Algorithmes géométriques adaptatifs

Frank Nielsen

N° d'ordre : Année 1996

THÈSE de DOCTORAT de l'UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

Spécialité : géométrie algorithmique

préparée à l'**INRIA Sophia Antipolis**

présentée par

Franck NIELSEN

pour obtenir le grade de DOCTEUR de l'UNIVERSITÉ de NICE-SOPHIA ANTIPOLIS
SCIENCES pour l'INGÉNIEUR

ALGORITHMES GÉOMÉTRIQUES ADAPTATIFS

Soutenue le 27 Septembre 1996 devant le jury composé de :

Président :	M.	Jean-Claude	BERMOND
Directeur de thèse :	M.	Jean-Daniel	BOISSONNAT
Rapporteurs :	M.	Bernard	CHAZELLE
	M.	Hiroshi	IMAI (今井。浩)
	M.	Michel	POCCHIOLA
Examinateurs :	M.	Michel	COSNARD
	M.	Jean-Marc	STEYAERT
	Mme.	Mariette	YVINEC

À ceux qui m'ont aimé,
À ceux qui m'aiment,
À ceux qui m'aimeront.

À ma famille,

À Caroline,

Remerciements

Je remercie Jean-Daniel Boissonnat pour m'avoir accueilli à l'INRIA dans l'équipe PRISME. Il m'a fait découvrir la géométrie algorithmique, un domaine fascinant à la frontière des mathématiques, de l'algorithmique, de la combinatoire et d'encore bien d'autres disciplines. Je remercie Mariette Yvinec pour avoir lu bon nombre de premières versions de documents et avoir fait des commentaires constructifs. Une thèse sans dessins ou croquis n'aurait sans doute pas autant d'attrait. Je remercie Jean-Pierre Merlet pour son outil de dessin JPdraw et sa hot-line. Merci également à Monique Teillaud et Olivier Devillers, Sylvain Lazard, Stéphane Nullans, Luc Tancredi, Hervé Brönnimann, Andreas Fabri, Pascal Desnoguès, Francis Avnaim et tous les autres pour leurs nombreuses discussions et leurs contributions à la bonne ambiance du projet. Je remercie Bernhard, Herta et Philipp Geiger pour leur sympathie et m'avoir si bien accueilli lors de mon passage à Princeton. Merci Bernhard pour les dessins ! (pages 3, 77 et 133)

Un petit clin d'œil également à Cyrille Gauclin, Robert Stahr, Imad Zoghlami, Jérôme Declerck, Myriam Ribière, Christophe Cointe, Monica Crubezy, Silvano Dal-Zilio et Ranan Fraer.

Je tiens à remercier Matthew Katz pour sa collaboration qui a débuté lors de son séjour post-doctoral dans l'équipe. Cela nous a permis de nous concentrer sur un domaine de recherche d'intérêt commun et ce fût une expérience très enrichissante.

Lors de cette thèse j'ai eu l'occasion d'échanger quelques idées et de collaborer avec plusieurs équipes. Je remercie Kurt Mehlhorn, Stefan Schirra et Christoph Burnikel pour m'avoir accueilli au Max Planck Institut für Informatik (Saarbrücken, Allemagne). J'aimerais également remercier Mordecai Golin, Sunil Arya et Sui Win Cheng pour leur accueil chaleureux à HKUST, Hong Kong University of Science and Technology. Je remercie Hiroshi Imai pour m'avoir reçu au sein de l'Université de Tôkyô (Tôdai Daigaku). Lors de ce long séjour, j'ai pu apprécier à la fois les compétences scientifiques et les qualités humaines de toute son équipe. Merci à Minako Okada-san, Mary Inaba-san, Hayase-san, Nakade-san, Shibuya-san. Merci également à Tetsuo

Asano-san pour m'avoir invité à Ôsaka, à Keiko Imai-san et Takeo Asano-san pour m'avoir invité à l'université de Chûo (Tôkyô). Je remercie également Mikiko Kodachi, Mayumi Ishizuka, Midori Yanagi, Ta Mura Nori Aki et Takao Goto qui ont fait preuve d'une grande patience et gentillesse pendant mes "premiers essais" en japonais. Qu'ils en soient remerciés !

Je suis très honoré que Messieurs Bernard Chazelle, Hiroshi Imai et Michel Pocchiola aient bien voulu être rapporteurs de cette thèse. Je remercie également tous les membres du jury d'avoir participé à la soutenance du vendredi 27 septembre 1996 : Messieurs Bermond, Boissonnat, Cosnard, Pocchiola, Steyaert et Madame Yvinec.

Je tiens à exprimer toute ma gratitude à Monique Teillaud qui a relu le manuscrit et corrigé quelques fautes d'orthographes et erreurs typographiques.

Quelques années de travail sont entrecoupées de vacances... merci aux schtroumphs: Christophe Rapine, Patrice Calégari, Pierre Baraduc et Olivier Beaumont. En souvenir de nos périples à Amsterdam, Prague, Londres, etc. Merci à Gilles Hubert, Benoît Urgelli, Grégoire Molinatti pour mon initiation aux sports extrêmes...

Finalement, je remercie le secrétariat de l'École Doctorale des Sciences pour l'Ingénieur pour sa grande aide lors des démarches administratives.

Voilà, il ne me reste plus qu'à vous souhaiter bonne lecture...

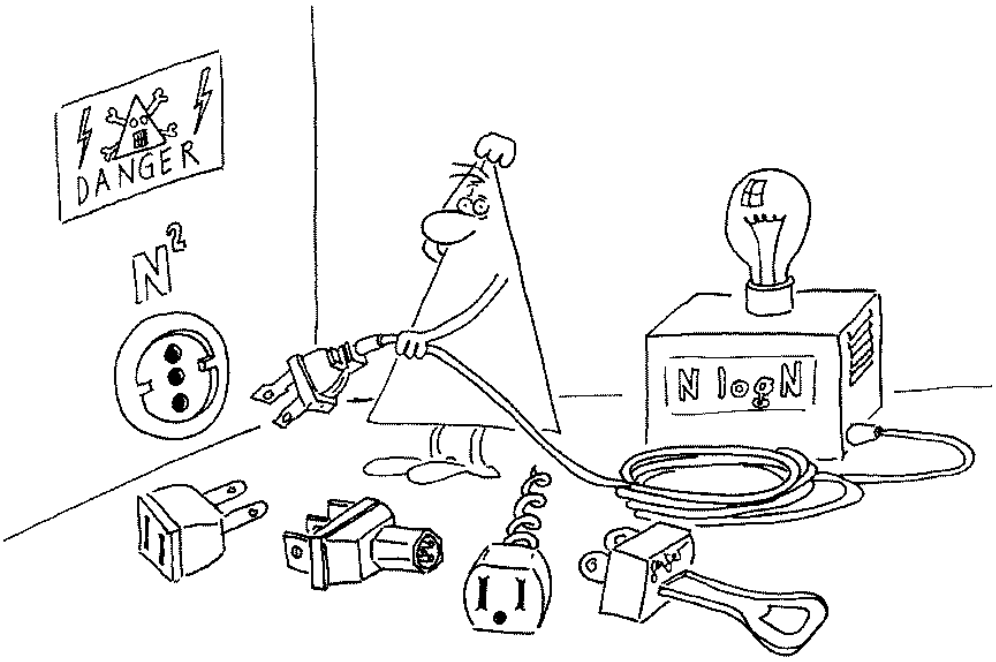
Table des matières

Première partie

État de l'art sur les algorithmes géométriques adaptatifs

Chapitre 1

Introduction



Dans cette première partie, nous définissons les algorithmes adaptatifs en insérant cette classe dans l'arbre taxinomique des familles d'algorithmes. Nous comparons ces algorithmes aux algorithmes traditionnels en mettant en valeur leurs points forts mais également leurs inconvénients.

Nous décrivons très succintement les résultats fondamentaux concernant les algorithmes géométriques adaptatifs. Puis, dans un second temps, nous détaillons quelques *paradigmes* permettant d'obtenir des algorithmes adaptatifs et les illustrons en considérant le problème du calcul de l'enveloppe

convexe de points, un des problèmes fondamentaux en géométrie algorithmique.

Cette partie permet de définir le vocabulaire et les techniques de base sur lesquelles reposent les principaux travaux de cette thèse. De plus, nous montrons que le choix des paramètres, qualifiés de *significatifs* (c'est-à-dire exhibant une propriété combinatoire sur la taille de la sortie ou de paramètres intrinsèques au problème), est crucial pour l'obtention d'algorithmes efficaces. En effet, nous montrons que certains paramètres significatifs ne peuvent pas donner lieu à des algorithmes eux-mêmes adaptatifs. Autrement dit, bien que nous puissions visualiser un paramètre qui pour un même nombre de données intervient de manière significative dans la taille de la structure construite ou dans l'agencement des données, nous ne pouvons pas tenir compte de ce paramètre dans la construction d'algorithmes.

Nous décrivons également dans cette partie un nouvel algorithme optimal pour le calcul des maxima d'un ensemble de points du plan.

Signalons au lecteur qu'à la fin de ce manuscrit se trouve un récapitulatif des principales notations (page 197) ainsi qu'un index par nom d'auteurs (page 211) et un index par mots clefs (page 215).

Chapitre 2

Taxonomie des algorithmes et algorithmes géométriques adaptatifs

2.1 Un rapide survol de la géométrie algorithmique

Nous introduisons dans ce chapitre le concept d'algorithmes géométriques adaptatifs en l'insérant dans l'arbre taxonomique de l'algorithmique. L'Algorithmique est une discipline dont les fondements mathématiques furent réunis pour la première fois dans "The Art of Programming" de Knuth [Knu73a, Knu81, Knu73b]. Le lecteur intéressé par une introduction à l'algorithmique pourra se référer aux ouvrages de base [FGS90, AHU74, AHU83, Sed83, Sed92, CLR90] et aux monographies synthétiques [Meh84a, Meh84b]. Nous n'aborderons pas dans cette thèse l'algorithmique des structures de données qui consiste à manipuler efficacement une information donnée sous forme codée (le lecteur intéressé pourra se reporter au livre de Tarjan [Tar87]).

Les propriétés d'objets géométriques de base (points, triangles, cercles, etc.) furent premièrement résumées dans les "Éléments" d'Euclide [Euc56]. Ces propriétés sont généralement utilisées dans la preuve de correction partielle des algorithmes géométriques. Elles donnent lieu également à des briques de base (oracle ou algorithmes en temps $O(1)$) qui servent à construire des algorithmes manipulant un ensemble de n données.

La géométrie algorithmique consiste à bâtir des algorithmes permettant de résoudre des problèmes de nature géométrique. Ces algorithmes reposent à la fois sur la construction de structures de données (carte des trapèzes, graphe d'incidence, etc.) mais aussi sur la manipulation efficace de celles-

ci. La géométrie algorithmique intervient dans de nombreuses applications comme la conception de circuits VLSI (Very Large Scale Integration), l'imagerie médicale, les systèmes d'information géographique (GIS), la conception de produits manufacturés, etc (cf. la CG Impact Task Force, rapport sur la géométrie algorithmique et ses applications [Cha96]). Tout au long de la lecture de cette thèse, le lecteur pourra se reporter aux ouvrages classiques de base de la géométrie algorithmique [PS85, Ede87, BY95a].

Après un essor considérable, l'informatique s'est heurtée à l'intractabilité de certains problèmes qualifiés de NP-difficiles (ou bien encore synonymement de NP-durs) ou NP-complets. L'excellent ouvrage de Garey et Johnson [GJ79] définit et décrit ces notions en les illustrant grâce à une panoplie de problèmes. Il est à noter que certains problèmes géométriques d'énoncé simple comme la triangulation de poids minimal (Minimum Weight Triangulation) n'ont pas été montrés NP-difficiles bien qu'aucun algorithme polynomial n'existe à ce jour. Ce problème, à lui seul, retient aujourd'hui l'attention de la communauté et représente un véritable défi : donner un algorithme polynomial ou bien montrer qu'il est NP-difficile.

Parmi les objets géométriques usuels auxquels nous nous référencerons dans la suite, on trouve, schématiquement, les polytopes dont les propriétés sont décrites dans les livres [Ber74, Brö83, Grü67, MS71, Zie94] et les diagrammes de Voronoï [OBS92, Boo89]. Le livre d'Overmars [Ove83] décrit les fondements de la maintenance dynamique d'algorithmes géométriques, c'est-à-dire l'insertion, la suppression ou encore la modification d'objets, dans des structures géométriques.

Les concepts d'algorithmes randomisés et de dérandomisation, faisant intervenir la théorie des probabilités [AS93b, ES74], sont détaillés dans l'ouvrage de Mulmuley [Mul94] mais aussi dans l'ouvrage en Français de Boissonnat et Yvinec [BY95a].

Les propriétés combinatoires d'objets géométriques sont détaillées dans les ouvrages [GW93a, GW93b, HD60a, HDK64]. Nous les utiliserons tout au long de cette thèse.

Finalement, avant d'introduire la notion d'algorithme adaptatif, concluons sur le fait que la géométrie algorithmique est une discipline utilisant et fournissant des résultats en robotique [BL89], en optimisation combinatoire [PS82], en théorie de Ramsey [GRS80], en théorie des langages [Aga91, SA95], etc. En ce sens, la géométrie algorithmique peut être perçue comme une discipline fédératrice.

Avec l'essor des machines parallèles, la plupart des algorithmes séquentiels fondamentaux ont pu être parallélisés avec succès (cf. [Akl85b, AL93]). Nous décrirons succintement quelques algorithmes adaptatifs sur machines parallèles dans cette partie afin de montrer les difficultés supplémentaires

rencontrées dans la parallélisation. Il y a fort à parier que cette branche de la géométrie algorithmique connaitra un rapide développement dans les prochaines années où la puissance d'un processeur sera bridée [1] par les contraintes physiques de propagation des électrons, etc. À l'heure actuelle, la plupart des constructeurs augmentent la rapidité des puces électroniques en associant sur une même puce plusieurs processeurs, et en réduisant ainsi le coût de communication.

2.2 Définition et utilité des algorithmes adaptatifs

Nous introduisons dans cette partie l'objet de cette thèse: les algorithmes géométriques adaptatifs. La complexité de la plupart des structures géométriques, définies à partir de n objets de base, varie entre deux extrêmes. Par exemple, prenons le cas de l'enveloppe convexe de n points en position générale dans l'espace euclidien de dimension d. Nous considérons dans la suite la dimension d fixée. La taille de cette enveloppe convexe varie du simplexe à 2^d faces (dont deux dites impropres) aux polytopes maximaux de taille $O(n^{\lfloor \frac{d}{2} \rfloor})$. Ainsi, en dimension 4, la taille de l'enveloppe convexe de points varie de $O(1)$ à $O(n^2)$. La table 2.1 illustre la variation de taille pour des structures de données géométriques usuelles. Historiquement, les premiers algorithmes construisant ces structures de données ne considérèrent pas ce facteur de sensibilité, et dans les meilleurs cas, furent optimaux vis-à-vis de la taille de la sortie dans le pire des cas.

On dira qu'un algorithme est *adaptatif* (adaptive ou sensitive en anglais ou encore *sensible*) si la complexité de celui-ci varie en fonction de la taille d'un paramètre, par exemple la taille de la structure géométrique construite. Cette structure géométrique intervient naturellement comme structure de données dans les domaines d'application de la géométrie algorithmique.

Les premiers algorithmes adaptatifs conçus furent naturels dans le sens où la taille de la sortie apparue dans leur complexité de par leur construction même. Un des problèmes majeurs consiste à définir et à utiliser de manière algorithmique des paramètres qualifiés de *significatifs* c'est-à-dire des paramètres liés à la taille de la sortie ou à des facteurs intrinsèques du problème.

Dans un deuxième temps, en tenant compte de la distribution des données, les algorithmiciens construisirent et analysèrent des algorithmes spécifiquement conçus pour une distribution de données. La table 2.2 résume les

1. La loi empirique d'Amdahl pourrait bien ne plus être satisfaite dans les prochaines années.

TAB. 2.1 – *Quelques structures géométriques usuelles et l'intervalle de variation de leur taille (dimension d fixée).*

Structure géométrique	de	à
Enveloppe convexe	$O(1)$	$O(n^{\lfloor \frac{d}{2} \rfloor})$
Diagramme de Voronoï	$O(n)$	$O(n^{\lceil \frac{d}{2} \rceil})$
Enveloppe supérieure de simplexes	$O(1)$	$O(n^{d-1}\alpha(n))$
Une cellule d'un arrangement de simplexes	$O(1)$	$O(n^{d-1}\log n)$
Carte des trapèzes	$O(n)$	$O(n^2)$
Points maximaux	$O(1)$	n
Union de rectangles	$O(1)$	$O(n^2)$
m cellules d'un arrangement de droites	$O(m)$	$O(n^{\frac{2}{3}}m^{\frac{2}{3}} + n + m)$

TAB. 2.2 – *Taille en moyenne de structure géométriques obtenues à partir d'un nuage aléatoire de points (cf. [(Ke85]).*

Structure géométrique	Distribution	Taille moyenne
Points extrêmes	Uniforme dans un carré	$O(\log n)$
	Uniforme dans un cercle	$O(\sqrt{\log n})$
	Uniforme dans un hypercube	$O(n^{\frac{d-1}{d+1}})$
Points maximaux	Axes indépendants	$O(\log^{d-1} n)$

principales tailles, en moyenne, des structures de données usuelles en géométrie algorithmique construites à partir de données provenant d'une distribution connue *a priori*. Dans certains cas, un même algorithme aura une complexité moyenne différente variant en fonction de la distribution des données. Ainsi bien que $\Omega(n \log n)$ soit une borne inférieure pour calculer l'enveloppe convexe de n points du plan, il existe de nombreux algorithmes linéaires pour des ensembles de points uniformément distribués dans un carré, un cercle, etc.

Certains algorithmes adaptatifs ne peuvent traiter les cas dégénérés de manière efficace (par exemple pour le calcul de l'enveloppe convexe de points en grandes dimensions). Prenons le problème suivant : calculer les points d'intersection d'un ensemble $\mathcal{S}$ de n segments. Soit I le nombre de paires de segments sécants (points d'intersection) de $\mathcal{S}$ ($0 \leq I \leq \frac{n(n-1)}{2}$) et I' le nombre de points d'intersection distincts ($I' \leq I$). Chazelle et Edelsbrunner [CE88, CE92] donnèrent un algorithme adaptatif optimal en temps $O(n \log n + I)$ pour calculer ces I points d'intersection. Notons qu'il existe des ensembles de segments $\mathcal{S}$ dégénérés pour lesquels $I = \Theta(n^2)$ mais $I' = 1$. Si l'algorithme a une complexité en $O(n \log n + I')$ alors on dira que cet algorithme est *réellement adaptatif* (truly output-sensitive). L'espace mémoire de cet algorithme est toutefois en $O(n + I')$. Il fut plus tard réduit à une taille linéaire par Balaban [Bal95] qui proposa une approche algorithmique différente. Dans ce cas précis, il existe un algorithme réellement adaptatif pour calculer les points d'intersection (et la carte des trapèzes). Signalons que pour le calcul d'enveloppes convexes de points en dimension $d \geq 5$ [Cha95b, AR96], bien que de bons algorithmes adaptatifs aient été récemment décrits, on ne peut pas garantir les mêmes bornes de complexité lorsque les points sont en position dégénérée. La simulation de la simplicité d'Edelsbrunner et Mücke [EM88, EM90] ne permet pas dans le cadre d'algorithmes adaptatifs de s'affranchir directement de l'hypothèse de position générale. En effet, la taille de la sortie de points perturbés peut être radicalement différente de celle des points originels. Les problèmes de précision en géométrie algorithmique sont importants puisque les structures géométriques construites dépendent de tests numériques. Bien que beaucoup de travaux aient été réalisé dans ce domaine, aucune solution, à ce jour, n'allie à la fois vitesse et fiabilité [BKM+95, BMS94, BMS94, EC91, EC92, Yap87, Yap90, MNS+96].

On peut également étudier la complexité en mémoire (c'est-à-dire la taille mémoire utilisée), l'approximation d'une heuristique ou bien encore le temps de requête en fonction de paramètres significatifs, c'est-à-dire des paramètres variant pour une taille fixe des données, en reflétant une propriété combinatoire des objets calculés. Ainsi, il n'est plus rare de qualifier un algorithme

de output-sensitive, precision-sensitive, structure-sensitive, query-sensitive, etc.

La troisième partie de cette thèse montrera un algorithme adaptatif approximant la valeur minimale requise pour percer un ensemble de boîtes isothétiques. On montrera que l'approximation obtenue est elle-même adaptative à la valeur optimale.

Nous distinguons, dans le vaste champ des algorithmes, les algorithmes dits incrémentaux, dynamiques, semi-dynamiques, randomisés, dérandomisés, en ligne, statiques, etc (la plupart des notions sont définies dans les ouvrages algorithmiques de base). En fonction du problème considéré, certains types d'algorithmes ne pourront fournir des algorithmes adaptatifs. Par exemple, en considérant le cas de l'enveloppe convexe de points, on ne peut obtenir un algorithme adaptatif en ligne puisque l'insertion d'un seul point peut détruire la plus grande partie de la structure géométrique déjà construite.

Chapitre 3

Algorithmes adaptatifs pour le calcul d'enveloppe convexe et problèmes connexes

Nous considérons dans cette partie les algorithmes adaptatifs à la taille de la sortie. Dans la suite, nous notons n, le nombre de points (données), et h un paramètre dépendant de la sortie. Dans le cas du plan, notons h le nombre de points extrêmes.

Les algorithmes sensibles à la sortie pour le calcul d'enveloppes convexes sont récents et sont axés suivant cinq grandes méthodologies (paradigmes[1]) des algorithmes adaptatifs:

- L'approche *papier cadeau* (gift wrapping) qui donne lieu à des algorithmes simples mais non optimaux en $O(n \times h)$.
- L'approche *fusion-avant-conquête* (marriage-before-conquest ou *mariage-avant-conquête*) qui à l'inverse de *diviser-pour-régner* (divide-and-conquer) calcule d'abord le résultat de la fusion puis résoud les sous-problèmes générés.
- L'approche *dérandomisation* (derandomization) qui permet de choisir les échantillons aléatoires intervenant dans des algorithmes randomisés de façon à obtenir le temps escompté.
- L'approche *prétraitement paresseux* (lazy preprocessing). Généralement, la solution d'un problème peut se trouver en prétraitant les données (le plus souvent par un tri), puis en parcourant les données prétraitées on

1. Ce mot est malencontreusement utilisé pour désigner une méthodologie. En fait, on appelle *paradigme* un exemple, bien choisi, reflétant la méthodologie (cf. Petit Robert).

trouve en temps linéaire $O(n)$ la solution. Ce paradigme utilise ce fait et montre comment on peut décomposer ce prétraitement et le raffiner en fonction de la taille de la sortie. Il est encore appelé lazy sweep line.

– L'approche composite *paquets/requêtes* ou *paquets/mariage avant conquête* (grouping scheme) qui permet d'obtenir une batterie d'algorithmes efficaces. Nous illustrerons ceux-ci dans la deuxième partie de cette thèse en considérant un algorithme adaptatif pour le calcul de l'enveloppe convexe d'objets planaires.

Nous citons dans la table 3 (page 13) les principaux résultats des algorithmes adaptatifs de calcul d'enveloppe convexe de points dans l'espace euclidien $\mathbb{E}^d$ connus à ce jour. On désigne par ϕ_k le nombre de k-faces, par h le nombre de sommets de l'enveloppe convexe ($h = \phi_0$) et par F le cardinal de l'ensemble des faces de l'enveloppe convexe: $F = \sum_{i=0}^{d-1} \phi_i$. Nous utilisons les notations classiques de complexité pour les algorithmes parallèles: t est le temps de calcul par processeur, p est le nombre de processeurs et w est le travail de l'algorithme: $w = p \times t$.

Nous détaillons les caractéristiques de ces algorithmes autour des quatre premiers paradigmes puis nous décrivons la résolution de problèmes voisins comme celui des vecteurs maximaux, etc. L'enveloppe convexe est une structure géométrique fondamentale et importante en géométrie algorithmique. Par exemple, en utilisant des dualités classiques (notamment décrites dans le livre de Boissonnat et Yvinec [BY95a] ou dans [ES92]), l'enveloppe convexe de points de $\mathbb{R}^{d+1}$ (ou dualement l'intersection de demi-espaces) peut servir à la construction de diagrammes de puissance à poids additifs de points de $\mathbb{R}^d$. On peut également calculer le diagramme de Voronoï à poids additifs ou multiplicatifs par un algorithme de calcul d'enveloppe convexe de points en dimension $\mathbb{R}^{d+2}$ [BY95a].

Dans la deuxième partie de cette thèse, nous nous sommes intéressés au calcul de l'enveloppe convexe d'objets planaires. Nous décrirons dans cette partie une approche composite combinant à la fois le paradigme *mariage avant conquête* mais aussi le paradigme de *décomposition en paquets*.

3.1 Le paradigme fusion avant conquête

On appelle algorithme *adaptatif*, un algorithme dont la complexité, pour une taille fixée des données, dépend de paramètres (par exemple, la taille de la sortie ou des paramètres caractérisant les données). Ces paramètres sont eux-mêmes qualifiés de *significatifs*. *Diviser pour régner* est une méthode générale qui consiste à diviser classiquement le problème en deux sous-problèmes

Dimension	Borne inférieure	Complexité	Méthode	Réferences
2	$\Omega(n \log h)$	$O(nh)$	Marche de Jarvis	[Jar77, Jar73, CK70]
		$\Theta(n \log h)$	Fusion-Conquête	[KS82, KS86]
		$p = O(n^{1-\epsilon}),\ t = O(n^{\epsilon} \log h)$	EREW PRAM	[Akl84, Akl85a]
		$p = O(n),\ t = \tilde{\mathrm{O}}(\log h)$	modèle SCAN	[EJ88]
		$t = \tilde{O}(\log h \log\log n), w = \tilde{O}(n \log h)$	Randomisation CRCW	[GS96]
		$t = \tilde{O}(\log n \log h), w = \tilde{O}(n \log h)$	Randomisation CRCW	[GS96]
3	$\Omega(n \log h)$	$O(n \log^2 h)$	Fusion-Conquête	[ES91]
		$O(nh)$	Papier Cadeau	[PS85, CK70, Dwy90]
		$O(n \log h)$	QuickHull+ *cond. d'équilibre*	[BDH93]
		$\tilde{\mathrm{O}}(n \log h)$	Randomisation	[CS89]
		$\Theta(n \log h)$	Dérandomisation	[CM94]
		$O(n \log h)$	Requêtes Adaptatives	[Cha95b]
		$t = O(\log^3 n), w = O(n \log h)$	Division/Décimation par.	[AGR95]
		$t = \tilde{O}(\log h \log\log^2 n), w = \tilde{O}(n \log h)$	Randomisation CRCW	[GS96]
$d = 4$	$\Omega(n \log h + F)$	$O((n+F) \log^2 h)$	Fusion-Conquête	[CSY95]
$d = 5$	$\Omega(n \log h + F)$	$O((n+F) \log^3 h)$	Division-Décimation	[AR96]
$d \geq 6$	$\Omega(n \log h + F)$	$O(nh^{\lfloor \frac{d}{2} \rfloor - 1} + h^{\lfloor \frac{d}{2} \rfloor})$	QuickHull+ *cond. d'équilibre*	[BDH93]
		$O(n^2 + F \log h)$	Effeuillage	[Sei86a]
		$O(n\phi_{d-1} + \phi_{d-2} \log \phi_{d-2})$	Papier Cadeau	[PS85]
		$O((n + F^{d-2}) \log^{d-1} F)$	Requêtes Adaptatives	[Cha95b]
		$O((n + F^{\lfloor \frac{d-1}{2} \rfloor}) \log^{d-1} F)$	Division-Décimation	[AR96]

de tailles quasi-équivalentes[2] (ou équilibrée) puis à résoudre récursivement et à fusionner les solutions (cf. algorithme 3.1). Le paradigme *fusion avant conquête* consiste à diviser le problème en sous-problèmes puis à fusionner les solutions des sous-problèmes (sans calculer toute la solution) et enfin à résoudre récursivement les sous-problèmes, qui ont été préalablement filtrés par le résultat de la fusion, de tailles désormais dépendantes de la fusion (cf. algorithme 3.2). L'idée fut introduite par Kirkpatrick et Seidel dans [KS82, KS86] pour le calcul d'enveloppes convexes de points dans le plan puis se généralisa au calcul d'enveloppes convexes de points dans l'espace $\mathbb{E}^3$ [ES91] et en dimensions supérieures [CSY95, AR96] ainsi qu'au calcul des vecteurs maximaux [KS85, Sei86b] avec toutefois une adaptation notable. Dans la partie 3.1.6 (page 24) nous étudierons de manière générale la complexité des paradigmes *diviser pour régner* et *fusion avant conquête*.

Algorithme 3.1: *Le paradigme diviser pour régner.*

Fonction Diviser-Pour-Régner($\mathcal{D} = \{D_1, ..., D_n\}$) : solution ;
début
si $n < b$ alors renvoyer *Trivial*($\mathcal{D}$);
sinon début
$\mathcal{S}_1$ =*Diviser-Pour-Régner*($\mathcal{D}_1 = \{D_1, ..., D_{\lfloor \frac{n}{2} \rfloor}\}$);
$\mathcal{S}_2$ =*Diviser-Pour-Régner*($\mathcal{D}_2 = \{D_{\lceil \frac{n+1}{2} \rceil}, ..., D_n\}$);
$\mathcal{S}$ =*Fusion*($\mathcal{S}_1, \mathcal{S}_2$);
renvoyer $\mathcal{S}$;
fin;
fin.

3.1.1 Un algorithme adaptatif optimal de tri

Nous illustrons dans l'encadré 3.3 l'approche mariage avant conquête en décrivant un algorithme de tri d'un tableau $A := A[1..n]$ de n entiers dont seulement p sont distincts. L'approche est différente de la méthode diviser

2. Si l'on considère $\mathcal{D} = \{D_1, ..., D_n\}$ l'ensemble des données, on divise le problème en k sous-problèmes $\mathcal{D}_1, ..., \mathcal{D}_k$ tel que $\forall\ i,\ |\mathcal{D}_i| \leq \lceil \frac{n}{k} \rceil$. Généralement, en algorithmique séquentielle, k est égal à deux.

Algorithme 3.2: *Le paradigme fusion avant conquête.*

Fonction Fusion-Avant-Conquête($\mathcal{D} = \{D_1, ..., D_n\}$) : solution ;
début
si $n < b$ alors renvoyer *Trivial*($\mathcal{D}$);
sinon début
$\mathcal{D}_1 = \{D_1, ..., D_{\lfloor \frac{n}{2} \rfloor}\}$;
$\mathcal{D}_2 = \{D_{\lceil \frac{n+1}{2} \rceil}, ..., D_n\}$;
$\mathcal{S}'$=*Fusion*($\mathcal{D}_1$,$\mathcal{D}_2$);
$\mathcal{D}'_1$ =*Filtrer*($\mathcal{D}_1$,$\mathcal{S}'$);
$\mathcal{D}'_2$ =*Filtrer*($\mathcal{D}_2$,$\mathcal{S}'$);
$\mathcal{S}_1$ =*Fusion-Avant-Conquête*($\mathcal{D}'_1$);
$\mathcal{S}_2$ =*Fusion-Avant-Conquête*($\mathcal{D}'_2$);
renvoyer $\{\mathcal{S}_1, \mathcal{S}', \mathcal{S}_2\}$;
fin;
fin.

pour régner car elle résoud la fusion des sous-problèmes avant même d'obtenir les solutions des sous-problèmes. Le résultat de l'algorithme est donné sous la forme d'une liste ordonnée d'éléments $< x, p_x >$ signifiant que l'élément x est présent p_x fois consécutivement dans la liste triée.

Algorithme 3.3: *L'approche mariage avant conquête TriMariage*(A) *pour trier* A.

Fonction TriMariage$(A = [A_1, ..., A_n])$: solution ;
début
si $n = 0$ alors renvoyer $\emptyset$;
sinon début
$x :=$ Médiane(A);
(* On utilise l'algorithme de Blum*et al.* [BFP$^+$72] *);
$A_1 := \{A[i] \mid A[i] < x\}$;
$A_2 = \{A[i] \mid A[i] > x\}$;
$A_x = (x)$;
(* L'élément x est présent $p_x = n - |A_1| - |A_2|$ fois *)
$A'_1 :=$ TriMariage(A_1);
$A'_2 :=$ TriMariage(A_2);
renvoyer $A'_1 \circ < x, p_x > \circ A'_2$;
fin;
fin;

Cet algorithme tire partie du fait que le tableau $\underbrace{(x, ..., x)}_{p_x}$ rempli de x de taille p_x est déjà trié. Soit $c(n, p)$ la complexité de l'algorithme *TriMariage* pour trier n éléments dont p sont différents. Notons $n_1 = |A_1|$, $n_2 = |A_2|$ et p_1 (respectivement p_2) le nombre d'éléments différents dans A_1 (resp. dans A_2) , alors on a:

$$c(n,p) = \begin{cases} O(n) & \text{si } p \leq 1, \\ c(n_1, p_1) + c(n_2, p_2) + O(n) & \text{sinon.} \end{cases} \tag{3.1}$$

Théorème 1 *L'algorithme 3.3 trie n données dont p sont différentes en temps $O(n \log p)$ et espace mémoire linéaire.*

Démonstration : Considérons l'équation (3.1) de la complexité de l'algorithme 3.3. On montre par récurrence sur le couple (n, p) que $c(n, p) \leq$

$Cn \log p$. Soulignons le fait que pour un n donné la taille de la sortie, p, peut varier de 1 à n. On a:

$$c(n,p) = \begin{cases} An & \text{si } p \leq 2 \\ c(n_1,p_1) + c(n_2,p_2) + Bn & \text{sinon.} \end{cases} \tag{3.2}$$

où A et B sont des constantes dépendant du calcul de la médiane et de la partition du tableau.

Si $p \leq 2$ alors $c(n,p) \leq c(n,2) = An \leq Cn$ pour $C \geq A$.

Si $p > 2$ alors on a:

$$c(n,p) \leq Bn + Cn_1 \log p_1 + Cn_2 \log p_2$$

avec $n_1, n_2 \leq \frac{n}{2}$ et $p_1 + p_2 + 1 = p$. Ainsi, on obtient l'inégalité suivante:

$$c(n,p) \leq Bn + C\frac{n}{2}\log(p_1 p_2) \leq Bn + C\frac{n}{2}\log \max_{p_1+p_2+1=p}\{(p_1 p_2)\}$$

Or $p_1 p_2$ contraint à $p_1 + p_2 + 1 = p$ est une fonction convexe[3] maximisée pour $p_1 = p_2 = \frac{p-1}{2} \leq \frac{p}{2}$. Ainsi, nous obtenons l'inégalité suivante:

$$c(n,p) \leq (B - C)n + Cn \log p$$

Clairement, $c(n,p) \leq Cn \log p$ pour $C \geq \max\{B, A\}$. □

Notons qu'une analyse plus détaillée montre que $c(n,p) \in [\Omega(n + p \log p), O(n \log p)]$. Ainsi, parfois on peut trier certains tableaux en temps linéaire si $p \log p < O(n)$. Le lien avec le tri par clefs est alors direct puisque dans le tri par clefs (bucket sort) on a $p \leq 2^d$ si bien que le tri mariage avant conquête donnerait un résultat analogue $O(n \log 2^d) = O(dn)$. En résumant, on a expliqué comment on pouvait raffiner un paramètre de la sortie (un critère qui reste néanmoins global) et trouver de nouveaux algorithmes tenant compte du critère de la taille des entrées n mais aussi de la taille de la sortie $p \leq n$ ($n \log p \leq n \log n$). Dans son ouvrage général sur les tris, K. Mehlhorn [Meh84b], décrit un tri en temps moyen $\tilde{O}(n)$ pour n réels répartis uniformément dans $[0,1]$. Il a été également démontré que tout algorithme randomisé pour le tri a une complexité $\Omega(n \log n)$. Il n'y a en fait aucun paradoxe puisque les modèles de calcul (ou de machine) sont différents dans les deux cas. Nous utilisons le modèle **RAM** où les opérations élémentaires se font en temps constant (quelque soit la longueur d'encodage

3. Plus généralement, on a l'inégalité de Hölder suivante: soit $\sum_{i=0}^{j} n_i = n$ alors $\sum_{i=0}^{j} n_i^a \leq n^a j^{1-a}$.

des données). Dans le tri en temps moyen $\tilde{O}(n)$, on se base sur des techniques de rangement dans des casiers (bucketting) et on fait donc appel à la partie entière[4].

Qu'en est-il de la borne inférieure pour trier n nombres dont p sont différents? Seidel et Kirkpatrick ont montré [KS82, KS86] la borne inférieure $\Omega(n \log p)$. Afin de mieux cerner les différences avec la borne générale $\Omega(n \log n)$, nous donnons ci-dessous succintement l'argument.

