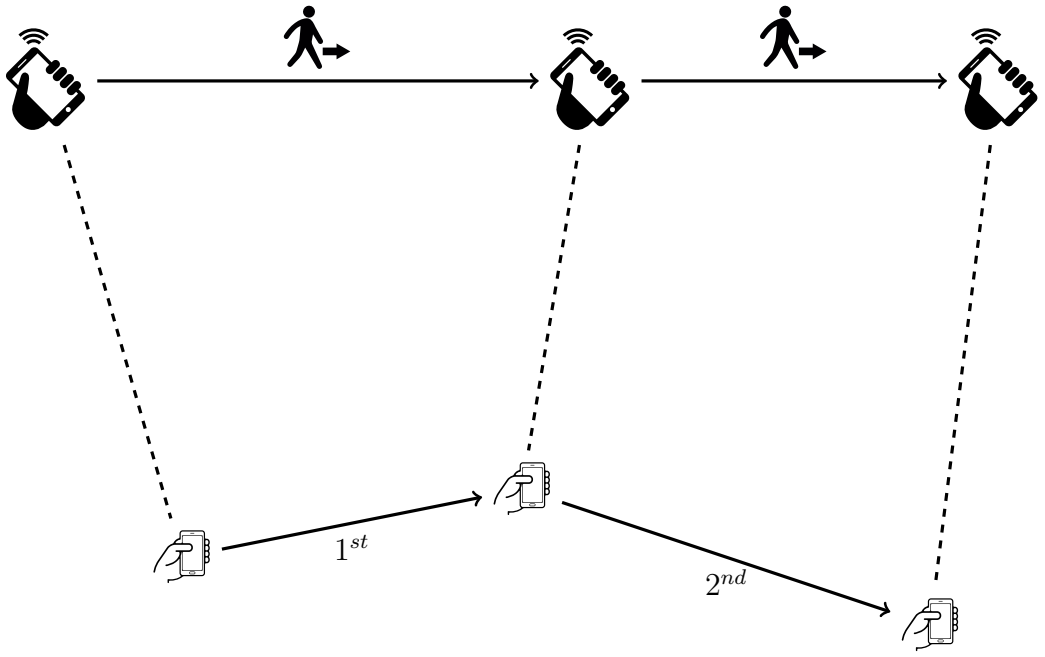


Figure D.11: Generalization of Two-Step Movement (G2SM): The Reference Point (black) is another mobile phone. Another mobile terminal (white) can localized itself thanks to the movement of the moveable reference point nearby.

- The RP is initially located at $A(x_A, y_A)$ which is known.
- The MT is initially at position B, which is however unknown, denoted by coordinates (x_B, y_B).
- C and D are the positions of the MT and RP, respectively, during their movement. Assume that RP can localize itself, so that $D(x_D, y_D)$ is known. However, $C(x_C, y_C)$ is not given.
- MT is capable of measuring the distance between itself and the RP, i.e., distances AB and DC are deterministic. For example, this can be done by measuring the received signal strength or using other standard techniques.
- MT is capable of measuring the distance and angle of its movement so that BC and the angle $\alpha \in (0, 2\pi]$ (with respect to the x-axis) are deterministic.

