UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE: INFORMATIQUE
Laboratoire de Recherche en Informatique (LRI)

Discipline: *INFORMATIQUE*

THÈSE DE DOCTORAT SUR TRAVAUX

soutenue le 05/07/2013
par

Christian KONRAD

Computations on Massive Data Sets: Streaming Algorithms and Two-Party Communication

Introduction française

Titre français: Calculs sur des Grosses Données: Algorithmes de Streaming et Communication entre Deux Joueurs

Directeur de thèse:	Frédéric Magniez	Directeur de recherche (CNRS, Université Paris Diderot)
Composition du jury		
Rapporteurs:	Christoph Dürr	Directeur de recherche (CNRS, Université P. et M. Curie)
	Andrew McGregor	Assistant Professor (University of Massachusetts, Amherst)
Examinateurs:	Magnús Halldórsson	Professor (Reykjavik University)
	Claire Mathieu	Directrice de recherche (CNRS, ENS Paris)
	Alain Denise	Professeur (Université Paris-Sud)

Abstract

Aujourd'hui, le traitement des grosses données est un enjeu important en informatique. Dans cette thèse on considère deux modèles de calcul qui abordent des problèmes qui se posent lors du traitement des grosses données.

Le premier modèle est le *modèle de streaming*. Les algorithmes classiques supposent que l'accès aux données peut se faire de façon aléatoire. Lors du traitement des grosses données, un tel accès est trop couteux. Pour cette raison, les algorithmes de streaming ont un accès restreint aux données: ils lisent les données de façon sequentielle (par passage) une fois ou peu de fois. De plus, les algorithmes de streaming utilisent une mémoire d'accès aléatoire de taille sous-linéaire dans la taille des données. L'accès séquentiel et la mémoire de taille sous-linéaire sont des limitations drastiques. Le but principal de cette thèse est l'exploration des limites et des capacités du modèle de streaming.

Le deuxième modèle est le *modèle de communication*. Lors du traitement des données par plusieurs entités de calcul situées à des endroits différents, l'échange des message pour la synchronisation de leurs calculs est souvent un goulet d'étranglement. Pour cette raison, il est préférable de minimiser la quantité de communication. Un modèle particulier est la communication à sens unique entre deux participants. Dans ce modèle, deux participants calculent un résultat en fonction des données qui sont partagées entre eux et la communication se réduit à un seul message du premier participant au deuxième. Ce modèle a des liens forts avec le modèle de streaming: une borne inférieure sur la taille du message est également une borne inférieure sur la taille de la mémoire d'un algorithme de streaming. Étudier les algorithmes de streaming du point de vue de communication permet d'enrichir notre compréhension des algorithmes de streaming.

Dans ce travail on étudie les quatres problèmes suivants dans le contexte du streaming et de la communication:

1. *Les couplages dans le modèle de streaming.* L'entrée du problème est un flux d'arêtes d'un graphe $G = (V, E)$ avec $n = |V|$. On recherche un algorithme de streaming qui calcule un couplage de grande taille en utilisant une mémoire de taille $O(n \operatorname{polylog} n)$. L'algorithme glouton remplit ces contraintes et calcule un couplage de taille au moins $1/2$ fois la taille d'un couplage maximum. Une question ouverte depuis longtemps demande s'il est possible de calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum en un passage si aucune hypothèse sur l'ordre des

arêtes dans le flux est faite. Nous montrons qu'il est possible de calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum en un passage si les arêtes du graphe sont dans un ordre uniformément aléatoire. De plus, nous montrons qu'avec deux passages on peut calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum sans contraintes sur l'ordre des arêtes.

2. *Les semi-couplages en streaming et en communication à sens unique entre deux participants.* Un semi-couplage dans un graphe biparti $G = (A, B, E)$ est un sous-ensemble d'arêtes qui couple tous les sommets de type A exactement une fois aux sommets de type B de façon pas forcement injective. L'objectif est de minimiser le nombre de sommets de type A qui sont couplés au même sommets de type B. Ce problème est équivalent au problème d'ordonnancement des tâches de même longueur sur des machines équivalentes tout en respectant des contraintes d'affectation définies par des arêtes d'un graphe biparti. Pour ce problème, nous montrons un algorithme de streaming qui trouve un bon compromis entre la taille de la mémoire et le facteur d'approximation: pour tout $0 \leq \epsilon \leq 1$, notre algorithme calcule une $O(n^{(1-\epsilon)/2})$-approximation en utilisant une mémoire de taille $\tilde{O}(n^{1+\epsilon})$. De plus, nous montrons des bornes supérieures et des bornes inférieurs pour la complexité de communication entre deux participants pour ce problème et des nouveaux résultats concernant la structure des semi-couplages.
3. *Validité des fichiers XML dans le modèle de streaming.* Un fichier XML (eXtended Markup Language) est une séquence de balises ouvrantes et fermantes. Une DTD (Document Type Definition) est un ensemble de contraintes de validité locales d'un fichier XML. Nous étudions des algorithmes de streaming pour tester si un fichier XML satisfait les contraintes décrites dans une DTD. Notre résultat principal est un algorithme de streaming qui fait $O(\log n)$ passages, utilise 3 flux auxiliaires et une mémoire de taille $O(\log^2 n)$ (n étant la taille du fichier XML). De plus, pour le problème de validation des fichiers XML qui décrivent des arbres binaires, nous présentons un algorithme en un passage et en mémoire $\tilde{O}(\sqrt{n})$ ainsi qu'un algorithme qui utilise une mémoire de taille $O(\log^2 n)$ et qui fait un passage de gauche à droite et un passage de droite à gauche.
4. *Correction d'erreur pour la distance du cantonnier.* On étudie le problème de communication à sens unique suivant: Alice et Bob ont des ensembles de n points sur une grille en d dimensions décrite par $[\Delta]^d$ pour un entier Δ. Alice envoie un échantillon de petite taille à Bob qui, après réception, déplace ses points pour que la distance du cantonnier entre les points d'Alice et les points de Bob diminue. La distance du cantonnier entre deux ensembles de n points est le poids du couplage parfait de poids minimal, où la contribution d'un couplage de deux points est la distance euclidienne entre ces points.