Théorème 2 *Trier n nombres dont p sont différents nécessite $\Omega(n \log p)$ opérations de comparaison dans le modèle des arbres algébriques de décision de degré borné d.*

Preuve: On considère le problème suivant :

Problème 1 (multiset size verification problem — MSVP) *Étant donné n nombres, déterminer combien sont distincts.*

S'il y a p nombres différents, il suffit de prouver que l'ensemble

$$M_p = \{(z_1, ..., z_n) \in \mathbb{R}^n \mid |\{z_1, ..., z_n\}| = p\}$$

a au moins $p!p^{n-p}$ composantes connexes disjointes. En effet, il suffit de fixer $z_1, ..., z_p$ comme des entiers distincts dans $[p] = \{1, ..., p\}$ et de choisir les $(n-p)$ autres z_i $(i > p)$ comme étant un entier de $[p]$. Il y a ainsi $p!p^{n-p}$ composantes connexes disjointes dans M_p. On utilise alors un théorème de Ben-Or [BO83, BOKR86] sur les arbres algébriques de décision à degré borné qui stipule que si W est le nombre de composantes connexes alors une borne inférieure, sur ce modèle de machine, est $\Omega(\log W)$. En appliquant directement ce théorème, nous obtenons la borne inférieure du problème 1 $\Omega(\log p!p^{n-p}) = \Omega(n \log p)$.

Clairement on ramène le problème du tri au problème de vérification du multi-ensemble (multiset) en temps linéaire. Ainsi, $\Omega(n \log p)$ est une borne inférieure pour le tri de n nombres dont p sont différents. □

La figure 3.1 montre les résultats de l'implantation du tri rapide (mergesort) adaptatif en C++. On a fixé le nombre de réels à trier à 10000 et on fait varier le nombre de réels différents entre 1 et 6000. Pour un n fixé, la complexité de l'algorithme est $O(\log p)$ où p est le nombre de nombres différents. Les résultats obtenus valident l'analyse et permettent de déterminer expérimentalement la constante cachée dans la notation $O()$.

4. cf. l'ouvrage d'introduction à la géométrie algorithmique de Preparata et Shamos [PS85]. Le problème MAX GAP (on se donne n réels et on veut trouver la distance la plus grande entre deux réels consécutifs) est $\Omega(n \log n)$-dur dans le modéle **RAM** mais peut être résolu en temps linéaire si l'on dispose de la partie entière.

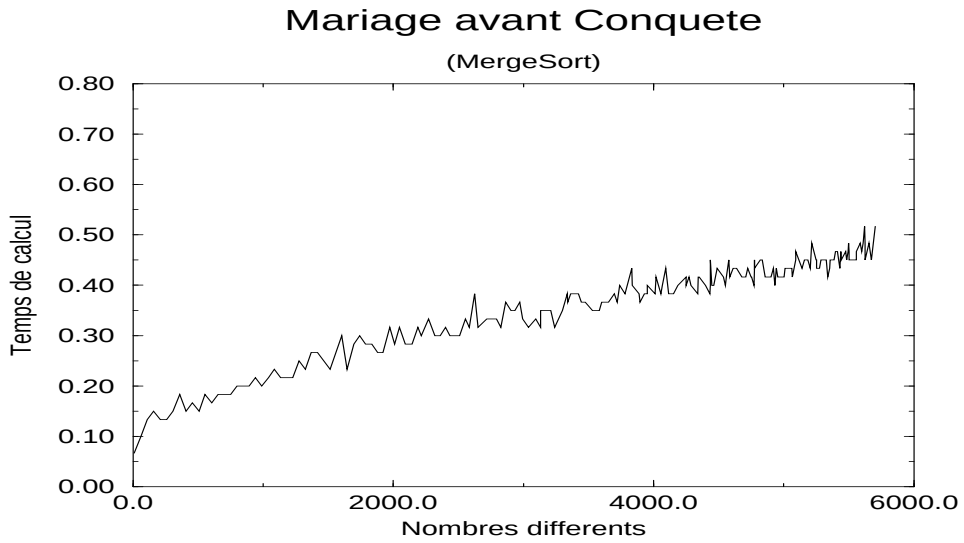


FIG. 3.1 – *Performances du tri adaptatif.*

Notons que l'algorithmique des tris ne s'aurait s'arrêter là! Nous pouvons affiner cet algorithme de manière à le rendre plus efficace en tenant compte de données partiellement triées, entrelacées, etc. Récemment, de nouveaux algorithmes [McI93] basés sur l'entropie du tableau A ont été décrits et s'avèrent plus performants à la fois en théorie mais aussi en pratique. L'algorithmique du tri est une branche bien active qui permet de mettre souvent de nouveaux paradigmes en pratique.

3.1.2 Algorithme adaptatif de calcul d'enveloppes convexes dans le plan [KS86]

3.1.3 Description de l'algorithme

On considère un ensemble $\mathcal{P}$ de n points du plan ($\mathcal{P} = \{p_1, ..., p_n\}$). On désire calculer l'enveloppe convexe de $\mathcal{P}$: $\mathcal{CH}(\mathcal{P})$. Soit h le nombre de points de $\mathcal{P}$ sur l'enveloppe convexe, l'algorithme calcule cette enveloppe convexe en temps optimal $\Theta(n \log h)$ en utilisant le paradigme de fusion avant conquête. L'algorithme détermine l'enveloppe convexe supérieure, c'est-à-dire ce qui est visible depuis le point situé à l'infini $(0, +\infty)$. L'enveloppe convexe inférieure sera déterminée de manière symétrique en transposant les points $p_i = (x_i, y_i)$ de $\mathcal{P}$ en $p'_i = (x_i, -y_i)$ puis en calculant leur enveloppe convexe supérieure[5].

5. On note $\partial\mathcal{CH}(\mathcal{P})$ le bord de l'enveloppe convexe de $\mathcal{P}$ qui est l'union des bords des enveloppes convexes supérieure et inférieure: $\partial\mathcal{CH}(\mathcal{P}) = \partial\mathcal{CH}^+(\mathcal{P}) \cup \partial\mathcal{CH}^-(\mathcal{P})$. On

On fusionne les deux enveloppes convexes en $O(h)$, où h désigne le nombre d'arêtes de $\mathcal{CH}(\mathcal{S})$. On suppose les points de $\mathcal{P}$ en position générale[6]. Le cas trivial du calcul de $\mathcal{CH}(\mathcal{P})$ est celui où $\mathcal{P}$ est constitué d'un unique point p. On a alors $\mathcal{CH}(P) = [p]$. On divise les données en partitionnant en deux paquets $\mathcal{P}_1$ et $\mathcal{P}_2$ ($\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$ et $\mathcal{P}_1 \leq \mathcal{P}_2$ de tailles quasi-équivalentes ($\mid |\mathcal{P}_1| - |\mathcal{P}_2| \mid \leq 1$) en calculant la médiane x_m des abscisses des points de $\mathcal{P}$ grâce à l'algorithme de Blum et al.([AHU74]). Soit $p_m = (x_m, y_m)$ le point de $\mathcal{P}$ (p_m est défini de manière unique lorsque $\mathcal{P}$ ne contient pas deux points de même abscisse) qui a pour abscisse x_m. On partitionne $\mathcal{P}$ comme suit:

- $\mathcal{P}_1 = \{p_i | p_i \in \mathcal{P} \text{ et } x_i \leq x_m\}$.
- $\mathcal{P}_2 = \{p_i | p_i \in \mathcal{P} \text{ et } x_i > x_m\}$.

On cherche alors à fusionner $\mathcal{CH}^+(\mathcal{P}_1)$ avec $\mathcal{CH}^+(\mathcal{P}_2)$ sans connaître ces enveloppes convexes supérieures qui seront calculées après avoir filtré les données (cf. algorithme 3.2). Cela revient à chercher le pont[7] (cf. [PS85, Ede87]) entre $\mathcal{CH}^+(\mathcal{P}_1)$ et $\mathcal{CH}^+(\mathcal{P}_2)$ ou de façon équivalente le pont entre $\mathcal{P}_1$ et $\mathcal{P}_2$. Cette étape peut être réalisée en temps $O(|\mathcal{P}_1| + |\mathcal{P}_2|)$ grâce à la programmation linéaire (cf. [Dye82, Dye84, Meg83b, Meg82]). En effet, cela revient à minimiser l'ordonnée du point d'intersection de la droite $x = x_m$ avec la droite support de $\mathcal{CH}^+(\mathcal{S})$ au point $\partial\mathcal{CH}(\mathcal{P}) \cap (x = x_m)$, c'est-à-dire de trouver la droite $y = ax + b$ dont les paramètres (a, b) optimisent le programme linéaire suivant:

$$\underline{\text{Programme linéaire}} : \left\{ \begin{array}{c} \underline{\text{minimise } ax_m + b} \\ \\ y_1 - ax_1 - b \leq 0 \\ y_2 - ax_2 - b \leq 0 \\ \cdots \\ y_n - ax_n - b \leq 0 \end{array} \right.$$

utilise alors le fait que $\mathcal{CH}^-(\mathcal{P}) = \mathcal{CH}^+(\mathcal{P}')$ où $\mathcal{P}'$ est l'ensemble des points défini par $\mathcal{P}' = \{(x_i, -y_i) | (x_i, y_i) \in \mathcal{P}\}$.

6. On dit que $\mathcal{P}$ est en position générale s'il n'existe pas trois points de $\mathcal{P}$ colinéaires. Cette condition n'est pas restrictive dans les algorithmes non-adaptatifs puisque l'on peut simuler la *simplicité* (cf. [EM88, EM90]). Toutefois, nous ne pouvons pas appliquer directement la simulation de la simplicité dans les algorithmes adaptatifs puisque la taille de la sortie de l'ensemble de points perturbé peut être radicalement différente de celle de l'ensemble de points originel. Toutefois ici, il est aisé de s'affranchir de cette condition.

7. On définit le *pont* ici comme l'arête de $\partial\mathcal{CH}^+(\mathcal{S})$ qui intersecte la droite verticale $x = x_m$.

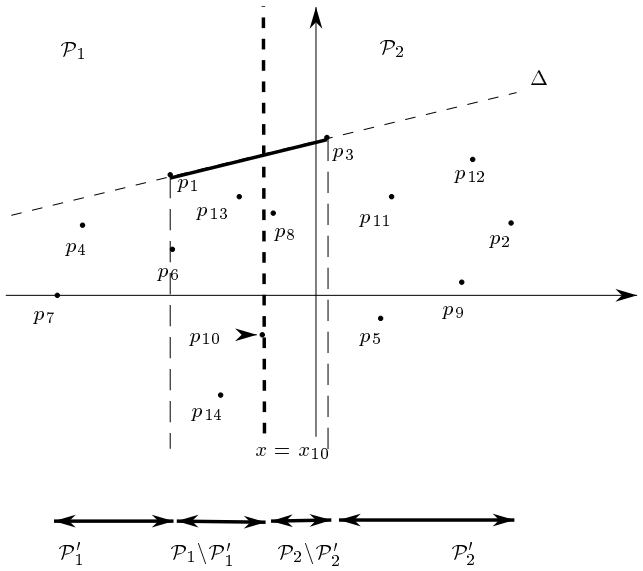


FIG. 3.2 – *Premier étage de récursion de l'algorithme sur $\mathcal{P}$.*

On trouve ainsi aisément le pont puisque sa droite support est $\Delta : y = ax + b$ et que ses extrémités sont les deux points p_i ($p_i \in \mathcal{P}_1$) et p_j ($p_j \in \mathcal{P}_2$) vérifiant $y_i - ax_i - b = 0$ et $y_j - ax_j - b = 0$. On traite à ce niveau les éventuels cas dégénérés (points confondus ou colinéaires).

On élimine dès lors les points de $\mathcal{P}_1$ et $\mathcal{P}_2$ qui sont en deçà de la droite Δ et dont l'abscisse est comprise entre x_i et x_j. Cette opération permet de filtrer les données et de trouver l'enveloppe convexe supérieure de $\mathcal{P}'_1$ et $\mathcal{P}'_2$ par application récursive de l'algorithme sur les ensembles de points $\mathcal{P}'_1$ et $\mathcal{P}'_2$ définis ci-après.

$$\mathcal{P}'_1 = \{p = (x, y) \in \mathcal{P}_1 | x \leq x_i\}$$

$$\mathcal{P}'_2 = \{p = (x, y) \in \mathcal{P}_2 | x \geq x_j\}$$

Dans [KS82, KS86] les auteurs donnent une méthode basée sur la sélection (méthode dite de bisection-décimation) qui permet de trouver le pont en temps linéaire $O(n)$ et évite de recourir à la programmation linéaire (cette méthode est néanmoins basée sur des constatations de programmation linéaire à deux variables [Dye84, Dye82, DFM84]). Dans la deuxième partie de cette thèse, nous considérons une approche composite intégrant le paradigme mariage avant conquête pour calculer l'enveloppe convexe d'objets planaires quelconques.

Algorithme 3.4: *Calcul de* $\mathcal{CH}^+(\mathcal{P})$.

Fonction $\mathcal{CH}^+(\mathcal{P} = \{p_1, ..., p_n\})$: envconvsup ;
début
si $n < 2$ alors renvoyer $\mathcal{P} = [p]$;
sinon début
m=*MédianeAbscisse*($\mathcal{P}$);
$\mathcal{P}_1 = \{p = (x, y) \in \mathcal{P} | x \leq x_m\}$;
$\mathcal{P}_2 = \{p = (x, y) \in \mathcal{P} | x > x_m\}$;
$\mathcal{S}'$=*FusionPont*($\mathcal{P}_1$,$\mathcal{P}_2$);
(* FusionPont renvoit l'arête $\mathcal{S}' = [p_i, p_j]$ qui coupe la droite $x = x_m$ *)
$\mathcal{P}'_1$ =*FiltrerPoint*($\mathcal{P}_1$,$\mathcal{S}'$);
$\mathcal{P}'_2$ =*FiltrerPoint*($\mathcal{P}_2$,$\mathcal{S}'$);
$\mathcal{S}_1 = \mathcal{CH}^+(\mathcal{P}'_1)$;
$\mathcal{S}_2 = \mathcal{CH}^+(\mathcal{P}'_2)$;
renvoyer $\mathcal{S}_1 \circ \mathcal{S}' \circ \mathcal{S}_2$;
fin;
fin.

3.1.4 Un exemple

On considére un ensemble de 14 points dont on veut calculer l'enveloppe convexe supérieure (cf. figure 3.2). La médiane en abscisse de

$$\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}\}$$

est x_{10}. On sépare donc $\mathcal{P}$ en deux paquets $\mathcal{P}_1$ et $\mathcal{P}_2$

$$\mathcal{P}_1 = \{p_1, p_4, p_6, p_7, p_{10}, p_{13}, p_{14}\}$$

et

$$\mathcal{P}_2 = \{p_2, p_3, p_5, p_8, p_9, p_{10}, p_{11}, p_{12}\}.$$

On trouve par programmation linéaire le pont (p_1, p_3) qui permet de filtrer $\mathcal{P}_1$ en $\mathcal{P}'_1 = \{p_6, p_7\}$ et $\mathcal{P}_2$ en $\mathcal{P}'_2 = \{p_2, p_5, p_9, p_{11}, p_{12}\}$. On résoud alors de façon récursive les sous-problèmes $\mathcal{P}'_1$ et $\mathcal{P}'_2$.
Note : Il ne s'agit pas de diviser les données en une partition mais plutôt de diviser le problème en sous-problèmes. Ici, les points de $\mathcal{P}'_1$ ne peuvent intervenir dans la solution de $\mathcal{P}'_2$ (les problèmes générés sont indépendants). De plus, tout objet construit est persistant, c'est-à-dire tout objet construit ne sera jamais détruit (les faces calculées sont persistantes dans le cas d'enveloppe convexe; algorithme de type *glouton*).

3.1.5 Complexité de l'algorithme

L'algorithme du calcul de l'enveloppe convexe supérieure d'un ensemble de points $\mathcal{P}$ est décrit dans l'encadré 3.4. Il est aisé de remarquer que le paradigme de fusion avant conquête a été utilisé (comparer les algorithmes 3.2 et 3.4). La complexité s'exprime à la fois en fonction de la taille des entrées n et de la sortie h. On a

$$c(n, h) = O(n) + c(n_1, h_1) + c(n_2, h_2),$$

$$c(1, 1) = 1,$$

avec $n_1, n_2 \leq \lceil \frac{n}{2} \rceil$ et $h_1 + h_2 + 1 = h$. En effet, chaque arête de l'enveloppe convexe supérieure n'est trouvée qu'une seule fois puisqu'elle a été détectée par un pont $[p_i, p_j]$ et que les points $p = (x, y)$ restant après le filtre vérifient $x \notin]x_i, x_j[$. Ainsi les médianes en abscisse des nouveaux paquets $\mathcal{P}'_1$ et $\mathcal{P}'_2$ ne seront pas dans l'intervalle $[x_i, x_j]$ et les nouvelles arêtes détectées seront bien différentes des précédentes. L'analyse de la complexité est identique à celle décrite dans la section 3.1.1. Notons que nous pouvons affiner cet algorithme en modifiant le filtrage des points. On peut utiliser la méthode

throw-away [AT78] par exemple. Cette méthode consiste à calculer quatre points extrémaux qui permettent d'éliminer beaucoup de points pour des données suivant les distributions usuelles.

Kirkpatrick et Seidel ont prouvé la borne inférieure du calcul de l'enveloppe convexe de points en dimension 2 en $\Omega(n \log h)$ en formalisant le problème sous forme d'arbre de décision algébrique (cf. [BO83, BOKR86]).

Théorème 3 *L'algorithme de Kirkpatrick et Seidel [KS82, KS86] calcule l'enveloppe convexe de n points de* $\mathbb{E}^2$ *par un algorithme adaptatif en temps optimal* $\Theta(n \log h)$.

Remarque 3.1 *Notons qu'à partir de l'équation de complexité (3.1), nous obtenons l'encadrement suivant:* $O(n + h \log h) \leq c(n, h) \leq O(n \log h)$. *Par exemple l'algorithme aura un coût* $O(n + h \log h)$ *si, dès la première étape,* $|\mathcal{P}'_1| + |\mathcal{P}'_2| = O(h)$. *Toutefois, l'algorithme n'est pas optimal dans toutes les configurations; prenons le cas d'un nuage de points uniformément distribués dans un carré (*$h = \tilde{O}(\log n)$*). L'algorithme aura une complexité en moyenne de* $\tilde{O}(n \log \log n)$ *alors que de nombreux algorithmes linéaires existent dans ce cas précis [Mau84, BGJR91]. Néanmoins, il est facile de modifier cet algorithme en gardant une complexité en* $O(n \log h)$ *et dont la complexité dans le cas de points uniformément répartis dans un carré soit* $\tilde{O}(n)$. *Remarquons que Mac Queen et Toussaint [MT85] discutèrent sur l'ultimalité de cet algorithme en pratique et en proposèrent deux variantesen modifiant la phase de filtrage.*

3.1.6 L'analyse de la complexité revisitée

On peut modéliser l'algorithme décrit dans l'encadré 3.4 par l'intermédiaire d'un arbre $\mathcal{T}$ dont les nœuds contiennent l'ensemble de points courant. Ainsi, la racine aura pour nœud l'ensemble de départ $\mathcal{P}$ et ses fils gauche et droit seront respectivement $\mathcal{P}'_1$ et $\mathcal{P}'_2$. Calculer un pont à un nœud v donné dont l'ensemble de points correspondant est $\mathcal{P}_v$ coûte $O(|\mathcal{P}_v|)$. On associe à l'arbre $\mathcal{T}$ son coût $c(\mathcal{T})$ défini comme la somme des coûts de ses nœuds: $c(\mathcal{T}) = \sum_{v \in \mathcal{T}} c(\mathcal{P}_v)$. Cet arbre a exactement h nœuds. Les fils gauche et droit d'un nœud v ont au plus $\frac{|\mathcal{P}_v|}{2}$ points; nous dirons que cet arbre a un facteur de décimation $\frac{1}{2}$ (fading factor). Considérons l'arbre binaire complet $\mathcal{T}'$ à h nœuds ayant comme facteur de décimation $\frac{1}{2}$. Son coût est trivialement plus élevé que $\mathcal{T}$, c'est-à-dire $c(\mathcal{T}') \geq c(\mathcal{T})$. Or, $c(\mathcal{T}') = \sum_{i=0}^{\lfloor \log_2 h \rfloor} O(2^i \times \frac{n}{2^i}) = O(n \log h)$. Ce type d'analyse a notamment été utilisé dans l'analyse de l'algorithme d'Edelsbrunner et Shi [ES91] de calcul d'enveloppe convexe de points

TAB. 3.1 – *Comparaison des complexités entre un algorithme de type diviser-pour-régner et un algorithme de type fusion-avant-conquête.*

α	$\alpha = 1$	$\alpha > 1$
Diviser-pour-régner	$O(n \log n)$	$O(n^\alpha)$
Fusion-avant-conquête	$O(n \log h)$	$O(n^\alpha)$

en dimension trois, puis repris par Chan et Snoeyink [CSY95], et Amato et Ramos [AR96].

D'une manière générale, les équations de complexité $c(n)$ générées par le paradigme *diviser-pour-régner* sont du type:

$$c(n) = \begin{cases} O(1) & \text{si } n < k, \\ O(n) + O(n^\alpha) + c(\lfloor \frac{n}{2} \rfloor) + c(\lceil \frac{n}{2} \rceil) & \text{sinon.} \end{cases} \tag{3.3}$$

où k est une constante et α est un paramètre reflétant le coût $O(n^\alpha)$ de l'étape de fusion des solutions des sous-problèmes. Les équations de complexité $c(n, h)$ générées par le paradigme *fusion-avant-conquête* sont, quant à elles, du type:

$$c(n,h) = \begin{cases} O(1) & \text{si } h < k, \\ O(n) + O(n^\alpha) + c(n_1, h_1) + c(n_2, h_2) & \text{sinon.} \end{cases} \tag{3.4}$$

avec $n_1, n_2 \leq \frac{n}{2}$, où k est une constante et α est un paramètre reflétant le coût $O(n^\alpha)$ de l'étape de fusion des solutions des deux sous-problèmes avant même d'avoir calculé complètement ces solutions.

On peut modéliser le déroulement d'un algorithme de type fusion-avant-conquête par un arbre $\mathcal{T}$ d'appels récursifs (défini ci-dessus). Clairement, l'arbre quasi-complet $\mathcal{T}'$ à h nœuds a un coût plus élevé que $\mathcal{T}$. Ainsi, nous obtenons la majoration suivante:

$$c(n,h) \leq O(n) + \sum_{i=0}^{\lceil \log h \rceil} 2^i (\frac{n}{2^i})^\alpha.$$

Remarquons que dans ce modèle, on n'élimine aucune donnée dans les nœuds internes.

La table 3.1 met en valeur les différents types de complexité obtenus pour $\alpha \in [1, +\infty)$. Notons que le cas intéressant pour l'approche fusion-avant-conquête est $\alpha = 1$.

Nous donnons ci-dessous un lemme (Lemme 3.2) sur la complexité d'un algorithme adaptatif, qui à partir de n données initiales, en crée $d\beta n$ au

premier niveau de l'arbre de récursion, puis $(d\beta)^2 n$ au second, etc. Ce type d'analyse peut être utilisé lorsqu'on n'arrive pas à partitionner complètement les données pour les sous-problèmes générés (par exemple, si les données ne sont pas ponctuelles).

Lemme 3.2 *Soit $\mathcal{T}$ un arbre d'arité d à h nœuds tel que chaque nœud ait au plus une fraction β ($0 \leq \beta \leq 1$) des données de son père. Si le coût d'un nœud est linéaire en son nombre de données et si la racine a n données alors le coût total de l'arbre est majoré par: $O(n \log_d h)$ si $d\beta = 1$, $O(nh^{\log_d(d\beta)})$ si $d\beta > 1$ et $O(n)$ si $d\beta < 1$.*

3.1.7 Un ultime algorithme?

Dans le papier originel de Kirkpatrick et Seidel [KS86], les auteurs posent la question de l'ultimalité de leur algorithme en choisissant comme titre du papier: "The Ultimate Planar Convex Hull Algorithm?". Tout d'abord, remarquons que l'algorithme décrit est optimal à une constante multiplicative près dans la notation $O()$ (cette constante fut d'ailleurs améliorée par Chan *et al.* en considérant une méthodologie différente [CSY95]). Nous insistons sur le fait que cet algorithme est optimal pour le modèle real RAM et pour les paramètres n et h. Toutefois, il ne serait pas surprenant de trouver de nouveaux paramètres ou un nouveau modèle de machine pour lesquels nous puissions obtenir un algorithme de meilleure efficacité. Notons également que ces algorithmes ne tiennent pas compte de la distribution (statistique) des données et que nous pouvons aisément calculer en temps moyen linéaire l'enveloppe convexe d'un nuage de points uniformément réparti dans un polygone convexe [GS88, Gol90]. La taille moyenne de cette enveloppe convexe est $h = O(\log n)$ [Dev91]. Si nous appliquons sur ce nuage de points l'algorithme mariage-avant-conquête, nous obtenons un algorithme de complexité $\Theta(n \log\log n)$ qui est donc moins bon que certains algorithmes statistiques. Nous pouvons également tenir compte des contraintes sur l'ensemble de données; par exemple si nos n points sont répartis sur k droites connues, nous obtenons un algorithme de calcul d'enveloppe convexe en temps $O(n+k \log h)$ (avec $h \leq k$), etc.

3.1.8 Paramètres significatifs

Comment trouver de nouveaux ou d'autres paramètres de sensibilité à la sortie ou aux données? Il suffit d'exhiber par rapport aux paramètres actuels, deux situations dont les paramètres soient identiques mais dont la combinatoire d'une quantité, qui définira le nouveau paramètre, soit différente. Ayant

trouvé ce nouveau paramètre, on essaye de créer un algorithme dont la complexité en temps de calcul (ou en toute autre ressource critique) dépende de celui-ci.

Toutefois, comme nous le ferons remarquer à plusieurs reprises dans cette thèse, tous les paramètres ne sont pas significatifs, c'est-à-dire être un paramètre intervenant dans l'analyse de la complexité d'un algorithme.

Prenons, pour illustrer ce principe, l'union d'intervalles. Soit $\mathcal{I} = \{I_1, ..., I_n\}$ un ensemble de n intervalles. On désire calculer l'union $\cup\mathcal{I} = \cup_{i=1}^{n} I_i$. En triant les $2n$ extrémités et en faisant une marche de gauche à droite, nous pouvons calculer $\cup\mathcal{I}$ en temps $O(n \log n)$. Pour un n donné, en examinant quelques unions, on remarque que le nombre de composantes connexes $c = \mathcal{C}(\mathcal{I})$ de $\cup\mathcal{I}$ varie de 1 à n. On peut alors considérer deux paramètres n et c et essayer d'obtenir un algorithme dont la complexité soit meilleure que $O(n \log n)$, e.g. $o(n \log n)$ dans le cas où $c = O(1)$. Dans ce cas précis, nous ne pouvons obtenir de tels algorithmes; en effet, nous montrerons dans cette thèse comment nous pouvons calculer l'union en $\Theta(n \log c^*(\mathcal{I}))$ où $c^*(\mathcal{I})$ est le nombre minimal de points perçant $\mathcal{I}$. Notons qu'il existe des configurations de $\mathcal{I}$ pour lesquelles $\mathcal{C}(\mathcal{I}) = 1$ mais $c^*(\mathcal{I}) = \lceil \frac{n}{2} \rceil$. Toutefois cela ne suffit pas pour montrer la borne inférieure. Il suffit de montrer que trouver l'existence d'une seule composante connexe a une complexité de $\Omega(n \log n)$ opérations élémentaires. Nous illustrons ce fait en considérant le problème suivant:

Problème 2 (UNIFORM GAP) *Soit $\mathcal{S}$ un ensemble de n réels $x_1, ..., x_n$ et ϵ un réel strictement positif. Déterminer si les différences consécutives de $\mathcal{S}$ sont uniformément égales à ϵ.*

Lee et Wu [LW86] ont démontré que UNIFORM GAP demande $\Omega(n \log n)$ opérations élémentaires sur le modèle d'arbre de décision algébrique.

En effet, supposons que l'on puisse avoir un algorithme de complexité $f(n) = o(n \log n)$ qui calcule la composante connexe de $\mathcal{I}$ dans le cas où $\mathcal{C}(\mathcal{I}) = 1$. Alors, par réduction en temps linéaire nous pouvons répondre OUI au problème UNIFORM GAP (voir ci-dessous). Or UNIFORM GAP est un problème ayant pour borne inférieure $\Omega(n \log n)$. D'où la contradiction. Ainsi il n'existe pas d'algorithme adaptatif, sur le modèle RAM, pour le paramètre $c = \mathcal{C}(\mathcal{I})$, qui batte l'algorithme (naïf) optimal. Nous prouvons dans [Nie94] qu'il n'existe pas d'algorithme adaptatif pour calculer l'intersection du bord d'un polytope avec le bord d'un objet convexe en dimension d.

Finalement, nous montrons l'importance du modèle de machine en montrant que les bornes inférieures dépendent du modèle considéré. Gonzalez [Gon75] constata également ce phénomène en donnant un algorithme linéaire pour MAX GAP(cf. [PS85], pp. 260) si on utilise la partie

entière alors que la borne inférieure est $\Omega(n \log n)$ sur les arbres de décision algébrique.

Dans ce même modèle, il devient alors également aisé d'obtenir un algorithme linéaire pour UNIFORM GAP. L'encadré 3.5 décrit notre algorithme.

Algorithme 3.5: *Un algorithme linéaire pour* UNIFORM GAP. *On suppose* $\min_{i=1}^{n} D_i = 0$.

Fonction UNIFORMGAP($\mathcal{D} = \{D_1, ..., D_n\}$, ϵ) : booléen ;
début
pour i de 0 à $n-1$ faire $T[i] := 0$;
pour i de 1 à n faire
    $k = \lfloor \frac{D_i}{\epsilon} \rfloor$;
    si $k > n-1$ alors renvoyer faux;
    si $\frac{D_i}{\epsilon} \neq k$ alors renvoyer faux;
        sinon $T[k] = 1$;
si $\sum_{i=0}^{n-1} T[i] \neq n$ alors renvoyer faux;
        sinon renvoyer vrai;
fin;

En insérant de nouveaux paramètres dans l'étude de problèmes, nous pouvons espérer construire des algorithmes plus performants. Par exemple, calculer la distance maximale entre deux points d'un ensemble de points est $\Omega(n \log n)$-dur [Bro80]. Toutefois, si on calcule le plus petit disque englobant et que nous regardons les points sur son bord, deux cas de figure peuvent se présenter : soit il y a exactement deux points et dans ce cas là, ils définissent le diamètre de l'ensemble de points. Soit il y a exactement trois points. Dans ce cas, on peut éliminer tous les points contenus dans le triangle défini par ces points et appliquer l'algorithme standard (cf. [PS85], page 177). Notons que puisque le calcul du plus petit disque englobant demande un temps linéaire (problème de type Programmation Linéaire [Wel91]) nous pouvons calculer dans le premier cas la distance maximale en temps linéaire.

D'une manière générale, la question philosophique sous-jacente est la notion d'optimalité en absolu, c'est-à-dire pour tous les paramètres "imaginables"...

3.1.9 Algorithme parallèle adaptatif pour le calcul de l'enveloppe convexe de points de $\mathbb{E}^2$

On décrit dans ce paragraphe comment on peut paralléliser l'algorithme précédant sur un modèle parallèle EREW PRAM SIMD. L'idée sous-jacente est de paralléliser chacune des étapes de l'algorithme [KS82, KS86] (le lecteur trouvera tous les détails de cette parallélisation dans [Akl84, Akl85a]). On dispose de $n^{1-\epsilon}$ processeurs, $0 < \epsilon < 1$, numérotés $P_1, ..., P_{n^{1-\epsilon}}$. L'algorithme calcule l'enveloppe convexe supérieure d'un ensemble $\mathcal{S}$ de n points dans le plan $\mathbb{E}^2$ en temps $O(n^\epsilon \log h)$. Le travail fourni par l'algorithme est ainsi optimal:

$$\underbrace{W(n)}_{\text{travail}} = \underbrace{C(n)}_{\text{coût}} \times \underbrace{P(n)}_{\text{nombre de processeurs}}$$

$$W(n) = O(n^\epsilon \log h) \times O(n^{1-\epsilon}) = O(n \log h)$$

La difficulté à surmonter dans ce cas est l'étape de récursivité. En effet, si on sépare les données en deux paquets comme dans l'algorithme séquentiel alors on devra résoudre deux sous-problèmes de taille $\lceil \frac{n}{2} \rceil$ avec en tout $n^{1-\epsilon}$ processeurs. Par hypothèse de récurrence, on sait résoudre un problème de taille n en temps $c(n)$ avec $n^{1-\epsilon}$ processeurs. Pour résoudre un sous-problème de taille $\lceil \frac{n}{2} \rceil$, il faudrait avoir $(\frac{n}{2})^{1-\epsilon}$ processeurs alors que l'on en dispose de $\frac{n^{1-\epsilon}}{2}$ (il en faut une moitié pour le paquet S_1 et une autre pour S_2). On doit donc diviser $\mathcal{S}$ de façon à résoudre récursivement le calcul de l'enveloppe convexe supérieure en parallèle sur tous les paquets et en utilisant les $n^{1-\epsilon}$ processeurs. On procède comme suit:

Déterminer les $(2^{\frac{1}{\epsilon}} - 1)$ points $p_1, ..., p_{2^{\frac{1}{\epsilon}}-1}$ de $\mathcal{S}$ qui partagent suivant leur abscisse $\mathcal{S}$ en $2^{\frac{1}{\epsilon}}$ paquets de taille $\frac{n}{2^{\frac{1}{\epsilon}}}$ constitués des points dont les abscisses se trouvent entre deux abscisses de points consécutifs $[x(p_i), x(p_{i+1})]$. Par convention, on choisit p_0 (respectivement $p_{2^{\frac{1}{\epsilon}}}$) comme étant le point ayant l'abscisse minimale (resp. maximale). On note S_i^1 le paquet $[x(p_{2i}), x(p_{2i+1})]$ et S_i^2 le paquet $[x(p_{2i+1}), x(p_{2i+2})]$ avec $i \in [0, 2^{\frac{2}{\epsilon}}]$. L'union des sous-ensembles de points de $S_i^1, i \in [1, 2^{\frac{1}{\epsilon}} - 1]$ (resp. $S_i^2, i \in [1, 2^{\frac{1}{\epsilon}} - 1]$) est notée S_1 (resp. S_2). On calcule alors les ponts aux droites $x = x(p_i),\ i \in [1, 2^{\frac{1}{\epsilon}} - 1]$ en utilisant une version parallèle *BridgePar()* de l'algorithme du calcul du pont *Bridge()*.

On appelle alors la procédure *UpperConvexHullPar()* sur S_1 (resp. S_2) en utilisant $\frac{n^{1-\epsilon}}{2^{\frac{1}{\epsilon}}-1}$ processeurs pour chaque paquet S_i^1 (respectivement sur S_i^2). L'algorithme dans le cas où $\epsilon = \frac{1}{2}$ est décrit dans l'encadré 3.6.

Le calcul du pont à $\Delta_i, i \in [1, 2^{\frac{1}{\epsilon}} - 1]$ se fait par appel de la procédure *BridgePar()* qui parallélise la fonction séquentielle *Bridge()*. On assigne à

Algorithme 3.6: *Algorithme parallèle ($\epsilon = \frac{1}{2}$) de calcul de l'enveloppe convexe supérieure d'un ensemble de points $\mathcal{S}$ de $\mathbb{E}^2$:*

Procedure *UpperConvexHullPar*$(\mathcal{S} = \{P_1, ..., P_n\}, k, m)$
(* Détermine l'enveloppe convexe supérieure entre $x = x(P_k)$ et $x = x(P_m)$ *)
début
Calculer p_1, p_2, p_3 les 3 points de $\mathcal{S}$ qui partitionnent $\mathcal{S}$
en paquets de taille quasi-équivalentes:
$S_1^1 = \{p \in \mathcal{S} | x(p) \leq x(p_1)\}$;
$S_2^1 = \{p \in \mathcal{S} | x(p_1) < x(p) \leq x(p_2)\}$;
$S_1^2 = \{p \in \mathcal{S} | x(p_2) < x(p) \leq x(p_3)\}$;
$S_2^2 = \{p \in \mathcal{S} | x(p) > x(p_3)\}$;

Pour chaque droite $\Delta_i : x = x(p_i), i \in [1...3]$, calculer le pont à Δ_i;
(* on note $[p_{i_l}, p_{j_l}]$ le pont à Δ_l *)
Faire en parallèle
si $i_1 = k$ alors renvoyer i_1
sinon renvoyer *UpperConvexHullPar*(S_1^1, k, i_1);
si $j_1 = i_2$ alors renvoyer j_1
sinon renvoyer *UpperConvexHullPar*(S_2^1, j_1, i_2);
fin; (* fin faire *)

Faire en parallèle
si $j_2 = i_3$ alors renvoyer j_2
sinon renvoyer *UpperConvexHullPar*(S_1^2, j_2, i_3);
si $j_3 = m$ alors renvoyer m
sinon renvoyer *UpperConvexHullPar*(S_2^2, j_3, m);
fin; (* fin faire *)
fin

chaque processeur $P_i, i \in [1, n^{1-\epsilon}]$ un paquet de taille n^ϵ. Le processeur P_i apparie alors ces points en $\lfloor \frac{n^\epsilon}{2} \rfloor$ paires (p_a, p_b) et calcule la pente de la droite : $\frac{y(p_b)-y(p_a)}{x(p_b)-x(p_a)}$. On calcule alors la médiane des pentes K par l'algorithme parallèle optimal en travail *SelectPar()* décrit dans l'annexe B en $O(n^\epsilon)$ avec $O(n^{1-\epsilon})$ processeurs. On calcule le (ou les, dans le cas où le pont a cette pente) point qui maximise $y(p_i) - K \times x(p_i)$ en $O(n^\epsilon)$ (d'abord en séquentiel sur chaque processeur puis on réalise un préfixe parallèle). On diffuse à tous les processeurs l'indice du point réalisant le maximum en $O(\log n^\epsilon)$. Chaque processeur épure ses n^ϵ points en tenant compte du couple (p_a, p_b) et de la position de la droite support de pente K. Puis on reconcentre les données dans un tableau en diffusant l'adresse de base et les sommes partielles. Cette opération se fait en $O(\log n^{1-\epsilon} + n^\epsilon) = O(n^\epsilon)$. Après ces étapes, on relance la procédure *BridgePar()* sur au plus $\lceil \frac{3n}{4} \rceil$ points. La complexité de *BridgePar()*, $b(n)$, est donnée par l'équation de récurrence:

$$b(n) = \begin{cases} O(1) & \text{si } n < 4 \\ n^\epsilon + b(\frac{3n}{4}) & \text{sinon} \end{cases}$$

Trivialement, on a $b(n) = O(n^\epsilon)$ (on suppose $b(n) = K \times n^\epsilon$ et on trouve la condition sur K: $K \geq 1 + (\frac{3}{4})^\epsilon$).