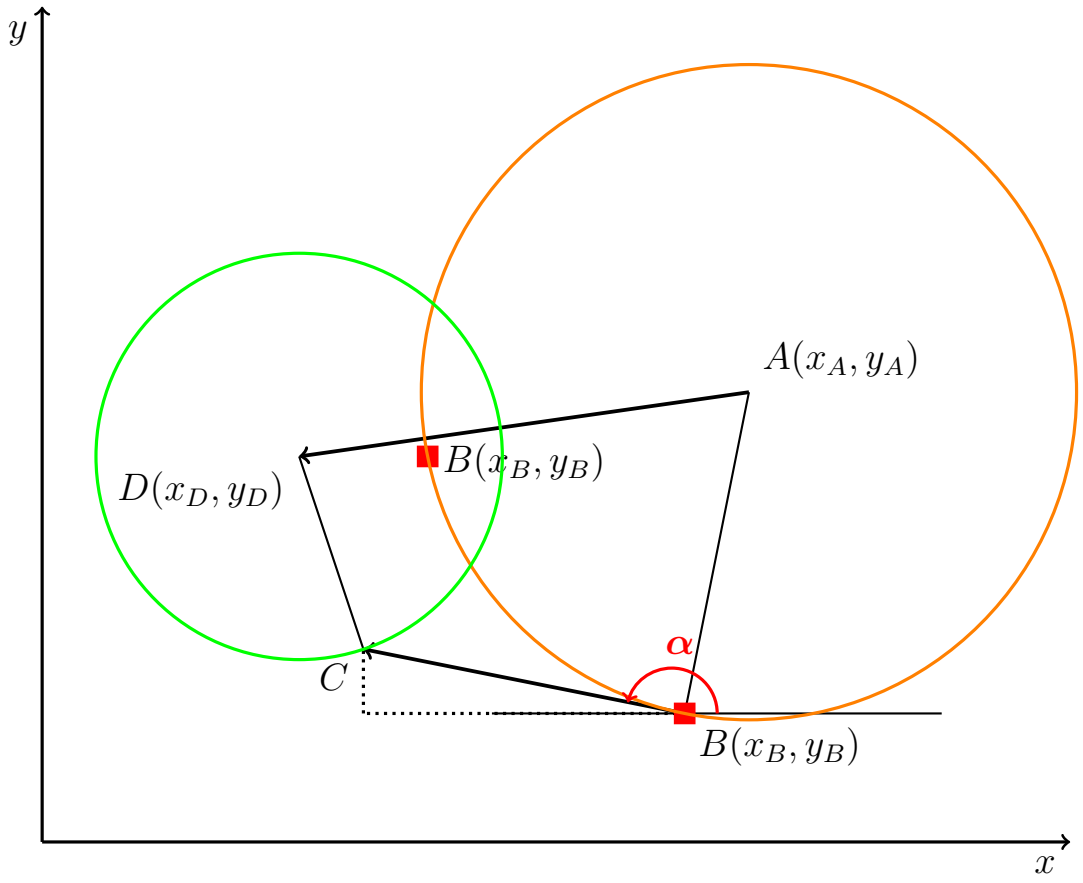


Figure D.12: Generalized One-Step Movement (GSM).

Theorem 6. *If $A(x_A, y_A)$, $D(x_D, y_D)$, AB, DC, BC, and α are known, Algorithm 8 give stwo possible solutions of B, denoted by (x_B, y_B), satisfying*

$$\begin{aligned} & x_B(BC\cos\alpha - x_D + x_A) + y_B(BC\sin\alpha - y_D + y_A) \\ & = x_D BC cos\alpha + y_D BC\sin\alpha \\ & - \frac{AB^2 + BC^2 - DC^2 - (x_A^2 + y_A^2) + (x_D^2 + y_D^2)}{2}. \end{aligned} \tag{D.9}$$

Proof. The equations of the two circles centered at $A(x_A, y_A)$ and $D(x_D, y_D)$ after the first movement (Fig. D.12) can be written as follows:

$$\begin{aligned} AB^2 &= (x_B - x_A)^2 + (y_B - y_A)^2. \\ DC^2 &= (x_C - x_D)^2 + (y_C - y_D)^2. \end{aligned} \tag{D.10}$$

Algorithm 8 Generalized One-Step movement algorithm

Require: $A(x_A, y_A), D(x_D, y_D), AB, DC, BC, \alpha$;

function OneStepExt($A(x_A, y_A)$, $D(x_D, y_D)$, AB, DC, BC, α)
  ▷ Precompute a, b, c such that $ax_B + by_B = c$;
  $a = BC\cos\alpha - x_D + x_A$;
  $b = BC\sin\alpha - y_D + y_A$;
  $c = x_D BC cos\alpha + y_D BC\sin\alpha - (AB^2 + BC^2 - DC^2 - (x_A^2 + y_A^2) + (x_D^2 + y_D^2))/2$;
  ▷ Compute x_B, y_B;
  if $b == 0$ **then**
    $x_B = c/a$;
    $y_{B1} = y_A + \sqrt{AB^2 - (x_B - x_A)^2}$;
    $y_{B2} = y_A - \sqrt{AB^2 - (x_B - x_A)^2}$;
    return $\{B1(x_B, y_{B1}), B2(x_B, y_{B2})\}$;
  else
    $d = -a/b$;
    $e = c/b$;
    $\Delta = (x_A - d(e - y_A))^2 - (1 + d^2)(x_A^2 + (e - y_A)^2 - AB^2)$;
    $x_{B1} = (x_A - d(e - y_A) + \sqrt{\Delta})/(1 + d^2)$;
    $y_{B1} = dx_{B1} + e$;
    $x_{B2} = (x_A - d(e - y_A) - \sqrt{\Delta})/(1 + d^2)$;
    $y_{B2} = dx_{B2} + e$;
    return $\{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\}$;
  end if
end function

where

$$\begin{aligned} x_C &= x_B + BCcos\alpha. \\ y_C &= y_B + BCsin\alpha. \end{aligned} \tag{D.11}$$

Substitute (D.11) to (D.10), we have

$$\begin{aligned} AB^2 &= (x_B - x_A)^2 + (y_B - y_A)^2. \\ DC^2 &= (x_B + BC\cos\alpha - x_D)^2 + (y_B + BC\sin\alpha - y_D)^2. \end{aligned} \tag{D.12}$$

which gives the equality

$$\begin{aligned} DC^2 - AB^2 &= 2x_B(BC\cos\alpha - x_D + x_A) - x_A^2 \\ &+ 2y_B(BC\sin\alpha - y_D + y_A) - y_A^2 \\ &+ (BC\cos\alpha - x_D)^2 + (BC\sin\alpha - y_D)^2. \end{aligned}$$

and can be rewritten as:

$$\begin{aligned}&2x_B(BC\cos\alpha - x_D + x_A) + 2y_B(BC\sin\alpha - y_D + y_A)\\&= 2x_D BC\cos\alpha + 2y_D BC\sin\alpha + DC^2 - AB^2 - BC^2\\&+(x_A^2 + y_A^2) - (x_D^2 + y_D^2).\end{aligned}$$

Thus,

$$\begin{aligned}&x_B(BC\cos\alpha - x_D + x_A) + y_B(BC\sin\alpha - y_D + y_A)\\&= x_D BC\cos\alpha + y_D BC\sin\alpha\\&-\frac{AB^2 + BC^2 - DC^2 - (x_A^2 + y_A^2) + (x_D^2 + y_D^2)}{2}.\end{aligned}$$