   Pour tout $k > 0$ nous montrons qu'il y a un protocole de communication avec coût de communication $\tilde{O}(kd)$ tel que les déplacements des points effectués

par Bob aboutissent à un facteur d'approximation de $O(d)$ par rapport aux meilleurs déplacements de d points. De plus, nous montrons que notre protocole est presque optimal.

Contents

Chapter 1

Introduction

En 2010, la quantité de données créées ou répliquées étaient plus que un zettabyte (un milliard de terabytes) [GR11]. Aujourd'hui, on est confronté à une explosion des données immense. Des expériences dans la domaine de physique des particules au centre de recherche CERN génèrent jusqu'à un petabyte (un million de terabytes) chaque seconde et bien que presque toutes ces données soient effacées à l'instant, jusqu'à 25 petabytes de données sont sauvegardées chaque jour[1]. Le centre de calcul climatique allemand (Deutsches Klimarechenzentrum, DKRZ) utilise une base de données de 60 petabytes de données climatiques[2]. Google, Youtube, Facebook et des autres compagnies Internet ont des bases de données de plusieurs petabytes pour y stocker des textes, des vidéos, de la musique, des photos et des statistiques des utilisateurs[3]. Entre autres, les dispositifs de stockage moins chers ont rendu possible le phénomène du *Big Data* [GR11].

Actuellement, l'extraction des informations utiles des grosses données est une tâche importante et cela constitue un véritable enjeu pour l'informatique moderne. Le fait que les données ne sont plus stockées localement mais plutôt éparpillées partout dans l'Internet ou même effacées immédiatement après leur création pose des grands problèmes pour les algorithmes qui les traitent. Dans cette thèse de doctorat on considère deux modèles de calcules qui affrontent ces problèmes. Ces modèles de calcul sont le *modèle de streaming* (modèle des flux) et le *modèle de la communication*. Le modèle de streaming s'adresse au problème d'*accès aux données*. Dans le modèle de la communication, on étudie la quantité de communication nécessaire lorsque plusieurs unités de calcul traitent des grosses données de façon distribuée. L'accès aux données et la communication sont des problèmes fondamentaux dans le domaine des algorithmes de grosses données et souvent ils se constituent en des goulots d'étranglements. Figure 1.1 illustre ces deux problèmes.

Les algorithmes qui traitent des grosses données doivent s'occuper du problème d'accès aux données. Aujourd'hui, les ordinateurs standards possèdent une mémoire vive de taille jusqu'à 16 gigabytes. Un algorithme peut rapidement accéder aux données stockées dans cette mémoire de façon aléatoire. Cependant, lors du traitement des données de taille beaucoup plus

[1] http://www.v3.co.uk/v3-uk/news/2081263/cern-experiments-generating-petabyte

[2] http://www.treehugger.com/clean-technology/meet-the-worlds-most-powerful-weather-supercomputer.html

[3] http://www.comparebusinessproducts.com/fyi/10-largest-databases-in-the-world

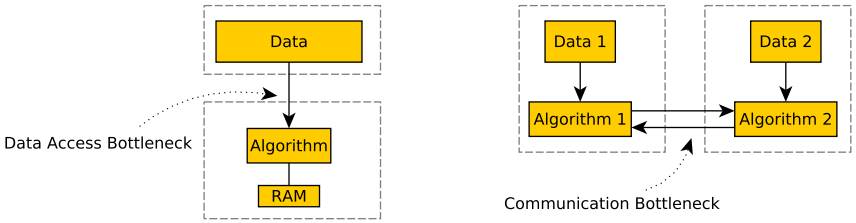

Figure 1.1: Gauche: Un algorithme qui traite des grosses données doit s'occuper du problème d'accès aux données. Droite: Si les données sont traitées de façon distribuée, il est préférable de minimiser la quantité de communication.

grande que la taille de la mémoire vive d'un ordinateur, on ne peut pas supposer que l'accès aux données peut se faire de façon aléatoire: Les grosses données sont beaucoup trop larges pour que la mémoire vive d'un ordinateur puisse les contenir. Cela exclut donc la possibilité de transférer les données dans la mémoire vive avant leur traitement. De plus, si les données sont stockées sur un périphérique externe ou sur un ordinateur qui n'est connecté que par un réseau lent comme l'Internet, l'accès aux données est coûteux et il faut considérer des alternatives. L'alternative qu'on considère dans cette thèse est le modèle de streaming. Un algorithme de streaming reçoit un flux de données. Lors l'arrivée des données, l'algorithme de streaming les traite immédiatement et il utilise une petite mémoire vive qui est souvent de taille polylogarithmique en taille des données. Dans la Section 1.1 on donnera une introduction plus détaillée aux algorithmes de streaming.

Un autre problème se pose lorsque plusieurs entités de calcul qui sont situés à des endroits différents traitent des grosses données de façon collaborative. Dans cette situation, les participants au calcul échangent des messages pour coordonner leurs calculs. Il faut remarquer que si la taille des données était petite, les participants pouvaient transmettre leurs données à un collaborateur central qui ensuite pourrait traiter toutes les données de façon locale. Cependant, dans beaucoup d'applications, la transmission des données est un goulot d'étranglement. Ceci est surtout vrai dans le cas des grosses données, car habituellement les réseaux connecteurs sont lents. Pour cette raison, la quantité des messages échangés devrait être minimisée. La *complexité de la communication* est le domaine de l'informatique qui s'intéresse à la quantité de communication nécessaire pour mener à bien le calcul d'une fonction sur des données distribuées de façon collective. On donnera une brève introduction à la complexité de la communication dans la Section 1.2.