En notant $c(n, h)$ la complexité en temps de la procédure *UpperConvexHullPar()* (avec h la taille de l'enveloppe convexe supérieure), on a l'équation de récurrence:

$$c(n,h) = \begin{cases} O(1) & \text{si } n < 3, \\ cn^\epsilon + \max_{h=h_1+h_2}(\max_{h_1^j} c(|S_j^1|, h_1^j) + \max_{h_2^j} c(|S_j^2|, h_2^j)) & \text{sinon.} \end{cases}$$

On trouve en suivant la même analyse que [KS82, KS86] que $c(n, h) = O(n^\epsilon \times \log h)$. Le travail fourni par l'algorithme est optimal puisque $W(n) = \Theta(n \log h)$. On peut également modéliser le déroulement de l'algorithme par un arbre ayant comme facteur de décimation $\frac{1}{4}$ et analyser le coût de cet arbre (cf. Section 3.1.1). Le temps de calcul est alors celui du plus long chemin de la racine à une feuille.

3.1.10 Algorithme de calcul d'enveloppe convexe adaptatif dans l'espace $\mathbb{E}^3$

Nous décrivons ci-dessous le calcul d'enveloppe convexe d'un ensemble de n points $\mathcal{P} = \{p_1, ..., p_n\}$ dans l'espace euclidien $\mathbb{E}^3$ (on note O-XYZ le repère associé). L'algorithme suit la méthodologie de *fusion avant conquête*. Nous résumons les caractéristiques en mettant en relief le paradigme. Les

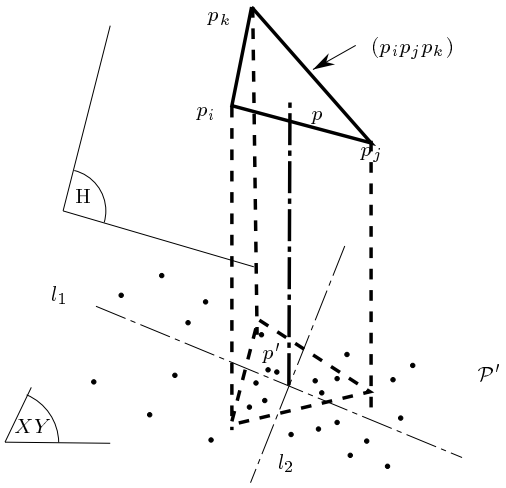


FIG. 3.3 – *Calcul d'une facette (2-face) de $\mathcal{P}$ par programmation linéaire.*

preuves ébauchées ou mentionnées sont décrites dans [ES91]. Nous montrons par la description de cet algorithme la difficulté à utiliser l'approche mariage avant conquête en dimensions supérieures.

3.1.11 Description de l'algorithme

On suppose, dans la suite, que les points de $\mathcal{P}$ sont en position générale, c'est-à-dire qu'il n'existe pas de quadruplet $(p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4})$ de $\mathcal{P}$ tel que $p_{i_1}, p_{i_2}, p_{i_3}$ et p_{i_4} soient coplanaires et qu'il n'existe pas de triplet $(p_{i_1}, p_{i_2}, p_{i_3})$ tel que p_{i_1}, p_{i_2} et p_{i_3} soient colinéaires. On cherche à calculer l'enveloppe convexe supérieure $\mathcal{CH}^+(\mathcal{P})$ puisque l'on pourra calculer $\mathcal{CH}^-(\mathcal{P})$ et calculer $\mathcal{CH}(\mathcal{P})$ en fusionnant le bord des enveloppes convexes en temps $\Theta(h)$.

Le cas trivial (cas d'arrêt de la conquête correspondant à la constante b de l'algorithme décrit dans l'encadré 3.2) est celui de l'enveloppe convexe de 4 points (en dimension d, le d-simplexe est consitué de $d+1$ points). Cette enveloppe convexe est un tétraèdre (3-simplexe). On divise le problème de calculer l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$ en 4 sous-problèmes dont les sous-ensembles correspondants sont de tailles quasi-équivalentes $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ et $\mathcal{P}_4$ tels que $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2 \uplus \mathcal{P}_3 \uplus \mathcal{P}_4$ (on partitionne les données). On réalise cette partition en projetant verticalement $\mathcal{P}$ en $\mathcal{P}_{XY} = \mathcal{P}'$ sur le plan XY puis par programmation linéaire en partitionnant $\mathcal{P}'$ par deux droites l_1 et l_2, qui se coupent en p' (ham sandwich cut). Les droites l_1 et l_2 divisent le plan XY

en quatre quadrants tel que chaque quadrant contient au plus $\lceil \frac{n}{4} \rceil$ points de $\mathcal{P}'$. Cette étape se réalise grâce à l'algorithme de Megiddo en temps linéaire (cf. [Meg85], méthodologie bisection-décimation, prune-and-search).

$$l_1 : a_1 x + b_1 y + c_1 = 0,$$

$$l_2 : a_2 x + b_2 y + c_2 = 0,$$

$$p' = (p'_x, p'_y) = \left(-\frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right).$$

On cherche alors à fusionner les sous-problèmes générés par les sous-ensembles $\mathcal{P}_i,\ i \in \{1,2,3,4\}$ suivant la droite passant par p' et parallèle à l'axe Z (on note l la droite passant par p' et parallèle à Z). On résoud un problème de programmation linéaire à trois variables pour trouver le plan H, support de $\mathcal{P}$, qui minimise l'altitude (par rapport à Z) du point d'intersection défini par le plan H avec l.

$$\underline{\text{Programme linéaire}} : \left\{ \begin{array}{l} \underline{\text{minimise } ap'_x + bp'_y + c} \\ \\ z_1 - ax_1 - by_1 - c \leq 0 \\ z_2 - ax_2 - by_2 - c \leq 0 \\ \qquad \cdots \\ z_n - ax_n - by_n - c \leq 0 \end{array} \right.$$

Une fois que l'on a trouvé H par résolution du programme linéaire, on recherche en $O(n)$ les trois points p_i, p_j, p_k qui se trouvent sur H. Ils définissent le triangle $(p_i p_j p_k)$ (cf. figure 3.3). L'équation du plan H support au triangle $(p_i p_j p_k)$ est

$$\begin{vmatrix} x & y & z & 1 \\ p_{i_x} & p_{i_y} & p_{i_z} & 1 \\ p_{j_x} & p_{j_y} & p_{j_z} & 1 \\ p_{j_x} & p_{j_y} & p_{j_z} & 1 \end{vmatrix} = 0.$$

On élimine (filtrage des données) les points de $\mathcal{P}'_i$ qui se trouvent dans $(p'_i p'_j p'_k)$ (le triangle en projection dans le plan XY) comme l'indique la figure 3.3. On obtient ainsi quatre paquets définis par le partitionnement des deux droites l_1 et l_2 après élimination des points à l'intérieur du triangle $(p'_i p'_j p'_k)$: $\mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}'_3$ et $\mathcal{P}'_4$. On doit maintenant garantir que les paquets qui définissent les sous-problèmes n'agissent pas les uns sur les autres. On définit

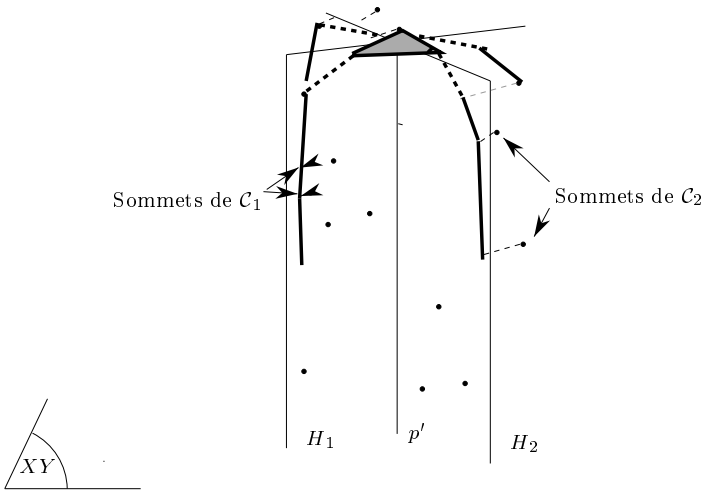


FIG. 3.4 – *Projection des points de $\mathcal{P}$ sur H_1 suivant $H_2 \cap H$ et sur H_2 suivant $H_1 \cap H$ et calcul des enveloppes convexes supérieures $\mathcal{C}_1$ et $\mathcal{C}_2$.*

le plan H_1 (respectivement H_2) passant par l_1 (respectivement l_2) et perpendiculaire au plan XY. On projette $\mathcal{P}$ sur H_1 (respectivement H_2) suivant la direction $H_2 \cap H$ (respectivement $H_1 \cap H$) en $\mathcal{P}_{H_1}$ (respectivement $\mathcal{P}_{H_2}$) et on calcule l'enveloppe convexe $\mathcal{CH}(\mathcal{P}_{H_1})$ (respectivement $\mathcal{CH}(\mathcal{P}_{H_2})$) (cf. figure 3.4). On note $\mathcal{C}_1$ (respectivement $\mathcal{C}_2$) la ligne polygonale définie par les points de $\mathcal{CH}(\mathcal{P}_{H_1})$ (respectivement $\mathcal{CH}(\mathcal{P}_{H_2})$) relevés dans l'espace $\mathbb{E}^3$ et $\mathcal{C}'_1$ (respectivement $\mathcal{C}'_2$) comme la projection de $\mathcal{C}_1$ (respectivement $\mathcal{C}_2$) sur le plan XY. Ces deux lignes polygonales $\mathcal{C}'_1$ et $\mathcal{C}'_2$ sont des pseudo-segments (c'est-à-dire se coupent en au plus un point) qui partitionnent $\mathcal{CH}(\mathcal{P}')$ en un ensemble de régions $\mathcal{R}$. La figure 3.5 montre un partitionnement de $\mathcal{P}'$ suivant $\mathcal{C}'_1$ et $\mathcal{C}'_2$. La ligne polygonale $\mathcal{C}'_1$ (respectivement $\mathcal{C}'_2$) est monotone suivant l_1 (respectivement l_2) et passe par le triangle $(p_i p_j p_k)$ (les directions de projection assurent cette propriété). L'ensemble $\mathcal{P}'$ est ainsi divisé en un ensemble de régions $\mathcal{R}$. Pour chaque région R de $\mathcal{R}$, on détermine les points de $\mathcal{P}'$ (et donc de $\mathcal{P}$ en les relevant) qui se trouvent à l'intérieur de chaque région (on montre qu'il y en a au plus $\lceil \frac{3n}{4} \rceil$) et on calcule récursivement les différentes enveloppes convexes supérieures correspondant aux régions (cf. algorithme 3.7).

3.1.12 Complexité de l'algorithme

Nous renvoyons le lecteur à l'article [ES91] pour les justifications précises de l'algorithme (notamment sa validité). L'algorithme a une complexité

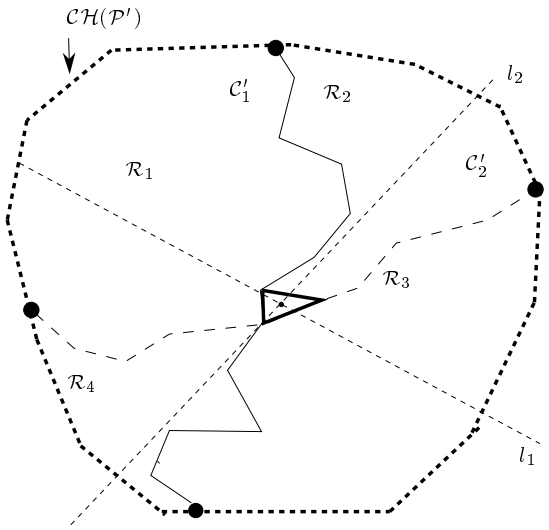


FIG. 3.5 – *Partitionnement en régions de $\mathcal{P}'$ suivant $\mathcal{C}_1$ et $\mathcal{C}_2$.*

$O(n \log^2 h)$ en temps et $O(n)$ en mémoire. En effet, si l'on note $c(n,h)$ la complexité de l'algorithme pour une taille des données de n et une taille h de la structure du résultat, nous avons les relations suivantes :

$$c(n,h) = \begin{cases} O(n) & \text{si } h \leq 3, \\ O(n \log h) + \sum_{i=1}^{k} c(|\mathcal{R}'_i|, h_i) & \text{sinon.} \end{cases}$$

À chaque niveau de l'arbre de récursion, l'algorithme travaille sur des régions polygonales fermées $\mathcal{R}'_1$, ..., $\mathcal{R}'_i$ de frontières $\mathcal{B}'_1$, ..., $\mathcal{B}'_i$. Notons $|\mathcal{R}'_i|$ le nombre de points de $\mathcal{P}'$ situés dans la région $\mathcal{R}'_i$ et $|\mathcal{B}'_i|$ le nombre d'arêtes de la frontière $\mathcal{B}'_i$ de $\mathcal{R}'_i$. Le coût de l'algorithme à cet *étage de récursion* est

$$\sum_{i=1}^{k} |\mathcal{R}'_i| \log h \leq (n + \sum_{i=1}^{k} |\mathcal{B}'_i|) \log h$$

Or chaque frontière $\mathcal{B}'_i$ de $\mathcal{R}'_i$ est un polygone simple (monotonie suivant les droites séparatrices). Une arête de $\mathcal{B}'_i$ apparait au plus dans deux régions. Ainsi nous pouvons majorer la taille totale du partionnement par:

$$\sum_{i=1}^{k} \mathcal{B}'_i \leq 6h - 12 \leq 6n.$$

Le coût total à chaque niveau de récursion est donc majoré par $7n \log h$. Il reste donc à majorer la hauteur de l'arbre afin de déduire la complexité

finale. Comme chaque région $\mathcal{R}'_i$ a au plus $\lceil \frac{3n}{4} \rceil$ points de $\mathcal{P}$, on en déduit que la hauteur de l'arbre est $O(\log h)$ (l'arbre a $O(h)$ nœuds et chaque nœud de l'arbre a au plus $\frac{3}{4}$ des points de son père). La complexité de l'algorithme est $O(n \log^2 h)$ en temps et $O(n)$ en mémoire. Cet algorithme de calcul d'enveloppe convexe de points de $\mathbb{E}^3$ n'est pas optimal (cf. l'algorithme optimal randomisé de Clarkson et Shor dérandomisé par Chazelle et Matousek [CS89, CM94]). Le goulot d'étranglement de cet algorithme se situe au niveau du partage des données en paquets (les régions $\mathcal{R}$) tels que la solution de chaque paquet soit indépendante des autres (définition d'un sous-problème indépendant). Cette étape nécessite le calcul d'une enveloppe convexe de points dans les plans H_1 et H_2 puis le partitionnement de $\mathcal{P}$ suivant $\mathcal{C}_1$ et $\mathcal{C}_2$ afin d'assurer le fractionnement correct des données.

Théorème 4 *On peut calculer l'enveloppe convexe de n points de $\mathbb{E}^3$ par un algorithme sensible à la sortie en temps $O(n \log^2 h)$ et en espace mémoire $O(n)$ par le paradigme fusion avant conquête.*

En conclusion, l'approche mariage-avant-conquête semble bien adaptée aux calculs de structures planaires mais s'adapte difficilement en dimensions supérieures. Deux approches essentielles permettent d'obtenir des algorithmes adaptatifs en grandes dimensions pour le calcul d'enveloppe convexe:

- l'approche papier cadeau basée sur des requêtes avec un prétraitement adaptatif [Cha95b] (cette technique fut également indépendamment utilisée pour le calcul d'enveloppe convexe d'objets [NY95])
- l'approche *diviser pour régner* basé sur l'échantillonnage géométrique en sélectionnant des sous-problèmes respectant un invariant global à chaque étape de récursion [CSY95, AR96].

3.2 Le paradigme prétraitement paresseux [KS85]

La méthode que nous détaillons ci-dessous est générale et peut s'appliquer à beaucoup d'algorithmes non-adaptatifs (c'est-à-dire, ici, non-sensibles à la sortie) qui reposent sur un prétraitement des données. Nous illustrons cette méthode par le problème du calcul des *vecteurs maximaux* ou *maxima* d'un ensemble de points [KS85]. Nous décrivons tout d'abord un algorithme non-adaptatif qui une fois le prétraitement calculé permet de trouver les vecteurs maximaux en temps linéaire. Puis nous montrons dans la partie 3.2.1

Algorithme 3.7: *Calcul de* $\mathcal{CH}^+(\mathcal{P})$ *dans* $\mathbb{E}^3$.

Fonction $\mathcal{CH}^+(\mathcal{P} = \{p_1, ..., p_n\}, \mathcal{B})$: envconvsup ;
début
si $n \leq 3$ alors renvoyer *Triangle*($\mathcal{P}$);
sinon début
p=*HamCut*($\mathcal{P}$);
(* p est le point d'intersection des deux droites l_1 et l_2 *)
$\mathcal{S}'$=*FusionFacette*($\mathcal{P}$,p);
(* FusionFacette renvoit le triangle $\mathcal{S}' = (p_i, p_j, p_k)$ qui coupe la droite $\{(p_x, p_y, z) | z \in \mathbb{R}\}$ *)
$\mathcal{P}'$ =*FiltrerPoint*($\mathcal{P}$,$\mathcal{S}'$);
$\mathcal{C}_1$=$\mathcal{CH}^+(\mathcal{P}'_{H_1})$;
$\mathcal{C}_2$=$\mathcal{CH}^+(\mathcal{P}'_{H_2})$;
$\mathcal{S} \leftarrow \emptyset$;
Pour chaque région $\mathcal{R}'_i$ limitée par $\mathcal{B}'_i$ de
$\mathcal{P}'$ partitionné par $\mathcal{C}'_1$ et $\mathcal{C}'_2$
faire
$\mathcal{S} \leftarrow \mathcal{S}' \circ \mathcal{CH}^+(\mathcal{R}_i, \mathcal{B}'_i)$;
fin faire;
renvoyer $\mathcal{S}$;
fin;
fin.

TAB. 3.2 – *Bornes sur le problème des vecteurs maximaux.*

Dimension	Hypothèses	Complexité $c_d(n,h)$	Remarque
$2,3$		$\Theta(n \log h)$	
$d \geq 4$		$O(n \log^{d-2} h)$	
$d \geq 4$	$n \geq h^2$	$O(n \log^{d-3} h)$	Optimal pour $d = 4$
$d \geq 4$	$n \geq h^{2+\alpha},\ \alpha > 0$	$\Theta(n \log h)$	

comment utiliser un prétraitement adaptatif afin d'obtenir un algorithme sensible à la sortie. Soit $\mathcal{P}$ un ensemble de n points $\mathcal{P} = \{p_1, ..., p_n\}$ de l'espace euclidien $\mathbb{E}^d$ alors on cherche le sous-ensemble $\mathcal{M}$ de $\mathcal{P}$ des vecteurs maximaux. On dit que p_i est maximal s'il n'existe pas de points p de $\mathcal{P}$ tel que p domine p_i ($p \succ p_i$ ou indifféremment $p_i <_M p$). On rappelle que $p \succ p_i$ où $p = (x_1, ..., x_d)$ et $p_i = (y_1, ..., y_d)$ ssi $\forall i \in [1, d],\ x_i > y_i$. On note également $\leq_L$ l'ordre lexicographique usuel :

$$x = (x_1, ..., x_d), y = (y_1, ..., y_d) \in \mathbb{E}^d,$$
$$x \geq_L y \Leftrightarrow \begin{cases} x = y \\ \text{ ou} \\ \exists\ k | \forall\ i \leq k\ x_i = y_i \text{ et } x_{k+1} > y_{k+1} \end{cases}$$

On note $c_d(n,h)$ la complexité du problème qui consiste à trouver les vecteurs maximaux d'un ensemble $\mathcal{P}$ de n points de $\mathbb{E}^d$ de façon adaptative. On résume dans le tableau 3.2 les bornes connues de $c_d(n,h)$. On a $c_d(n,h) = \Omega(n \log h), \forall d \geq 2$ [KLP75] (on remarque que les sommets de l'enveloppe convexe de $\mathcal{P}$ sont des points maximaux).

On étend naturellement la notation "$\leq_M$" aux ensembles comme suit : on dira que $p \leq_M R$ si et seulement si il existe un point $q \in R$ tel que q domine p, c'est-à-dire $p \leq_M q$. De même, on dira que $p \not\leq_M R$ s'il n'existe aucun point de R dominant p.

On note par S/R l'ensemble des points de S qui ne sont pas dominés par des points de R. Soit

$$S/R = \{s \in S | s \not\leq_M R\}$$

On appelle problème d'*affichage*[8] (screening problem) le problème de calculer S/R étant donnés S et R. On note $r = |R|$ et $s = |S|$ et on désigne par $F_d(r, s)$ la complexité en temps de calcul de S/R. Kung, Luccio et Preparata ont démontré dans [KLP75] les résultats suivants:

- $F_1(r, s) = \Theta(r + s)$
- $F_2(r, s) = O((r + s) \log r)$
- Un ensemble fini $R \in \mathbb{R}^2$ peut être représenté par une structure de données telle que S/R peut être calculé en $O(s \log r)$ pour tout ensemble de points $S \in \mathbb{R}^2$. On peut, de plus, calculer cette structure de données de façon incrémentale en temps $O(r \log r)$.

Dans la suite, on explique un algorithme pour calculer les vecteurs maximaux en dimension 2 (basé sur la méthodologie *fusion avant conquête*), puis on montre comment le paradigme de prétraitement paresseux permet de calculer les points maximaux d'un ensemble de n points $\mathcal{P}$ dans l'espace euclidien $\mathbb{E}^d$. Finalement, nous décrivons un nouvel algorithme optimal simple pour calculer les maxima de $\mathcal{P}$ en dimension 2.

3.2.1 Le cas bidimensionnel

Soit S et R deux ensembles d'élements, on note $S \leq_L R$ ssi $\forall\ s \in S,\ \forall\ r \in R,\ s \leq_L r$. De même, on dira que $x \leq_L M$ ssi $\{x\} \leq_L M$. Les algorithmes décrits ci-après reposent sur le lemme suivant :

Lemme 1 *Soit T un ensemble de points de $\mathbb{R}^2$ union de deux ensembles disjoints: $T = S \uplus R$ avec $S \leq_L R$ alors*

- *$y \in R$ est maximal (dominant) dans T ssi y est maximal dans R*
- *$x \in S$ est maximal dans T ssi x est contenu et maximal dans S/R*

On peut tout d'abord implanter un algorithme basé sur le paradigme *Fusion-Avant-Conquête* comme le montre l'encadré 3.8 et la figure 3.6.

8. Ce problème est très utile dans le calcul de l'affichage de fenêtres sur un écran. On calcule les zones visibles uniquement (qui ne sont pas recouvertes par d'autres portions de fenêtres) puis on dessine uniquement à l'écran les zones obtenues.

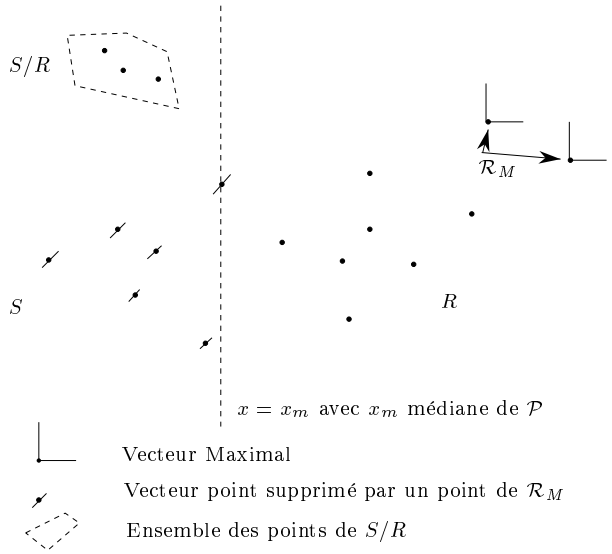


FIG. 3.6 – *Illustration du paradigme fusion avant conquête pour le calcul des vecteurs maximaux.*

Algorithme 3.8: *Le calcul des maxima par fusion avant conquête.*

Fonction Maxima($\mathcal{P} = \{p_1, ..., p_n\}$) : maxima ;
début
si $n = 1$ alors renvoyer p_1;
sinon début
m=*Médiane*($\mathcal{P}$,$\lceil \frac{n}{2} \rceil$);
$\mathcal{S} = \{p \in \mathcal{P} | p_x \leq m\}$
$\mathcal{R} = \{p \in \mathcal{P} | p_x > m\}$
$\mathcal{R}_M$=*Maxima*($\mathcal{R}$);
Calculer $\mathcal{Q} = \mathcal{S}/\mathcal{R}_M$;
(* On filtre les données suivant $\mathcal{R}_M$ *)
renvoyer *Maxima*($\mathcal{Q}$) $\uplus$ $\mathcal{R}_M$;
fin;
fin.

On fusionne (ici, on sélectionne en fait) en temps $O(F_{d-1}(|\mathcal{R}_M|, \frac{n}{2}))$ où $\mathcal{R}_M$ est l'ensemble des vecteurs maximaux de $\mathcal{R}$. On obtient alors l'équation de récurrence :

Lemme 2 *Soit $C_d(n,h)$ la complexité du calcul des h vecteurs maximaux de n points de $\mathbb{E}^d$ alors l'algorithme 3.8 calcule ces vecteurs maximaux en temps*

$$C_d(n,h) \leq C_d(\frac{n}{2}, |\mathcal{R}_M|) + F_{d-1}(\frac{n}{2}, |\mathcal{R}_M|) + C_d(\frac{n}{2}, h - |\mathcal{R}_M|) + O(n)$$

On remarque que la médiane a permis d'assurer un préordre lexicographique suivant la première coordonnée et on peut ainsi diminuer le problème d'une dimension.

Dans le cas bidimensionnel, en utilisant le fait que $F_1(r,s) = \Theta(r+s)$ on obtient directement un algorithme sensible à la sortie:

Corollaire 1 *Le calcul des vecteurs maximaux d'un ensemble de n points du plan peut être obtenu en temps optimal:*

$$c_2(n,h) = \Theta(n \log h).$$

3.2.2 Un algorithme optimal basé sur un prétraitement paresseux pour le calcul de vecteurs maximaux dans $\mathbb{E}^3$ en $\Theta(n \log h)$

Cet algorithme met en œuvre le paradigme de *paresse* ou *prétraitement adapté*. Généralement, on traite les données en les ordonnant puis on les balaye (sweep line) en mettant à jour la solution. Comme on doit assurer que le résultat en cours ne doit pas être détruit par les futures données non encore traitées, on choisit une marche (un sens de parcours) qui vérifie ce principe.

On donne tout d'abord le principe général avec prétraitement complet de l'algorithme (cf. algorithme 3.9) puis on montre les différentes étapes qui ont permis de le rendre sensible à la sortie.

On remarque tout d'abord que le résultat du test $x \leq_M M$ est celui de $x' \leq_M M'$ où $x' = (x_2, ..., x_d)$ et $A' = \{a' | a \in A\}$ puisque les données ont été triées par ordre lexicographique (suivant la première coordonnée puis en cas d'égalité suivant la deuxième, etc.). Analysons la complexité de l'algorithme 3.9:

$$c(n,h) = \underbrace{O(n \log n)}_{\text{tri lexicographique}} + \underbrace{O((n+h)\log h)}_{\text{marche}}$$

Algorithme 3.9: *Le calcul des maxima par une marche sur les données prétraitées.*

Fonction Maxima($\mathcal{P} = \{p_1, ..., p_n\}$) : maxima ;
début
$\mathcal{P}'$=*TriLexicographiqueDécroissant*($\mathcal{P}$);
$M \leftarrow \emptyset$;
Tant que $\mathcal{P}' \neq \emptyset$ faire
$x \leftarrow$ Premier($\mathcal{P}'$);
$\mathcal{P}' \leftarrow \mathcal{P}' \backslash \{x\}$;
si $x \not\leq_M M$ alors $M \leftarrow M \cup \{x\}$;
fin;
fin.

En effet, on peut répondre au test $x' \leq_M M'$ en $O(\log h)$ avec une structure de données adéquate [KLP75] et on insère un vecteur maximal en $O(\log h)$ dans cette structure. Le coût de cet algorithme est $O(n \log n)$. On voit que le tri des données en $O(n \log n)$ a un coût dominant dans la complexité de l'algorithme (pénalise le coût de la marche en $O((n+h)\log h) = O(n \log h)$). De plus, on remarque que les données de $S \backslash (S/R)$ (les vecteurs de S éliminés par les vecteurs maximaux de R) ont été triées inutilement. Pour améliorer cet algorithme et le rendre sensible à la sortie, on prétraite en triant lexicographiquement les données en k paquets ($\mathcal{P}_i,\ 1 \leq i \leq k$) de tailles au plus $\lceil \frac{n}{k} \rceil$ vérifiant :

$$\mathcal{P} = \uplus_{i=1}^{k} \mathcal{P}_i,$$

et

$$\mathcal{P}_i < \mathcal{P}_{i+1},\ \text{pour } 1 \leq i < k.$$

On réalise cette étape par l'algorithme de la médiane en itérant sur des paquets de taille de plus en plus petite en $\Theta(n \log k)$. On retrouve ainsi le tri qui considère n paquets de taille $\lceil \frac{n}{n} \rceil = 1$ en temps $\Theta(n \log n)$. L'algorithme 3.9 se transforme aisément pour tenir compte du partitionnement (cf. algorithme 3.10).

Analysons la complexité de cet algorithme: on trie tout d'abord les données en k paquets de taille au plus $\lceil \frac{n}{k} \rceil$ avec l'algorithme de la médiane

Algorithme 3.10: *Le calcul des maxima par une marche sur les données pré-traitées en k paquets.*

Fonction Maxima($\mathcal{P} = \{p_1, ..., p_n\}$) : maxima ;
début
$\mathcal{P}'$=*PaquetLexicographiqueDécroissant*($\mathcal{P}$,k);
(* on a $\mathcal{P}' = \uplus_{i=1}^{k} \mathcal{P}_i$ *)
$M \leftarrow \emptyset$;
(* on initialise les vecteurs maximaux à $\emptyset$ *)
Pour $i = k$ en descendant jusqu'à 1 faire
    $\mathcal{P}_i \leftarrow \mathcal{P}_i / M$;
    (* on élimine les vecteurs dominés de $\mathcal{P}_i$ *)
    Tant que $\mathcal{P}_i \neq \emptyset$ faire
        $x \leftarrow$ *LexicographiqueMax*($\mathcal{P}_i$);
        $M \leftarrow M \cup \{x\}$;
        $\mathcal{P}_i \leftarrow \mathcal{P}_i / \{x\}$;
        fin;
    fin;
fin.

récursif en $\Theta(n \log k)$ [AHU83]. Désignons par h le nombre de vecteurs maximaux $|M|$. On peut vérifier si un point p de $\mathcal{P}$ est maximal ou non en temps $O(\log h)$. L'épuration des paquets $\mathcal{P}_i/M$ se fait donc en $O(n \log h)$. Chaque point p maximal de $\mathcal{P}$ a été détecté en prenant l'élément lexicographique maximal d'un sous-paquet de taille au plus $\lceil \frac{n}{k} \rceil$. La complexité $c_3(n,h)$ de cet algorithme est :

$$c_3(n,h) = O(n \log k + n \log h + h\frac{n}{k}).$$

Toute la difficulté réside à trouver la bonne valeur de k qui assure une complexité sensible à la sortie. C'est-à-dire que le facteur $n \log k$ ne doit pas masquer le $n \log h$ et le facteur $h\frac{n}{k}$ ne doit pas être supérieur à $n \log h$. Si on choisit $k = h$, on obtient:

$$c_3(n,h) = \Theta(n \log h).$$

Malheureusement, on ne connait pas la valeur de h. Mais au fur et à mesure du déroulement de l'algorithme nous obtenons une approximation de h. L'idée est donc de fixer k au début de l'algorithme à une constante (disons $k_0 = 1$, ainsi $h_0 = 2^{k_0} = 2$ par exemple) puis de rentrer dans la boucle principale jusqu'à $|M| = 1 = 2^{k_0} - 1$ auquel cas nous savons que $h \geq 1$ et nous pouvons autoriser un raffinement du tri en $h_1 = 2h_0$ ($k_1 = k_0 + 1$) paquets de taille $\frac{n}{2^{k_1}}$ puis nous continuons la boucle principale jusqu'à $|M| = 2^{k_1} - 1$, etc. Nous aboutissons à l'algorithme 3.11.

À la sortie de l'algorithme, nous avons $k \leq \lceil \log_2(h+1) \rceil$. L'étape de partionnement des données Q (initialement $Q = \mathcal{P}$) est exécutée au plus $O(\log h)$ fois. Chaque partitionnement prend au plus un temps linéaire $O(n)$. La mise à jour du partitionnement prend donc un temps $O(n \log h)$. Notons t le nombre de fois où l'algorithme a appelé un screening problème, nous avons :

$$C_d(n,h) \leq O(n \log h) + \sum_{i=1}^{t} F_{d-1}(n_i, h),$$

$$\sum_{i=1}^{t} n_i = O(n).$$

Dans le cas tridimensionnel, nous pouvons pousser l'analyse de l'algorithme. En effet, nous avons $F_2(n_i, h) = O(n_i \log h)$. On en déduit ainsi facilement l'optimalité pour $d = 3$.

Corollaire 2 *L'algorithme 3.11 calcule les h vecteurs maximaux d'un ensemble de n vecteurs $\mathcal{P}$ en temps optimal pour $d = 2, 3$ en $\Theta(n \log h)$.*

Algorithme 3.11: *Algorithme adaptatif de calcul des vecteurs maximaux.*

Fonction Maxima($\mathcal{P} = \{p_1, ..., p_n\}$) : maxima ;
début
$M \leftarrow \emptyset$;
$k \leftarrow 1$;
$Q \leftarrow \mathcal{P}$;
(* Q est une queue de priorité contenant les paquets partitionnant $\mathcal{P}$ *)
Tant que $Q \neq \emptyset$ faire
    PaquetCourant$\leftarrow$ *Premier*(Q);
    PaquetCourant$\leftarrow$ PaquetCourant$/M$;
    Tant que PaquetCourant$\neq \emptyset$ faire
        $x \leftarrow$*LexicographiqueMax*(PaquetCourant);
        PaquetCourant$\leftarrow$PaquetCourant$\backslash\{x\}$;
        $M \leftarrow M \cup \{x\}$;
        si $|M| = 2^k - 1$ alors début
            Pour chaque $S \in Q$ avec $|S| \geq \frac{n}{2^k}$ faire
                Partitionner S en S_1, S_2 de taille au plus $\lceil \frac{|S|}{2} \rceil$;
                (* avec $S_1 <_L S_2$ *)
                Remplacer S par S_1 et S_2 dans Q;
                fin;
            $k \leftarrow k + 1$;
            fin;
        fin;
    fin; (* tant que *)
fin.

On voit qu'il faut raffiner l'opération de screening en dimension supérieure afin de donner un borne supérieure pour $\sum_{i=1}^{t} F_{d-1}(n_i, h)$. L'article de Kirkpatrick et Seidel [KS85] poursuit cette analyse en l'appliquant au problème du screening puis utilise cette borne et d'éventuelles conditions (celles données dans la table 3.2) afin de donner des résultats globaux sur le problème du calcul des vecteurs maximaux. Nous conseillons au lecteur intéressé de se rapporter à l'article [KS85] pour la suite de cette étude.