□

Let $a = BC\cos\alpha - x_D + x_A$, $b = BC\sin\alpha - y_D + y_A$ and $c = x_D BCcos\alpha + y_D BC\sin\alpha - (AB^2 + BC^2 - DC^2 - (x_A^2 + y_A^2) + (x_D^2 + y_D^2))/2$, the Eqn. (D.9) can be rewritten under the form $ax_B + by_B = c$ and be solved as follows:

- If $b = 0$ (or $BC\sin\alpha = y_D - y_A$), straightforwardly $x_B = c/a$ and $y_B = y_A \pm \sqrt{AB^2 - (x_B - x_A)^2}$.
- If $b \neq 0$, let $d = -a/b$, $e = c/b$, we see that now y_B is expressible as a function of x_B such that $y_B = dx_B + e$. Substituting y_B to the first equation of (D.10), we have

  $$(x_B - x_A)^2 + (dx_B + e - y_A)^2 = AB^2.$$

  which is a quadratic equation

$$(1 + d^2)x_B^2 - 2(x_A - d(e - y_A))x_B + x_A^2 + (e - y_A)^2 - AB^2 = 0. \quad \text{(D.13)}$$

The above equation (D.13) can be solved easily.

Algorithm 8 shows step-by-step how to compute the possible solutions of B. It outputs two points $B1(x_{B1}, y_{B1})$ and $B2(x_{B2}, y_{B2})$.

It outputs two points $B1(x_{B1}, y_{B1})$ and $B2(x_{B2}, y_{B2})$, which are the possible solution of B.

Remark 3. *It is clear that one of the two points, $B1(x_{B1}, y_{B1})$ and $B2(x_{B2}, y_{B2})$, must be the position of the MT (or both of them are the position, if $B1$ and $B2$ are identical).*

Remark 4. *In the specific case where RP position is fixed (i.e., $AD = 0$ such that $x_A = x_D$ and $y_A = y_D$), the above generalized Algorithm 8 (G1SM) will become the 1SM Algorithm and Equation (D.9) can be simplified as:*

$$x_B \cos\alpha + y_B \sin\alpha \;=\; x_A \cos\alpha + y_A \sin\alpha - \frac{(AB^2 + BC^2 - AC^2)}{2BC}.$$

D.6.2 The Generalized Two-Step Movement (G2SM)

After the first movement, we have two possible locations of the MT given by G1SM using Algorithm 8, but cannot determine which one is the true location. We need to resolve this ambiguity. It is natural to think about performing an additional movement. The basic idea is simple: a Generalized Two-Step Movement (G2SM) is a combination of two consecutive G1SMs where each move gives two possible positions (in which one of these two positions must be the true position). It is clear that by comparing the results of the two G1SMs, we can determine the location of the MT, given that the results of the two G1SMs are not redundant.

Fig. D.13 depicts how G2SM works. While the RP moves from D to E, the MT carries out the second movement from C to F in the direction of angle β, which is measured from the positive x-axis counter-clockwise. The distance CF and β are known by the MT, whereas the distance EF from the MT to the up-to-date position of RP is measured from the received signal strength by standard techniques. Because the position of RP is always accurately tracked, its coordinate $E(x_E, y_E)$ is known. The underlying idea is that, we now consider the movement of G2SM case similarly as that of G1SM case in which the starting point of MT is now B and the ending point is F, regarding the two positions of RP are A and E. We can compute the distance BF and the angle γ analytically (see Algorithm 9: line 8–15) and then use the method of Algorithm 8 to determine B. **Algorithm 9** details how 2SM works. By comparing the results from the two 1SM's computation, we determine the location of the MT.

Remark 5. *The G2SM requires the MT to change the moving direction such that $\beta \neq \alpha$ and $\beta \neq \alpha \pm \pi$ (or the RP is changing its direction), otherwise the ambiguity cannot be eliminated since the system of equations obtained from the second movement would be equivalent to that of the first one.*