1.1 Algorithmes de Streaming

Un objectif de cette thèse de doctorat est l'exploration des limites et des capacités du *modèle de streaming*. Les *algorithmes de streaming* appartiennent à la catégorie des algorithmes des grosses données. Ils s'attaquent au problème que souvent les données sont trop grandes pour qu'ils rentrent dans la mémoire vive d'un ordinateur. Un algorithme de streaming utilise une mémoire vive de taille sous-linéaire dans la taille des données.

Le modèle de streaming se caractérise également par une restriction de la méthode d'accès aux données à un accès séquentiel. Tandis que dans le modèle RAM un algorithme peut accéder aux données en ordre aléatoire, un algorithme de streaming lit les données par passages de *gauche* à *droite*. Figure 1.2 illustre ceci.

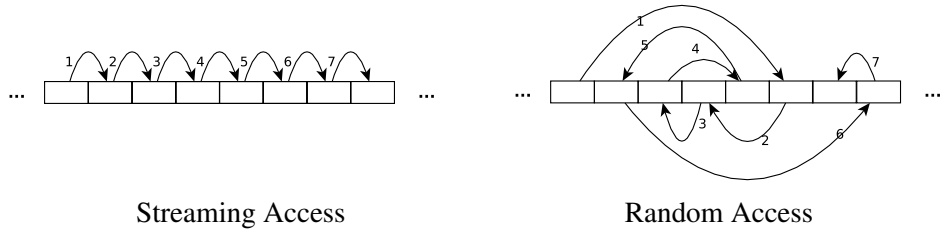

Figure 1.2: Accès Aléatoire et Accès Séquentiel. Les algorithmes de streaming accèdent aux données de façon séquentielle.

Limiter la taille de la mémoire vive est nécessaire pour obtenir une définition significative du modèle de streaming: Si la mémoire vive était de taille linéaire dans la taille des données, un algorithme de streaming pourrait copier toutes les données dans la mémoire vive en utilisant un passage sur les données. Ensuite, il pourrait traiter la copie des données avec un accès aux données aléatoire.

Visionner des flux de données vidéo depuis l'Internet sont un bon exemple pour un algorithme de streaming. Bien que souvent les fichiers vidéo soient large, à tout moment de la visualisation, le visionneur de la vidéo n'a besoin que de la petite partie de la vidéo qui encode les images et le son du moment. La vidéo est donc téléchargée et elle arrive au visionneur comme un flux de données. Le visionneur montre les données au moments de leurs arrivées et ensuite il les jette immédiatement. Cette manière de procéder permet de visualiser des vidéos de grande taille en utilisant une mémoire vive de petite taille.

Le but principal dans la conception des algorithmes de streaming est de minimiser la taille de la mémoire vive. On illustre cela avec l'exemple suivant:

Problem 1 (Compter le nombre d'éléments distincts)**.** *Soit $A = A[1], \ldots, A[n]$ une séquence de n éléments et $\mathcal{U}$ un ensemble fini. Combien d'éléments distincts y-a-t il dans A?*

On note $F_0(A)$ le nombre d'éléments distincts dans A. Le nombre d'éléments distincts est également le 0ème *moment de fréquence* d'un ensemble de fréquences. Les moments de fréquence sont définis comme suit. Pour tout $u \in \mathcal{U}$ on note $f_A(u)$ le nombre d'occurrences de u dans A. Le $i$ème moment de fréquence F_i est défini comme $F_i(A) := \sum_{u \in \mathcal{U}} f_A(u)^i$.

Dans le modèle RAM, on peut facilement obtenir un algorithme pour compter le nombre d'éléments distincts qui utilise peu d'espace comme suit. L'algorithme utilise un compteur et il vérifie pour tout $u \in \mathcal{U}$ si u fait partie de A en traversant A de façon séquentielle. Cette stratégie nécessite une mémoire de taille $O(\log |\mathcal{U}|)$ et elle engendre un algorithme de streaming qui fait $|\mathcal{U}|$ passages sur A. Généralement, dans le modèle de streaming on souhaite obtenir des algorithmes qui font un nombre de passages constantes sur les données et le cas spécial d'un

seule passage est particulièrement intéressant: Combien de mémoire nécessite un algorithme de streaming qui fait un seule passage sur le flux de données?

Un algorithme de streaming en un passage pourrait stocker un bit pour tout $u \in \mathcal{U}$ et il vérifierait si u apparaît dans le flux. Cet algorithme nécessite une mémoire de taille $O(|\mathcal{U}|)$ et il s'avère que cet algorithme est optimal parmi les algorithmes déterministes: Dans [AMS96] il est montré que tout algorithme de streaming en un passage qui sort un nombre Y tel que $|Y - F_0(A)| \leq 0.1F_0(A)$ nécessite une mémoire de taille $\Omega(|\mathcal{U}|)$.

Cette borne inférieure souligne deux propriétés caractéristiques pour le modèle de streaming. Premièrement, pour beaucoup de problèmes, tout algorithme de streaming déterministe en un passage nécessite une mémoire linéaire et la randomisation est un outil nécessaire pour obtenir des algorithmes de streaming en mémoire sous-linéaire. Deuxièmement, la précédente borne inférieure utilise la notion d'*approximation*. En streaming, il est rarement possible de résoudre un problème de façon exacte en utilisant une mémoire sous-linéaire et en conséquence on conçoit des algorithmes d'approximation. Dans le cas du comptage des éléments distincts il y a un algorithme de streaming randomisé optimal en un passage qui utilisé une mémoire de taille $O(1/\epsilon^2 + \log |\mathcal{U}|)$ et il calcule une $(1 \pm \epsilon)$-approximation, cet-à-dire il calcule une valeur Y tel que $|Y - F_0(A)| \leq \epsilon F_0(A)$ pour tout $\epsilon > 0$. Ce résultat est grâce à Kane et al. [KNW10] et il conclut une longue séquence de recherche sur ce problème qui était initiée par Flajolet et Martin en 1983 [FM83].