3.2.3 Un nouvel algorithme optimal pour le calcul des maxima dans le plan

Nous montrons comment l'approche mariage avant conquête, en utilisant le calcul du "pont", peut s'appliquer au calcul des maxima d'un ensemble $\mathcal{P}$ de n points du plan Euclidien. Nous adoptons l'algorithme standard de mariage avant conquête décrit dans l'encadré 3.4. Nous avons donc à décrire le calcul du "pont" $[p_i p_j]$ à la droite verticale $x = m$ ainsi que l'opération de filtrage. On définit $p_j = (x_j, y_j)$ comme étant le point de $\mathcal{P}_2$ ayant l'ordonnée maximale, c'est-à-dire $y_j = \max\{y_k | p_k = (x_k, y_k) \in \mathcal{P}_2\}$. $p_i = (x_i, y_i)$ est alors simplement défini comme étant le point de $\mathcal{P}_1$ de plus grande abscisse ayant une ordonnée plus grande que y_j. On élimine alors tous les points autres que p_i et p_j se trouvant dans le quadrant $(-\infty, x_j] \times (-\infty, y_i]$, c'est-à-dire au Sud-Ouest du sommet (x_j, y_j) (la zone $(x_i, x_j) \times (y_j, y_i)$ est vide de points). La figure 3.7 précise les notations.

Théorème 5 *L'approche mariage avant conquête permet de calculer les maxima d'un ensemble de n points de* $\mathbb{E}^2$ *en temps* $\Theta(n \log h)$ *et espace mémoire linéaire, où h est le nombre de maxima.*

La preuve de ce théorème et de la correction totale de l'algorithme se transcrivent directement de celle donnée dans la partie 3.1.12. La table 3.3 donne les résultats de l'implantation de cet algorithme en C++ sur un SUN4 IPX.

3.3 Randomisation et dérandomisation rendent les algorithmes adaptatifs

On illustre ce principe en prenant le cas du calcul de l'enveloppe convexe de points dans l'espace euclidien de dimension 3 ($\mathbb{E}^3$). Le lecteur intéressé pourra se réferer aux articles fondateurs de la randomisation [Mul94, Cla88a, CS89, Cla86, Cla87, GG93] mais aussi aux articles [BDS+90, BDS+92]. La

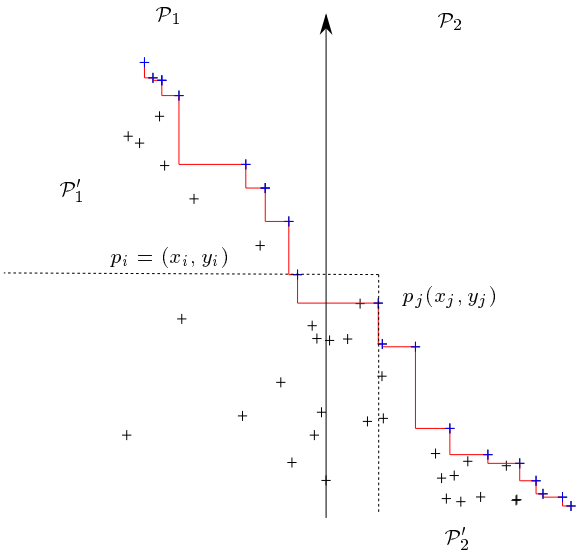


FIG. 3.7 – *Les maxima définissent une structure en escalier* ($|\mathcal{P}| = 100$ *et* $|\mathcal{M}(\mathcal{P})| = 18$ *où* $\mathcal{M}(R)$ *est l'ensemble des maxima de* R*).*

TAB. 3.3 – *Temps d'exécution de l'algorithme sur un* SUN4 IPX.

n	$\lvert\mathcal{M}(\mathcal{P})\rvert$	temps en seconde
128	30	0
256	43	0
512	62	0.016666
1024	78	0.016666
2048	102	0.033332
4096	143	0.08333
16384	322	0.316654
32768	468	0.649974
65536	639	1.33328
131072	909	2.91655
262144	1254	5.84977

dérandomisation, quant à elle, fut étudiée dans [CF88, CF90, CM94]. Les principes fondamentaux de la randomisation sont également détaillés dans le livre de géométrie algorithmique [BY95a].

3.3.1 Principe de la randomisation et de la dérandomisation

Généralement lorsqu'on étudie un problème, on en donne une méthode constructive de résolution dont on évalue le coût dans le pire des cas. L'idée de la randomisation est de donner le temps escompté (moyen) de l'algorithme. On note par $\tilde{O}(c(n))$ la complexité, en moyenne, d'un algorithme randomisé. Le formalisme introduit dans [Cla88a] est basé sur la définition de régions et d'objets, et de conflits entre objets et régions (cf. page 83 pour une illustration de la méthode). La randomisation incrémentale qui consiste à insérer un par un les objets ne fait aucunement intervenir la distribution des données mais l'ordre dans lequel elles sont traitées. On considère les $n!$ permutations des entrées et pour chaque permutation, on détermine le temps de calcul de l'algorithme. On détermine alors l'espérance du temps de calcul de l'algorithme avec n entrées lorsque l'on considère les $n!$ permutations possibles comme équiprobables (algorithme randomisé incrémental). Un algorithme randomisé peut également choisir un échantillon aléatoire R (un sous-ensemble des données initiales de S: $R \subset S$) et l'insére dans la structure de régions construites jusqu'alors en déterminant les nouveaux conflits de type *objet-région*. Nous détaillerons ce paradigme lors de la construction de l'enveloppe convexe d'objets planaires (page 82). Certains algorithmes randomisés tiennent compte de la complexité du résultat dans leur complexité: ils sont alors randomisés et sensibles à la sortie (par exemple, le calcul de la carte des trapèzes d'un ensemble de segments).

La randomisation permet d'atteindre beaucoup de bornes inférieures de complexité avec des algorithmes simples (on peut citer le calcul de l'enveloppe convexe de points mais aussi l'intersection de segments, etc.). Toutefois, la complexité donnée pour ces algorithmes est une espérance de temps de calcul et non pas une borne supérieure. Une fois un algorithme randomisé trouvé, on cherche généralement à le dérandomiser en déterminant correctement les échantillons aléatoires que l'on insère sans que le coût de cette détermination soit prépondérant dans la complexité finale de l'algorithme (pour les algorithmes randomisés suivant le schéma *diviser-pour-régner*). La détermination de ces échantillons de façon déterministe a été étudiée et appliquée dans les articles de Chazelle, Friedman et Matousek [CF88, CF90, CM94].

Nous illustrons les principes de la randomisation et de la dérandomisation qui combinés forment un paradigme de sensibilité à la sortie pour l'étude de

l'enveloppe convexe de points en dimension 3.

3.3.2 Un algorithme randomisé adaptatif pour le calcul de l'enveloppe convexe de points de $\mathbb{E}^3$

On s'intéresse au problème dual : le calcul de l'intersection de n demi-espaces. On considère un ensemble $\mathcal{S} = \{S_1, ..., S_n\}$ de n demi-espaces de l'espace euclidien $\mathbb{E}^3$. On désire calculer l'intersection $\mathcal{P}(\mathcal{S})$ de ces demi-espaces de façon adaptative. On peut déterminer en temps linéaire $\Theta(n)$ si l'intersection $\mathcal{P}(\mathcal{S})$ est vide ou non (et dans ce cas là trouver un point p^* à l'intérieur du polytope intersection) par programmation linéaire (cf. [Meg84, Meg82, Meg83b, Cla88b]). Considérons un point p^* contenu dans l'intersection de ces demi-espaces $\mathcal{S} = \{S_1, ..., S_n\}$ alors la polarité de centre p^* de ces demi-espaces (S_i^* est le pôle de l'hyperplan qui limite le demi-espace S_i) donne un ensemble $\mathcal{S}^* = \{S_1^*, ..., S_n^*\}$ tel que :

$$(\bigcap_{i=1}^{n} S_i)^* = \mathrm{conv}_{i=1}^{n}(S_i^*).$$

L'idée directrice de l'algorithme est de filtrer rapidement les demi-espaces redondants S_i (respectivement les points redondants S_i^*) définis comme ceux vérifiant $\mathcal{P}(\mathcal{S}\backslash\{S_i\}) = \mathcal{P}(\mathcal{S})$ (respectivement $\mathrm{conv}(\mathcal{S}^*\backslash\{S_i^*\}) = \mathrm{conv}(\mathcal{S}^*)$).

L'algorithme est basé sur la relation entre $\mathcal{P}(R)$ et $\mathcal{P}(\mathcal{S})$ où R est un échantillon aléatoire de $\mathcal{S}$. Soit R un échantillon aléatoire de $\mathcal{S}$ ($R \subset \mathcal{S}$). On calcule $\mathcal{P}(R)$ (le polytope intersection de l'échantillon R) puis on décompose ses facettes (2-faces) en trapèzes en faisant passer à chaque sommet de la facette un translaté d'un plan h choisi arbitrairement (cf. figure 3.8). On achève alors la décomposition en la rendant volumique : pour chaque trapèze obtenu précédemment, on prend la fermeture convexe de ce trapèze avec le point intérieur p^* à $\mathcal{P}(\mathcal{S})$. On aboutit ainsi à un partitionnement de $\mathcal{P}(R)$ en zones convexes élémentaires $\Delta(R)$. Pour chaque région $F \in \Delta(R)$, on note $F \wedge \overline{\mathcal{S}}$ l'ensemble des demi-espaces de $\mathcal{S}$ qui ne contiennent pas F entièrement dans leur demi-espace (on désigne par $|F|$ le nombre de demi-espaces ainsi définis).

Lemme 3 *La valeur moyenne de* $\sum_{F \in \Delta(R)} |F|$ *est* $O(n)$.

La preuve de ce lemme se trouve dans [CS89], pp. 411. On remarque que le polytope intersection $\mathcal{P}(\mathcal{S})$ est l'union des régions $F \cap \mathcal{P}(\mathcal{S})$ sur toutes les régions $F \in \Delta(R)$:

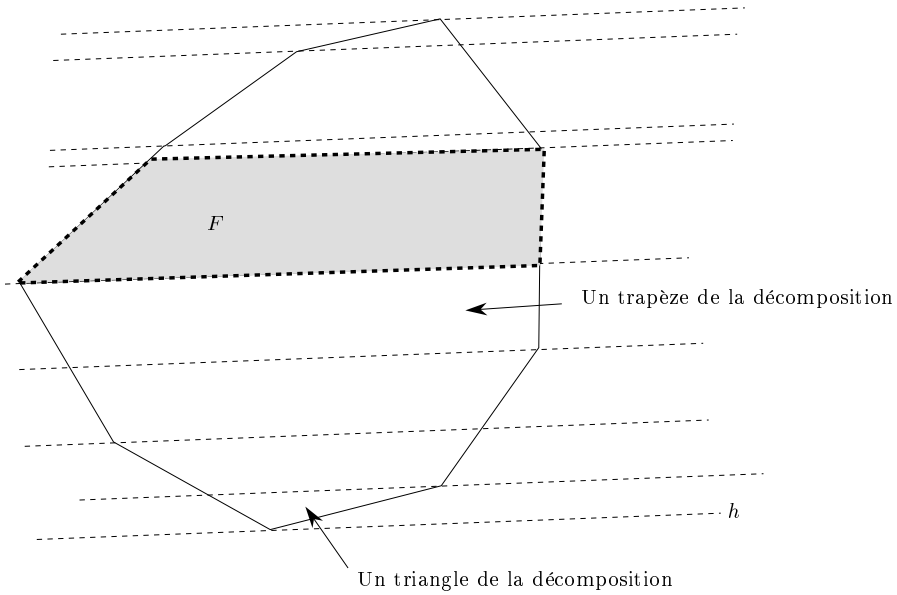


FIG. 3.8 – *Décomposition en trapèzes par un plan h d'une facette du polytope intersection $\mathcal{P}(R)$.*

$$\mathcal{P}(\mathcal{S}) = \bigcup_{F \in \Delta(R)} F \cap \mathcal{P}(\mathcal{S}).$$

Les demi-espaces qui contribuent aux régions non-triviales de $F \cap \mathcal{P}(\mathcal{S})$ (c'est-à-dire $F \cap \mathcal{P}(\mathcal{S}) \neq \emptyset,\ F \cap \mathcal{P}(\mathcal{S}) \neq \mathcal{P}(\mathcal{S})$) sont les complémentaires par rapport à $\mathcal{S}$ des demi-espaces de $F \wedge \overline{\mathcal{S}}$. Puisque chaque demi-espace non redondant détermine un sommet de $\mathcal{P}(\mathcal{S})$, il est inutile de considérer toutes les régions de $F \in \Delta(R)$ mais plutôt les régions F de $\Delta(R)$ qui contiennent au moins un sommet de $\mathcal{P}(\mathcal{S})$. On note $\Delta^*(R)$ l'ensemble de ces régions ($\Delta^*(R) \subset \Delta(R)$). Si le polytope intersection contient h sommets, on a le lemme suivant:

Lemme 4 (Clarkson et Shor [CS89]) *La valeur moyenne de $\sum_{F \in \Delta^*(R)} |F|$ est $O(\frac{n}{r}) \times h$ pour un échantillon aléatoire de taille r.*

Il suffit maintenant de déterminer la taille des échantillons aléatoires au cours de l'algorithme. On ne connait pas la valeur de h au départ. Supposons que $r = h^2$, alors la valeur moyenne de $\sum_{F \in \Delta^*(R)} |F|$ (partie du polytope intersection $\mathcal{P}(\mathcal{S})$ déjà filtré) sera $O(\frac{n}{h})$. Cette constatation fournit un procédé rapide d'élimination des demi-espaces redondants à condition de connaître:

1. Une estimation de h

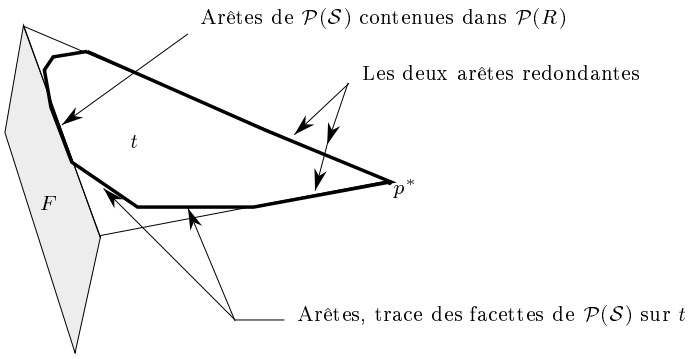


FIG. 3.9 – *Élimination des demi-espaces redondants grâce au calcul de* $t \cap \mathcal{P}(\mathcal{S})$.

2. Un algorithme de calcul rapide de $\Delta^*(R)$, et des ensembles $F \wedge \overline{\mathcal{S}}$, $\forall\, F \in \Delta^*(R)$.

Les régions $F \in \Delta(R)$ et ses ensembles caractéristiques correspondants $F \wedge \overline{\mathcal{S}}$ peuvent être obtenus grâce à un algorithme randomisé de calcul du polytope intersection en temps $\tilde{\mathrm{O}}(n \log r)$ (cf. [CS89]). Il faut maintenant filtrer les régions qui ne participent pas au polytope intersection final, c'est à dire les régions $F \in \Delta^*(R)$. Pour cela, considérons une région F de $\Delta(R)$ et sélectionnons un triangle t défini par p^* et deux sommets adjacents de F, alors le polygone $P_t = t \cap \mathcal{P}(\mathcal{S})$ peut être déterminé par l'algorithme de Kirkpatrick et Seidel [KS82, KS86] en temps déterministe optimal $\Theta(|F| \log h_t)$ où h_t est le nombre de faces de P_t. Toutes les faces de P_t (les arêtes du polygone convexe) sauf deux (celles qui rejoint le point p^*) correspondent à la trace de 2-faces de $\mathcal{P}(\mathcal{S})$ (et aux arêtes lorsqu'elles proviennent de F), si bien que le résultat de cet algorithme appliqué aux régions F de $\Delta(R)$ peut être calculé en temps $O(n \log h')$ où h' est le nombre total de faces de $\mathcal{P}(\mathcal{S})$ identifiées. Si une région $F \in \Delta(R)$ ne contient aucun sommet de $\mathcal{P}(\mathcal{S})$ alors les polygones correspondant aux faces de F (décomposée en triangles) déterminent complètement la structure de $F \cap \mathcal{P}(\mathcal{S})$ (soit cette intersection est F, soit $\emptyset$, ou alors elle permet d'identifier les arêtes de F qui interviennent dans $\mathcal{P}$) et on peut la déterminer en temps $O(|F|)$ (cf. figure 3.9). Ainsi les régions $F \in \Delta^*(R)$ peuvent être calculées en temps $O\left(n(\log r + \log h')\right)$. Il faut maintenant résoudre le problème de l'estimation de h ou trouver des bornes inférieures sur h (afin que le travail fourni sur les échantillons aléatoires R lors du calcul de $\mathcal{P}(R)$ par un algorithme randomisé non adaptatif ne dépassent pas celui du calcul de l'intersection par un algorithme sensible à la sortie). À chaque étape de l'algorithme, on utilise une estimation h^* de

h. Initialement, on fixe h^* a une constante. Par exemple $h^* = 10$. Puis à chaque étape courante on ajuste l'estimation courante de h^* comme suit: si $h^* > |\mathcal{S}|$ alors calculer $\mathcal{P}(\mathcal{S})$ en utilisant un algorithme randomisé en temps $\tilde{O}(n \log n)$, sinon, on calcule $\Delta^*(R)$ et $F \wedge \overline{\mathcal{S}}$ avec un échantillon aléatoire de taille $r = |R| = h^*$. Si la moitié des demi-espaces a été éliminée alors on ne change pas h^* sinon on fixe h^* à $h^* \leftarrow (h^* \times h')^2$.

Théorème 6 *Étant donné un ensemble $\mathcal{S}$ de n demi-espaces, on peut calculer le polytope intersection de ces demi-espaces par un algorithme randomisé en temps $\tilde{O}(n \log h)$ où h désigne la complexité du résultat.*

L'étude de la complexité est décrite dans [CS89]. L'algorithme général est récapitulé dans l'encadré 3.12.

3.3.3 Dérandomisation de l'algorithme

La dérandomisation permet de déterminer les échantillons "aléatoires" des algorithmes randomisés ayant les caractéristiques voulues. En particulier, elle permet de s'affranchir des *mauvais cas* (qui sont rares au sens de la fréquence). Chazelle et Matoušek ont dérandomisé l'algorithme de Clarkson et Shor dans [CM94]. Ils aboutissent ainsi à un algorithme déterministe optimal sensible à la sortie pour le calcul de l'enveloppe convexe de points:

Théorème 7 *Étant donné un ensemble $\mathcal{P}$ de n points de $\mathbb{E}^3$, on peut calculer son enveloppe convexe de façon déterministe optimale en temps $\Theta(n \log h)$ et en espace mémoire $O(n)$.*

On s'intéresse plus particulièrement à déterminer les sommets de l'enveloppe convexe. En effet, si on connaît les h sommets de l'enveloppe convexe, alors on peut construire le graphe d'incidence par un algorithme non adaptatif en temps $O(h \log h)$ comme par exemple le *diviser-pour-régner* de Preparata et Hong [PH77]. On peut définir une sortie dite *faible* comme étant les points de $\mathcal{P}$ qui se trouvent sur le bord de l'enveloppe convexe et une sortie qualifiée de *forte* comme étant les sommets de l'enveloppe convexe. On dira qu'un algorithme est *réellement sensible à la sortie* si son temps de calcul dépend de la taille de la sortie forte. Dans le cas d'ensembles de points hautement dégénérés, les deux sorties peuvent varier considérablement. On considére dans la suite la sensibilité faible à la sortie afin de faciliter la compréhension. Le même résultat peut être obtenu pour la sensibilité forte à la sortie en considérant des détails plus techniques. On invite le lecteur à consulter les articles [Cla87, Mat91a, Mat91b] pour de plus amples détails sur les techniques de dérandomisation. On considère le problème dual qui est de trouver

Algorithme 3.12: *Calcul du polytope intersection $\mathcal{P}(\mathcal{S})$ par élimination des demi-espaces redondants..*

Fonction EliminationDemiEspaces($\mathcal{S} = \{S_1, ..., S_n\}$) : ens-demi-espaces ;
début
$h^* \leftarrow 10$;
Tant que $\mathcal{S} \neq \emptyset$ faire
$n \leftarrow |\mathcal{S}|$;
R =*EchantillonAléatoire*($\mathcal{S}$,h^*);
$\mathcal{S} \leftarrow \mathcal{S} \backslash R$;
$\mathcal{P}(R)$=*CalculerPolytopeIntersection*(R);
$\Delta(R)$ =*Découper*($\mathcal{P}(R)$,h);
$h' \leftarrow 0$;
Pour chaque région $F \in \Delta(R)$ faire
Pour chaque triangle t de F faire
$P_t = t \cap \mathcal{P}(\mathcal{S})$;
$h' \leftarrow h' + n_f$;
(* n_f est le nombre de faces de $\mathcal{P}$ identifiées par $\Delta(R)$ *)
(* On utilise l'algorithme de Kirkpatrick et Seidel *)
$\mathcal{S}$=*Épurer*($\mathcal{S}, P_t$);
(* on garde les demi-espaces de $\mathcal{S}$ qui coupent
les régions $F \in \Delta^*(R)$ *)
fin;
fin;
si $|S| > \lceil \frac{n}{2} \rceil$ alors $h^* \leftarrow (h^* h')^2$;
fin; (* tant que *)
fin.

les plans non-redondants de $\mathcal{P}^* = \mathcal{H}$. On peut déterminer en temps optimal linéaire $\Theta(n)$ si l'intersection des demi-espaces de $\mathcal{P}^*$ est vide ou non et dans ce cas déterminer un point p à l'intérieur (cf. [Dye84, Meg84]).

3.3.4 Construction d'une triangulation $\Delta(\mathcal{H})$ du polytope intersection $\cap\mathcal{H}$

Chaque facette (2-face) de $\mathcal{H}$ est triangulée à partir du sommet le plus petit dans l'ordre lexicographique (cette triangulation est dénommée en anglais bottom-vertex triangulation). Puis, chaque triangle ainsi défini est fermé par convexité avec le point intérieur p. La décomposition de $\cap\mathcal{H}$ est ainsi un complexe simplicial (pour plus d'informations sur les complexes simpliciaux et leurs nombreuses propriétés, cf. [BY95a]) formé de $O(|\mathcal{H}|)$ simplexes, noté $\Delta(\cap\mathcal{H})$.

3.3.5 Conflits entre simplexes et polytope intersection

Pour un simplexe s de la décomposition $\Delta(\mathcal{H})$, on note $\mathcal{H}_s$ l'ensemble des plans coupant s. On utilise les deux résultats suivants :

Lemme 5 *(i) Soit R un échantillon aléatoire de $r = |R|$ plans de $\mathcal{H}$, alors il existe deux constantes C et C' telles que les deux conditions ci-dessous soit vérifiées avec une probabilité non nulle:*

(a) Pour chaque simplexe s de $\Delta(R)$, on a $|\mathcal{H}_s| \leq C \times \frac{|\mathcal{H}|}{r}$.

(b) $\displaystyle\sum_{s \in \Delta(R)} |\mathcal{H}_s| \leq C' \times |\mathcal{H}|$

(ii) Étant donné $\mathcal{H}$ et r, on peut calculer de façon déterministe un échantillon R vérifiant (a) et (b), de taille $|R| = r$ en temps polynomial en $|\mathcal{H}|$.

On obtient la deuxième partie de ce lemme grâce aux probabilités conditionnelles de Raghavan et Spencer (cf. [Mat91a]). Il faut noter que la complexité de (ii) est un polynôme en $|\mathcal{H}|$ de grand degré. Cette étape est alors la plus coûteuse des algorithmes dérandomisés. Pour pallier ce sur-coût, on fait appel au concept d'ϵ-approximation qui permet de choisir un échantillon représentatif R' de l'échantillon aléatoire R et qui garantit les propriétés (a) et (b) pour R à partir de R'.

3.3.6 Concept d'ϵ-approximation

On dit que $A \subset \mathcal{H}$ est une ϵ-approximation de $\mathcal{H}$ (pour les simplexes, avec $0 < \epsilon < 1$) si pour chaque simplexe s de $\Delta(\mathcal{H})$ on a:

$$\left| \frac{|\mathcal{H}_s|}{|\mathcal{H}|} - \frac{|A_s|}{|A|} \right| \leq \epsilon.$$

On utilise le lemme suivant (plus d'explications sont fournies dans l'article [Mat91b]):

Lemme 6 *Il existe une constante $\alpha > 0$ telle qu'étant donné $\mathcal{H}$ et r, $r < n^\alpha$, une $(\frac{1}{r})$-approximation de $\mathcal{H}$ de cardinal $O(r^d)$ peut être calculée de façon déterministe en temps $O(n \log r)$.*

On en déduit alors le corollaire suivant:

Corollaire 3 *Il existe une constante $\alpha > 0$, telle qu'étant donné $\mathcal{H}$ et r, $r < n^\alpha$, on peut calculer de façon déterministe en temps $O(n \log r)$ un échantillon R satisfiant les propriétés (a) et (b).*

Preuve (idée): On calcule l'échantillon R en deux étapes. Tout d'abord, on trouve une $(\frac{1}{r})$-approximation A de $\mathcal{H}$ en temps $O(n \log r)$, de cardinal $O(r^3)$. Puis on applique (ii) pour calculer l'échantillon R qui a les propriétés relatives à A. Si r est assez petit alors le temps de cette seconde étape sera dominé par $O(n \log r)$. On utilise alors l'ϵ-approximation pour montrer par transitivité que l'échantillon R qui a les propriété relatives à A respecte également ces propriétés pour $\mathcal{H}$ avec des constantes plus importantes. □

Avant de décrire l'algorithme final, nous introduisons les deux derniers lemmes. Le lemme suivant permet de remplacer la construction incrémentale randomisée par son équivalent déterministe:

Lemme 7 *Étant donné $\mathcal{H}$ et $\Delta(R)$ définis par le lemme 5, on peut calculer l'ensemble $\mathcal{H}_s$ pour chaque simplexe $s \in \Delta(R)$ en temps $O(n \log |R|)$.*

Nous esquissons la preuve de manière à rendre plus claire l'implantation de l'algorithme:

Preuve: Pour chaque plan H de $\mathcal{H}$, nous tracons les interactions (conflits) de H sur la décomposition en simplexes de $\Delta(R)$. Une fois que nous connaissons un simplexe s qui est coupé par H, nous pouvons tracer toutes les incidences restantes de H en temps proportionnel à leur nombre par un algorithme de recherche sur un graphe (on utilise le graphe de la triangulation). Il reste à décrire l'algorithme qui permet de trouver une interaction (conflit).

Il suffit de trouver un sommet de $\cap R$ séparé de p par H (p est le point à l'intérieur de $\cap\mathcal{H}$). Le problème dual consiste à trouver une facette du polyèdre séparant un point-requête $p^* = p$ (p centre de polarité) de l'infini. Il s'agit donc là d'un problème de localisation de points dans l'espace euclidien bidimensionnel $\mathbb{E}^2$. Ce problème a une solution optimale décrite dans [Kir83]. On peut ainsi trouver un simplexe laissant une trace sur H (ou alors affirmer qu'il n'existe pas de simplexe laissant une trace sur $\mathcal{H}$) en temps $O(\log r)$ avec un prétraitement en temps $O(r)$. Le temps résultant pour le calcul de $\mathcal{H}_s$ est au plus $O(n \log r) + \sum_{s \in \Delta(R)} |\mathcal{H}_s| = O(n \log r)$. □

Lemme 8 *Soit s un simplexe et $\mathcal{H}_s$ l'ensemble des plans laissant une trace sur s, alors on peut décider si s contient un sommet de $\cap\mathcal{H}$ par un algorithme déterministe en temps $O(|\mathcal{H}_s| \log h)$, où h désigne le nombre de sommets de $\cap\mathcal{H}$.*

On peut maintenant décrire l'algorithme optimal de calcul de l'enveloppe convexe sensible à la sortie. On se base sur le lemme suivant:

Lemme 9 *Soit $\mathcal{H}$ un ensemble de n plans de $\mathbb{E}^3$, et $b \leq n$ un paramètre. Il existe une constante B et un algorithme déterministe, tel que si $\cap\mathcal{H}$ a au plus b sommets, alors l'algorithme calcule $\cap\mathcal{H}$ en temps au plus $Bn \log b$. L'algorithme termine en temps au plus $Bn \log b$ si le polytope intersection $\cap\mathcal{H}$ contient plus de b sommets. Dans ce cas, il signale que $h > b$.*

On considére alors le lemme 9 afin de construire une suite de valeurs $(b_i)_{i \in \mathbb{N}^*}$ en fixant b_1 à une constante assez grande et $b_{i+1} = b_i^2$. On lance alors l'algorithme avec $b = b_1$; s'il trouve $b > b_1$ alors on le relance avec b_2, etc. Nous obtenons la solution pour $b_i \geq h > b_{i-1}$. Ainsi le temps total d'exécution est:

$$\sum_{i \in \mathbb{N}^* | b_i < h} Bn \log b_i \leq Bn \sum_{i=1}^{\log \log h} 2^i = O(n \log h).$$

On laisse volontairement la preuve du lemme 9 (il suffit d'assembler les différents résultats avec $r = b$) dans le papier [CM94] afin de ne pas alourdir l'étude des algorithmes adaptatifs.

3.4 La méthodologie *Papier Cadeau*

On décrit dans cette partie une méthodologie fréquemment utilisée pour le calcul d'enveloppe convexe d'objets de l'espace euclidien. Cette méthode

fut originellement proposée par Chand et Kapur [CK70] dans le cas des points. L'analyse de la technique ne fut faite qu'en 1982 !par Bhattacharya *et al.* [ATB82]. L'idée est de calculer les $(d-1)$-faces par adjacence en parcourant les $(d-2)$-faces de la $(d-1)$-face considérée et en cherchant la nouvelle facette contenant la $(d-2)$-face courante. On part d'une facette f initiale (trouvée par programmation linéaire par exemple) puis on enrobe l'enveloppe convexe à partir de cette facette. Le lecteur trouvera cette méthode dans [PS85]. On aborde ci-dessous le cas de l'enveloppe convexe d'objets dans l'espace euclidien $\mathbb{E}^2$ puis on illustre le fonctionnement général de la méthode pour le cas d'un ensemble de points de $\mathbb{E}^d$. Bien qu'extrêmement simple, cette méthodologie couplée avec le paradigme *groupement par paquets* (décrit dans la deuxième partie) permet d'obtenir les meilleurs algorithmes de calcul d'enveloppe convexe de points connus à ce jour.

3.4.1 Le cas bidimensionnel : la marche de Jarvis [Jar77, Jar73]

On considère un ensemble $\mathcal{P}$ de n objets du plan $\mathbb{E}^2$ dont on veut calculer l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$. Pour cela, on enrobe l'enveloppe convexe de ces objets par un espace affine de dimension 1, c'est-à-dire une droite. On part d'un objet $\mathcal{O}$ ($\mathcal{O} \in \mathcal{P}$) qui se trouve sur le bord de l'enveloppe convexe (trivialement, on peut prendre l'objet O qui contient sur son bord ∂O le point extrémal droit p : $p \in \mathcal{O}$) puis on cherche le prochain objet $\mathcal{O}'$ en pivotant la droite verticale autour de p jusqu'à rencontrer un objet de $\mathcal{P}$. On trace alors le segment rejoignant ces objets et support de $\mathcal{P}$. On réitère ce procédé à partir de $\mathcal{O}'$ jusqu'à ce qu'on retombe sur p. Comme nos objets ne sont généralement pas ponctuels, on fait coulisser le point p sur $\mathcal{O}$ dans le sens de la marche afin que la droite l reste tangente à $\mathcal{O}$ (cf. figure 3.11). Si on veut assurer la continuité de la marche, on assure que l'on tourne toujours dans le même sens. Dans le cas d'enveloppe convexe d'objets non convexes, cette continuité ne peut être assurée (cf. figure 3.10) (on remédie à ce problème en calculant d'abord l'enveloppe convexe de chaque objet).

Prenons le cas de l'enveloppe convexe de n points du plan $\mathcal{P} = \{p_1, ..., p_n\}$ (cf. figure 3.12). On choisit d'abord le point ayant l'abscisse maximale. Soit p_1 ce point (quitte à renommer les points), alors on regarde les angles α_i que font les droites $(p_1 p_i)$ avec la droite verticale et on choisit le point p_j qui réalise $\alpha_j = \min_{i=1..n} \alpha_i$. On résume dans l'algorithme 3.13 les différentes étapes et le mécanisme d"enrobage.

La complexité de cet algorithme est trivialement le nombre de fois que l'on rentre dans la boucle `Repeter` fois le temps d'exécution d'une boucle

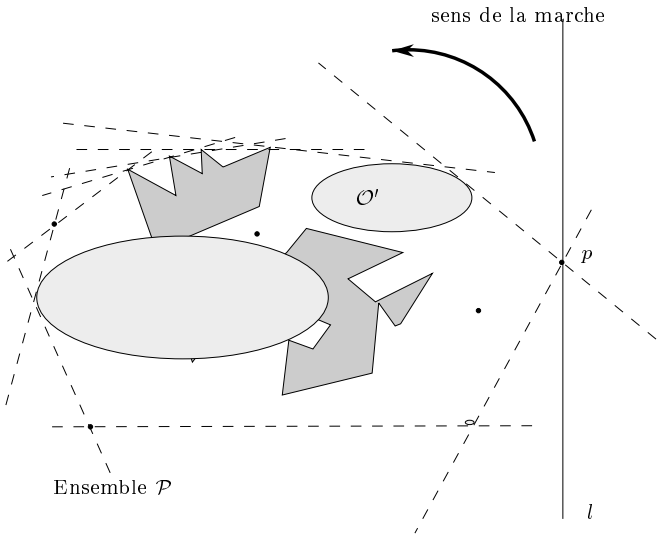


FIG. 3.10 – *La marche de Jarvis dans le cas général d'enveloppe convexe d'objets du plan.*

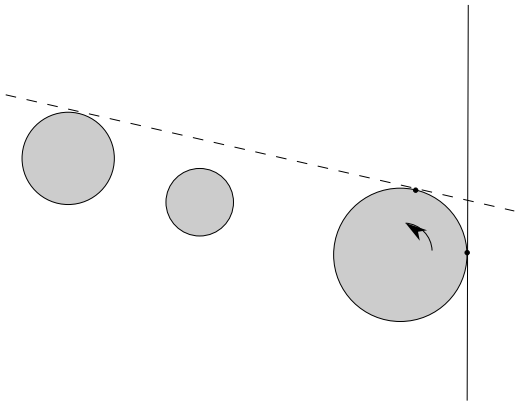


FIG. 3.11 – *Coulissement du point p le long de $\partial\mathcal{O}$.*

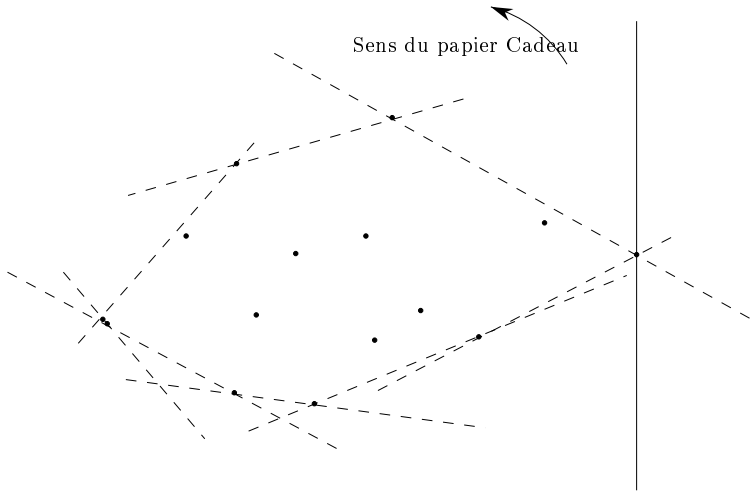


FIG. 3.12 – *Marche de Jarvis sur un ensemble de points du plan.*

Algorithme 3.13: *L'algorithme de la marche de Jarvis pour un ensemble de points.*

Procedure *Jarvis*($\mathcal{P} = \{p_1, ..., p_n\}$);
début
p=*MaximumAbscisse*($\mathcal{P}$);
$p_c = p$
α_c=*MinAngle*($\mathcal{P}$,p);
(* angle pris par rapport à la droite verticale *)
Répeter
    np =*PointMinAngle*($\mathcal{P}$,p_c);
    α_{np}=*MinAngle*($\mathcal{P}$,p_c);
    (* np est le nouveau sommet de l'enveloppe convexe *)
    $p_c = np$;
    $\alpha_c = \alpha_{np}$
Jusqu'à $np = p$;
fin

`Repeter`. Si l'enveloppe convexe de $\mathcal{P}$ a h arêtes (et donc également h sommets) alors on exécute h fois la boucle. À chaque exécution de la boucle, on cherche un minimum sur un ensemble de n angles. L'algorithme a donc une complexité en $O(n \times h)$. On peut corroborer ce résultat avec des distributions de points particulières et comparer avec le résultat de l'algorithme de type *diviser-pour-régner* de Preparata et Hong [PH77]. Le lecteur intéressé par le calcul de l'espérance de h suivant des modéles de répartition des points dans l'espace pourra se référer aux études [AW91, Dev91, Dwy89, Efr65, Ray70] et à la table 2.2 page 8. La technique utilisée par Jarvis est générale puisqu'il s'agit d'envelopper l'objet que l'on calcule (en l'occurence ici, l'enveloppe convexe).