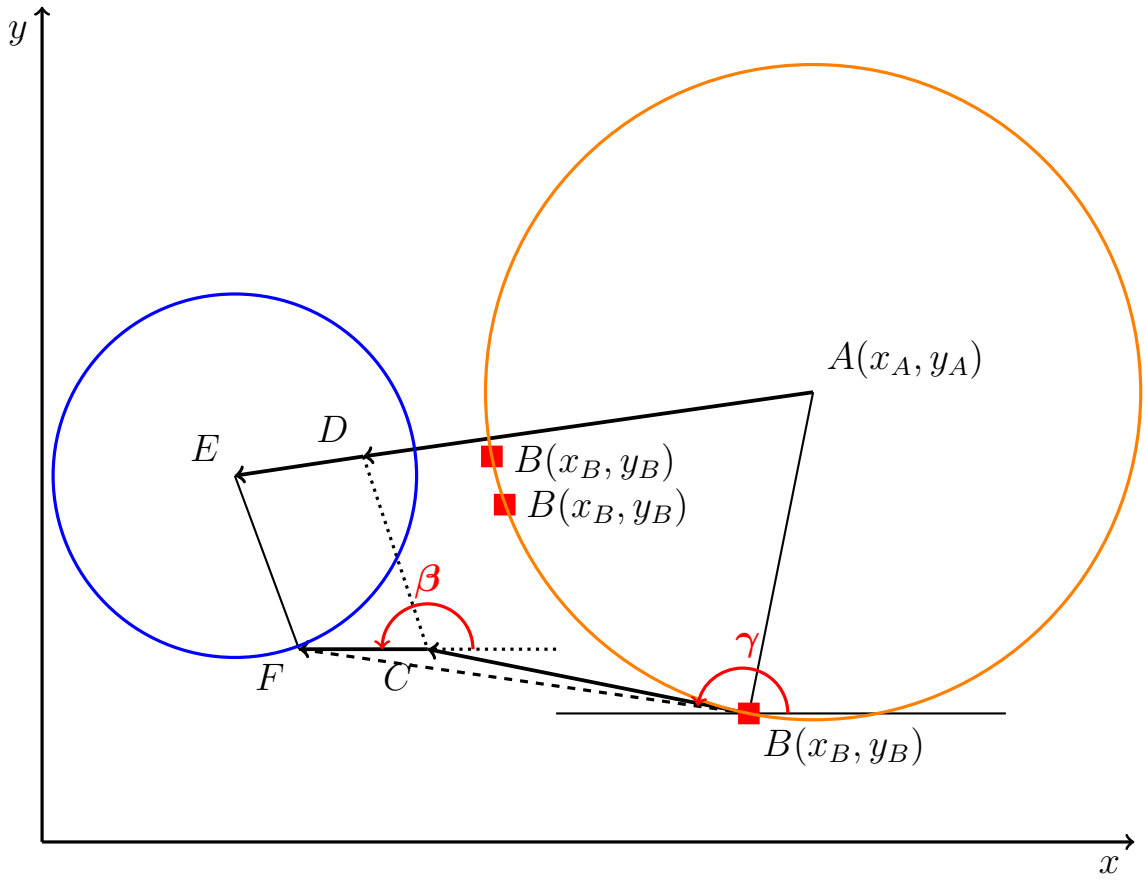


Figure D.13: Generalized Two-Step Movement (Generalized-2SM).

In practice with estimation error or system imperfection, say the existence of noise, we may not obtain a common solution from the two G1SM's computations, i.e., the two possible solutions obtained from the first movement are different from the solutions obtained from the second movement (see Fig D.13). To solve this problem, we can choose the pair of points in $\{\overline{B1}, \overline{B2}, \overline{B3}, \overline{B4}\}$ that have the minimum Euclidean distance, i.e., solving $\min\{d(P_i, P_j)|P_i \neq P_j\}$, where $P_i, P_j \in \{\overline{B1}, \overline{B2}, \overline{B3}, \overline{B4}\}$ and $d(P_i, P_j)$ is used to denote the Euclidean distance of points P_i and P_j, and then takes their mean position. One may also consider a possible optimization problem to improve the result in combining the original data sets instead of the above.

Algorithm 9 Generalized Two-Step movement algorithm

Require: $A(x_A, y_A)$; ▷ Initial position of RP

function TwoStepExt($A(x_A, y_A)$)
  RP makes the first movement from A to D; obtaining $D(x_D, y_D)$;
  In parallel, MT makes the first movement from B to C; measuring AB, DC, BC, α;
  ▷ Compute two locations $B1$ and $B2$;
  $\{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\}$ = OneStepExt($A(x_A, y_A)$, $D(x_D, y_D)$, AB, DC, BC, α);
  RP makes the second movement from D to E; obtaining $E(x_E, y_E)$;
  In parallel, MT makes the second movement from C to F; measuring CF, EF, β; make sure that $\beta \neq \alpha \pm \pi$;
  ▷ Change in x-coordinate after the second move;
  $X = BC\cos\alpha + CF\cos\beta$;
  ▷ Change in y-coordinate after the second move;
  $Y = BC\sin\alpha + CF\sin\beta$;
  $BF = \sqrt{X^2 + Y^2}$;
  $\cos\gamma = X/BF$;
  $\sin\gamma = Y/BF$;
  Compute $\gamma \in [0; 2\pi)$ from $\cos\gamma$ and $\sin\gamma$;
  ▷ Compute two locations $B3$ and $B4$;
  $\{B3(x_{B3}, y_{B3}), B4(x_{B4}, y_{B4})\}$ = OneStepExt($A(x_A, y_A)$, $E(x_E, y_E)$, AB, EF ,BF, γ);
  ▷ Determine the MT location $B(x_B, y_B)$ from the set of $B1$, $B2$, $B3$ and $B4$;
  $B(x_B, y_B) = \{B1(x_{B1}, y_{B1}), B2(x_{B2}, y_{B2})\} \cap \{B3(x_{B3}, y_{B3}), B4(x_{B4}, y_{B4})\}$;
  return $B(x_B, y_B)$;
end function

D.6.3 Simulation Result

The performance of the G2SM method is investigated by simulation. Parameters used are the same as those used when studying 2SM (see Section D.5-C). Since the RP is also mobile (movable), we have to set the values of AD and DE. Here, we consider they are proportional to AB and in three movement ranges: $[0.1, 0.2] \times AB$ (i.e., small move), $[0.2, 0.5] \times AB$ (i.e., medium move), and $[0.5, 1] \times AB$ (i.e., large move). In the simulation, we consider that the RP moves in a distance which is at most AB (assuming that AB is the signal coverage range of the RP). Secondly, we consider $AD = DE$, for simplicity. However, the movement direction of the RP does