1.2 Complexité de la Communication

La *Complexité de la Communication* est un domaine de l'informatique qui a des forts liens avec les algorithmes de streaming. Dans le modèle de communication, la donnée est partagée parmi plusieurs joueurs. Les joueurs échangent des messages pour pouvoir collectivement calculer une fonction qui dépend de toute la donnée. En complexité de communication, le but est de concevoir des protocoles de communication avec le moins de communication possible et de prouver des bornes inférieurs sur la quantité de communication nécessaire pour calculer une fonction. La *complexité de la communication déterministe/randomisé de k joueurs* d'une fonction f est la quantité de communication minimale nécessaire pour tout protocole déterministe/randomisé exécuté par les k joueurs pour calculer f.

Considérons le problème de deux joueurs de déterminer si deux chaînes de n bit sont égales. Ce problème est illustré dans Figure 1.3 et on le définit comme suit:

Problem 2 (Égalité de deux chaînes de n bits)**.** *Alice a une chaîne binaire $X \in \{0,1\}^n$ et Bob a une chaîne binaire $Y \in \{0,1\}^n$. Alice et Bob échangent des message pour pouvoir déterminer si $X = Y$. Combien de communication est nécessaire?*

Regardons le protocole déterministe suivant pour le problème d'égalité. Alice envoie le bit $X[1]$ à Bob et ensuite Bob vérifie si $X[1] = Y[1]$. Si $X[1] \neq Y[1]$, Bob signale que les deux chaînes sont différentes. En cas d'égalité, Bob envoie le bit $Y[2]$ à Alice et elle le compare à $X[2]$. Si $X[2] \neq Y[2]$ elle signale que les deux chaînes sont différentes. Alice et Bob répètent cette démarche jusqu'à ce qu'un indice i tel que $X[i] \neq Y[i]$ a été trouvé ou tous les bits ont été comparés.

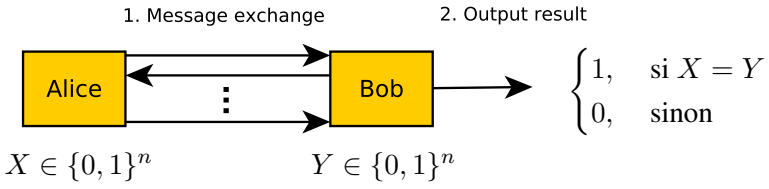

Figure 1.3: Two-Party Communication Setting. Here, Alice and Bob both hold n-bit strings and they want to determine whether their strings are equal or not. Alice and Bob communicate in order to achieve this task. In this illustration, Bob outputs the result.

Clairement, dans le cas ou $X = Y$, le coût de communication du protocole est $\Omega(n)$. Regardons le cas $X \neq Y$. Dans le cas pire, le coût de communication de ce protocole est toujours $\Omega(n)$ car il se peut que $X[i] = Y[i]$ pour tout $i < n$ et $X[n] \neq Y[n]$. Un résultat connu est que la complexité de communication déterministe de la fonction d'égalité est $\Omega(n)$ (par exemple [KN97]), et notre protocole est donc optimal.

Pourtant on peut considérablement réduire la quantité de communication en utilisant de la randomisation. Soit h une fonction de hachage appropriée. Alice calcule la valeur $h(X)$ et elle l'envoie à Bob. Bob vérifie si $h(X) = h(Y)$. Il y a des fonctions de hachage h tel que $|h(X)| \in O(\log n)$, si $X \neq Y$ la probabilité de $h(X) = h(Y)$ est au plus $1/n$ et si $X = Y$ alors $h(X) = h(Y)$ toujours. Une telle fonction de hachage engendre un protocole de communication randomisé de deux joueurs avec un coût de communication de $O(\log(n))$. Pour plus d'informations, regarder [KN97]. Cet exemple montre qu'il y a des fonction telle qu'il y a un écart exponentiel entre la complexité de la communication déterministe et la complexité de la communication randomisé.

Dans cette thèse, on s'intéresse principalement à la *communication à sens unique*. Dans cette situation, la donnée est partagée entre k joueurs et la communication est à sens unique: Joueur un envoie un message à joueur deux qui ensuite envoie un message à joueur trois. Cette procédure se répète jusqu'à ce que joueur k reçoit un message de joueur $k-1$ et joueur k sort le résultat du protocole. Considérons le cas $k = 2$ et on notons les joueurs *Alice* et *Bob*. Dans cette situation, Alice calcule un message en fonction de sa partie de la donnée et elle l'envoie à Bob. Ensuite, Bob calcule le résultat de la fonction en fonction de sa partie de la donnée et le message qu'il a reçu d'Alice. Le précédent protocole randomisé pour la fonction d'égalité est un protocole à deux joueurs à sens unique.

Il y a un lien inhérent entre la communication à sens unique et les algorithmes de streaming à un passage: Un algorithme de streaming à un passage et mémoire s pour un problème P engendre un protocole de communication à sens unique à deux joueurs avec un message de taille $O(s)$ pour le problème P si le flux des données est partagé entre les deux joueurs tel que Alice reçoit le préfixe et Bob reçoit le suffixe. Ça se voit comme suit. Alice déroule l'algorithme de streaming sur sa partie de la donnée et ensuite elle envoie l'état de sa mémoire à Bob. Puis, Bob initialise sa mémoire avec le message d'Alice et il continue l'algorithme de streaming sur sa partie de la donnée. Ce lien implique qu'une borne inférieure sur la taille du message dans un protocole à sens unique et à deux joueurs et également une borne inférieure

sur la taille de la mémoire d'un algorithme de streaming. Pour cette raison, il est éclairant d'étudier les algorithmes de streaming du point de vue de la complexité de la communication.