La méthodologie utilisée précédemment est celle du *papier cadeau* (gift wrapping). Ici on enrobe par rapport aux arêtes, on le qualifie encore en anglais d'edge-based gift wrapping.

3.4.2 La méthode *papier cadeau* en dimension supérieure

On désire calculer l'enveloppe convexe de n points en position générale dans l'espace euclidien $\mathbb{E}^d$. Le d-polytope $\mathcal{CH}(\mathcal{P})$ obtenu est simplicial, c'est-à-dire que chaque facette ($(d-1)$-face) est un $(d-1)$-simplexe déterminé par exactement d sommets (points de $\mathcal{P}$). Cette hypothèse n'est pas restrictive puisque l'on peut simuler la *simplicité* (cf. [EM88, EM90]). Toutefois, la sensibilité à la sortie devient faible et non forte (on peut y remédier en décrivant les différents cas qui peuvent se produire). Nous renvoyons le lecteur au début de ce chapitre pour les détails concernant cette discussion. L'algorithme est basé sur le théorème d'adjacence des polytopes simpliciaux.

Théorème 8 *Dans un polytope simplicial, chaque sous-facette ($(d-2)$-face) est partagée par exactement deux facettes et deux facettes F_1 et F_2 sont adjacentes par e (partage une même sous-facette e) si et seulement si e est déterminée par un sous-ensemble de $(d-1)$ sommets communs à F_1 et F_2.*

L'idée générale est alors de construire l'enveloppe convexe à partir d'une facette en sélectionnant les sous-facettes (peaks) de la facette courante et en construisant par le théorème d'adjacence les nouvelles facettes. On doit néanmoins vérifier que la facette regardée n'a pas déjà été construite !

3.4.3 Construction

Soit $\mathcal{P} = \{p_1, ..., p_n\}$ l'ensemble des points de $\mathbb{E}^d$ dont nous voulons calculer l'enveloppe convexe et soit F une facette connue (que l'on nomme *initiale*)

de $\mathcal{CH}(\mathcal{P})$. Le mécanisme d'enrobage est le suivant : prendre une sous-facette ($(d-2)$-face) e de F et déterminer la facette F' adjacente à F par e. F' est une facette de l'enveloppe convexe dont le simplexe ne diffère que d'un sommet par rapport à F. Il faut déterminer le point p' de $\mathcal{P}\backslash\text{Sommet}(F)$ tel que tous les autres points se trouvent d'un même côté de l'hyperplan aff($\underbrace{e \cup \{p'\}}_{F'}$) défini par l'espace affine engendré par la sous-facette e et le point p'.

Nous cherchons ainsi le point p' de $\mathcal{P}\backslash\text{Sommet}(F)$ tel que aff($F' : e \cup \{p'\}$) forme "le plus petit angle" possible (suivant un sens que nous détaillons dans la suite) avec aff(F). L'algorithme est résumé dans l'encadré 3.14.

Algorithme 3.14: *Algorithme Papier Cadeau pour le calcul d'enveloppe convexe de points de* $\mathbb{E}^d$.

Procédure *PapierCadeau*($\mathcal{P} = \{p_1, ..., p_n\}$);
début
$\mathcal{Q} \leftarrow \emptyset$;
(* Ensemble des facettes calculées *)
$\mathcal{F} \leftarrow \emptyset$;
(* Ensemble des sous-facettes des facettes déjà trouvées *)
$F \leftarrow$*FaceInitiale*($\mathcal{P}$);
$\mathcal{Q} \leftarrow F$;
Tant que $\mathcal{Q} \neq \emptyset$ faire
    $F \leftarrow$*Premier*($\mathcal{Q}$);
    (* on prend une première facette de $\mathcal{Q}$ pas encore traitée *)
    $T \leftarrow$*Sous-Facettes*(F);
    Pour chaque $e \in T \cap \mathcal{F}$ faire
        $F' =$*Facette*(e,$\mathcal{P}\backslash$Sommet(F));
        $\mathcal{F} \leftarrow (\mathcal{F} \oplus F')$
        (* l'opérateur $\oplus$ est celui de la différence symétrique *)
        $\mathcal{Q} \leftarrow F'$
        fin; (* fin pour *)
    Afficher(F);
    fin; (* fin tant que *)
fin

3.4.4 Sélectionner le bon point

On considère l'ensemble des hyperplans passant par la sous-facette e et un point p de $\mathcal{P}\backslash\text{Sommet}(F)$ et nous cherchons parmi ces hyperplans celui qui réalise l'angle minimum avec l'hyperplan support à F. On choisit l'angle maximal (la convexité de l'enveloppe assure $\alpha < \pi$). On définit ainsi une nouvelle facette F'. La comparaison des angles (test d'orientation) peut être réalisée via les cotangentes aux hyperplans définis précédemment. En effet, considérons le vecteur $\vec{\mathbf{n}}$ unitaire normal à F et $\vec{\mathbf{a}}$ un vecteur unitaire normal à e (suivant une 1-face p_1p_2) et à $\vec{\mathbf{n}}$. Notons $\mathbf{v_k}$ le vecteur $\overrightarrow{p_2p_k}$ alors la cotangente de l'angle entre l'hyperplan support à F et l'hyperplan support à e et passant par p_k est exprimée par:

$$\rho_k = -\frac{\mathbf{v_k} \cdot a}{\mathbf{v_k} \cdot n}.$$

et nous choisissons le point p_j satisfiant :

$$\rho_j = \max_{k=1..n} \rho_k. \tag{3.5}$$

On remarque que dans le cas simplicial, la dernière équation a une unique solution. La figure 3.13 montre l'espace affine (d-hyperplan) aff(F) et l'ensemble des hyperplans support à e et à un point p de $\mathcal{P}\backslash\text{Sommet}(F)$.

3.4.5 Les derniers détails

- On trouve une face initiale F, par exemple par programmation linéaire en temps linéaire $O(n)$.

- On génére les $(d-2)$-faces d'une facette F en considérant les $(d-2)$-simplexes générés par les $(d-1)$ sommets de la facette originelle F.

- Pour déterminer si une sous-facette e est candidate pour trouver de nouvelles facettes, il faut déterminer si elle ne se trouve que dans une seule facette de $\mathcal{Q}$. On utilise la structure de données $\mathcal{F}$ afin de déterminer si elle se trouve dedans ou non (cette notion correspond à celle d'horizon définie par Seidel). On choisit pour $\mathcal{F}$ un arbre de recherche équilibré sur les $(d-2)$-faces (chaque $(d-2)$-face pointe sur deux $(d-1)$-faces (facettes)).

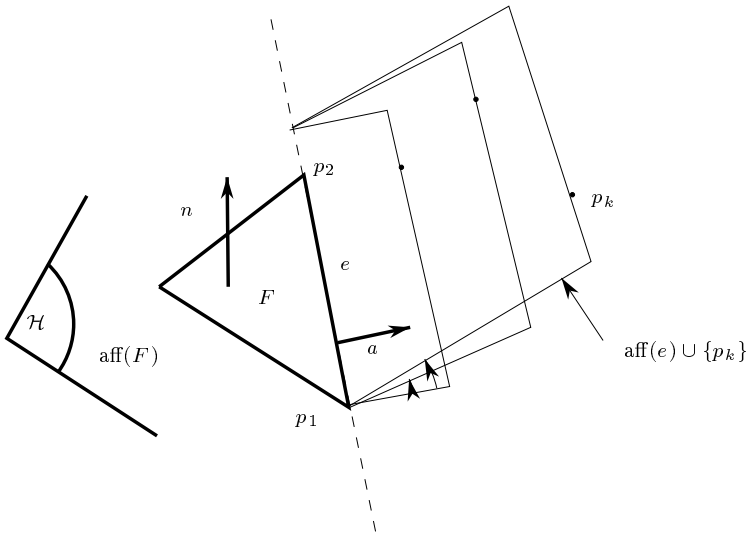


FIG. 3.13 – *Construction d'une facette à partir d'un peak d'une facette adjacente.*

3.4.6 Rassemblons le tout

On sélectionne une nouvelle facette par application du test (3.5) en $O(n)$ et on met à jours les sous-facettes en $O(d\phi_{d-1} \log \phi_{d-2}) = O(\phi_{d-2} \log \phi_{d-2})$ où ϕ_i est le nombre de i-faces de l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$. Ainsi, l'algorithme a une complexité en $O(n \times \phi_{d-1} + \phi_{d-2} \log \phi_{d-2})$. On trouvera plus d'informations dans le livre de Preparata et Shamos [PS85] et dans celui de Boissonnat et Yvinec [BY95a].

3.5 Calcul de l'enveloppe convexe de points en dimension supérieure par *effeuillage* [Sei86a]

On présente dans cette partie la technique utilisée par Seidel pour la construction du polytope représentant l'enveloppe convexe de points en dimension $d \geq 4$ (dans l'espace $\mathbb{E}^d$). Le problème est différent de ceux en dimension 2 et 3 puisque le nombre de 0-faces (ϕ_0), c'est-à-dire le nombre de sommets de ce polytope, n'est plus proportionnel au nombre total de faces du polytope ($\phi = \sum_{i=0,\ldots,d-1} \phi_i$). L'algorithme consiste au départ à sélectionner les points de $\mathcal{P}$ qui sont sur le bord de l'enveloppe convexe $\partial\mathcal{CH}(\mathcal{P})$ puis de

construire les $(d-1)$-faces en respectant un ordre dit d'effeuillage (shelling order). On distingue deux problèmes de construction:

- La construction du graphe d'adjacence du polytope enveloppe convexe qui construit les $(d-1)$-faces (facet enumeration) et leurs relations d'adjacence.

- La construction du graphe d'incidence complet du polytope enveloppe convexe qui construit les i-faces, $0 \leq i \leq d-1$ (facial lattice) et leurs relations d'incidence.

On suppose encore une fois, sans perte de généralité, que les points de $\mathcal{P}$ sont en position générale (sinon on simule la simplicité [EM88, EM90]). La structure du graphe d'incidence d'un polytope peut être trouvée dans [BY95a, PS85, Ede88, Ede87]. On présente tout d'abord des propriétés sur les relations d'adjacence d'un polytope puis on décrit la notion d'*effeuillage* avant de donner l'algorithme final. Nous décrivons en annexe (page 207) les propriétés mathématiques du graphe d'adjacence.

3.5.1 Effeuillage (shelling) d'un polytope

Un effeuillage d'un d-polytope $\mathcal{P}$ est une liste ordonnée de ses facettes $F_1, ..., F_n$ $(n = \phi_{d-1}(\mathcal{P}))$ telle que pour tout $i,\ 1 \leq i \leq n$:

$$F_i \cap (\cup_{j<i} F_j)\,,$$

soit l'union des k premières $(d-2)$-faces $(k \leq \phi_{d-2}(F_i))$ de F_i pour un effeuillage de F_i donné. L'effeuillage correspond à donner un ordre sur les faces du polytope afin de controller non seulement l'adjacence entre faces mais aussi l'ordre dans lequel elles apparaîtront.

Lemme 10

$$\forall\ i,\ 1 \leq i \leq n,\ \cup_{j<i} F_j \textit{ est une } (d-1)\textit{-boule topologique}$$

On invite le lecteur à consulter l'article de référence [Sei86a] pour plus de détails sur l'effeuillage et la preuve de l'existence d'un effeuillage pour tout polytope (cf. [BM71]).

3.5.2 Effeuillage suivant le long d'une droite orientée

On considère un effeuillage particulier: celui réalisé par l'ordre induit des facettes avec une droite orientée (straight line shelling) coupant $\mathcal{P}$, le polytope enveloppe convexe. La construction peut se comprendre aisément grâce à la métaphore d'un voyageur qui commence sa route sur la droite à l'intérieur du polytope $\mathcal{P}$ et qui regarde au fur et à mesure de sa marche les faces visibles. Quand il arrive à l'infini, le voyageur approche le polytope par son autre côté et regarde cette fois-ci les faces qui disparaissent lors de son avancement. Sa route se termine quand il a rejoint son point de départ. L'ordre dans lequel apparaissent les faces est un *effeuillage* particulier, dénommé *effeuillage par une droite orientée*. La notion d'*effeuillage* a également été utilisée dans la construction de diagramme de Delaunay de points contraints dont la complexité est sensible à la sortie ([BCDT91a, BCDT91b, BCDT96]). Des considérations plus mathématiques peuvent être trouvées dans les articles [BM71, Goo92].

3.5.3 L'énumération des facettes de l'enveloppe convexe d'un ensemble $\mathcal{P}$ de n points

On considère un ensemble $\mathcal{P}$ de n points de l'espace euclidien $\mathbb{E}^d$. On désire énumerer les facettes ($(d-1)$-faces) de son enveloppe convexe $\mathcal{CH}(\mathcal{P})$. On suppose les points en position générale. Le polytope enveloppe convexe est alors simplicial. Les facettes de $\mathcal{P}$ sont donc des $(d-1)$-simplexes. Énumérer une facette revient à donner les d points qui l'engendrent (spanning points). On suppose que l'origine est contenue à l'intérieur de $\mathcal{P}$ (sinon, on détermine en $\Theta(n)$ un point intérieur et on change l'origine. Cf. [Meg82, Meg83b]). Nous mimons alors la marche titanesque du voyageur comme précédemment:

On considère la droite orientée par le vecteur $\vec{\mathbf{a}}$ paramétrée par x: Δ : $x(t) = -\frac{1}{t} \times a$ avec $-\infty < t < \infty$. Notons $F_1, ..., F_N$ les facettes de $\mathcal{CH}(\mathcal{P})$ trouvées selon l'effeuillage suivant Δ. Pour $1 \leq i \leq N = \phi_{d-1}(\mathcal{P})$, on note $x(t_i)$ l'intersection de l'hyperplan engendré par la facette F_i avec la droite Δ. On note que $t_i < t_j$ si $i < j$. Bien sûr cet ordre n'est pas connu puisque l'on ne connait pas l'enveloppe convexe de $\mathcal{P}$. Toutefois, on se base sur son existence pour construire les faces dans cet ordre. On introduit d'abord la notion d'horizon puis on décrit l'algorithme.

3.5.4 L'horizon (horizon)

La notion d'horizon est une notion de visibilité du polytope relative à la position du voyageur ($t \in \mathbb{R}$). On définit une face G comme face-horizon à

l'instant t si et seulement si G est incidente à deux facettes F_i et F_j de $\mathcal{CH}(\mathcal{P})$ avec $t_i < t$ et $t_j > t$. On désigne par horizon-ridge (horizon-ridge) l'ensemble des $(d-2)$-face-horizons et par horizon-peak (horizon-peak) l'ensemble des $(d-3)$-face-horizons. Les propriétés des horizons sont importantes. Elles sont résumées dans les lemmes ci-dessous :

Lemme 11 *Soit G une $(d-2)$-face de $\mathcal{CH}(\mathcal{P})$ et F_i et F_j les deux facettes adjacentes à G, on a*

(i) G est une sous-facette de l'horizon-ridge pour $t \in [t_i, t_j)$.

(ii) Si G est une $(d-2)$-face de l'horizon pour t alors l'hyperplan engendré par G et $x(t)$ supporte $\mathcal{CH}(\mathcal{P})$.

Lemme 12 *Pour $t_1 < t < t_n$, l'horizon à t est isomorphe à un $(d-1)$-polytope.*

Ce dernier lemme permet de considérer les $(d-3)$-faces. En effet, une $(d-3)$-face horizon (peak-face) G est incidente à exactement deux $(d-2)$-faces G_1 et G_2. Notons H_G l'hyperplan engendré par les faces G, G_1 et G_2, et $x(t_G)$ l'abscisse temporelle où H_G coupe Δ. Supposons par hypothèse de récurrence que nous avons déjà trouvé les i premières facettes dans l'ordre de l'effeuillage et que nous connaissons l'horizon à t_{i-1}. On désire trouver F_i et mettre à jour l'horizon au temps t_i. On réalise ceci en maintenant un ensemble $\mathcal{H}$ d'hyperplans parmi lequels l'un d'eux contient F_i. Ces hyperplans sont mis à jour à chaque étape de l'algorithme. L'hyperplan de H qui réalise la "première" coupure avec Δ sera celui qui contiendra la face F_i.

En vertu des propriétés d'effeuillage de polytope, l'intersection $F_i \cap (\cup_{j<i} F_j)$ est l'union des $(d-2)$-faces. On distingue les deux cas suivants:

1. L'intersection consiste en plusieurs $(d-2)$-faces horizon(ridge-faces). Deux de ces $(d-2)$-faces sont incidentes à une même $(d-3)$-face G (propriété de la boule topologique). Il s'ensuit que F_i doit être contenu dans H_G. Ainsi on met à jour $\mathcal{H}$ en rajoutant les hyperplans H_G.

2. L'intersection consiste en une seule $(d-2)$-face horizon (ridge-face). Dans ce cas on ne peut déduire F_i de la structure d'horizon. Néanmoins, F_i doit contenir un sommet $p \in \mathcal{P}$ qui n'est pas contenu dans G. De plus p n'est contenu dans aucune autre face F_j, $j < i$. Supposons que nous connaissions tous les sommets de $\mathcal{P}$ qui sont sur $\partial\mathcal{CH}(\mathcal{P})$ et que pour chaque sommet sur le bord de l'enveloppe convexe on connaisse la première facette F_p de l'effeuillage qui le contient alors on connait H_p l'hyperplan engendré par F_p et $x(t_{F_p})$. On inclut dans $\mathcal{H}$ l'ensemble de ces hyperplans H_p.

On obtient alors le lemme suivant pour la construction de l'enveloppe convexe:

Lemme 13 *Soit $\mathcal{H}_i$ l'ensemble des hyperplans définis au temps t_i comme précédemment en enlevant les hyperplans H_ϵ tels que $t_\epsilon < t_{i-1}$ alors l'hyperplan H_ϵ ayant le plus petit t_ϵ contient la prochaine face F_i.*

L'algorithme en découle naturellement. On doit cependant résoudre le problème de trouver les sommets de $\mathcal{CH}(\mathcal{P})$ et pour chacun de ces sommets trouver la *première face* qui le contient. On peut résoudre ce problème par programmation linéaire. Nous précisons ces problèmes de programmation linéaire en annexe (page 207).

3.5.5 Assemblons les différentes briques pour décrire l'algorithme

Grâce à l'hypothèse de position générale des points de $\mathcal{P}$, chaque k-face F de $\mathcal{CH}(\mathcal{P})$ est engendrée par exactement $(k+1)$ sommets de l'enveloppe convexe. L'algorithme doit garder trace des $(d-2)$-faces de l'horizon (horizon-ridge) et des $(d-3)$-faces de l'horizon (horizon-peak) afin de les mettre à jour rapidement. Elles sont stockées dans un $(d-1)$-graphe régulier avec les ridges comme nœuds et les peaks comme arêtes (calque les relations d'incidence d'un IDAG [BY95b]). On désigne par *graphe d'horizon* ce graphe. On doit également maintenir une queue de priorité pour $\mathcal{H}$ contenant les valeurs de t_p et t_G et un dictionnaire des $(d-1)$-ensembles de $\mathcal{P}$. L'algorithme est détaillé dans l'encadré 3.15.

On ne détaille pas ici les mises à jour du graphe d'incidence et du dictionnaire. Ces informations peuvent être trouvées dans l'article originel [Sei86a].

3.5.6 Analyse de la complexité

Si l'on considère la dimension d fixée alors chaque programme linéaire peut être résolu en temps optimal $O(n)$ (searching-and-pruning, [Meg84]). Pour sélectionner les points qui sont sur le bord de l'enveloppe convexe, on utilise n fois la programmation linéaire soit une complexité en $O(n^2)$. Ce temps quadratique peut être réduit en $O(n^{2-\frac{2}{\lfloor \frac{d}{2} \rfloor + 1} + \epsilon})$ $\forall\, \epsilon > 0$ en utilisant les résultats sur l'optimisation des requêtes linéaires [Mat93, MS92]. A chaque fois que l'on affiche une facette, on a au plus $O(d^2)$[9] éléments du graphe d'horizon qui changent et autant de requêtes au dictionnaire et dans la file

9. Correspond à l'établissement d'une clique à d sommets.

Algorithme 3.15: *Construction des facettes de $\mathcal{CH}(\mathcal{P})$ pour un ensemble de points en dimension d ($d \geq 4$).*

début
Pour chaque point $p \in \mathcal{P}$ faire
Résoudre un programme linéaire
Si le programme linéaire est infaisable alors éliminer p
sinon calculer t_p et $Q_p \subset \mathcal{P}$ l'ensemble des $(d-1)$ points définissant la facette.
Mettre les valeurs de t_p dans une queue prioritaire $\mathcal{Q}$ et
Entrer les valeurs de Q_p dans un dictionnaire $\mathcal{D}$.
fin; (* faire *)

$t \leftarrow$*Minimum*($\mathcal{Q}$);
AfficherFace(t);
(* on donne les d sommets de la facette trouvée *)
Construire le graphe d'horizon;
(* on construit les liens entre le dictionnaire $\mathcal{D}$ et les noeuds *)
Effacer de $\mathcal{Q}$ les points q tel que $t_q = t_p$

Tant que $\mathcal{Q} \neq \emptyset$ faire
$t \leftarrow$*Minimum*($\mathcal{Q}$);
si t est réalisé ($t = t_p$) par un sommet $p \in \mathcal{P}$ alors
Afficher($Q_p \cup \{p\}$);
Mettre à jour le graphe d'horizon en utilisant $\mathcal{D}$;
Enlever t_p de la queue de priorité;
finsi;
si il existe un ensemble Γ d'horizon-peaks g ($t = t_g$)
Afficher($\cup_{g \in \Gamma} g$);
(* c'est une nouvelle facette *)
finsi;
fin; (* fin faire *);
fin

de priorité. Comme la taille du dictionnaire est $O(n)$ et celle de la file de priorité au plus $O(n^{\lfloor \frac{d}{2} \rfloor})$, le temps mis pour énumerer les facettes dans la boucle `tant que` est $O(F_N \log n)$. On aboutit donc au théorème suivant:

Théorème 9 *L'enveloppe convexe d'un ensemble de n points $\mathcal{P}$ en dimension d peut être calculé par un algorithme énumeratif en temps $O(n^{2-\frac{2}{\lfloor \frac{d}{2} \rfloor + 1}+\epsilon} + F \log n)$ $\forall\ \epsilon > 0$ où F est le nombre de facettes de $\mathcal{CH}(\mathcal{P})$.*

Après avoir résolu les n programmes linéaires, nous pouvons éliminer les points qui n'interviennent pas dans l'enveloppe convexe. Si bien que le terme $F \log n$ peut être remplacé par un $F \log h$.

La construction du graphe d'incidence peut se faire de manière analogue. Nous en donnons les détails en annexe (page 207) afin de ne pas alourdir cet état de l'art.

Une autre approche considérée par Avis et Fukuda [AF92b, AF91a, AF91b, AF92a] pour l'énumération des sommets d'un polytope défini par l'intersection de demi-espaces consiste à calculer par programmation linéaire un sommet initial. Puis, on fixe ce sommet comme étant une solution optimale pour un problème de programmation linéaire et on renverse (reverse search) les règles de pivot pour décrire, par l'intermédiaire d'un arbre couvrant, tous les sommets du polytope. Ce genre de technique a permis également d'énumerer les triangulations régulières d'un ensemble de points [MII96].

Chapitre 4

Implantation de l'algorithme [KS82, KS86]

L'implantation de l'algorithme de Kirkpatrick et Seidel a été réalisée en C++.

La figure 4.1 montre le résultat du programme sur un ensemble $\mathcal{P}$ de 100 points de $\mathbb{E}^2$ (34 points sont sur le bord de l'enveloppe convexe). On ne donne ici que les temps de calcul obtenus sur un SUN4 (cf. table 4.1). La figure 4.2 montre les performances du programme.

En comparant notre implantation à celles de LEDA (Library for Efficient Data Structure, MPI Germany), on constate que l'algorithme implanté tourne moins vite qu'une marche de Jarvis ou de Graham pour des valeurs de n inférieures à 100000. Le comportement optimal asymptotique est repéré pour des valeurs de n supérieures à 250000 en pratique, pour des distributions de points uniformes dans un carré. Si on implante l'algorithme en utilisant un seul tableau afin d'éviter l'allocation d'un nouveau tableau temporaire,

TAB. 4.1 – *Temps d'exécution de l'algorithme sur un* SUN4.

n	$t(n)$ en μs
100000	75530312
10000	7483034
1000	916630
100	49998
10	< 1

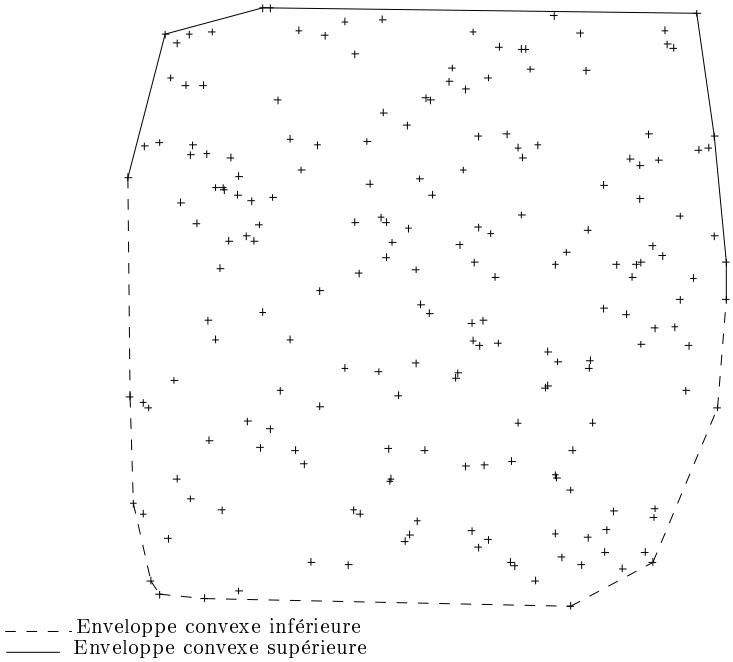


FIG. 4.1 – *Enveloppe convexe d'un ensemble* $\mathcal{P}$ *de* 100 *points de* $\mathbb{E}^2$ *(Sortie* JpDraw *).*

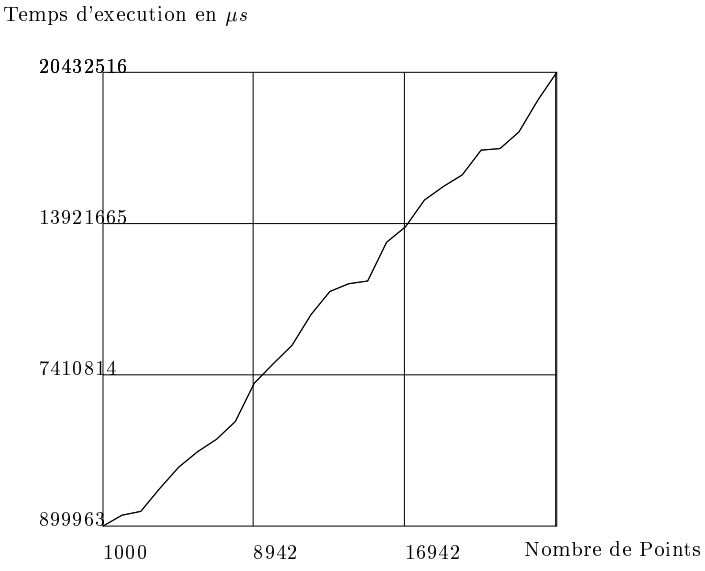


FIG. 4.2 – *Temps d'exécution pour* $n \in [1000, 25000]$ *par incrément de* 1000.

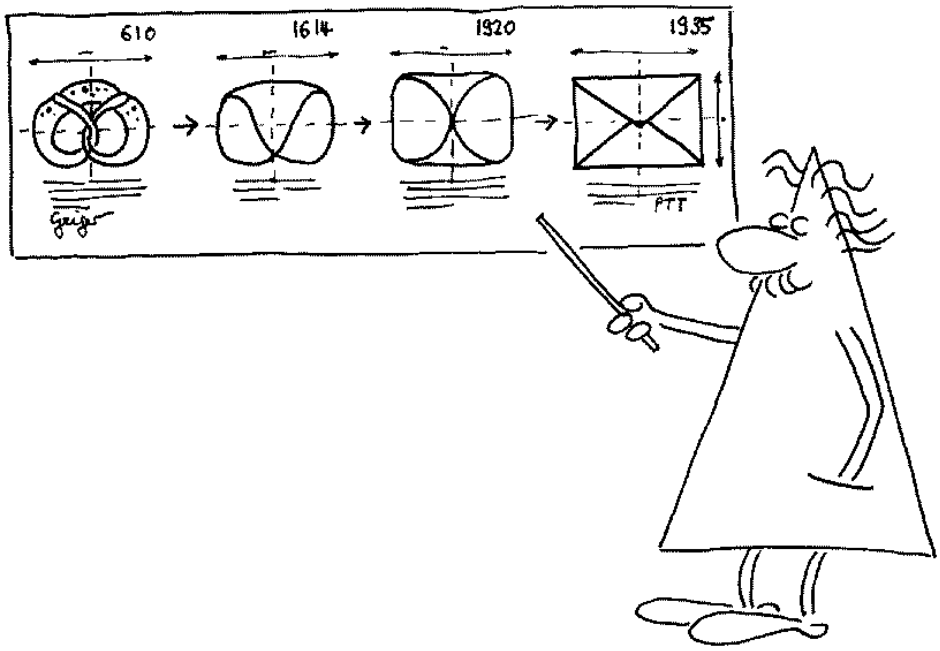
algorithme dit *en place* (e.g. par la méthode de Dijkstra), on s'apercoit que celui-ci est nettement moins rapide en pratique. En pratique, nous pouvons calculer la médiane en temps moyen $2n + o(n)$ en utilisant la méthodologie du tri rapide, encore appelé tri de Hoare.

Deuxième partie

Le calcul d'enveloppes

Chapitre 1

Algorithmes pour le calcul d'enveloppes et problèmes connexes



Dans les deux prochains chapitres de cette thèse, nous mettons en œuvre une méthodologie basée sur le prétraitement adaptatif des données afin d'obtenir des algorithmes adaptatifs. Nous décrivons dans le chapitre 2 un algorithme basé sur une approche composite combinant à la fois l'approche *ma-*

riage avant conquête mais aussi le paradigme *groupement par paquets* pour des objets géométriques non ponctuels (autrement dit des objets différents de points). Puis nous considérons le calcul des k premières couches convexes ou des maxima d'un ensemble de points du plan euclidien. Nous proposons pour ce problème un algorithme adaptatif utilisant la méthodologie *groupement par paquets*.

1.1 Un algorithme adaptatif pour calculer l'enveloppe d'objets

1.1.1 Calcul de l'enveloppe convexe par une enveloppe supérieure de fonctions

On s'intéresse dans cette partie à la transformation du calcul de l'enveloppe convexe à celui du calcul d'une envelope supérieure. Nous commençons tout d'abord par expliquer cette transformation pour le cas des points du plan euclidien $\mathbb{E}^2$ puis nous l'étendons aux cas d'objets planaires arbitraires. Cette transformation se transpose aisément en toutes dimensions.

1.1.2 Le cas des points

Soit $\mathcal{P} = \{P_1 = (x_1, y_1), ..., P_n = (x_n, y_n)\}$ un ensemble de n points de l'espace euclidien $\mathbb{E}^2$. On désire calculer l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$ de $\mathcal{P}$, c'est-à-dire le plus petit objet convexe (au sens de l'inclusion) contenant $\mathcal{P}$. On peut se limiter au calcul de l'enveloppe convexe supérieure $\mathcal{CH}^+(\mathcal{P})$ de $\mathcal{P}$ qui est définie comme la partie de l'enveloppe convexe visible depuis le point $(0, +\infty)$. On définit de manière analogue l'enveloppe convexe inférieure $\mathcal{CH}^-(\mathcal{P})$ de $\mathcal{P}$. L'enveloppe convexe $\mathcal{CH}(\mathcal{P})$ se déduit trivialement en temps linéaire des enveloppes convexes supérieure et inférieure.

On considére la fonction support de l'enveloppe convexe qui à une pente s associe la droite support $L(s)$ à $\mathcal{CH}^+(\mathcal{P})$ de pente s. Soit $\mathcal{C}(s)$ le ou les points de contact de la droite support $L(s)$ avec $\mathcal{P}$. L'enveloppe convexe supérieure peut-être décrite en utilisant $L(\cdot)$. En effet, on peut enrober l'enveloppe convexe supérieure par le faisceau de droites supports obtenu lorsqu'on fait varier s de $-\infty$ à $+\infty$. Lorsqu'on examine pour s variant progressivement de $-\infty$ à $+\infty$ le ou les points de contact $\mathcal{C}(s)$, on s'aperçoit que $\mathcal{C}(s)$ change un nombre fini de fois. Plus particulièrement, nous obtenons la séquence ordonnée de gauche à droite des sommets de l'enveloppe convexe supérieure. Nous allons calculer pour $s \in (-\infty, +\infty)$ le ou les points de contact de $\mathcal{P}$ pour la

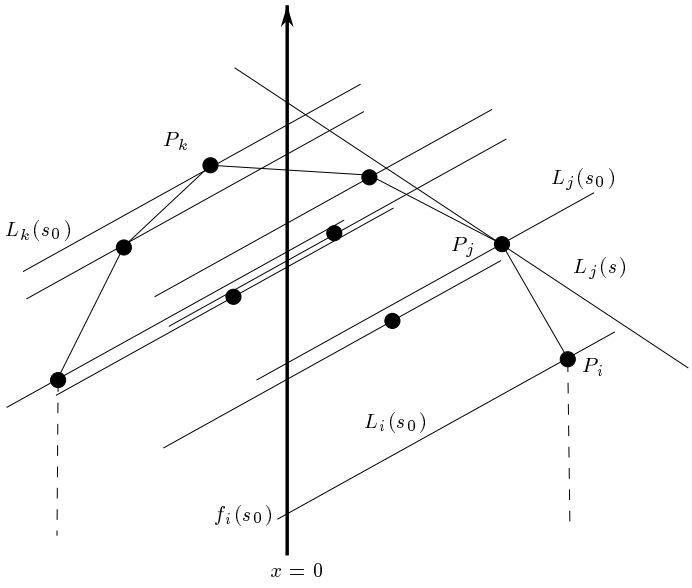


FIG. 1.1 – *L'enveloppe convexe peut être enrobée par $L(\cdot)$. Pour une pente s_0 fixée, le point P_k de $\mathcal{P}$ sur $L(s_0)$ est celui vérifiant $f_k(s_0) = \max_{i=1}^{n} f_i(s_0)$.*

droite support à $\mathcal{CH}(\mathcal{P})$ de pente s. Pour un point $P_i = (x_i, y_i)$ appartenant à $\mathcal{P}$, nous considérons la droite $L_i(s)$ passant par P_i de pente s. On a:

$$L_i(s) : y = s(x - x_i) + y_i.$$

Les droites $\mathcal{L}(s) = \{L_i(s),\ 1 \leq i \leq n\}$ sont mutuellement parallèles entre elles. Clairement, pour une pente donnée s_0 les points (un ou deux points de contact si $\mathcal{P}$ est en position générale) de contacts de $L(s_0)$ sont les points situés sur les droites n'admettant aucune droite de $\mathcal{L}(s_0)$ "au-dessus" d'elle (cf. figure 1.1). En effet, considérons une droite $L_i(s_0)$ passant par P_i. S'il existe une droite $L_j(s_0)$ au-dessus de $L_i(s_0)$ alors P_j est au-dessus de $L_i(s_0)$ et le point P_i n'est pas un point de contact de $L(s_0)$. Pour comparer deux droites parallèles $L_1 : y = s(x - x_1) + y_1$ et $L_2 : y = s(x - x_2) + y_2$, nous choisissons arbitrairement une droite non horizontale, par exemple la droite verticale $x = 0$, et nous comparons les ordonnées des points appartenant à L_1 et L_2 pour l'abscisse $x = 0$. Ainsi, si $y_1 - sx_1 > y_2 - sx_2$, $y_1 - sx_1 = y_2 - sx_2$ ou $y_1 - sx_1 < y_2 - sx_2$ nous disons que L_1 est au-dessus, égale ou en-dessous de L_2 et on écrit $L_1 > L_2$, $L_1 = L_2$ ou $L_1 < L_2$.