	$AD = DE = [0.1, 0.2] \times AB$			$AD = DE = [0.2, 0.5] \times AB$			$AD = DE = [0.5, 1] \times AB$		
	$e_d = 1\%$ $e_a = 1^\circ$	$e_d = 2\%$ $e_a = 2^\circ$	$e_d = 5\%$ $e_a = 5^\circ$	$e_d = 1\%$ $e_a = 1^\circ$	$e_d = 2\%$ $e_a = 2^\circ$	$e_d = 5\%$ $e_a = 5^\circ$	$e_d = 1\%$ $e_a = 1^\circ$	$e_d = 2\%$ $e_a = 2^\circ$	$e_d = 5\%$ $e_a = 5^\circ$
$AB = 1(BC = CF = 0.1AB)$	0.1139	0.2011	0.4121	0.0677	0.1189	0.2736	0.0461	0.0820	0.1808
$AB = 1(BC = CF = 0.2AB)$	0.0855	0.1518	0.3310	0.0597	0.1126	0.2432	0.0473	0.0832	0.1803
$AB = 1(BC = CF = 0.5AB)$	0.0509	0.0918	0.2119	0.0462	0.0877	0.1802	0.0413	0.0765	0.1678
$AB = 5(BC = CF = 0.1AB)$	0.5433	0.9654	2.1378	0.3346	0.6336	1.3726	0.2431	0.4196	0.9274
$AB = 5(BC = CD = 0.2AB)$	0.4162	0.7548	1.6035	0.3058	0.5562	1.1936	0.2361	0.4100	0.9044
$AB = 5(BC = CF = 0.5AB)$	0.2567	0.4452	1.0373	0.2349	0.4349	0.9156	0.2148	0.3852	0.8364
$AB = 10(BC = CF = 0.1AB)$	1.1319	1.9817	4.1819	0.7184	1.1993	2.7077	0.4064	0.7965	1.8415
$AB = 10(BC = CF = 0.2AB)$	0.8090	1.5358	3.2446	0.6189	1.2082	2.4429	0.4292	0.8366	1.8521
$AB = 10(BC = CF = 0.5AB)$	0.5059	0.9164	2.0490	0.4579	0.8819	1.9196	0.4539	0.7590	1.6926

Table D.4: G2SM with multi-sampling: the average positioning error (in meter) under various AB, AD, BC, and noise levels.

not need to be fixed and we thus generate it to be uniformly distributed in $(0; 2\pi]$. Note that the proposed algorithms are not limited to above numerical settings.

Table D.4 shows the simulation result of G2SM localization with multi-sampling. For each setup, we conduct 10^5 runs of simulation to obtain the average performance. Same as 2SM with multi-sampling, we perform 1000 sampling near the end point of each movement at G2SM. As expected, it can be seen that the estimation error in determining the position of the MT increases as the noise power increases. However, there is a correlation between the movement distance of the two devices (the MT and RP) and the resulting error. If the RP moves a bit (see $AD = DE = [0.1, 0.2] \times AB$) but the MT moves a lot (see the case $BC = CF = 0.5AB$), there is substantial error decrease. However, when RP moves a lot (see $AD = DE = [0.5, 1] \times AB$), the error is independent of how much the MT moves. Another interesting observation is that a substantial movement of either the MT or the RP is sufficient for achieving good performance. Overall, the average error is within about 15% of AB. In the best case, the average error is less than 5% of AB.

D.7 Conclusion

In this chapter, we have proposed a new method called Two-Step Movement (2SM) to estimate the position of MT. It requires only one reference point (RP) by exploiting useful information given by the position change of the MT or user movement. One can therefore reduce the number of RPs required and lower the system cost. Furthermore, a Generalization of the Two-Step Movement (G2SM) to Device-