Nous démontrons ce lien à l'aide de la fonction d'égalité décrite ci-dessus. Supposons qu'un algorithme de streaming avec mémoire s et en un passage sait décider si la première moitié d'un flux de bit A de longueur $2n$ et égale à la deuxième moitié, cet-à-dire $\forall 1 \leq i \leq n : A[i] = A[i+n]$. Un tel algorithme de streaming engendre un protocole de communication à deux joueurs et à sens unique pour la fonction d'égalité avec un coût de communication s. Comme avant, appelons X la donnée d'Alice et Y la donnée de Bob. Alice déroule l'algorithme de streaming sur X et ensuite elle envoie l'état de sa mémoire à Bob. Bob initialise sa mémoire avec le message reçu d'Alice et il continue l'algorithme de streaming sur Y. Ceci est illustré dans la Figure 1.4.

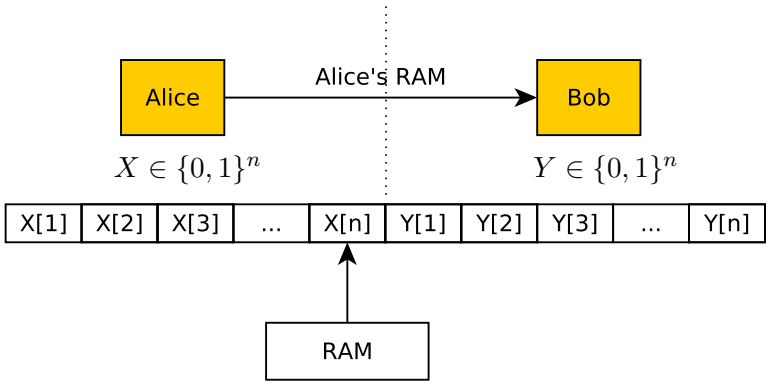

Figure 1.4: Lien entre les algorithmes de streaming et la communication entre deux joueurs. Un algorithme de streaming à un passage et mémoire s pour un problème P engendre un protocole de communication à sens unique de deux joueurs avec un message de taille $O(s)$ pour le problème P si le flux de donnée est partagé entre les deux joueurs tel que Alice reçoit le préfixe et Bob reçoit le suffixe. Alice déroule l'algorithme de streaming sur sa donnée et ensuite elle envoie l'état de sa mémoire à Bob. Puis, Bob initialise sa mémoire avec le message d'Alice et il continue l'algorithme de streaming sur sa donnée.

Comme nous l'avons déjà discuté, la complexité de la communication déterministe de la fonction d'égalité est $\Omega(n)$. Comme la communication à sens unique entre deux joueurs est un cas spécial de la communication à deux joueurs, on en déduit que la complexité de la communication à sens unique entre deux joueurs de la fonction d'égalité est également $\Omega(n)$. Un algorithme de streaming déterministe pour le problème d'égalité nécessite donc également une mémoire de $\Omega(n)$. Nous répétons l'argumentation précédente: Si il y avait un algorithme de streaming déterministe avec une mémoire $o(n)$ pour le problème d'égalité, alors cet algorithme engendrait un protocole de communication déterministe à sens unique et à deux joueurs avec un message de taille $o(n)$. Cependant, c'est une contradiction car il y a une borne inférieure linéaire sur la complexité de la communication déterministe à sens unique et à deux joueurs pour le problème d'égalité.

Dans cette thèse, nous étudions plusieurs problèmes dans le modèle de streaming et de communication. Dans la section suivante nous introduisons brièvement ces problèmes pour que le lecteur obtienne une vue global sur les domaines de recherche considérés dans ce document.

1.3 Problèmes Considérés

Les couplages dans le modèle de streaming. L'entrée du problème est un flux d'arêtes d'un graphe $G = (V, E)$ avec $n = |V|$. On recherche un algorithme de streaming qui calcule un couplage de grande taille en utilisant une mémoire de taille $O(n \operatorname{polylog} n)$. L'algorithme glouton remplit ces contraintes et calcule un couplage de taille au moins $1/2$ fois la taille d'un couplage maximum. Une question ouverte depuis longtemps demande s'il est possible de calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum en un passage si aucune hypothèse sur l'ordre des arêtes dans le flux est faite. Nous montrons qu'il est possible de calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum en un passage si les arêtes du graphe sont dans un ordre uniformément aléatoire. De plus, nous montrons qu'avec deux passages on peut calculer un couplage de taille strictement supérieur à $1/2$ fois la taille d'un couplage maximum sans contraintes sur l'ordre des arêtes.

Les semi-couplages en streaming et en communication à sens unique entre deux participants. Un semi-couplage dans un graphe biparti $G = (A, B, E)$ est un sous-ensemble d'arêtes qui couple tous les sommets de type A exactement une fois aux sommets de type B de façon pas forcement injective. L'objectif est de minimiser le nombre de sommets de type A qui sont couplés au même sommets de type B. Ce problème est équivalent au problème d'ordonnancement des tâches de même longueur sur des machines équivalentes tout en respectant des contraintes d'affectation définies par des arêtes d'un graphe biparti. Pour ce problème, nous montrons un algorithme de streaming qui trouve un bon compromis entre la taille de la mémoire et le facteur d'approximation: pour tout $0 \leq \epsilon \leq 1$, notre algorithme calcule une $O(n^{(1-\epsilon)/2})$-approximation en utilisant une mémoire de taille $\tilde{O}(n^{1+\epsilon})$. De plus, nous montrons des bornes supérieures et des bornes inférieurs pour la complexité de communication entre deux participants pour ce problème et des nouveaux résultats concernant la structure des semi-couplages.