Notons f_i la fonction qui à une pente donnée s associe la valeur $f_i(s) = y_i - sx_i$ et $\mathcal{F} = \{f_i, 1 \leq i \leq n\}$. $P_i \in \mathcal{C}(s)$ si $f_i(s) = \max_{j=1}^{n}\{f_j(s)\}$. On détermine alors aisément $\mathcal{C}(\cdot)$ en calculant l'enveloppe supérieure $\mathbf{E}_{\mathcal{F}}$ des

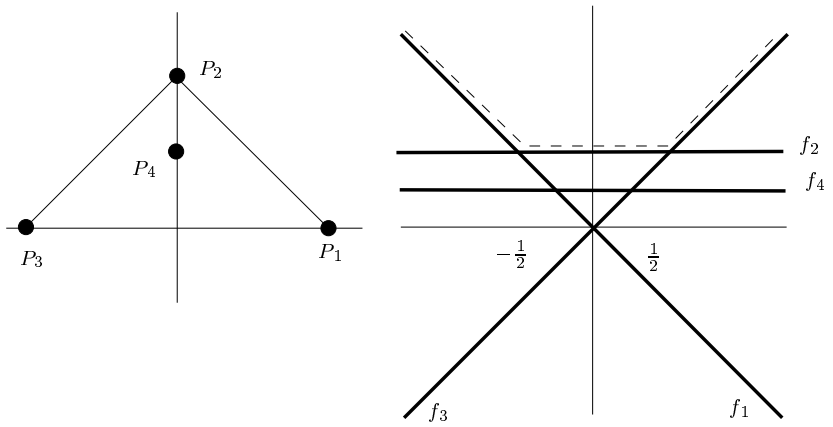


FIG. 1.2 – *Correspondance entre l'enveloppe supérieure* $\mathbf{E}_{\mathcal{F}}$ *et l'enveloppe convexe supérieure* $\mathcal{CH}^+(\mathcal{P})$.

fonctions f_i:

- À un sommet v de $\mathbf{E}_{\mathcal{F}}$ correspond un segment de l'enveloppe convexe supérieure. On détermine aisément les extrêmités du segment en considérant les deux droites L_i et L_j, issues respectivement des fonctions f_i et f_j, qui se coupent en v_{ij}. En effet, le segment est $[P_iP_j]$.
- À un arc de $\mathbf{E}_{\mathcal{F}}$ correspond un point de $\mathcal{P}$ appartenant à l'enveloppe convexe supérieure $\mathcal{CH}^+(\mathcal{P})$.

Considérons l'exemple suivant: $\mathcal{P} = \{P_1 = (1,0), P_2 = (0,1), P_3 = (-1,0), P_4 = (0,\frac{1}{2})\}$. Nous avons $\mathcal{CH}(\mathcal{P}) = P_1 \circ P_2 \circ P_3$ et $\mathcal{CH}^+(\mathcal{P}) = P_1 \circ P_2 \circ P_3$. Les fonctions caractéristiques associées aux points sont $\mathcal{F}(s) = \{f_1(s) = -s, f_2(s) = -1, f_3(s) = s, f_4(s) = \frac{1}{2}\}$. La figure 1.2 montre la relation entre l'enveloppe supérieure $\mathbf{E}_{\mathcal{F}}$ et l'enveloppe convexe supérieure $\mathcal{CH}^+(\mathcal{P})$. On peut résumer dans le tableau ci-dessous la correspondance :

	$\mathbf{E}_{\mathcal{F}}$	$\mathcal{CH}^+(\mathcal{P})$
$s \in (-\infty, -\frac{1}{2})$	arête de $f_1(\cdot)$	sommet P_1
$s = \frac{1}{2}$	intersection de $f_1(\cdot)$ et $f_2(\cdot)$	arête $[P_1P_2]$
$s = (\frac{1}{2}, -\frac{1}{2})$	arête de $f_2(\cdot)$	sommet P_2
$s = \frac{1}{2}$	intersection de $f_2(\cdot)$ et $f_3(\cdot)$	arête $[P_2P_3]$
$s = (\frac{1}{2}, +\infty)$	arête de $f_3(\cdot)$	sommet P_3

1.1.3 Le cas des objets plans

Considérons le cas des objets planaires. Un objet O est dit *lisse* si, pour tout $\theta \in [0, 2\pi)$, il existe un unique point de contact pour la droite support à O de pente θ. Par exemple, un disque ou une ellipse sont lisses mais un carré ou un polygone ne le sont pas. Soit $\mathcal{O} = \{O_1, ..., O_n\}$ un ensemble de n objets lisses du plan. Le même formalisme peut être appliqué si ce n'est que nous considérons les fonctions suivantes (elles ne sont toutefois plus linéaires) :

$$L_i : y = s(x - x_i(s)) + y_i(s),$$

où $P_i(s) = (x_i(s), y_i(s))$ est le point du bord $\partial\mathcal{CH}(O_i)$ de l'enveloppe convexe supérieure de O_i qui admet une droite support à O_i de pente s. Notons $\mathcal{L} = \{L_i, 1 \leq i \leq n\}$ et $\mathcal{F} = \{y_i(s) - sx_i(s), 1 \leq i \leq n\}$, alors nous pouvons déduire de l'enveloppe supérieure $\mathbf{E}_{\mathcal{F}}$ de $\mathcal{F}$ l'enveloppe convexe supérieure $\mathcal{CH}^+(\mathcal{O})$ de $\mathcal{O}$.

On peut résumer ainsi le lien entre l'enveloppe convexe supérieure et l'enveloppe supérieure:

> Le calcul de l'enveloppe convexe supérieure d'une famille d'objets planaires se ramène au calcul de l'enveloppe supérieure de ses fonctions supports.

Afin d'éclaircir le cas des objets plans, nous considérons le cas des cercles $\mathcal{C}$ du plan. Un cercle A est défini par son centre $C(A)$ et son rayon $r(A)$: $A =< C(A), r(A) >$. Un cercle $< C(A), r(A) >$ peut être paramétrisé par $\theta \in [0, 2\pi)$ comme suit:

$$A(\theta) = \begin{cases} x(\theta) = r(A)\cos(\theta) + x(C(A)) \\ y(\theta) = r(A)\sin(\theta) + y(C(A)) \end{cases}$$

L'équation de la droite tangente à $A(\theta)$ est donnée par l'expression

$$< \overrightarrow{AA(\theta)}, \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} >= r(A).$$

Soit la droite d'équation $y = \frac{r(A)}{\sin\theta} + y(C(A)) - (x - x(C(A)))\cot\theta$. On cherche a déterminer le point $B(s) \in \partial A$ qui admet la droite support à A de pente s. Il en découle que $B(s) = A(\arctan(-\frac{1}{s}))$.

1.1.4 Remarques et conclusion

Nous avons exhibé une relation entre le calcul de l'enveloppe convexe supérieure d'une famille d'objets et l'enveloppe supérieure des fonctions supports associées aux objets. Nous pouvons déduire la complexité de l'enveloppe convexe d'objets par la complexité de l'enveloppe supérieure de ses fonctions supports. Ainsi, il n'est pas étonnant de retrouver que la complexité de l'enveloppe convexe de n points en dimension d est celle de l'intersection de n demi-espaces, c'est-à-dire $O(n^{\lfloor \frac{d}{2} \rfloor})$. En effet, l'enveloppe supérieure de n hyperplans de $\mathbb{E}^d$ peut être perçue comme l'intersection de n demi-espaces.

Ainsi, on pourra étudier les fonctions supports des objets afin d'en déduire une borne supérieure sur la complexité de leur enveloppe convexe. Considérons n objets planaires dont les fonctions supports associées se coupent mutuellement en s points, alors on en déduit que la taille de l'enveloppe convexe de ces objets est $O(\lambda(n,s))$ où $\lambda(n,s)$ est la taille maximale d'une séquence de Davenport-Schinzel de paramètres (n,s) [ASS89, Kla95]. En étudiant le cas des disques, il devient aisé de montrer que la taille de l'enveloppe convexe est linéaire. Notez que par le calcul de l'enveloppe supérieure des fonctions supports nous connaissons pour $s \in (-\infty, +\infty)$ l'objet qui admet la droite support à l'enveloppe convexe supérieure pour la pente s. Toutefois, on ne peut pas directement déduire la face de l'enveloppe convexe supérieure intersectée par une droite verticale Δ donnée, c'est-à-dire le pont, puisqu'il faudrait connaitre la pente de la droite support à l'enveloppe convexe supérieure dont le (un) point de tangence à l'enveloppe convexe appartient à la droite verticale Δ.

1.1.5 Calcul de l'enveloppe convexe d'un ensemble d'objets convexes par un algorithme incrémental randomisé en ligne

Nous proposons dans cette partie un algorithme randomisé incrémental qui calcule l'enveloppe convexe d'un ensemble d'objets convexes. Soit $\mathcal{S} = \{O_1, ..., O_n\}$ un ensemble de n objets planaires. L'ensemble $\mathcal{S}$ est dit de type m si pour toute paire d'objets de $\mathcal{S}$ leur enveloppe convexe est de taille inférieure ou égale à $2m$. On montre dans cette thèse par réduction au calcul d'enveloppe supérieure que $|\mathcal{CH}(\mathcal{S})| \leq 4\lambda(n,m)$. L'algorithme est incrémental, c'est-à-dire les objets sont ajoutés un par un, et en ligne, c'est-à-dire qu'il n'est pas nécessaire de connaitre à l'avance les objets. Nous décrivons briévement le formalisme de Clarkson et Shor [CS89] pour la construction incrémentale randomisée. Puis nous détaillons une structure de données: *le*

graphe d'influence [BY95b] et nous concluons en donnant l'algorithme dans ce formalisme en utilisant le graphe d'influence.

1.1.6 Formalisme objets/régions selon Clarkson et Shor [CS89]

Généralement en géométrie algorithmique beaucoup de problèmes peuvent se formuler abstraitement à l'aide d'objets, de régions et de *conflits* entre les objets et les régions.

- Les objets sont des éléments d'un univers $\mathcal{O}$ et constituent les données du problème. Dans le cas de l'enveloppe convexe d'objets convexes planaires, on considére l'univers $\mathcal{O}$ des objets convexes planaires et $\mathcal{S} \subset \mathcal{O}$ est un sous-ensemble de taille $|\mathcal{S}| = n$.

- Les régions quant à elles sont des éléments d'un univers de régions noté $\mathcal{F}$. Dans le cas de l'enveloppe convexe, les éléments de $\mathcal{F}$ sont des sous-ensembles connexes bien définis du plan. Les régions sont définies par des objets. Plus formellement, il existe un entier b strictement positif et une relation δ entre l'ensemble des sous-ensembles de $\mathcal{O}$ de cardinalité au plus b et l'ensemble des régions $\mathcal{F}$. Chaque élément de $\mathcal{F}$ est défini par au plus b objets et la relation δ qui les lie.

Soit $\mathcal{S}$ un ensemble d'objets ($\mathcal{S} \subset \mathcal{O}$), on définit $\mathcal{S}^b$ comme étant l'ensemble des sous-ensembles de $\mathcal{S}$ de cardinalité au plus b :

$$\mathcal{S}^b = \{\mathcal{V} \subseteq \mathcal{S} \mid |\mathcal{V}| \leq b\}.$$

Une région $F \in \mathcal{F}$ est définie par l'ensemble $\mathcal{S}$ d'objets s'il existe un élément $E \in \mathcal{S}^b$ en relation avec F. On note $\mathcal{F}(\mathcal{S})$ l'ensemble des régions définies par l'ensemble des objets $\mathcal{S}$:

$$\mathcal{F}(\mathcal{S}) = \{F \in \mathcal{F} \mid \exists E \in \mathcal{S}^b \; E\delta F\}$$

Dans le cas de l'enveloppe convexe, nous utilisons les arcs d'objets définis par trois objets de $\mathcal{S}$. Puisque l'enveloppe convexe d'objets convexes planaires est une suite alternée d'arcs et de segments bitangents, nous associons à chaque arc la région du plan délimitée par cet arc et les deux segments bitangents qui le délimitent. Les deux segments bitangents adjacents à un arc a de l'enveloppe convexe supportent à la fois a mais également un autre objet. La région d'un arc est donc définie par trois objets ($b = 3$). On pourrait également utiliser le formalisme de birégion [BY95a] pour définir les régions uniquement par des arcs d'objet. Chaque région est caractérisée par

un sous-ensemble de $\mathcal{O}$ appelé *domaine d'influence*. Dans le cas de l'enveloppe convexe, le domaine d'influence associé à une région est l'union des demi-plans ouverts tangents à la courbe $s_1 a s_2$ où s_1 et s_2 sont les deux segments bitangents à l'arc a. Un objet $O \in \mathcal{O}$ est en conflit avec une région $F \in \mathcal{F}$ si son intersection avec le domaine d'influence de cette région est non vide.

Soit F une région de $\mathcal{F}$ et $\mathcal{S}$ un nombre fini d'objets, on note $\mathcal{S}(F)$ l'ensemble des objets de $\mathcal{S}$ en conflit avec la région F (son domaine d'influence). On appelle *largeur* de F par rapport à $\mathcal{S}$ le nombre $|\mathcal{S}(F)|$ d'objets de $\mathcal{S}$ en conflit avec F.

> Calculer l'enveloppe convexe d'objets convexes planaires $\mathcal{S}$ se ramène à construire l'ensemble des régions définies par ces objets de largeur nulle par rapport à $\mathcal{S}$.

Dans ce qui suit, on suppose que la relation δ est fonctionnelle, c'est-à-dire que les données du problème sont en position générale. On note $\mathcal{F}_j(\mathcal{S})$ le sous-ensemble de $\mathcal{F}(\mathcal{S})$ constitué par les régions de largeur j par rapport à $\mathcal{S}$:

$$\mathcal{F}_j(\mathcal{S}) = \{F \in \mathcal{F}(\mathcal{S}) \mid |\mathcal{S}(F)| = j\}.$$

1.1.7 Le graphe d'influence

Nous décrivons dans cette partie une structure de données appelée *graphe d'influence* qui est très utile pour la construction incrémentale randomisée. Soit $\mathcal{R} \subseteq \mathcal{S}$ un sous-ensemble courant des objets déjà traités. On appelle régions de largeur courante nulle les régions définies par des éléments de $\mathcal{R}$ de largeur nulle. Le graphe d'influence (IDAG) est un graphe orienté connexe et acyclique possédant une racine. Un nœud peut avoir plusieurs pères. On appelle *feuille* du graphe d'influence les nœuds ne possédant pas de fils. Le graphe d'influence est caractérisé par les deux propriétés suivantes:

- L'ensemble $\mathcal{F}_0(\mathcal{R})$ des régions de largeur courante nulle est une bijection avec un sous-ensemble de l'ensemble des feuilles du graphe d'influence.

- Le domaine d'influence d'une région associé à un nœud du graphe d'influence est inclus dans l'union des domaines d'influence des régions associés aux pères de ce nœud.

La première propriété permettra de localiser rapidement les régions de largeur nulle tandis que la seconde permettra de parcourir efficacement le graphe d'influence afin de détecter les conflits occasionnés par l'ajout d'un objet. Afin d'actualiser rapidement le graphe d'influence lorsque nous ajoutons un nouvel objet, nous supposons les trois hypothèses d'actualisation suivantes:

- Pour tout objet O de l'univers des objets $\mathcal{O}$ et toute région F de l'univers $\mathcal{F}$, on peut répondre en temps constant si O est en conflit avec F. Ici, temps constant signifie temps dépendant de b.

- Le nombre de fils de chaque nœud du graphe d'influence est borné.

- À chaque insertion d'un nouvel objet, la phase de mise à jour peut-être effectuée en un temps proportionnel au nombre de régions de largeur courante nulle en conflit avec l'objet introduit à cette étape.

Nous citons ci-après le principal théorème de Boissonnat et al. [BDS+92]:

Théorème 10 *Soit un algorithme en ligne qui construit un graphe d'influence et satisfait les hypothèses d'actualisation. Le graphe d'influence correspondant à un ensemble $\mathcal{S}$ de taille n occupe en moyenne un espace mémoire*

$$\tilde{O}\left(n\sum_{r=1}^{n}\frac{f_0(r,\mathcal{S})}{r}\right).$$

L'algorithme traite un ensemble d'objets S de taille n en temps moyen

$$\tilde{O}\left(\sum_{r=1}^{n}\frac{f_0(\lfloor\frac{r}{2}\rfloor,\mathcal{S})}{r^2}\right).$$

Une analyse non amortie montre que la n^e étape de l'algorithme s'effectue en temps moyen

$$\tilde{O}\left(\sum_{r=1}^{n}\frac{f_0(r,\mathcal{S})}{r}\right).$$

où $f_0(r,\mathcal{S})$ est l'espérance mathématique du nombre de régions de largeur nulle définie par un échantillon aléatoire de taille r de $\mathcal{S}$.

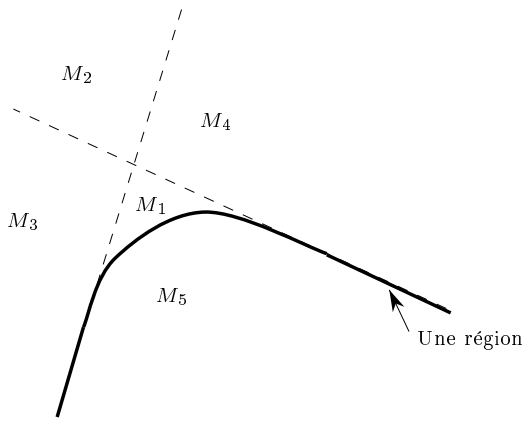


FIG. 1.3 – *Une région définie par un arc d'objet et deux segments bitangents limitant l'arc ($b = 3$ objets). Le domaine d'influence est constitué par les morçeaux M_1, M_2, M_3, M_4.*

1.1.8 Application à l'enveloppe convexe

Soit $\mathcal{S} = \{O_1, ..., O_n\}$ un ensemble de n objets planaires convexes de type m. L'algorithme incrémental en ligne est le suivant (son analyse est randomisée) :

Initialisation. $i = 0$, $\mathcal{R} = \emptyset$ et $\mathcal{G} = \emptyset$.

Boucle. Tant qu'il y a des objets à traiter, prendre un objet courant O, $\mathcal{R} = \mathcal{R} \cup \{O\}$ et actualiser le graphe d'influence $\mathcal{G}$.

Résultat. Renvoyer toutes les régions de largeur nulle de $\mathcal{G}$.

À chaque itération, la localisation du nouvel objet O à insérer dans le graphe d'influence courant $\mathcal{G}$ fournit l'ensemble $L(O)$ des régions de $\mathcal{G}$ en conflit avec O. Tout d'abord on détecte le conflit entre une région et un objet en calculant l'intersection de son domaine d'influence avec cet objet (se fait en temps constant si les objets ont une complexité bornée).

Chaque région en conflit avec O peut donner naissance à au plus trois nouvelles régions. Il faut faire une étude exhaustive en se basant sur le découpage du plan induit par un domaine d'influence donné. Un domaine d'influence partitionne le plan en 5 morçeaux. Nous examinons le cas où l'objet O est en conflit avec $1, 2, 3, 4$ ou 5 morçeaux (cf. figure 1.3).

Dans le graphe d'influence, une feuille est créée pour chaque nouvelle région (segment bitangent ou arc d'objet) et on la relie aux pères dont les nœuds sont associés aux arcs d'objets de $L(O)$ de telle façon que l'union

des domaines d'influence des pères contienne le domaine d'influence de la nouvelle région.

Pour implanter efficacement l'algorithme, on ajoute au graphe d'influence des pointeurs bidirectionnels entre les régions de largeur courante nulle qui sont adjacentes (c'est-à-dire que deux arcs d'objet sont reliés si et seulement si l'enveloppe convexe contient ces deux arcs d'objet et l'unique segment bitangent les reliant).

Chaque nœud du graphe d'influence a au plus trois fils et le temps de calcul nécessaire à la mise à jour du graphe d'influence et de ces pointeurs bidirectionnels est clairement proportionnel au nombre de régions en conflit avec l'objet inséré.

Pour un ensemble $\mathcal{S}$ de n objets convexes planaires de type m, l'espérance de la taille de l'enveloppe convexe est inférieure à la borne supérieure de sa complexité, et est donc $f_0(n, \mathcal{S}) = O(\lambda(n, m))$. L'algorithme randomisé incrémental en ligne a donc un temps de calcul moyen en $\tilde{O}(n\beta(n, s)\log n)$ [1] en utilisant un espace mémoire $\tilde{O}(n\beta(n, s))$ et un coût amorti par insertion de $\bar{O}(\beta(n, s)\log n)$ où $\beta(n, 2) = 1$ et $\beta(n, s)$ est une fonction croissant extrêmement lentement, ayant rapport avec les longueurs maximales des suites de Davenport-Schinzel [ASS89]. En dimension supérieure ce procédé ne s'applique malheureusement pas directement puisqu'une $(d-1)$-face peut être bordée par un nombre arbitrairement grand d'autres $(d-1)$-faces (on ne borne plus b). L'algorithme que nous obtenons est incrémental semi-dynamique et randomisé (ou son analyse est randomisée).

1.1.9 Résultats

Un ensemble d'objets planaires est de type m si la taille de l'enveloppe convexe de deux objets est bornée par $2m$. Dans le prochain chapitre, nous présentons un algorithme basé sur une approche composite utilisant à la fois le paradigme mariage-avant-conquête mais aussi la méthodologie groupement en paquets pour calculer l'enveloppe convexe d'un ensemble de n objets planaires de type m (m fixé). L'algorithme est adaptatif, c'est-à-dire que son temps de calcul dépend à la fois de la taille des entrées mais aussi de la taille de l'enveloppe convexe. Le principal ingrédient de cet algorithme est une méthode linéaire pour calculer un *pont*, c'est-à-dire une facette de l'enveloppe convexe coupant une droite donnée. Nous obtenons un algorithme dont la complexité est $O(n\beta(h, m)\log h)$. Ici $\beta(h, 2) = O(1)$ et $\beta(h, m)$ est

1. D'après le théorème 10, on sait que l'algorithme a un temps moyen de calcul en $\tilde{O}(\sum_{r=1}^{n} \frac{f_0(\lfloor \frac{r}{2} \rfloor, \mathcal{S})}{r^2})$. On a $f_0(\lfloor \frac{r}{2} \rfloor, \mathcal{S}) \leq f_0(r, \mathcal{S})$ et $\frac{f_0(r,\mathcal{S})}{r} \leq \beta(r, m)$. Soit $n\sum_{r=1}^{n} \frac{\beta(r,m)}{r} \leq n\beta(n, m)\sum_{i=1}^{n} \frac{1}{r} \leq O(n\beta(n, m)\log n)$.

une fonction qui croit extrêmement lentement. Il en découle que nous pouvons calculer en temps optimal $\Theta(n \log h)$ l'enveloppe convexe de disques, d'objets convexes homothétiques, d'objets non-recouvrants (ne se coupant pas deux à deux). La méthode décrite dans le prochain chapitre peut s'appliquer également au calcul de l'enveloppe supérieure de fonctions. En particulier, nous obtenons un algorithme optimal en $\Theta(n \log h)$ pour calculer l'enveloppe supérieure de segments.

1.2 Un algorithme adaptatif pour calculer les k premières couches convexes ou maximales

Soit $\mathcal{S}$ un ensemble de n points du plan euclidien, $\mathcal{CH}(\mathcal{S})$ son enveloppe convexe et $\mathcal{V}(\mathcal{S})$ l'ensemble des sommets sur le bord de l'enveloppe convexe $\mathcal{CH}(\mathcal{S})$. Soit $\mathcal{S}_1 = \mathcal{S}$ et $\mathcal{S}_{i+1} = \mathcal{S}_i \backslash \mathcal{V}(\mathcal{S}_i)$ pour $i \geq 1$. Les couches convexes de $\mathcal{S}$ sont définies par la séquence emboîtée de polygones convexes: $\mathcal{CH}(\mathcal{S}_1)$, ..., $\mathcal{CH}(\mathcal{S}_l)$ où l est le premier entier tel que $\mathcal{S}_{l+1} = \emptyset$. Le chapitre 3 présente un algorithme pour calculer les k premières couches convexes dont la complexité dépend à la fois de la taille des entrées mais aussi de la taille de la sortie, c'est-à-dire un algorithme adaptatif. Nous donnons un algorithme pour calculer les k premières couches convexes en temps $O(n \log H_k)$ où H_k est le nombre de points définissant les k premières couches. Soit $\mathcal{M}(\mathcal{S})$ le nombre de points dominants (maxima) de $\mathcal{S}$ et $\mathcal{S}_1 = \mathcal{S}$, $\mathcal{S}_{i+1} = \mathcal{S}_i \backslash \mathcal{M}(\mathcal{S}_i)$ pour $i \geq 1$. Nous pouvons également calculer les k premières couches des maxima de $\mathcal{S}$ en temps $O(n \log H_k)$ time, où H_k est le nombre de maxima sur les k premières couches de la décomposition en couches de maxima de $\mathcal{S}$. Calculer seulement les k premières couches est intéressant pour beaucoup de problèmes de géométrie algorithmique. Par exemple, cet algorithme permettra d'accélerer le prétraitement d'algorithmes calculant les k-ensembles (k-sets) et dualement les k-niveaux (k-levels). Cet algorithme est également utile en reconnaissance de motifs (pattern recognition), en statistique, etc.

1.3 Conclusion et perspectives.

La méthodologie décrite dans le prochain chapitre (chapitre 2) permet d'obtenir des algorithmes adaptatifs dans le plan. Toutefois, force nous est de constater que les extensions en dimensions supérieures ne sont pas directes. L'approche mariage-avant-conquête couplée avec le paradigme groupement en paquets permet d'obtenir des algorithmes adaptatifs dont la complexité

en temps de calcul s'avère être proche de l'optimal. Nous distingons deux problèmes ayant trait au calcul de l'enveloppe supérieure de segments et à la décomposition convexe du plan:

Problème 3 (Décomposition convexe) *Soit $\mathcal{S}$ un ensemble de n points de l'espace euclidien $\mathbb{E}^3$. Peut-on calculer sa décomposition en couches convexes en temps optimal $\Theta(n \log n)$ ou peut-on trouver une meilleure borne inférieure ? Actuellement, le meilleur algorithme a une complexité en $O(n^{1+\epsilon})$ [Cha95b, AM93] (auparavant on connaissait un algorithme en temps $O(n^{\frac{3}{2}} \log n)$ [Ede87]). Il faut noter que cet algorithme repose sur les procédures de lancer de rayons.*

Problème 4 (Enveloppe supérieure) *On considère un ensemble $\mathcal{S} = \{S_1, ..., S_n\}$ de n segments du plan euclidien $\mathbb{E}^2$. On montre dans cette thèse qu'il est possible de calculer l'enveloppe supérieure $\mathbf{E}_\mathcal{S}$ de façon sensible à la sortie en temps optimal $\Theta(n \log h)$. Soit v le nombre d'extremités différentes, alors la complexité de l'enveloppe supérieure de ces n segments dépend également du nombre extrémités distinctes: $|\mathbf{E}_\mathcal{S}| = O(v \log v)$ [AHK+91] (notons que le résultat $|\mathbf{E}_\mathcal{S}| = \Theta(v\alpha(v))$ stipulé dans le papier est erroné). Notez que $\frac{v}{2} \leq n = \Omega(v^2)$. Arkin et al. [AHK+91] décrivent un algorithme qui calcule une cellule de l'arrangement en temps $O(n\alpha(n) \log n + v \log^3 v)$. Peut-on dans ce cas précis (en tenant compte du paramètre v) calculer l'enveloppe supérieure de fonctions en temps meilleur que $O(n \log h)$, par exemple $O(n + v \log h)$?*

Notons que le paradigme de groupement en paquets s'applique aisément à beaucoup de problèmes. Toutefois, dans le cas du calcul de l'union d'un ensemble $\mathcal{S}$ de n objets de taille descriptive constante, on peut montrer que la détection d'un "*trou*" nécessite un temps $\Omega(n \log n)$ si bien qu'une borne inférieure pour le calcul de l'union de $\mathcal{S}$ est $\Omega(n \log n + h)$ où h est la taille de l'union, c'est-à-dire le nombre d'arcs définissant son bord. Güting [Güt84] considéra le cas de boîtes isothétiques dans le plan et donna un algorithme optimal, basé sur un balayage du plan, pour ce problème en construisant une structure de données élaborée. Il serait intéressant de considérer les objets c-orientés (définis par l'intersection de demi-plans d'au plus c directions distinctes) et de donner un algorithme adaptatif ayant une complexité semblable dans ce cas. Notons que nous pouvons calculer par un algorithme adaptatif le contour extérieur d'une composante connexe en modélisant son calcul par des lancers de rayon suivant son bord. Ainsi, si $\mathcal{S}$ est un ensemble de cercles, ou de polygones convexes, nous obtenons le bord de l'union de cette composante connexe en temps $O((n + h)\text{polylog } h)$ [AS90, AS93a, AM92, AM93, dB92].

L'approche *groupement par paquets* semble également prometteuse en parallélisme. Soit $\mathcal{P}_1, ..., \mathcal{P}_P, P$ processeurs d'une machine à gros grains [DFRC93, DF93]. Une des approches consisterait à réaliser $P = \frac{n}{p}$ paquets de p données répartis sur les P processeurs, puis de prétraiter ces données parallèlement en temps (séquentiel) $t(p)$. Une fois le prétraitement effectué on réalise itérativement h requêtes globales sur ces P paquets. On effectue une requête globale en choisissant un gagnant global parmi les P gagnants locaux $G_1, ..., G_P$ où G_i est le gagnant global du paquet traité par le processeur P_i (par une opération de préfixe paralléle). Soient $r(P)$ le temps de requête dans une structure de données construite dans l'étape du prétraitement et $s(P)$ le temps d'une opération de préfixe sur P processeurs, la complexité de cet algorithme est $O\left(\frac{n}{p}t(p) + h \times (r(p) + s(p))\right)$. On s'aperçoit immédiatement que pour des faibles valeurs de h cette méthode permettrait d'obtenir des algorithmes adaptatifs sur machine à gros grains. Toutefois lorsque h grandit, il devient un facteur limitant de cet algorithme parallèle. Il faut dans ce cas essayer de décomposer le problème en sous-problèmes indépendants [Nie96a] et équilibrer les données de manière à ce que la charge par processeur soit du même ordre de grandeur que la charge moyenne.

Finalement, considérons le problème suivant qui a des applications en biologie moléculaire (docking problem) et en planification de trajectoires (motion planning):

Problème 5 *(*Docking problem*) Soient $\mathcal{A}$ et $\mathcal{B}$ deux ensembles de n et m disques du plan respectivement. On considére $\mathcal{A}$ fixe et on cherche les positions de $\mathcal{B}$ (en autorisant les translations et rotations de l'union $\cup\mathcal{B}$ des disques de $\mathcal{B}$) dites admissibles telles que $\cup\mathcal{B}$ n'intersecte pas $\cup\mathcal{A}$.*

Nous montrons dans [DN96] que la complexité de l'espace de configurations est $O((mn)^{2+\epsilon})$ pour $\epsilon > 0$ puis décrivons un algorithme adaptatif pour calculer l'espace de configurations en $O(h^2 mn \log(mn))$ où $h \leq mn$ décrit la complexité, dans un certain sens, de la sortie. Récemment, en utilisant le résultat de Sharir et Schwarzkopf [SS96], nous obtenons un algorithme randomisé (théorique) qui calcule une cellule de l'espace de configuration en temps $\tilde{O}((mn)^{2+\epsilon})$.

Chapitre 2

An output-sensitive convex hull algorithm for planar objects

(Cette partie est disponible en rapport de recherche INRIA Numéro 2575 [NY95]. Ce travail fût présenté au 4^{e} atelier de travail Israelien de géométrie combinatoire et algorithmique (Avril 95) et est à paraître dans la revue International Journal of Computational Geometry and its Applications [NY96].)

Abstract

A set of planar objects is said to be of type m if the convex hull of any two objects has its size bounded by $2m$. In this paper, we present an algorithm based on the marriage-before-conquest paradigm to compute the convex hull of a set of n planar convex objects of fixed type m. The algorithm is output-sensitive, i.e. its time complexity depends on the size h of the computed convex hull. The main ingredient of this algorithm is a linear method to find a *bridge*, i.e. a facet of the convex hull intersected by a given line. We obtain an $O(n\beta(h,m)\log h)$-time convex hull algorithm for planar objects. Here $\beta(h,2) = O(1)$ and $\beta(h,m)$ is an extremely slowly growing function. As a direct consequence, we can compute in optimal $\Theta(n \log h)$ time the convex hull of disks, convex homothets, non-overlapping objects. The method described in this paper also applies to compute lower envelopes of functions. In particular, we obtain an optimal $\Theta(n \log h)$-time algorithm to compute the upper envelope of line segments.

Mots clef : Computational geometry, Convex hull, Upper Envelope, Output-sensitive algorithms, Marriage before conquest.

2.1 Introduction

Convex hull has been of main interest for years in computational geometry. Many articles have considered the case of points where general paradigms have been used or purposely developed. Worst-case optimal space and time algorithms have been established for sets of points in dimension d [Cla87, Cha91, Brö95]. However, the convex hull of n points in general position in a d-dimensional space ranges from the d-simplex with 2^{d+1} faces to maximal polytopes of size $O(n^{\lfloor \frac{d}{2} \rfloor})$ (see Ref. [Grü67, McM70]). We are interested in designing algorithms whose time complexity depends on both the input and output sizes: the so-called output-sensitive algorithms.

Optimal output-sensitive algorithms for points are known only in dimensions 2 and 3 by the time being. D.G. Kirkpatrick and R. Seidel [KS82, KS86] gave the first optimal output-sensitive algorithm in dimension 2. Their algorithm is based on a new paradigm: *marriage-before-conquest*. H. Edelsbrunner and W. Shi [ES91] gave an $O(n \log^2 h)$-time algorithm to compute the h facets of the convex hull of n points of $\mathbb{E}^3$ using the same paradigm. K.L. Clarkson and P.W. Shor [CS89] described an output-sensitive randomized algorithm for computing the convex hull of a set of points in dimension 3. The expected complexity of their algorithm is optimal. Their algorithm uses as a basic primitive the deterministic algorithm of D.G. Kirkpatrick and R. Seidel and was derandomized later on by B. Chazelle and J. Matoušek [CM92]. In higher dimensions ($d \geq 4$), for a long time the best known solution was the algorithm of R. Seidel [Sei86a] which after an $O(n^2)$-time preprocessing step , finds the facets of a convex hull in a shelling order at a logarithmic cost per facet. The preprocessing step was reduced later [Mat93, MS92] on to $O\left(n^{2-\frac{2}{\lfloor \frac{d}{2} \rfloor + 1}+\epsilon}\right)$ for any $\epsilon > 0$. Recently, T. Chan et al. [CSY95] have investigated the case of points in four dimensions, achieving an $O((n+h)\log^2 h)$-time algorithm for computing the convex hull of a set of n points where h denotes the output-size. In higher dimensions, T. Chan [Cha95b] realized many improvements on the convex hull computations and related problems, combining the gift-wrapping method of D.R. Chand and S.S. Kapur [CK70] and G.F. Swart [Swa85] with recent results on data structures for ray shooting queries in polytopes (developed by P.K. Agarwal et J. Matoušek [AM93] and refined by J. Matoušek and O. Schwarzkopf [MS93]).

There are numerous applications in which the convex hull of objects can be used effectively to make certain decision problems easier. For example, an empty intersection between the convex hull of two groups of objects implies an empty intersection between the two groups. Testing the intersection of two convex hulls is easier to handle than for general blue/red

groups [DS91]. Motion planning is also easier to study if the objects and obstacles are convex [BK87].

Computing the convex hull of a set of curved objects has been much less investigated. Computing the convex hull of a single planar object bounded by curves has been carefully studied [BK91, SV87, DS90] and several authors have generalized linear-time algorithms for computing the convex hull of a simple planar polygon [BEG84, MA79, SW86, GY83]. In the case of a family of n planar disks, optimal $\Theta(n \log n)$-time convex hull algorithms have been designed [Rap92, BCD$^+$92].

We consider the following problem: given a collection $\mathcal{O} = \{O_1, ..., O_n\}$ of n convex objects, compute in an *output-sensitive* manner the convex hull $\mathcal{CH}(\mathcal{O})$, i.e. the smallest convex object containing $\mathcal{O}$. In the general case, the usual way to compute the convex hull of $\mathcal{O}$ is to compute the lower and the upper envelopes of $\mathcal{O}$ and to consider the unique object bounded by these envelopes. Then, one can apply to this single planar object one of the convex hull algorithms mentioned above. A classical output-sensitive algorithm to compute the convex hull $\mathcal{CH}(\mathcal{O})$ is Jarvis's march [Jar73] which runs in $O(nh)$ where h denotes the output-size. In this paper, we generalize the marriage-before-conquest approach of R. Seidel and D.G Kirkpatrick [KS86] in the case of planar objects.