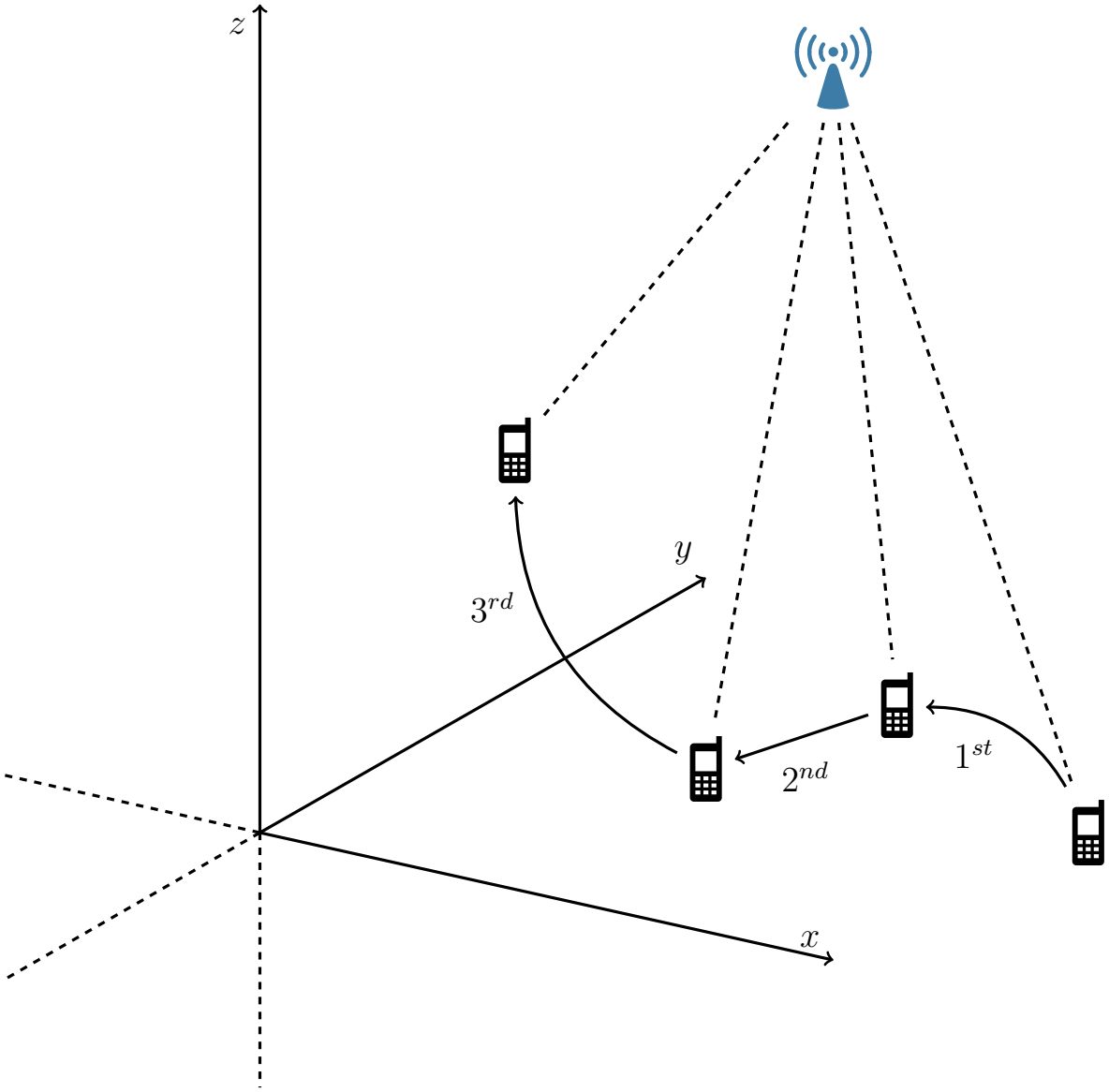


Figure D.14: Three-Step Movement (3SM) in three-dimensional (3D) space. The position of the device can be determined after three moves of the user carrying it.

To-Device context is also described as it allows the unique Reference Point to move or to be another mobile device. Analytical result shows that the user position can be derived and given in simple closed-form expression with low complexity. Simulation is conducted to study its performance under noisy environment. It is possible to achieve average error within about 10% of the distance between the RP and MT, or even less. Note that further analysis of noise impact and issues related to reflection and refraction of signals are important to improve the proposed method. Also in this

chapter, we first combine the 2SM method with multi-sampling technique to improve the positioning performance. Simulation result shows an error decrease of $15\% - 30\%$. Secondly, we propose the generalized localization method G2SM by utilizing device movement in which both the MT and RP are allowed to move. The position of MT can be determined analytically and in simple closed-form expression. Simulations are conducted to study its performance under various setup and noise levels. Results show that an average error within about 15% of the distance between the MT and RP can be realized. Since G2SM would allow a MT to locate itself through a peer mobile device, it has potential applications in future large D2D or multi-hop systems. Our method, thanks to the reliance on a single reference point, makes a lot of sense in the context of Internet of Things (IoT) such as home or business office area. It should be also noted that our method can be easily extended to localization in 3D coordinates (see Figure D.14). Together with practical implementation, they are subject to future work.

Bibliography

[1] I. L. Stats, "Internet usage & social media statistics," http://www.internetlivestats.com/, 2014.

[2] "The statistics portal for market data, market research and market studies," http://www.statista.com/.

[3] http://news.netcraft.com/archives/category/web-server-survey/.

[4] L. Page and S. Brin, "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 107–117, 1999.

[5] L. Page, S. Brin, M. Rajeev, and W. Terry, "The pagerank citation ranking: Bringing order to the web," Stanford University, Technical Report, 1998.

[6] M. Jelasity, G. Canright, and K. Eng-monsen, "Asynchronous distributed power iteration with gossip-based normalization," in *Euro-Par 2007, volume 4641 of Lecture Notes in Computer Science*. Springer-Verlag, 2007, pp. 514–525.

[7] Y. Wang and D. J. DeWitt, "Computing pagerank in a distributed internet search system," in *IN VLDB*, 2004, pp. 420–431.

[8] F. Mathieu and L. Viennot, "Local Aspects of the Global Ranking of Web Pages," in *6th International Workshop on Innovative Internet Community Systems (I2CS)*, Neuchâtel, Switzerland, Jun. 2006, pp. 493–506. [Online]. Available: https://hal.inria.fr/inria-00160799

[9] K. Avrachenkov and N. Litvak, "Decomposition of the google pagerank and optimal linking strategy," Enschede, the Netherlands, 2004, imported from MEMORANDA. [Online]. Available: http://doc.utwente.nl/65897/

[10] "The size of world wide web (the internet)," http://www.worldwidewebsize.com/.

[11] "Twitter," https://twitter.com/.

[12] "Facebook," https://facebook.com/.