Validité des fichiers XML dans le modèle de streaming. Un fichier XML (eXtended Markup Language) est une séquence de balises ouvrantes et fermantes. Une DTD (Document Type Definition) est un ensemble de contraintes de validité locales d'un fichier XML. Nous étudions des algorithmes de streaming pour tester si un fichier XML satisfait les contraintes décrites dans une DTD. Notre résultat principal est un algorithme de streaming qui fait $O(\log n)$ passages, utilise 3 flux auxiliaires et une mémoire de taille $O(\log^2 n)$ (n étant la taille du fichier XML). De plus, pour le problème de validation des fichiers XML qui décrivent des arbres binaires, nous présentons un algorithme en un passage et en mémoire $\tilde{O}(\sqrt{n})$ ainsi qu'un algorithme qui utilise une mémoire de taille $O(\log^2 n)$ et qui fait un passage de gauche à droite et un passage de droite à gauche.

Correction d'erreur pour la distance du cantonnier. On étudie le problème de commu-

nication à sens unique suivant: Alice et Bob ont des ensembles de n points sur une grille en d dimensions décrite par $[\Delta]^d$ pour un entier Δ. Alice envoie un échantillon de petite taille à Bob qui, après réception, déplace ses points pour que la distance du cantonnier entre les points d'Alice et les points de Bob diminue. La distance du cantonnier entre deux ensembles de n points est le poids du couplage parfait de poids minimal, où la contribution d'un couplage de deux points est la distance euclidienne entre ces points.

Pour tout $k > 0$ nous montrons qu'il y a un protocole de communication avec coût de communication $\tilde{O}(kd)$ tel que les déplacements des points effectués par Bob aboutissent à un facteur d'approximation de $O(d)$ par rapport aux meilleurs déplacements de d points. De plus, nous montrons que notre protocole est presque optimal.

Chapter 2

Modèle de Streaming

Un flux de donnée de longueur n est une séquence d'éléments $A = A[1] \dots A[n]$ et chaque élément $A[i]$ appartient à un ensemble $\mathcal{U}$. L'ensemble $\mathcal{U}$ dépend de l'application précise. Il se peut que ça soit des nombres, des lettres, des arêtes d'un graphe, des balises XML ou tout autre ensemble fini. Dans cette thèse, on suppose qu'on peut accéder à un élément du flux $A[i]$ en temps $O(1)$. De plus, pour rendre la présentation des résultat plus facile, on suppose que la longueur du flux est connue en avance à nos algorithmes de streaming. Ceci n'est pas une vraie limitation: On peut légèrement modifier nos algorithmes pour qu'ils marchent sans connaître n en avance.

Commençons par formellement définir un algorithme de streaming.

Definition 1 (Algorithme de Streaming)**.** *Un* algorithme de streaming ***S*** *à $p(n)$ passages, mémoire $s(n)$ et temps de mis-à-jour $t(n)$ est un algorithme tel que pour tout flux de donnée $A = A[1] \dots A[n]$ avec $A[i] \in \mathcal{U}$ et ensemble $\mathcal{U}$:*

1. ***S*** *fait au plus $p(n)$ passages sur le flux A,*
2. ***S*** *utilise une mémoire vive de taille $s(n)$,*
3. ***S*** *ne dépasse pas le temps $O(t(n))$ entre deux lectures consécutives des éléments du flux.*

De plus, le temps de prétraitement (le temps avant la lecture de $A[1]$) et le temps de posttraitement (le temps après la dernière lecture du flux et la sortie du résultat) est $O(t(n))$. Nous supposons que la lecture d'un élément du flux se fait en temps constant.

2.1 Brève Histoire et Applications

Les algorithmes de streaming s'appliquent lorsque la mémoire vive d'un ordinateur est trop petite pour contenir toute la donnée. La situation la plus naturelle est celle du traitement des *vrais* flux de donnée. Considérons un routeur de réseau qui doit prendre des statistiques telles que le nombre des adresses IP différentes des paquets IP qui passent par lui (problème du nombre des éléments distincts). Dans une telle situation, l'application impose un accès linéaire aux données et seulement un seule passage sur les données est possible.

Un autre domaine d'application est le traitement des grosses données stockées sur des disques externes ou des ordinateurs qui sont connectés par un réseau local ou l'Internet. C'est une situation typique pour des bases de données. Dans une telle situation, l'accès aléatoire aux données est très coûteux et l'accès aux données par flux est une bonne alternative qui se fait en pratique. Des tels scénarios justifient également l'étude des algorithmes de streaming à plusieurs passages car la donnée peut être relue plusieurs fois.

Les algorithmes de streaming ont été étudiés depuis 1978. Munro et Paterson [MP78] ont étudié la relation entre le nombre de passages et la taille de la mémoire nécessaire pour choisir le $k$ième plus large élément d'un flux. Ce travail et le travail de Flajolet and Martin [FM83] sur le problème du comptage approximatif sont parmi les premiers travaux sur les algorithmes de streaming. Un jalon dans le domaine des algorithmes de streaming est l'article "The space complexity of approximating the frequency moments" de Alon, Matias et Szegedy [AMS96] qui ont reçu le prix Gödel en 2005 pour ce travail. Citons le comité: " This concise work laid the foundations of the analysis of data streams using limited memory."[1]. Entre-temps ce papier a été cité plus que 1000 fois.

Aujourd'hui, il y a un grand nombre d'articles qui portent sur les algorithmes de streaming. Des problèmes concrets parvenus des domaines comme la statistique, la bio-informatique, les algorithmes de textes, la géométrie algorithmique, des problèmes de graphe, l'algèbre linéaire, des bases de données et beaucoup d'autres ont été étudiés dans le modèle de streaming. [Mut05] donne une excellente vue globale sur les algorithmes de streaming. Une liste de problèmes ouverts se trouve sur la page internet http://sublinear.info/.