Independently, T. Chan [Cha95b] gave a simple algorithm for computing the convex hull of a set of planar points. His algorithm can be adapted to handle the case of convex objects (although this is not described in Ref. [Cha95b]) within the same time bounds. Nevertheless, our algorithm is different and is interesting in its own right. It relies on an $O(n \log h + \delta h)$-time algorithm to compute the convex hull of n objects of fixed type m such that any object can be colored with a value in $\{1, ..., \delta\}$ and objects of a same color do not intersect pairwise[1], where h denotes the output-size. Thus, we obtain immediately an optimal $\Theta(n \log h)$ algorithm if we consider that the objects satisfy the hard-sphere model [HO94] or have only a few intersections (in that case, our algorithm is simpler than T. Chan's one [Cha95b]). Moreover, we solve the problem of computing in linear time a bridge, i.e. a facet of the convex hull intersecting a given oriented line. In the general case, we first transform our original set of objects $\mathcal{O}$ into another set $\mathcal{T}$ such that $\mathcal{CH}(\mathcal{O}) = \mathcal{CH}(\mathcal{T})$ with the objects in $\mathcal{T}$ being colored with at most $\delta = \lceil \frac{n}{h} \rceil$ objects and apply our basic algorithm.

Computing the convex hull of general planar convex objects differs from the case of points because the convex hull of two points p_1 and p_2 is the straight segment $[p_1 p_2]$ whereas the complexity of the convex hull of two

1. In particular, if any object intersect at most γ others then $\delta \leq \gamma + 1$.

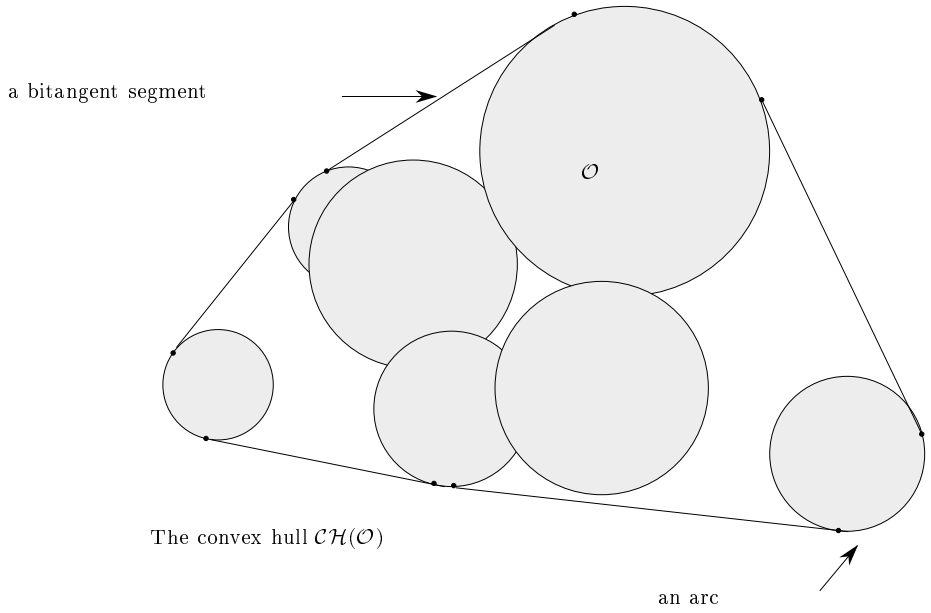


FIG. 2.1 – *A convex hull of disks ($m = 2$).*

planar convex objects O_i and O_j depends on the nature of these objects. We call *arc* a maximal piece of the boundary of $\mathcal{CH}(\mathcal{O})$ that is included in the boundary ∂O_i of an object O_i of $\mathcal{O}$. The boundary of $\mathcal{CH}(\mathcal{O})$ is an alternating sequence of arcs and *bitangent segments* (Figure 2.1). In the following, the arcs of $\mathcal{CH}(\mathcal{O})$ and its bitangent segments are called *facets*. In this paper, we shall consider sets of convex objects with the property that the convex hull of any two objects has bounded complexity (if the objects are non-convex and have fixed descriptive complexity, we can first compute their convex hulls in linear time). More precisely, a set of objects $\mathcal{O}$ is said to be of *type* m if the convex hull of any two objects of $\mathcal{O}$ has at most m arcs (or $2m$ facets). Let $|\mathcal{CH}(\mathcal{O})|$ denote the size of the convex hull $\mathcal{CH}(\mathcal{O})$, i.e. the number of facets (convex arcs and bitangent segments) of its boundary $\partial\mathcal{CH}(\mathcal{O})$. Then, $\mathcal{O}$ is of type m if $\forall\, i, j \in [1, n],\ |\mathcal{CH}(O_i, O_j)| \leq 2m$. For example, points have type 1, disks, convex homothets and non-overlapping objects have type 2, ellipsis have type 4, etc. Note that if $\mathcal{O}$ is of type m then the boundaries of any two convex objects of $\mathcal{O}$ cannot intersect in more than m points. Moreover, if q denotes the maximum number of intersection points between the boundaries of two distinct convex objects of $\mathcal{O}$, then $m = \max\{2, q\}$. Moreover, if the objects are bounded by closed convex curves then m is even.

Throughout this paper, we suppose that the type of set $\mathcal{O}$ is fixed. Moreover, each object in $\mathcal{O}$ is assumed to have a bounded descriptive size (for

instance, the boundary of each object is a curve of bounded degree) : in particular, this means that we can find in constant time the two supporting lines of an object with a given slope. Furthermore, we assume that the convex hull of two objects in $\mathcal{O}$ can be computed in constant time, where the constant depends on the type m.

This paper is organized as follows:

In section 2.2, we recall the complexity of the convex hull of n objects of type m.

In section 2.3, we first extend to the case of a set of convex objects of type m, the algorithm of D.G. Kirkpatrick and R. Seidel [KS82, KS86] to compute a *bridge*, i.e. the facet of the convex hull intersecting a given oriented line (subsection 2.3.1). Our algorithm is based on the searching-and-pruning paradigm and achieve an optimal $\Theta(n)$ time complexity to compute a bridge of a set of n convex objects of type m. Then, we present the scheme of the *marriage-before-conquest* approach (subsection 2.3.7). This scheme amounts to computing a bridge at a given oriented line, uses this bridge to filter the objects and to divide the problem into two independent sub-problems which are recursively solved. Finally, we refine the *marriage-before-conquest* algorithm in the case of a set partitionned into k subsets of non-overlapping objects, i.e. a set $\mathcal{O} = \cup_{i=1}^{k} \mathcal{P}_i$ where each $\mathcal{P}_i,\ i \in [1, k]$, is a collection of non-overlapping objects (subsection 2.3.8). The time complexity of the algorithm is $O(n \log h + hk)$. This algorithm is used as a basic primitive in the final algorithm. We also derive an $O(n \log h + \delta h)$-time algorithm to compute the convex hull of n objects of fixed type m where h denotes the output-size and δ is the maximal number of objects that an object can intersect.

In section 2.4, we describe the algorithm in the general case. We design an $O(n\beta(h, m) \log h)$-time convex hull algorithm where n is the number of objects, h denotes the output-size and $\beta(h, m)$ is a very slowly growing function related to the maximum length $\lambda(n, m)$ of a (n, m)-Davenport-Schinzel sequence [ASS89, Sha87, Sha88]. More precisely, $\beta(h, 2) = O(1)$ and $\beta(h, m) = O(2^{\alpha(h)^{c_m}})$ if $m > 2$, where c_m is an integer depending on m and $\alpha(\cdot)$ is the functional inverse of Ackermann's function. The algorithm is close to optimal with respect to both the input and output sizes since $\Omega(n \log h)$ is a lower bound [KS86].

In section 2.5, we adapt the method for computing upper envelopes of functions intersecting pairwise in at most m points and obtain an $O(n\beta(h, m+2) \log h)$-time algorithm. We improve slightly the algorithm in case of k-intersecting *generalized segments*, i.e. partially defined functions that intersect pairwise in at most k points. In that case, we obtain an $O(n\beta(h, k+$

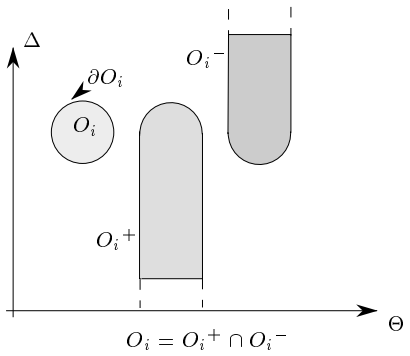


FIG. 2.2 – *Defining O_i^+ and O_i^-.*

$1)\log h$)-time algorithm which is $\Theta(n \log h)$ for line segments.

Finally, in section 2.6, we conclude and give several guidelines for future research.

2.2 Complexity of the Convex Hull of Convex Objects of Type m

In this section, we first examine the complexity of the convex hull $\mathcal{CH}(\mathcal{O})$ where $\mathcal{O}$ is a set of planar convex objects of type m.

Let us consider p^+ the point with coordinates $(0, +\infty)$ (resp. p^- the point with coordinates $(0, -\infty)$). Let us call *upper convex hull* (respectively *lower convex hull*) of $\mathcal{O}$ the convex hull $\mathcal{CH}^+(\mathcal{O}) = \mathcal{CH}(\mathcal{O}, p^-)$ (resp. $\mathcal{CH}^-(\mathcal{O}) = \mathcal{CH}(\mathcal{O}, p^+)$). We denote by O_i^+ (resp. O_i^-) the object $\mathcal{CH}^+(O_i, p^-)$ (resp. $\mathcal{CH}^-(O_i, p^+)$) (see Figure 2.2). Let $\mathcal{O}^+ = \{O_i^+ | O_i \in \mathcal{O}\}$ and $\mathcal{O}^- = \{O_i^- | O_i \in \mathcal{O}\}$. Then, $\mathcal{CH}^+(\mathcal{O}) = \mathcal{CH}(\mathcal{O}^+)$, $\mathcal{CH}^-(\mathcal{O}) = \mathcal{CH}(\mathcal{O}^-)$ and $\mathcal{CH}(\mathcal{O}) = \mathcal{CH}(\mathcal{O}^+) \cap \mathcal{CH}(\mathcal{O}^-)$.

We bound the complexity of the convex hull of convex objects of type m as follows:

Theorem 1 *In the worst-case, the complexity of the convex hull of n planar convex objects of type m is bounded by $4\lambda(n, m)$ where $\lambda(n, m)$ is the maximum length of an (n, m)-Davenport-Schinzel sequence [ASS89, Sha87, Sha88].*

Proof. Since the boundaries $\partial\mathcal{CH}(\mathcal{O}^+)$ and $\partial\mathcal{CH}(\mathcal{O}^-)$ of respectively $\mathcal{CH}(\mathcal{O}^+)$ and $\mathcal{CH}(\mathcal{O}^+)$ coincides at their extremities, the size $|\mathcal{CH}(\mathcal{O})|$ of the convex hull is at most $|\mathcal{CH}(\mathcal{O}^+)| + |\mathcal{CH}(\mathcal{O}^-)|$. We therefore focus on the upper bound

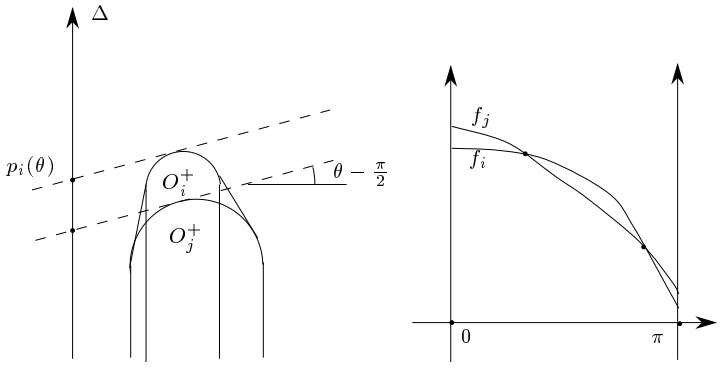


FIG. 2.3 – *Duality between the boundary of the upper convex hull* $\partial\mathcal{CH}^+(\mathcal{O})$ *and the upper envelope* $E_{\mathcal{F}}$ *of* $\mathcal{F}$.

of $|\mathcal{CH}(\mathcal{O}^+)|$. Since the convex hull $\mathcal{CH}(\mathcal{O}^+)$ is an alternating sequence of bitangent segments and arcs, we count the maximal number of arcs that can appear on the boundary of $\mathcal{CH}(\mathcal{O}^+)$, i.e. $\partial\mathcal{CH}(\mathcal{O}^+)$.

To each object O_i^+ of $\mathcal{O}^+$ we associate its *supporting* function $f_i(\cdot)$ defined as follows: $f_i(\theta)$ is defined over $[0,\pi]$ as the y-coordinate of the intersection point $p_i(\theta)$ of the y-axis Δ with the supporting line of O_i^+ with slope θ (see Figure 2.3). We get a collection $\mathcal{F} = \{f_i | 1 \leq i \leq n\}$ of n functions that are totally defined over the range $[0,\pi]$. As it is well known from duality, the upper convex hull $\mathcal{CH}(\mathcal{O}^+)$ is isomorph to the upper envelope $E_{\mathcal{F}}$ of $\mathcal{F}$, i.e. the pointwise maximum of the f_i's. An arc of $E_{\mathcal{F}}$ corresponds to an arc of $\partial\mathcal{CH}(\mathcal{O}^+)$, a vertex of $E_{\mathcal{F}}$ corresponds to a bitangent segment of $\partial\mathcal{CH}(\mathcal{O}^+)$.

Since $\mathcal{O}^+$ is of type m, two supporting functions f_i and f_j intersect in at most m points.

Therefore, the number of arcs of $\partial\mathcal{CH}(\mathcal{O}^+)$ is bounded by the maximal length $\lambda(n,m)$ of an (n,m)-Davenport-Schinzel sequence [ASS89, Sha87, Sha88]. Since the upper convex hull is an alternating sequence of arcs and bitangent segments, we get $|\mathcal{CH}(\mathcal{O}^+)| \leq 2\lambda(n,m)$. It follows that $|\mathcal{CH}(\mathcal{O})| \leq 4\lambda(n,m)$. □

Computing upper envelopes of real functions (defined over $\mathbb{R}$) that can mutually intersect in at most q points is a problem which has been extensively studied [SCK+86, EGS89, WS88]. A divide-and-conquer approach yields an $O(\lambda(n,q)\log n)$-time complexity algorithm. Therefore, one can compute the upper envelope of the 'dual' functions f_i's defined by objects in $\mathcal{O}^+$ in time $O(\lambda(n,m)\log n)$ if set $\mathcal{O}^+$ is of type m. Alternatively, the convex hull $\mathcal{CH}^+(\mathcal{O})$ can also be computed using the randomized incremental construc-

TAB. 2.1 – *Lower and upper bounds of $\lambda(n,m)$. $\alpha(n)$ is the functional inverse of Ackermann's function.*

$m \backslash \lambda(n,m)$	Lower Bound Ω	Upper Bound O
1	n	n
2	$2n-1$	$2n-1$
3	$\Omega(n \times \alpha(n))$	$O(n \times \alpha(n))$
4	$\Omega(n \times 2^{\alpha(n)})$	$O(n \times 2^{\alpha(n)})$
$m = 2s+1$	$\Omega(n \times 2^{O(\alpha(n)^{s-1})})$	$O(n \times \alpha(n)^{O(\alpha(n)^{s-1})})$
$m = 2s+2$	$\Omega(n \times 2^{O(\alpha(n)^{s})})$	$O(n \times 2^{O(\alpha(n)^{s})})$

tion of Clarkson [Cla88a] and Clarkson and Shor [CS88] in expected running time $\tilde{O}(\lambda(n,m)\log n)$.

2.3 Computing the Convex Hull of Colored Families of Convex Objects

In this section, we first show in subsection 2.3.1 how to compute in linear time a bridge, i.e. a facet of the convex hull intersected by a given ray. Then, we present in subsection 2.3.7 the marriage-before-conquest paradigm applied to the case of objects. Finally, we refine the algorithm in subsection 2.3.8 in the case of colored families of convex objects, i.e. families that can be partitionned into monochromatic subsets of pairwise non-intersecting objects.

2.3.1 Bridge of a Convex Hull

2.3.2 Definition and notations

The bridge of $\mathcal{O}$ at Δ is the unique facet of $\mathcal{CH}^+(\mathcal{O})$ that is intersected by Δ. The *bridge* at Δ is either an arc or a bitangent segment of $\mathcal{CH}(\mathcal{O})$. This section is devoted to the computation of the bridge at an oriented line Δ of a set of planar convex objects of fixed type m. The bridge facet is easily determined if one knows the line which supports $\mathcal{CH}(\mathcal{O})$ at the point $\Delta \cap \partial\mathcal{CH}(\mathcal{O})$ where Δ intersects the boundary of $\mathcal{CH}(\mathcal{O})$. Indeed, if this line is a supporting line for at least two objects in $\mathcal{O}$ then the bridge is a bitangent

segment whereas if this line is a supporting line of a single object $O_i \in \mathcal{O}$ then the bridge is an arc of $\mathcal{CH}(\mathcal{O})$ included in ∂O_i. In both cases, the two endpoints of the bridge can be found in linear time once this supporting line is known. Thus, we focus on the determination of the supporting line of $\mathcal{CH}(\mathcal{O})$ at point $\Delta \cap \partial\mathcal{CH}(\mathcal{O})$. Hereafter, this line is called the *supporting line* of the bridge at Δ.

Computing the supporting line of the bridge at Δ of n convex objects is a generalized linear program [SW92, MSW92, Ame93, Ame94] and can therefore be computed by a randomized algorithm in expected $\tilde{O}(n)$ time. Moreover, we can use the derandomized algorithm of B. Chazelle and J. Matoušek [CM93] in order to obtain a linear deterministic algorithm. Hereafter we give a more direct algorithm to compute in linear time the bridge at Δ. D.G. Kirkpatrick and R. Seidel [KS86] gave a deterministic optimal $\Theta(n)$ algorithm that computes a bridge for a set of n points using a searching-and-pruning procedure. We extend this algorithm to convex objects of fixed type m.

In order to follow the steps of this searching-and-pruning method, we first extend the main theorem of D.G. Kirkpatrick and R. Seidel [KS86]'s algorithm for computing the bridge of points to the case of convex objects that can be separated by a line parallel to Δ. Then, we introduce the *vertical decomposition* in order to obtain convenient sets of convex objects. We finally give the overall algorithm and analyze its time complexity.

2.3.3 The case of convex objects

Without loss of generality, consider that the direction of Δ is the direction of the y-axis, called the *vertical* axis. We denote by $x(p)$ the abscissa of point p. Kirkpatrick and Seidel proved the following lemma for a set $\mathcal{O}$ of points:

Lemma 2 (3.2, pp. 291 Ref. [KS86]) *Let p, q be a pair of points of $\mathcal{O}$ with $x(p) < x(q)$, let s_b be the slope of the supporting line of the bridge of $\mathcal{O}$ at Δ and let s be the slope of the straight line through p and q. If $s > s_b$ then p cannot be a point of the bridge of $\mathcal{CH}(\mathcal{O})$ at Δ. If $s < s_b$ then q cannot be a point of the bridge at Δ.*

Two objects O_1 and O_2 are said to be *x-separated* if they can be separated by a line parallel to Δ. Note that x-separated objects can be ordered along the x-axis. In the following, we note $x(O)$ the x-range of an object O, i.e. the projection of O onto the x-axis. Let (O_1, O_2) be a pair of x-separated objects. If O_1 is to the left of an oriented vertical line separating O_1 and O_2 then we note $x(O_1) < x(O_2)$ and O_1 (resp. O_2) is called the left (resp. right) object

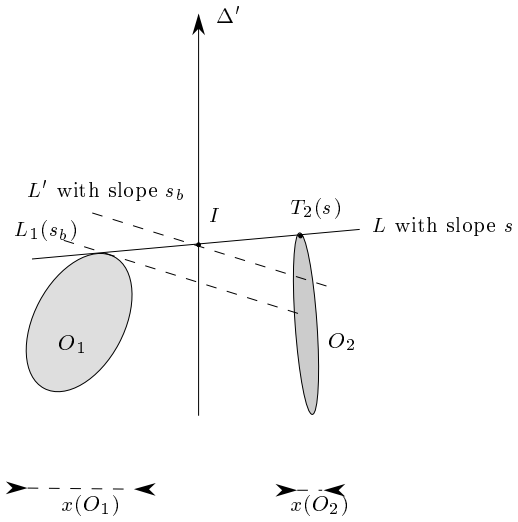


FIG. 2.4 – *Discarding O_1 when $s > s_b$.*

of the pair (O_1, O_2). An object $O \in \mathcal{O}$ is said to participate to the bridge at Δ if the supporting line of $\mathcal{CH}^+(\mathcal{O})$ at $\Delta \cap \partial\mathcal{CH}^+(\mathcal{O})$ is a supporting line of object O. We extend lemma 2 to the case of x-separated objects. Observe that if O_1 and O_2 are a pair of x-separated object, the boundary of the upper convex hull $\mathcal{CH}^+(O_1, O_2)$ has a unique bitangent segment.

Lemma 3 *Let (O_1, O_2) be a pair of x-separated objects with $x(O_1) < x(O_2)$, let s be the slope of the unique bitangent segment of $\partial\mathcal{CH}^+(O_1, O_2)$ and let s_b be the slope of the bridge of $\mathcal{O}$ at Δ. If $s > s_b$ then the left object O_1 of the pair cannot participate to the bridge at Δ. If $s < s_b$ then the right object O_2 of the pair cannot participate to the bridge at Δ.*

Proof. We only give the proof in case of $s > s_b$ (the other case is obtained by symmetric considerations). Let I be the intersection point between a separating line Δ' parallel to Δ and the affine hull L of the unique bitangent segment of $\partial\mathcal{CH}^+(O_1, O_2)$ (see Figure 2.4). Let s be the slope of L and define L' as the line passing through I with slope s_b. Let $L_1(s_b)$ be the tangent line to O_1 with slope s_b. $L_1(s_b)$ and L' are parallel lines (and can therefore be ordered along Δ'). Because of the convexity of O_1, if $s > s_b$ then $y(L' \cap \Delta') = y(L \cap \Delta') \geq y(L_1(s_b) \cap \Delta')$ and $L_1(s_b)$ is below L'. But the contact point $T_2(s) = L \cap O_2$ is strictly above L' if $s > s_b$. Therefore, point $T_2(s)$ is above $L_1(s_b)$ so that O_1 cannot participate to the bridge at Δ $\square$

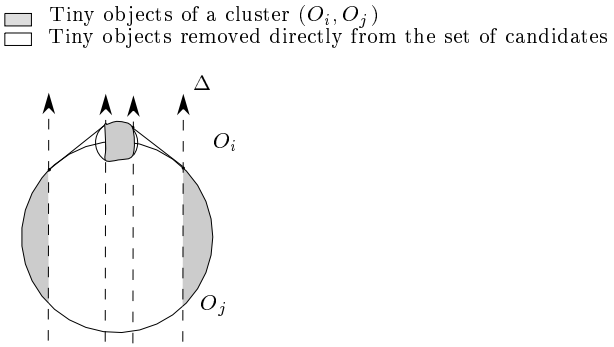


FIG. 2.5 – *Vertical decomposition of a pair of convex objects of type* 2.

2.3.4 Vertical decomposition

The vertical decomposition will give rise to a set of x-separated convex objects.

Let $\mathcal{O} = \{O_1, ..., O_n\}$ be a set of n planar convex objects of type m and $\mathcal{CH}^+(\mathcal{O})$ its upper convex hull. We decompose this upper convex hull by striping $\mathcal{CH}^+(\mathcal{O})$. To stripe $\mathcal{CH}^+(\mathcal{O})$, we draw through each vertex of $\partial\mathcal{CH}^+(\mathcal{O})$ a line parallel to Δ. These parallel lines induce a decomposition of each object O_i of $\mathcal{O}$ into sub-objects, called *tiny* objects in the following (see Figure 2.5). We only keep the tiny objects whose boundary participates to the boundary of the convex hull $\partial\mathcal{O}$. Note that each tiny object is defined from a single object and two vertical lines, and that two tiny objects arising from this decomposition are x-separated.

2.3.5 Algorithm

Let L be the line which supports $\mathcal{CH}(\mathcal{O})$ at point $\Delta \cap \partial\mathcal{CH}^+(\mathcal{O})$. Line L is a supporting line for some of the objects in $\mathcal{O}$. Our goal is to select from $\mathcal{O}$ the objects that touch L.

We pair up the convex objects of $\mathcal{O}$ into pairs (O_i, O_j). For each pair (O_i, O_j), we first compute $\mathcal{CH}^+(O_i, O_j)$, the upper convex hull of O_i and O_j and apply the vertical decomposition to $\mathcal{CH}^+(O_i, O_j)$. We discard from this decomposition all the tiny objects that do not have an arc of $\partial\mathcal{CH}^+(O_i, O_j)$ on their boundaries. Indeed, as these tiny objects do not appear on the boundary of $\mathcal{CH}^+(O_i, O_j)$, they also do not appear on the boundary of $\mathcal{CH}^+(\mathcal{O})$ and therefore cannot participate to any bridge of $\mathcal{CH}^+(\mathcal{O})$ (Figure 1).

We call *cluster* the set of remaining tiny objects of a pair. As each pair is of type at most m since $\mathcal{O}$ is of type m, we can deduce that there are at most $(m+1)$ convex tiny objects in a cluster. Then, we pair up all the tiny objects

within a cluster into at most $\lfloor \frac{m+1}{2} \rfloor$ tiny pairs. Note that we pair up only the tiny objects within a cluster since, as they are x-separated, they have type 2 whereas two tiny objects of different clusters have type m. In the following, we shall use lemma 3 to reduce the number of tiny objects in the clusters. As the slope s_b of the supporting line of $\mathcal{CH}(\mathcal{O})$ at $\Delta \cap \partial\mathcal{CH}^+(\mathcal{O})$ is of course unknown, the trick is to resolve tests like $s < s_b$ or $s > s_b$ using transitivity. A cluster is said to be *reduced* if it has only one remaining tiny object. The algorithm consists in an *initial step* where we pair up the objects in order to get non-reduced clusters and several rounds of *selecting* and *clustering* where we eliminate, round after round, the tiny objects. We describe the algorithm below:

Initial step. We pair up the objects and compute for each pair its vertical decomposition. This step gives rise to clusters of tiny objects. If an object O_i of a pair (O_i, O_j) is included in O_j then the cluster generated by this pair is reduced. In that case, we discard O_i and pair O_j again until all clusters are non-reduced.

Selecting and Clustering. A round:

- **Selecting.**
  - Pair up the tiny objects inside each cluster.
  - Compute the median s_m of the slopes of the bitangent segments of pairs of tiny objects (use the median algorithm of Blum et al. [BFP+72]).
  - Let $\mathcal{O}'$ be the subset of objects which contribute to the current collection of tiny objects. Then, find the supporting line $L(s_m)$ of $\mathcal{CH}^+(\mathcal{O}')$ with slope s_m and locate the contact points $L(s_m) \cap \mathcal{CH}^+(\mathcal{O}')$ with respect to Δ. To find the supporting line of $\mathcal{CH}^+(\mathcal{O}')$ with a given slope s_m, we find the object(s) which maximize
    $\max_{i=1..|\mathcal{O}'|}\{y_i - s_m x_i\}$ where point (x_i, y_i) is the contact point of the supporting line of O_i with slope s_m. In general, there is one or two such objects and therefore one or two contact points but there can be possibly more.
  - Discard tiny objects:
    - If there are contact points located in both sides of Δ then $s_m = s_b$. The supporting line of $\mathcal{CH}(\mathcal{O})$ at $\Delta \cap \partial\mathcal{CH}^+(\mathcal{O})$ is fully determined by a tiny pair whose bitangent segment has slope s_m.
    - If all the contact points are located at the left side of Δ then

$s_m > s_b$. We consider all tiny pairs with slope $s \geq s_m$ and discard the left tiny object t_1 of these pairs.

- If all the contact points are located at the right side of Δ then $s_m < s_b$. We consider all tiny pairs with slope $s \leq s_m$ and discard the right tiny object t_2 of these pairs.

– **Clustering.** This stage is required in order to obtain a set of non-reduced clusters for the next round. For each reduced cluster, we consider the original object $O \in \mathcal{O}$ which gave rise to the single tiny object of this cluster. We pair up these objects and compute for each pair its vertical decomposition. This step gives rise to new clusters. If an object O_i of a pair (O_i, O_j) is included in O_j then the cluster generated by this pair is reduced. In that case, we discard O_i and pair O_j again until all clusters are non-reduced. In the next round, we consider these new clusters together with the non-reduced clusters remaining from the last selecting step.

The algorithm halts whenever it finds a tiny pair whose slope equals the slope s_b of the supporting line of the bridge at Δ or if it remains only one tiny object. In the former case, the bridge is a bitangent segment and we find its two endpoints in linear time. In the latter case, there are two subcases: either the remaining tiny object does not intersect Δ and $\mathcal{CH}(\mathcal{O}) \cap \Delta = \emptyset$ or it defines the object whose boundary contains the bridge arc. In the latter subcase, the endpoints of the arc can be found in linear time.

2.3.6 Complexity analysis

Theorem 4 *The above algorithm computes the bridge of a set of n planar convex objects of fixed type m in optimal $\Theta(n)$ time and storage.*

Proof. Once we know the supporting line of the bridge at Δ, we can determine, in linear time, the nature of the bridge (arc or segment) and compute its two endpoints in linear time for a fixed type m. We therefore focus on the analysis of the searching-and-pruning algorithm.

Let l and k be respectively the number of tiny objects and the number of clusters (they are all non-reduced) present at the beginning of some round of the selecting and clustering steps. Then, we denote by $c(l, k)$ the cost of the algorithm from that stage. Let l' and k' be respectively the number of tiny objects and the number of non-reduced clusters at the end of that round, i.e. after the clustering step. We have the following recursive equation:

$$c(l,k) = \begin{cases} O(1) & \text{if } k \leq 1 \\ \alpha l + \beta(k - k') + c(l', k') & \text{otherwise,} \end{cases} \tag{2.1}$$

where α and $\beta = \beta_m$ are some constants (β depending on m).

Let r denote the number of clusters reduced during the *selecting* phase. Since we pair up the r reduced clusters to create new non-reduced clusters, we have $k' \leq k - r + \lfloor \frac{r}{2} \rfloor$. In the second part of equation (2.1), αl is the cost of the *selecting* phase, $\beta \frac{r}{2} \leq \beta(k - k')$ the cost of the *clustering* phase of the round and $c(l', k')$ the total cost of the remaining rounds. Each vertical decomposition of the convex hull of two objects costs $\beta_m = \beta$ if type m is fixed.

If $k = 1$ there is only one cluster of tiny objects. We can compute the convex hull of the at most two objects which give rise to the set of its tiny objects in time $O(1) = \gamma_m = \gamma$ if type m is fixed.

Let $\mathcal{S}_1$ ($|\mathcal{S}_1| = l'_1$) be the set of remaining tiny objects after the selecting phase of the current round and $\mathcal{S}_2$ ($|\mathcal{S}_2| = l'_2$) the set of tiny objects created during the clustering phase. Let $\mathcal{S}'$ ($|\mathcal{S}'| = l'|$) be the set of tiny objects at the beginning of the next round, i.e. $\mathcal{S}' \subseteq \mathcal{S}_1 \cup \mathcal{S}_2$. Clearly, we have $l' \leq l'_1 + l'_2$.

We prove that $l'_1 \leq \frac{5}{6} l$:

Assume that among the k clusters present at the beginning of the current round, k_o clusters have an odd number of tiny objects (say the first k_o clusters) and thus remain with an unpaired tiny object after the pairing of tiny objects while the $(k - k_o)$ other clusters have all their tiny objects paired. Finally, denote by a_i the number of pairs of tiny objects in the i-th cluster. We have the following equation:

$$l = \sum_{i=1}^{k_o} (2a_i + 1) + \sum_{i=k_o+1}^{k} 2a_i = (\sum_{i=1}^{k} 2a_i) + k_o \tag{2.2}$$

The *selecting* process removes a tiny object from half of the tiny pairs, so that $l'_1 \leq l - \frac{1}{2} \sum_{i=1}^{k} a_i$. Using equation (2.2), we obtain $l'_1 \leq \frac{3}{4} l + \frac{k_o}{4}$. As the number of tiny objects l is at least $2k + k_o$ and k_o ranges over $[0, k]$, we have $l \geq 3k_o$. Thus, $l'_1 \leq \frac{5l}{6}$.

Now, consider the number of created tiny objects during the *clustering* step. Clearly, $l'_2 \leq (k - k')(m + 1)$. l' is therefore upper-bounded by $\frac{5}{6} l + (k - k')(m + 1)$.

Then, it follows by induction on vector (k, l) ordered lexicographically that $c(l, k) \leq \delta l + (\beta + \delta(m + 1))k + \gamma$ for any $\delta \geq 6\alpha$.

Initially, $l \leq \lceil \frac{n}{2} \rceil (m + 1)$ and $k \leq \lceil \frac{n}{2} \rceil$ so that the complexity of all the rounds of the selecting and clustering step is upper bounded by $O(n)$ for any fixed type m.

The cost of the initial step is also $O(n)$. Thus we obtain an $\Theta(n)$-time algorithm to compute the bridge $\square$

2.3.7 Marriage-Before-Conquest Algorithm

In this section, we present the marriage-before-conquest strategy to compute the convex hull $\mathcal{CH}(\mathcal{O})$ of a set of n convex objects $\mathcal{O}$. We consider w.l.o.g. the computation of the upper convex hull since the boundary of $\mathcal{CH}(\mathcal{O})$ is obtained in $O(1)$ time from the boundaries of $\mathcal{CH}^+(\mathcal{O})$ and $\mathcal{CH}^-(\mathcal{O})$. Each object in $\mathcal{O}$ has two supporting lines parallel to the y-axis, called *walls*. Each wall is oriented as the y-axis. Let $\mathcal{W}$ be the set of walls and denote by $|\mathcal{W}| = w = 2n$ its cardinality. Let $\mathcal{R}$ be a range, i.e. an interval on the x-axis. We define a *slab* as the portion of the euclidean plane $\mathbb{E}^2$ between two lines parallel to Δ. The upper convex hull $\mathcal{CH}^+(\mathcal{O})$ can be described as an x-ordered sequence of facets. The following algorithm MarriageBeforeConquest$(\mathcal{W}, \mathcal{O}, \mathcal{R})$ computes a subsequence $\mathcal{MBC}(\mathcal{W}, \mathcal{O}, \mathcal{R})$ of the facets of $\mathcal{CH}^+(\mathcal{O})$ included in the slab $\mathcal{B} = \mathcal{R} \times (-\infty, +\infty)$.

Termination. If $w = 0$ then $\mathcal{MBC}(\mathcal{W}, \mathcal{O}, \mathcal{R}) = \emptyset$. Return $\mathcal{MBC}(\mathcal{W}, \mathcal{O}, \mathcal{R})$.

Divide. Find the median W_m of the walls $\mathcal{W}$. Split $\mathcal{W}$ into two balanced subsets $\mathcal{W}'_1 = \{W \in \mathcal{W} | x(W) \leq x(W_m)\}$ and $\mathcal{W}'_2 = \{W \in \mathcal{W} | x(W) \geq x(W_m)\}$.

Merge. Compute the bridge b at the median oriented line W_m.

Filter. Let $\mathcal{W}_1$ (respectively $\mathcal{W}_2$) be the subset of the walls of $\mathcal{W}'_1$ (resp. $\mathcal{W}'_2$) that do not intersect b. Let w_1 and w_2 denote respectively the cardinalities of sets $\mathcal{W}_1$ and $\mathcal{W}_2$. Let x_b^+ and x_b^- be respectively the abscissæ of the right and left endpoints of b. Let $\mathcal{R}_1 = \mathcal{R} \cap (-\infty, x_b^-)$, $\mathcal{B}_1 = \mathcal{R}_1 \times (-\infty, +\infty)$, $\mathcal{R}_2 = \mathcal{R} \cap (x_b^+, +\infty)$ and $\mathcal{B}_2 = \mathcal{R}_2 \times (-\infty, +\infty)$. Let $\mathcal{O}_1$ (resp. $\mathcal{O}_2$) be the set of objects in $\mathcal{O}$ that intersect slab $\mathcal{B}_1$ (resp. slab $\mathcal{B}_2$). Compute $\mathcal{W}_1$, $\mathcal{W}_2$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{O}_1$ and $\mathcal{O}_2$. Let $n_1 = |\mathcal{O}_1|$ and $n_2 = |\mathcal{O}_2|$.

Conquest. Call recursively MarriageBeforeConquest$(\mathcal{W}_1, \mathcal{O}_1, \mathcal{R}_1)$ and MarriageBeforeConquest$(\mathcal{W}_2, \mathcal{O}_2, \mathcal{R}_2)$ and return the ordered sequence of facets $\mathcal{MBC}(\mathcal{W}_1, \mathcal{O}_1, \mathcal{R}_1) \cup \{b\} \cup \mathcal{MBC}(\mathcal{W}_2, \mathcal{O}_2, \mathcal{R}_2)$.