[13] "Googleplus," https://plus.google.com/.

[14] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *J. Artif. Int. Res.*, vol. 10, no. 1, pp. 243–270, May 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622859.1622867

[15] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 591–600.

[16] M. Huang, Y. Yang, and X. Zhu, "Quality-biased ranking of short texts in microblogging services," in *In IJCNLP?11*, 2011.

[17] H. Huang, A. Zubiaga, H. Ji, H. Deng, D. Wang, H. Le, T. Abdelzaher, J. Han, A. Leung, J. Hancock, and C. Voss, "Tweet ranking based on heterogeneous networks," in *Proceedings of COLING 2012*. Mumbai, India: The COLING 2012 Organizing Committee, December 2012, pp. 1239–1256. [Online]. Available: http://www.aclweb.org/anthology/C12-1076

[18] P. Dey, A. Sinha, and S. Roy, "Social network analysis of different parameters derived from realtime profile," in *Distributed Computing and Internet Technology - 11th International Conference, ICDCIT 2015, Bhubaneswar, India, February 5-8, 2015. Proceedings*, 2015, pp. 452–455. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14977-6_50

[19] D. Hong, "Optimized on-line computation of pagerank algorithm," *CoRR*, vol. abs/1202.6158, 2012. [Online]. Available: http://arxiv.org/abs/1202.6158

[20] D. Hong, T. D. Huynh, and F. Mathieu, "D-iteration: diffusion approach for solving pagerank," 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01109532/document

[21] T. D. Huynh, D. Hong, G. Burnside, and F. Mathieu, "Analyzing methods computing pagerank vector of large matrix," 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01109536/document

[22] T. D. Huynh, F. Mathieu, and L. Viennot, "Liverank: How to refresh old crawls," in *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings*, 2014, pp. 148–160.

[23] ——, "LiveRank : comment faire du neuf avec du vieux ?" in *ALGOTEL 2014 – 16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Le Bois-Plage-en-Ré, France, Jun. 2014, pp. 1–4.

[24] T. D. Huynh, C. Shue Chen, and H. Siu-Wai, "Exploiting User Movement for Position Detection," in *IEEE Consumer Communications and Networking Conference*. Las Vegas, United States: IEEE, Jan. 2015, p. 6.

[25] ——, "Localization Method for Device-to-Device through User Movement," in *IEEE International Conference on Communications*, London, United Kingdom, 2015.

[26] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of the seventh international conference on World Wide Web 7*, ser. WWW7. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1998, pp. 107–117.

[27] L. Surhone, M. Tennoe, and S. Henssonow, *Perron-Frobenius Theorem*. VDM Publishing, 2010. [Online]. Available: https://books.google.fr/books?id=c1q5cQAACAAJ

[28] M. Bianchini, M. Gori, and F. Scarselli, "Inside pagerank," *ACM Trans. Internet Technol.*, vol. 5, no. 1, pp. 92–128, Feb. 2005.

[29] T. H. Haveliwala, "Topic-sensitive pagerank," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 517–526. [Online]. Available: http://doi.acm.org/10.1145/511446.511513

[30] A. N. Langville and C. D. Meyer, "Deeper inside pagerank," *Internet Mathematics*, vol. 1, p. 2004, 2004.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." no. 1999-66. Stanford InfoLab, 1999, Technical Report.

[32] S. Kamvar, T. Haveliwala, and G. Golub, "Adaptive methods for the computation of pagerank," Stanford InfoLab, Technical Report 2003-26, April 2003.

[33] T. Haveliwala, S. Kamvar, D. Klein, C. Manning, and G. Golub, "Computing pagerank using power extrapolation," Stanford InfoLab, Technical Report 2003-45, 2003.

[34] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Extrapolation methods for accelerating pagerank computations," in *In Proceedings of the Twelfth International World Wide Web Conference*. ACM Press, 2003, pp. 261–270.

[35] S. Abiteboul, M. Preda, and G. Cobena, "Adaptive on-line page importance computation," in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 280–290.

[36] C. Greif and D. Kurokawa", "A note on the convergence of sor for the pagerank problem," *"SIAM J. Scientific Computing"*, pp. 3201–3209, 2011.

[37] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Exploiting the block structure of the web for computing pagerank," Stanford InfoLab, Technical Report 2003-17, 2003. [Online]. Available: http://ilpubs.stanford.edu:8090/579/

[38] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[39] P. Boldi and S. Vigna, "The webgraph framework i: Compression techniques," in *In Proc. of the Thirteenth International World Wide Web Conference*. ACM Press, 2003, pp. 595–601.

[40] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on World Wide Web*. ACM Press, 2011.

[41] P. Boldi and S. Vigna, "The webgraph framework i: Compression techniques," in *In Proc. of the Thirteenth International World Wide Web Conference*. ACM Press, 2003, pp. 595–601.

[42] "Webgraph: Laboratory of webgraph algorithmic," http://webgraph.di.unimi.it/.

[43] I. Bordino, P. Boldi, D. Donato, M. Santini, and S. Vigna, "Temporal evolution of the uk web," in *ICDM Workshops*, 2008, pp. 909–918.

[44] B. D. Lubachevsky and D. Mitra, "A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius," *Journal of the ACM*, vol. 33, no. 1, pp. 130–150, 1986.

[45] O. Axelsson, *Iterative Solution Methods*. Cambridge University Press, 1996.