2.2 Flux de Graphe

Definition 2 (Flux de Graphe)**.** *Un* flux de graphe *est une séquence des arêtes d'un graphe.*

Pour assurer une présentation claire, nous supposons que nos algorithmes de streaming qui traitent des flux de graphe connaissent le nombre de nœuds et le nombre d'arêtes du graphe.

Premièrement, les flux de graphe étaient étudiés par Henzinger et al. [HRR99] et aujourd'hui ils constituent un domaine de recherche très actif. Beaucoup de problèmes de graphe comme le problème de l'ensemble indépendant [HHLS10], des problèmes de couplage [McG05, GKK12, KMM12], compter des triangles [JG05, BFL$^+$06] et des des sous-graphes arbitraires [KMSS12], la connexité de nœuds dans des graphes orientés [GO13], la sparsification [AGM12] et beaucoup d'autres ont été étudiés dans le modèle de streaming.

Feigenbaum et al. ont montré dans [FKM$^+$05] que vérifier des propretés simples comme la connexité nécessite une mémoire de taille $\Omega(n)$ (n est le nombre des nœuds). Ceci justifie l'étude du *modèle de semi-streaming* qui était introduit par Muthukrishnan [Mut05]. Dans le modèle de semi-streaming, un algorithme de streaming a le droit d'utiliser une mémoire de taille $O(n \operatorname{polylog} n)$. Nous écrirons $\tilde{O}(n)$ pour $O(n \operatorname{polylog} n)$.

Les difficultés des problèmes de graphes dans le modèle de streaming dépendent fortement de l'ordre d'arrivée des arêtes. Au moins les quatre ordres d'arrivées suivantes ont été étudiées:

[1] Cette citation est prise de: http://www.sigact.org/Prizes/Godel/2005.html

1. **Ordre adversaire.** Un algorithme ne fait aucune hypothèse sur l'ordre d'arrivée des arêtes. La plupart des travaux considèrent ce modèle.

2. **Ordre des nœuds.** Soit $G = (A, B, E)$ un graphe biparti. Les arêtes incidentes au même nœud de type A arrivent successivement [GKK12, Kap13]. Dans [BYKS02] et [BFL$^+$06], un flux de graphe qui respecte cette ordre est appellé *incidence stream* et ce concept est également appliqué pour des graphes générales.

3. **Ordre aléatoire.** Les arêtes arrivent dans une ordre uniformément aléatoire [DLOM02, GMV06, GM09, KMM12]. Ce modèle est motivé par le fait que quelques algorithmes fonctionnent bien en pratique et n'échouent que si les arêtes sont présentées dans très peu d'ordres d'arrivées artificielles. L'ordre d'arrivée adversaire est considérée comme trop pessimiste et loin de la réalité. L'ordre d'arrivée aléatoire est une façon d'analyser un cas moyen. Nous signalons que l'analyse de l'ordre d'arrivée aléatoire considère toujours un graphe pire.

4. **Ordre meilleure.** Les arêtes arrivent dans l'ordre souhaitée par l'algorithme de streaming. Ce modèle était étudié dans le contexte du *stream checking* [DSLN09]. Nous ne mentionnons ce modèle que pour compléter cette vue globale et nous ne le considérons plus au cours de ce document.

2.3 Algorithmes de Streaming Bidirectionnels

Pour certains problèmes, un algorithme de streaming économise beaucoup de mémoire si on autorise l'algorithme de faire un deuxième passage au sens inverse. Magniez et al. montrent dans [MMN10] que vérifier si une séquence de parenthèses de longueur n est bien parenthésée nécessite une mémoire de taille $\tilde{O}(\sqrt{n})$ dans un passage. De plus, il est connu que un algorithme de streaming à p passages nécessite une mémoire de taille $\Omega(\sqrt{n}/p)$ ([CCKM10] et indépendamment [JN10]). Dans [MMN10] Magniez et al. présentent également un algorithme de streaming bidirectionnel à deux passages qui fait un passage de gauche à droite et un passage de droite à gauche et qui utilise une mémoire de taille $O(\log^2 n)$.

Nous observons un phénomène similaire pour le problème de la validité des fichiers XML qui encode des arbres binaires [KM12]. Ce phénomène était également rencontré par François and Magniez [FM13] pour le problème de vérification des files de priorité.

Definition 3 (Algorithme de Streaming bidirectionnel)**.** *Un algorithme de streaming est* bidirectionnel *si il fait au moins un passage de gauche à droite et un passage de droite à gauche.*

2.4 Flux Auxiliaires

Streaming avec des flux auxiliaires est également connu sous le nom de streaming avec de la mémoire externe. Ce modèle a été étudié par exemple dans [GS05, GHS06, BJR07, BH12, KM12]. A côté du flux d'entrée des données, dans ce modèle un algorithme de streaming peut accéder des flux auxiliaires sur lesquels il peut écrire et desquels il peut lire. Cela équipe

l'algorithme avec une mémoire externe de grande taille. Cependant, notons que l'algorithme ne peut accéder à cette mémoire que de façon séquentielle. De plus, nous supposons que la longueur de chaque flux auxiliaire est $O(n)$ (n étant la longueur du flux d'entrée). Nous définissions un algorithme de streaming avec des flux auxiliaires comme suit:

Definition 4 (Algorithme de Streaming avec des Flux Auxiliaires)**.** *Un* algorithme de streaming ***S*** *à $p(n)$ passages, $k(n)$ flux auxiliaires, mémoire $s(n)$ et temps de mis-à-jour $t(n)$ est un algorithme tel que pour tout flux de donnée $A = A[1] \dots A[n]$ et $A[i] \in \mathcal{U}$ et ensemble $\mathcal{U}$:*

1. ***S*** *a accès à $k(n)$ flux auxiliaires pour lecture et écriture, le flux de donnée est en lecture seule,*
2. ***S*** *fait au plus $p(n)$ passages sur le flux π et les k flux auxiliaires,*
3. ***S*** *utilise une mémoire vive de taille $s(n)$,*
4. ***S*** *ne dépasse pas le temps $t(n)$ entre deux manœvres consécutives sur les flux (lectures ou écritures).*

De plus, le temps de prétraitement (le temps avant le premier manœvre sur un flux) et le temps de posttraitement (le temps après le dernier manœvre sur un flux et la sortie du résultat) est $O(t(n))$. Nous supposons que la lecture d'un élément du flux et l'écriture d'un élément sur le flux se font en temps constant.

Accéder des flux auxiliaires permet aux algorithmes de streaming de trier la donnée. Il est possible d'implémenter *merge sort* comme un algorithme de streaming avec trois flux auxiliaires et $O(\log n)$ passages [GS05] (n étant la longueur du flux de donnée).

Chapter 3

Contributions

Cette thèse se base sur les articles suivants:

- [KMM12]: Christian Konrad, Frédéric Magniez and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of the 24th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2012.
- [KR13]: Christian Konrad and Adi Rosén. Approximating Semi-Matchings in Streaming and in Two-Party Communication. In *Proceedings of 40th International Colloquium on Automata, Languages and Programming*, 2013.
- [KM12]: Christian Konrad and Frédéric Magniez. Validating XML Documents in the Streaming Model with External Memory. In *Proceedings of 15th International Conference on Database Theory*, 2012.
- [KYZ13]: Christian Konrad, Wei Yu and Qin Zhang. Budget Error-Correcting under Earth-Mover-Distance. Technical Report.

References

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. ACM. 10

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM. 4, 10

[BFL+06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 253–262, New York, NY, USA, 2006. ACM. 10, 11

[BH12] Paul Beame and Trinh Huynh. The value of multiple read/write streams for approximating frequency moments. *ACM Trans. Comput. Theory*, 3(2):6:1–6:22, January 2012. 11

[BJR07] Paul Beame, T. S. Jayram, and Atri Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 689–698, New York, NY, USA, 2007. ACM. 11

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 623–632, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. 11

[CCKM10] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 387–396, Washington, DC, USA, 2010. IEEE Computer Society. 11

[DLOM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 348–360, London, UK, UK, 2002. Springer-Verlag. 11

[DSLN09] Atish Das Sarma, Richard J. Lipton, and Danupon Nanongkai. Best-order streaming model. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation*, TAMC '09, pages 178–191, Berlin, Heidelberg, 2009. Springer-Verlag. 11

REFERENCES

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 745–754, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. 10

[FM83] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *FOCS*, pages 76–82, 1983. 4, 10

[FM13] N. François and F. Magniez. Streaming complexity of checking priority queues. In *Proceedings of 30th Symposium on Theoretical Aspects of Computer Science*, 2013. To appear. 11

[GHS06] Martin Grohe, André Hernich, and Nicole Schweikardt. Randomized computations on large data sets: tight lower bounds. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 243–252, New York, NY, USA, 2006. ACM. 11

[GKK12] A. Goel, M. Kapralov, and S. Khanna. On the communication and streaming complexity of maximum bipartite matching. In *SODA*, 2012. To appear. 10, 11

[GM09] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM J. Comput.*, 38(5):2044–2059, January 2009. 11

[GMV06] Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 733–742, New York, NY, USA, 2006. ACM. 11

[GO13] V. Guruswami and K. Onak. Superlinear lower bounds for multipass graph processing. *28th IEEE Conference on Computational Complexity*, 2013. 10

[GR11] John Gantz and David Reinsel. Extracting value from chaos. 2011. 1

[GS05] Martin Grohe and Nicole Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 238–249, New York, NY, USA, 2005. ACM. 11, 12

[HHLS10] Bjarni V. Halldórsson, Magnús M. Halldórsson, Elena Losievskaja, and Mario Szegedy. Streaming algorithms for independent sets. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 641–652, Berlin, Heidelberg, 2010. Springer-Verlag. 10

[HRR99] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. External memory algorithms. chapter Computing on data streams, pages 107–118. American Mathematical Society, Boston, MA, USA, 1999. 10

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *In COCOON*, pages 710–716, 2005. 10

[JN10] Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions. Technical Report TR10-071, Electronic Colloquium on Computational Complexity, `http://eccc.hpi-web.de/`, April 19 2010. Revised July 20, 2011. 11

[Kap13] Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of 24th ACM-SIAM Symposium on Discrete Algorithms*, 2013. To appear. 11

[KM12] Christian Konrad and Frédéric Magniez. Validating xml documents in the streaming model with external memory. In *Proceedings of 15th International Conference on Database Theory*, 2012. 11, 13

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of 15th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2012. 10, 11, 13

[KMSS12] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012. 10

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. 5

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM. 4

[KR13] Christian Konrad and Adi Rosén. Approximation Semi-Matchings in Streaming and in Two-Party Communication. In *Proceedings of the 40th international conference on Automata, languages and programming*, ICALP'13, 2013. 13

[KYZ13] Christian Konrad, Wei Yu, and Qin Zhang. Budget Error-Correcting under Earth-Mover Distance. Research report, 2013. submitted. 13

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th international workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th international conference on Randamization and Computation: algorithms and techniques*, APPROX'05/RANDOM'05, pages 170–181, Berlin, Heidelberg, 2005. Springer-Verlag. 10

[MMN10] F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. In *Proceedings of 42nd ACM Symposium on Theory of Computing*, pages 261–270, 2010. 11

[MP78] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, SFCS '78, pages 253–258, Washington, DC, USA, 1978. IEEE Computer Society. 10

[Mut05] S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005. 10