[46] H. A. van der Vorst, "Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 2, pp. 631–644, Mar. 1992. [Online]. Available: http://dx.doi.org/10.1137/0913035

[47] M. Najork and J. L. Wiener, "Breadth-first crawling yields high-quality pages," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 114–118. [Online]. Available: http://doi.acm.org/10.1145/371920.371965

[48] J. Cho and U. Schonfeld, "Rankmass crawler: A crawler with high personalized pagerank coverage guarantee," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07. VLDB Endowment, 2007, pp. 375–386. [Online]. Available: http://dl.acm.org/citation.cfm?id=1325851.1325897

[49] P. Boldi, M. Santini, and S. Vigna, "A large time-aware graph," *SIGIR Forum*, vol. 42, no. 2, pp. 33–38, 2008.

[50] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.

[51] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 591–600. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772751

[52] M. Gabielkov, A. Rao, and A. Legout, "Studying Social Networks at Scale: Macroscopic Anatomy of the Twitter Social Graph," in *ACM Sigmetrics 2014*, Austin, United States, Jun. 2014. [Online]. Available: https://hal.inria.fr/hal-00948889

[53] V. Grolmusz, "A note on the pagerank of undirected graphs," *CoRR*, vol. abs/1205.1960, 2012. [Online]. Available: http://arxiv.org/abs/1205.1960

[54] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse," in *USENIX Security 13*.

[55] ——, "Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse," in *USENIX Security 13*.

[56] M. Gabielkov, A. Rao, and A. Legout, "Studying Social Networks at Scale: Macroscopic Anatomy of the Twitter Social Graph," in *ACM Sigmetrics 2014*, Austin, United States, Apr. 2014. [Online]. Available: http://hal.inria.fr/hal-00948889

[57] C. Olston and M. Najork, "Web crawling," *Foundations and Trends in Information Retrieval*, vol. 4, no. 3, pp. 175–246, 2010.

[58] J. Cho and H. Garcia-Molina, "Effective page refresh policies for web crawlers," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 390–426, 2003.

[59] C. Olston and S. Pandey, "Recrawl scheduling based on information longevity," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 437–446.

[60] J. Cho and A. Ntoulas, "Effective change detection using sampling," in *VLDB '02*, 2002, pp. 514–525.

[61] Q. Tan, Z. Zhuang, P. Mitra, and C. L. Giles, "A clustering-based sampling approach for refreshing search engine's database," in *WebDB '07*, 2007.

[62] K. Radinsky and P. N. Bennett, "Predicting content change on the web," in *WSDM '13*. ACM, 2013, pp. 415–424.

[63] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins, "The discoverability of the web," in *WWW '07*. ACM, 2007, pp. 421–430.

[64] A. Java, X. Song, T. Finin, and B. Tseng, "Why we twitter: understanding microblogging usage and communities," in *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*. ACM, 2007, pp. 56–65.

[65] B. Krishnamurthy, P. Gill, and M. Arlitt, "A few chirps about twitter," in *Proceedings of the first workshop on Online social networks*. ACM, 2008, pp. 19–24.

[66] M. Gabielkov and A. Legout, "The complete picture of the twitter social graph," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 19–20.

[67] "Twitter graph 2010," https://dev.twitter.com/.

[68] "Colonel blotto game," http://en.wikipedia.org/wiki/Blotto_games/.

[69] G. Gartner and K. Rehrl, *Location Based Services and TeleCartography II: From Sensor Fusion to Context Models*. Lecture Notes in Geoinformation and Cartography, Springer, 2009.

[70] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Trans. Sys. Man Cyber - Part C*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.

[71] G. Mao, B. Fidan, and B. D. O. Anderson, "Wireless sensor network localization techniques," *Comput. Netw.*, vol. 51, no. 10, Jul. 2007.

[72] A. Kushki, K. N. Plataniotis, and A. N. Venetsanopoulos, *WLAN Positioning Systems: Principles and Applications in Location-Based Services*. Cambridge University Press, 2012.

[73] A. Cavallini, *iBeacons Bible*, http://meetingofideas.files.wordpress.com/2013/12/ibeacons-bible-1-0.pdf.

[74] M. Yasir, S.-W. Ho, and B. Vellambi, "Indoor localization using visible light and accelerometer," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 3341–3346.

[75] ——, "Indoor positioning system using visible light and accelerometer," *Journal of Lightwave Technology*, vol. 32, pp. 3306–3316, Oct 2014.

[76] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 1, pp. 13–32, Jan. 2009.

[77] M. Abdat, T.-C. Wan, and S. Supramaniam, "Survey on indoor wireless positioning techniques: Towards adaptive systems," in *International Conference on Distributed Framework and Applications*, Aug 2010.

[78] K. Al Nuaimi and H. Kamel, "A survey of indoor positioning systems and algorithms," in *International Conference on Innovations in Information Technology*, April 2011, pp. 185–190.

[79] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, 2013.

[80] P. Mirowski, D. Milioris, P. Whiting, and T. Kam Ho, "Probabilistic radio-frequency fingerprinting and localization on the run," *Bell Labs Technical Journal*, vol. 18, no. 4, pp. 111–133, Feb. 2014.

[81] R. Peng and M. Sichitiu, "Angle of arrival localization for wireless sensor networks," in *IEEE SECON*, Sep 2006, pp. 374–382.