We denote by $c(n, w, h)$ the complexity of the algorithm MarriageBeforeConquest running inside range $\mathcal{R}$ if there are w walls in $\mathcal{B}$, n objects intersecting $\mathcal{B}$ and h computed facets of $\mathcal{CH}^+(\mathcal{O})$ in $\mathcal{B}$. Each computed facet is intersected by at least one wall of $\mathcal{W}$, so that $h \leq |\mathcal{W}|$. We obtain the following equation:

$$c(n, w, h) = \begin{cases} O(n) & \text{if } h \leq 1 \\ c(n_1, w_1, h_1) + c(n_2, w_2, h_2) + O(n) & \text{otherwise} \end{cases} \tag{2.3}$$

The algorithm ensures that $w_1 + w_2 \leq w$ and $w_1, w_2 \leq \lceil \frac{w}{2} \rceil$ but it does not control n_1 nor n_2 ($n_1, n_2 \leq n$) so that its worst-case running time is $O(nh)$. At the end, we are left with an x-ordered alternating sequence of computed facets and *empty* slabs (i.e. slabs that do not contain any wall of $\mathcal{W}$). We can find the whole upper convex hull using Jarvis's algorithm inside each empty terminal slab. In the following section, we study a special case where we can bound the number of objects that participate to the upper convex hull inside a slab (parameter n_1 and n_2 of equation (2.3)). We will use this "special" case as a basic primitive in the final algorithm.

2.3.8 The Case of a Non-Overlapping Partition

Let $\mathcal{O}$ be a set of n objects of fixed type m. If we know that a partition $\uplus_{i=1}^{k} \mathcal{P}_i$ of set $\mathcal{O}$ of fixed type m into k subsets such that each subset $\mathcal{P}_i$, for $i \in [1, k]$, is a set of non-overlapping convex objects then we can derive an $O(n \log h + hk)$-time complexity algorithm to find the convex hull of $\mathcal{O}$. This result holds, for example if $\mathcal{O}$ is a set of non-overlapping convex objects, since in that case $k = 1$ and $m = 2$. Let $\mathcal{B}$ be a vertical slab where we want to compute the upper convex hull. Among the objects of $\mathcal{O}$ intersecting $\mathcal{B}$, we distinguish two mutually exclusive categories:

Category 1: The objects that have a wall inside $\mathcal{B}$.

Category 2: The objects that intersect $\mathcal{B}$ but do not have a wall inside $\mathcal{B}$: these objects are called the *spanning* objects hereafter.

Algorithm MarriageBeforeConquest is slightly modified, taking into account these two categories of objects inside each slab $\mathcal{B}$ (with associated range $\mathcal{R}$), as follows:

- We bound the number of objects to consider in slab $\mathcal{B}$ by selecting among the spanning objects, at most one object of each family $\mathcal{P}_i$. Indeed, $\mathcal{R}$ is included in the x-range of each spanning object. Thus, the spanning objects which belong to a given family $\mathcal{P}_i$ can be ordered along any vertical line included in $\mathcal{B}$ and only the topmost object can contribute to the upper convex hull in $\mathcal{B}$.
- We stop the recursive calls as soon as $w \leq k$ and run Jarvis's march in each resulting slab on the set of objects $\mathcal{O}_{\mathcal{B}}$ relevant for this slab. We have $|\mathcal{O}_{\mathcal{B}}| \leq 2k$ since there are at most k spanning objects and k objects of category 1. This Jarvis's march is initialized from the computed facet which intersects the rightmost vertical line limiting $\mathcal{B}$ and stopped when the leftmost vertical line limiting $\mathcal{B}$ is reached.

Theorem 5 *Let $\mathcal{O}$ be a set of n planar convex objects of fixed type m partitionned into k subsets of non-overlapping convex objects, then the convex hull of $\mathcal{O}$ can be computed in $O(n \log h + hk)$ time, where h is the size of the convex hull of $\mathcal{O}$.*

Proof. Let $c(n, w, h)$ denote the complexity of the above algorithm. We have:

$$c(n, w, h) = \begin{cases} O(hk) & \text{if } w \leq k \\ c(n_1, w_1, h_1) + c(n_2, w_2, h_2) + O(n) & \text{otherwise} \end{cases} \tag{2.4}$$

with $w_1 + w_2 \leq w$, $w_1, w_2 \leq \lceil \frac{w}{2} \rceil$, $n_1 \leq w_1 + k$ and $n_2 \leq w_2 + k$ since we keep, in each sub-slab $\mathcal{B}_1, \mathcal{B}_2$, at most k spanning objects and there are at most w_1 objects (resp. w_2 objects) that have a wall in slab $\mathcal{B}_1$ (resp. $\mathcal{B}_2$).

We consider the recursive time complexity equation (2.4) and link parameters n and w using the inequality: $n \leq w+k$; thus $c(n, w, h) \leq c(w+k, w, h)$ and from now on, we simply note $c(w, h)$ for $c(w + k, w, h)$. Bounding n by $w + k$ in equation (2.4), we obtain:

$$c(w, h) = \begin{cases} \alpha hk & \text{if } w \leq k \\ c(w_1, h_1) + c(w_2, h_2) + \beta(w + k) & \text{otherwise} \end{cases} \tag{2.5}$$

where α and β are some constants.

Note that $w_1 \leq \lceil \frac{w}{2} \rceil$, $w_2 \leq \lceil \frac{w}{2} \rceil$ and $h = h_1 + h_2 + 1$. We prove by induction on w that $c(w, h) \leq \gamma(w \log h + kh)$ for a suitable constant γ:

- If $w \leq k$ then $c(w, h) = \alpha kh$ by equation (2.5). So that $c(w, h) \leq \gamma(w \log h + kh)$ if $\gamma \geq \alpha$.
- Suppose that $c(w', h) = \gamma(w' \log h + kh)$ for $0 \leq w' < w$ and consider $c(w, h)$ with $w > k$. Using equation (2.5), it follows that:

  $$c(w, h) = \gamma(w_1 \log h_1 + kh_1 + w_2 \log h_2 + kh_2) + \beta(w + k)$$

  with $w_1, w_2 \leq \frac{w}{2}$ and $h_1 + h_2 + 1 = h$. Note that $\log(h_1 h_2)$ is maximized for $h_1 = h_2 = \frac{h-1}{2}$, thus:

  $$c(w, h) \leq \gamma(\frac{w}{2} \log \frac{h^2}{4} + kh) + \beta(w + k),$$

  $$c(w, h) \leq \gamma(w \log h + kh - w) + \beta(w + k).$$

  But $k < w$ by hypothesis, so that

  $$c(w, h) \leq \gamma(w \log h + kh) + (2\beta - \gamma)w,$$

  and $c(w, h) \leq \gamma(w \log h + kh)$ for suitable $\gamma \geq 2\beta$.

This proves that $c(w,h) \leq \gamma(w \log h + kh)$ for constant $\gamma = \max\{2\beta, \alpha\}$. Initially, $w = 2n$ (each of the n initial objects has two walls) so that the complexity of the algorithm is $O(n \log h + kh)$. □

As a direct consequence, we obtain a $\Theta(n \log h)$-time algorithm for computing the convex hull of non-overlapping convex objects. Note that our algorithm requires to know the partition of $\mathcal{O}$ into subsets of non-overlapping objects. We can define for a family of n objects its *intersection graph* $\mathcal{G}$ as follows: for each object $O_i \in \mathcal{O}$ we create a node and two different nodes are linked iff their corresponding objects intersect. If δ is the maximum degree of the nodes of $\mathcal{G}$, we know from the graph theory that there exists a partition of $\mathcal{O}$ into p subsets of non-overlapping objects such that $p \leq \delta + 1$. We can slightly modify our algorithm in order to take into account the paramater δ without knowing a partition into subsets of non-overlapping objects: choose a vertical line inside the slab and select from the spanning objects the object O that has the uppermost intersection point with that line. Then, we discard all the spanning objects that do not intersect O (this means that we only keep the spanning objects intersecting O). It is trivial to prove that all the spanning objects that do not intersect O are below O and therefore cannot participate to the upper convex hull. Thus, we obtain an $O(n \log h + \delta h)$-time algorithm to compute the upper convex hull of n objects of fixed type m where δ is the maximal number of intersection of any object with the others. For example, we can compute the convex hull of n hard-disks [HO94] in $\Theta(n \log h)$ (a family of disks in the hard-sphere model has the property that each disk intersects at most $O(1)$ others, i.e. $\delta = O(1)$). We also obtain an optimal $\Theta(n \log h)$-time algorithm if $\delta \leq O(\frac{n \log n}{\lambda(n,m)})$.

Note that the above algorithm computes the upper convex hull inside each terminal slab using Jarvis's march. If we skip this last phase of the algorithm, we are left with a subsequence of the facets of the convex hull. There is a terminal slab intersecting at most $2k$ objects between each pair of consecutive facets in the subsequence. Then, the algorithm is called PartialMBC and its complexity is still $O(n \log h + kh)$ but h is, here, the number of computed bridges (and not the total number of facets of the upper convex hull).

2.4 The General Case

In this section, we first present a convex hull algorithm assuming we know a good estimate h_e of the output-size h. To obtain a good estimate of the output-size, we have to compare the size h of the convex hull with some given

value p; we show in section 2.4.2 how to perform such comparisons. The final algorithm is given in section 2.4.3.

2.4.1 Given an Estimate of the Output-Size

Let h_e be an estimate of the output-size $h = |\mathcal{CH}^+(\mathcal{O})|$. The algorithm includes two steps: the first step computes from $\mathcal{O}$ a set $\mathcal{T}$ of objects partitionned into non-overlapping subsets such that $\mathcal{CH}^+(\mathcal{O}) = \mathcal{CH}^+(\mathcal{T})$. Then, in a second step, we apply the marriage-before-conquest algorithm of section 2.3.8 on $\mathcal{T}$. We describe the algorithm below:

Grouping. Group the n objects into $\lceil \frac{n}{h_e} \rceil$ groups of size h_e. For each group, we compute the vertical decomposition of the convex hull of its objects. Thus, we obtain from the groups a set $\mathcal{T}$ of $O(\lceil \frac{n}{h_e} \rceil \lambda(h_e, m))$ tiny objects partitionned into $\lceil \frac{n}{h_e} \rceil$ subsets of at most $\lambda(h_e, m)$ non-overlapping tiny objects.

Marriage-before-conquest. Let $\mathcal{W}$ be the set of walls corresponding to the tiny objects of $\mathcal{T}$. Let $\mathcal{R}$ be the x-range $(-\infty, +\infty)$.
Return MarriageBeforeConquest$(\mathcal{W}, \mathcal{T}, \mathcal{R})$ (see algorithm section 2.5).

Let us now analyze the complexity of the two steps:

Grouping. Computing the vertical decomposition of the upper convex hull of a group of h_e objects requires $O(\lambda(h_e, m) \log h_e)$ time: we first compute the upper envelope of the h_e objects by a divide-and-conquer algorithm and then run a walk-like convex hull algorithm on the resulting upper envelope [BK91]. The upper envelope has worst-case size $\lambda(h_e, m)$. Thus, the time required to compute the vertical decomposition of a group is $O(\lambda(h_e, m) \log h_e)$. Since there are $\lceil \frac{n}{h_e} \rceil$ groups, the total time complexity of this first step is $O(n \frac{\lambda(h_e,m)}{h_e} \log h_e)$.

Marriage-before-conquest. We run the marriage-before-conquest algorithm of section 2.3.8 onto the set of $O(\frac{n\lambda(h_e,m)}{h_e})$ tiny objects partitionned into $\lceil \frac{n}{h_e} \rceil$ subsets of non-overlapping objects. From the complexity analysis of section 2.3.8, this step requires $O(n \frac{\lambda(h_e,m)}{h_e} \log h + \frac{nh}{h_e})$ time.

The total time complexity of the algorithm is therefore $O\left(n \frac{\lambda(h_e,m)}{h_e} (\log h_e + \log h) + \frac{nh}{h_e}\right)$. Thus, if $h_e = h$ then the time required to compute the convex hull $\mathcal{CH}^+(\mathcal{O})$ is $O(n \frac{\lambda(h,m)}{h} \log h)$.

2.4.2 Comparing the Output-Size with a Given Value

In order to find a good estimate of h, we will need to determine if our current estimate (say p) is good (this means that p roughly equals to h) or not, i.e. to answer tests like $p > h$, $p = h$ or $p < h$.

Lemma 6 *There exists a deterministic algorithm that given an integer p, answers whether $h > p$ or not in $O(n\frac{\lambda(p,m)}{p}\log p)$ time.*

Proof. We design an algorithm which does not differ too much from the algorithm of section 2.4.1: the idea is to group the objects into $\lceil\frac{n}{p}\rceil$ groups of p objects and then run the marriage-before-conquest algorithm PartialMBC on the $\lceil\frac{n}{p}\rceil$ subsets of non-overlapping tiny objects resulting from the vertical decomposition of each group. Finally, we bound the number of facets computed by Jarvis's marches inside the terminal slabs. More precisely, we run Jarvis's march inside each "terminal" slab (a slab with at most $\frac{n}{p}$ walls) until we have computed a total of $\min\{p, h\}$ facets. We describe the algorithm below:

Let $a = 0$ (a denotes the number of computed facets).

Grouping. Group the n objects into $\lceil\frac{n}{p}\rceil$ groups of size p and compute the vertical decompositions of their convex hull. We obtain a set $\mathcal{T}$ of $O(\frac{n\lambda(p,m)}{p})$ tiny objects partitionned into $\lceil\frac{n}{p}\rceil$ non-overlapping subsets.

Marriage-before-conquest. Apply algorithm PartialMBC on the set $\mathcal{T}$ until each slab has less than $\lceil\frac{n}{p}\rceil$ walls, incrementing a each time we compute a bridge. If $a > p$ stop and return YES, i.e. $h > p$.

Jarvis's march. Fill the terminal slabs by running Jarvis's march inside each slab on a set of $O(\frac{n}{p})$ objects (at most $\lceil\frac{n}{p}\rceil$ spanning objects and $\lceil\frac{n}{p}\rceil$ objects that have a wall inside the slab), incrementing a and testing if $a > p$ each time we compute a new facet. If $a > p$ at some step then we stop the algorithm and return YES, i.e. $h > p$.

Default case. At this stage, we have computed the whole upper convex hull and $a = h$, the number of computed facets is less or equal to p. We return NO.

The overall cost of the grouping step is $O(n\frac{\lambda(p,m)}{p}\log p)$ as in section 2.4.1. The cost of the marriage-before-conquest algorithm is bounded by $O(n\frac{\lambda(p,m)}{p}\log p)$ since we stop the recursion process if the slab has less than $\lceil\frac{n}{p}\rceil$ walls. Indeed, we split into two balanced parts the walls of the tiny objects of $\mathcal{T}$ at each recursive call of the procedure. So that dividing the number of walls inside each slab by a factor $\lambda(p, m)$ amounts to computing

at most $\lambda(p,m)$ bridge facets. Thus, the cost of running PartialMBC is bounded by $O(n\frac{\lambda(p,m)}{p}\log\lambda(p,m))+n\frac{\lambda(p,m)}{p}=O(n\frac{\lambda(p,m)}{p}\log p)$ since $\log\lambda(p,m)=O(\log p)$. Let $c(n,p)$ denote the time complexity of this algorithm. Then, $c(n,p)=O(n\frac{\lambda(p,m)}{p}\log p)+O(\frac{n}{p}a')$ where a' is the number of computed facets during the Jarvis's march ($a'\leq a$). Clearly, $a'\leq p$ so that $c(n,p)=O(n\frac{\lambda(p,m)}{p}\log p)$. This proves the lemma. □

2.4.3 The Overall Algorithm

The scheme of the algorithm is to find a good estimate h_e of h, that is an estimate such that $h\leq h_e<h^2$, and to run the algorithm of section 2.4.1 with that estimate. The final algorithm is described below:

Initializing. Let $i=0$ and $p=2^{2^0}=2$.

Estimating. While $(p<h)$ do $p\leftarrow\min\{n,p^2\}$ (this means that $i\leftarrow i+1$ and $p=2^{2^i}$)

Computing. Compute the upper convex hull using $p=h_e=2^{2^i}$ (note that $h^2>p\geq h$).

Note that we use the algorithm of section 2.4.2 to perform tests like $p<h$ in the while-loop.

Let $c(n,h)$ be the cost of the algorithm, we obtain:

$$c(n,h)=O(1)+O\left(\sum_{i=0}^{\lceil\log\log h\rceil}n\frac{\lambda(2^{2^i},m)}{2^{2^i}}2^i\right)+O\left(n\frac{\lambda(h_e,m)}{h_e}\log h_e\right).$$

Let $\beta(p,m)$ be an upper bound of the ratio $\frac{\lambda(p,m)}{p}$ that satisfies $\beta(p^2,m)=O(\beta(p,m))$ like $\beta(p,m)\leq O(2^{\alpha(p)^{c_m}})$ with c_m an integer depending on m (this upper-bound is deduced from the maximal length of (n,s)-Davenport-Schinzel sequences, see Table 2.1). We bound $c(n,h)$ as follows:

$$c(n,h)\leq O\left(n\beta(h,m)\sum_{i=0}^{\lceil\log\log h\rceil}2^i\right)+O\left(n\beta(h^2,m)\log h^2\right),$$

$$c(n,h)=O(n\beta(h,m)\log h).$$

This yields the desired upper-bound $c(n,h)=O(n\beta(h,m)\log h)$.

Theorem 7 *There exists a deterministic algorithm that computes the upper convex hull of n planar convex objects of fixed type m in time $O(n\beta(h,m)\log h)$ using $O(n\beta(h,m))$ storage.*

This bound is very close to optimal since $\Omega(n \log h)$ is a lower bound [KS86]. In case of convex objects of type 2 (like disks, convex homothets, non-overlapping objects, etc.), the algorithm is truly optimal since $\frac{\lambda(h,2)}{h} = O(1)$
(see Ref. [KLPS86]). If $m > 2$ we do not know if our algorithm is optimal. We cannot reach the $\Omega(n \log h)$ lower bound (proved in Ref. [KS86]) with this method. Indeed, when grouping the objects into groups and computing their vertical decomposition, we create a set of tiny objects which is slightly supra-linear with respect to the original set of objects. This remark gives rise to the problem of the lower bound as soon as $m > 2$. Is $\Omega(n\frac{\lambda(h,m)}{h} \log h)$ a better lower bound for the convex hull problem? Can we group the objects in a better way so that the number of tiny objects obtained from the convex decomposition of the groups is less than $O(n\frac{\lambda(p,m)}{p})$ for a p-grouping?

In the following section, we show how this method can be used to compute upper envelopes of functions and line segments. In the latter case, we can improve the grouping step of the inputs so that we achieve an optimal $\Theta(n \log h)$-time algorithm in the case of line segments.

2.5 Computing Upper Envelopes

Let $\mathcal{F} = \{f_1, ..., f_n\}$ be a collection of n mono-variate, possibly partially defined, functions, all algebraic of some constant maximum degree. We denote by $E_{\mathcal{F}}$ its upper envelope, i.e. the pointwise maximum of the f_i's:

$$E_{\mathcal{F}}(x) = \max_{i \in \{1,...,n\}} \{f_i(x)\},$$

where $f_i(x)$ is the value of the function f_i at abscissa x or $-\infty$ if x does not belong to the domain of definition of f_i. Throughout this paper, we will use the term function for the mathematical object itself or its graph. Thus, in term of graph, the upper envelope of functions can be seen as the part of the graphs of the f_i's visible from viewpoint $(0, +\infty)$. If the functions are partially defined, then the observer (which stands at point $(0, +\infty)$) may see the point $(0, -\infty)$, i.e. there exists vertical rays emanating from $(0, +\infty)$ that do not collide with the function graphs. In order to unify the definition of the mathematical object *upper envelope* in case of partially defined functions, we add an extra function $f_{-\infty}(\cdot)$ such that $f_{-\infty}(x) = -\infty, \forall\ x \in \mathbb{R}$. Thus,

$$E_{\mathcal{F}}(x) = \max_{i \in \{1,...,n\}} \{f_i(x), f_{-\infty}(x)\} = \max_{i \in \{1,...,n\}} \{f_i(x), -\infty\}.$$

The upper envelope is a sequence of maximal visible portions of the original functions. Hereafter, we call *facet* of the upper envelope each maximal

visible portion of the original functions. A facet is fully determined by the function whose graph coincides with that facet, and its two endpoints. The size of the upper envelope $E_{\mathcal{F}}$ of $\mathcal{F}$, denoted by $|E_{\mathcal{F}}|$, is the number of facets of the upper envelope.

Set $\mathcal{F}$ is said of type m if any two functions of $\mathcal{F}$ intersect in at most m points. Line segments are of type 1, parabolæ are of type 2, ... Since the functions have a bounded descriptive size (algebraic functions of fixed degree), $\mathcal{F}$ is of fixed type m.

We can use the theory of Davenport-Schinzel [ASS89, Sha87, Sha88, SA95] to bound the complexity of the upper envelope $E_{\mathcal{F}}$ of $\mathcal{F}$. The maximal length $\lambda(n,m)$ of an (n,m)-Davenport-Schinzel sequence is almost linear in n for fixed m [ASS89, Sha87, Sha88]. It is well-known that the size of the upper envelope of n functions totally defined over $\mathbb{R}$ (resp. partially defined over $\mathbb{R}$) is bounded by $\lambda(n,m+1)$ (resp. $\lambda(n,m+3)$).

For example, line segments are partially defined functions intersecting pairwise in at most one point. Thus, the size of the upper envelope of n line segments is $\lambda(n,3) = O(n\alpha(n))$. Here $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function [WS88]. This bound is tight: M. Sharir and A. Wiernik [WS88] built a set of n line segments such that the size of their upper envelope is $\Omega(n\alpha(n))$. However for practical implementation, it is worth noting that $\alpha(n) \leq 4$ for $n \leq \text{tower}(65536)$ where tower(i) is a tower of 2 of length i, i.e. $\text{tower}(1) = 2$ and $\text{tower}(i+1) = 2^{\text{tower(i)}}$.

The methodology previously described for computing convex hulls can be applied for computing upper envelopes. We briefly recall the main steps. Computing the bridge at a given oriented line Δ, i.e. the facet of $E_{\mathcal{F}}$ intersected by Δ, can be done almost trivially in linear time: first, we select the function which has the highest intersection point with Δ. Let f be that function. Then, in a second step, we find the two endpoints (to the left and right of Δ) limiting the bridge facet. As type m is fixed, we can compute the two endpoints in linear time. We can also design a linear-time-per-facet algorithm (an analogous algorithm of Jarvis's march). Then we consider the case of a set of functions partitionned into k subsets of pairwise nonintersecting subsets. We obtain a $O(n \log h + kh)$-time upper envelope algorithm.

We define the vertical decomposition of a group of functions as the partially defined functions induced by striping vertically the upper envelope. We follow the same steps as those of the convex hull algorithm and obtain an $O(n\beta(h,m)\log h)$-time $O(n\beta(h,m))$-storage algorithm with $\beta(h,m) = O(2^{\alpha(h)^{c_m}})$ where $c_m = \lceil \frac{m}{2} \rceil$ if the functions are partially defined and $c_m = \lceil \frac{m}{2} \rceil - 1$ otherwise (see Table 2.1). Note that the complexity of the upper envelope depends on both the number of intersection points and if the functions

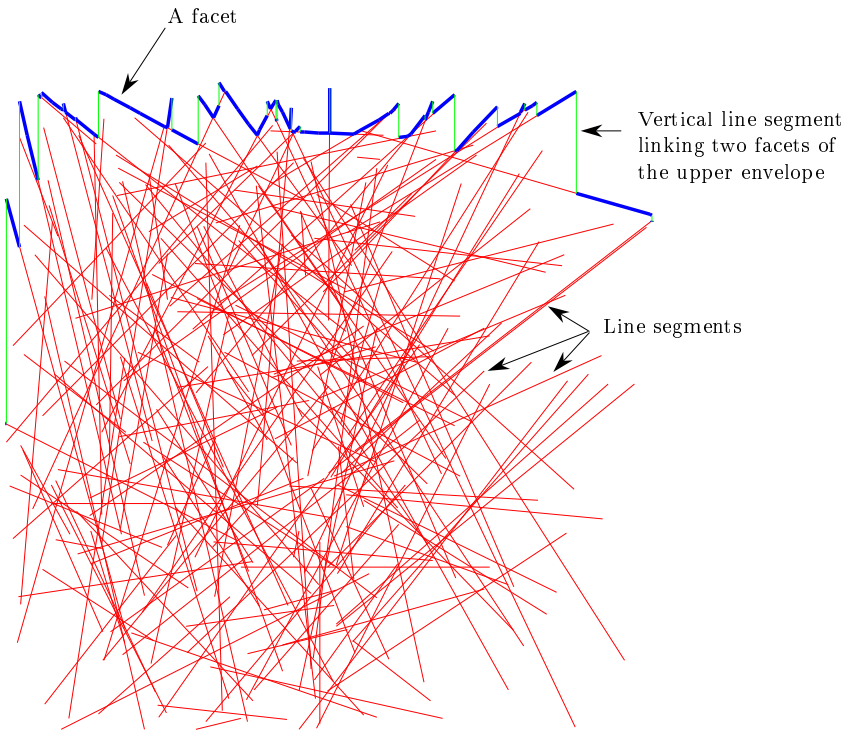


FIG. 2.6 – *Upper envelope of* 200 *line segments. Facets are shown in bold.*

are partially or totally defined.

Thus, for the case of line segments we obtain an $O(n\alpha(h)\log h)$-time algorithm. We show in the following section how we can reach the optimal bound $\Omega(n \log h)$ by adapting the technique due to J. Hershberger [Her89]. The main idea is to group the the line segments efficiently. A family of functions is said to be *k-intersecting* if the functions are intersecting pairwise in at most k points. A set of *k-intersecting generalized segments* is a family of partially defined functions that are k-intersecting.

2.5.1 An Improved Algorithm for *k*-Intersecting Segments

W.l.o.g. we consider the case of line segments. The generalization of the result to k-intersecting generalized segments is straightforward. The main idea is to create groups so that the size of the vertical decomposition of each group remains linear. We first compute a *lazy* interval tree as follows: consider the $2n$ endpoints of the line segments and compute by recursive application of the median algorithm [BFP$^+$72] a partition $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_p\}$ of the $2n$ endpoints so that each sheaf $\mathcal{P}_i$ has size $\frac{2n}{p}$ and the sheaves are x-ordered, i.e. $x(\mathcal{P}_i) < x(\mathcal{P}_j)$ for all $j > i$. We consider the following $p-1$ reference abscissæ and p x-ranges:

- For each sheaf $\mathcal{P}_i$, we associate the x-range X_i of the points $p_i \in \mathcal{P}_i$. Note that all the x-ranges of the sheaves are disjoint.
- Between two successive sheaves, we choose an abscissa a_i so that $X_i < a_i < X_{i+1}$, i.e an abscissa between two consecutive x-ranges of sheaves.

We build an interval tree $\mathcal{IT}$ as follows: each leaf of the interval tree corresponds to the x-range of a sheaf and each internal node to an abscissa separating the sheaves (see Figure 2.8). Then, we allocate the n line segments according to the lowest common ancestor of their two endpoints. At this step, all the segments are located into two kinds of sets:

- Those staying at a leaf of $\mathcal{IT}$. This means that the x-range of each of these line segments is included in the x-range of the sheaf. We say that these line segments are *unclassified*.
- Those lying in an internal node of $\mathcal{IT}$. This means that all the line segments, whose lowest common ancestor of the abscissæ of their endpoints is the abscissa a_i, cross the vertical line $x = a_i$. Their upper envelope is linear in the number of line segments. We say that these segments are *classified*.

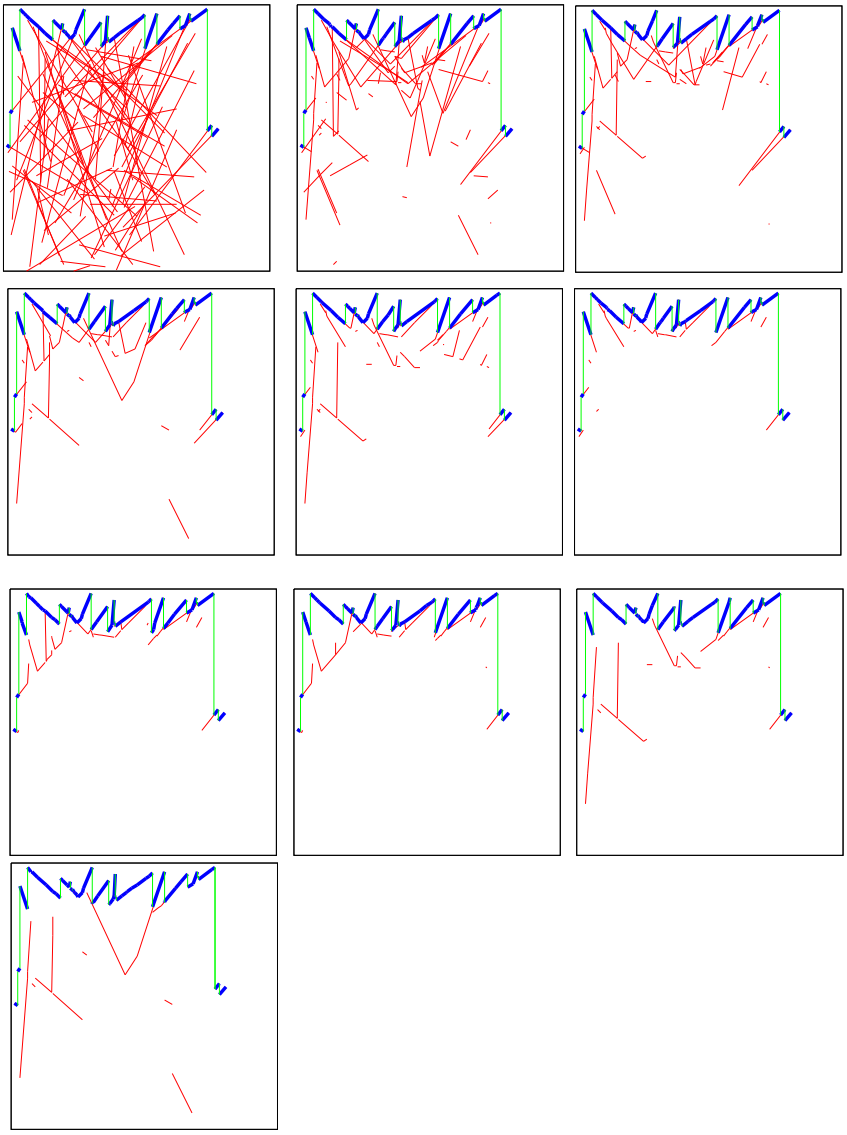


FIG. 2.7 – *The upper envelope of* 100 *line segments. The upper left drawing depicts the upper envelope of* 100 *line segments. Then from left to right, and top to bottom, we first compute groups of size* $10, 20, ..., 90$ *and apply the marriage-before-conquest algorithm on the set of line segments resulting from the vertical decompositions of their upper envelopes.*

Following the communication of J. Hershberger [Her89], we notice that the upper envelope of the line segments allocated into a same internal level of $\mathcal{IT}$ is linear in the number of line segments. Indeed, the upper envelope of the segments allocated to a given internal node is linear (see Ref. [Her89]) because all these segments cross a vertical line and the segments of two nodes of a same internal level are separated by a vertical line by virtue of the interval tree. Let n_i denote the number of line segments at level i, $1 \leq i \leq \lceil \log p \rceil$. By grouping the line segments of each internal level of the interval tree into groups of size p and computing for each group the vertical decomposition of their upper envelope, we obtain an $O(\frac{n}{p} + \log p)$-coloration, i.e. a partition of the original set of n line segments into $\sum_{i=0}^{\lceil \log p \rceil} \lceil \frac{n_i}{p} \rceil = O(\frac{n}{p} + \log p)$ subsets of pairwise non-intersecting line segments resulting from the vertical decomposition of their upper envelope. We also color the unclassified line segments (those staying at a leaf of the interval tree) as follows: to the i-th line segment attached to a given leaf of the interval tree, we give it the color $(i, 2)$. Here, 2 means the unclassified line segments. Note that $i \leq \lceil \frac{n}{p} \rceil$. Moreover, two line segments with color $(i, 2)$ do not intersect since they belong to two different sheaves and are therefore x-separated.

Thus, globally, after an $O(n \log p)$-preprocessing time required for building the lazy interval tree, we obtain a $O(\frac{2n}{p} + \log p)$-coloration of a new created set of $O(n)$ line segments which has the same upper envelope as $\mathcal{O}$. We run the $O(n \log h + kh)$-time algorithm upon this new set. Since $k = \frac{2n}{p} + \log p$, we obtain an $O(n \log h + \frac{2n}{p} h + h \log p) = O((n + h) \log h)$-time algorithm with linear storage.

For the case of k-intersecting generalized segments, we note that the complexity of the upper envelope of the n_i k-intersecting segments at the i-th level of the interval tree is $O(\lambda(n_i, k+1))$ [Her89]. It follows that the complexity of the upper envelopes (one upper envelope per group) of the n k-intersecting segments is $O(n\beta(h, k+1))$. Thus, we can compute the upper envelope of k-intersecting segments in time $O((n\beta(h, k+1) + h) \log h)$. The space requirement has also been reduced to $O(n\beta(h, k+1))$. A challenging problem is to design an algorithm that computes the upper envelope of n functions intersecting pairwise in at most m points in less than $O(\lambda(n, m+1) \log n)$ operations. Probably, if a better result is found, it may yield straightforwardly to a better output-sensitive algorithm since the crucial step of our method is to compute partitionned sets.

As a final remark, to underline the power of the grouping scheme, we show how in the case of line segments we can obtain again an $O(n \log h)$-time algorithm using ray shooting procedures. As before, we create $\lceil \frac{n}{p} \rceil$ groups of size p, compute their upper envelopes and preprocess these upper envelopes

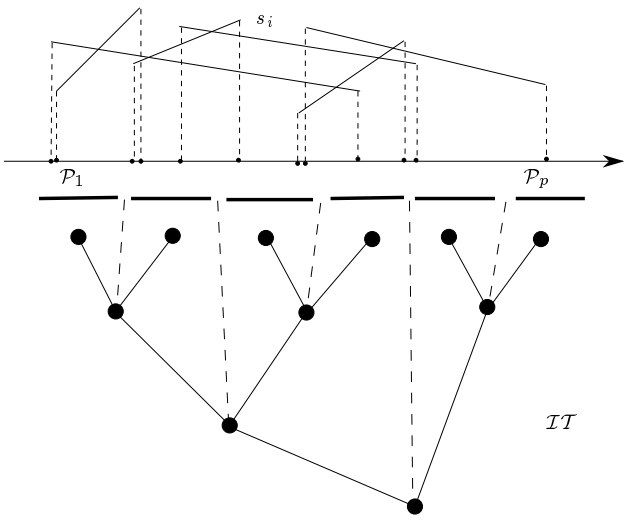


FIG. 2.8 – *Building the lazy interval tree.*

(which can be viewed as simple polygons, each of them of size $O(p\alpha(p))$) for ray shooting. For each group, the time for computing its upper envelope and preprocess it for ray shooting is $O(p \log p)$ [CEG+91, HS93]. Thus, the total time for the preprocessing step is $O(n \log p)$. The ray shooting query time of a group is $O(\log p)$.

Then, the procedure walks from $x = -\infty$ to $x = \infty$ as follows: consider that the algorithm at some stage has found a portion of the upper envelope (a line segment) and therefore knows (by the rightmost endpoint e of that facet) which line segment s will support the following portion of the upper envelope. Then, for each group (in fact each simple polygon), we shoot a ray from the endpoint e following the direction of s. Finally, among the $\frac{n}{p}$ terminations, we choose the one that shorten the most the line segment s. The cost of this algorithm is $O(n \log p + (\frac{n}{p} \log p + \frac{n}{p})h)$. If $p = h$ then the algorithm has time complexity $O(n \log h)$. We use again the technic of approximation in order to achieve that bound.

2.6 Concluding Remarks

We have applied the marriage-before-conquest paradigm to the computation of the convex hull of n planar convex objects of fixed type m. We first described a linear-time algorithm to compute the bridge of the convex hull at a given oriented line. Then, we investigated the case where the family of objects consists of k subsets of non-overlapping objects. For that case, we de-

signed an $O(n \log h + kh)$-time algorithm where h denotes the output-size. As a byproduct, we obtain an optimal $\Theta(n \log h)$-time algorithm for computing the convex hull of a set of non-overlapping objects. Moreover, if each object cannot intersect more than δ others then we design an $O(n \log h + \delta h)$-time algorithm. Finally, we transformed the problem of computing the convex hull of $\mathcal{O}$ to computing the convex hull of a set $\mathcal{T}$ partitioned into non-overlapping subsets such that $\mathcal{CH}(\mathcal{O}) = \mathcal{CH}(\mathcal{T})$. (We use nonoutput-sensitive algorithms in order to get $\mathcal{T}$.)

The size of the partition of $\mathcal{T}$, i.e. the number of non-overlapping subsets, depends on the size of the output. Since we do not know the output-size, we iteratively estimate it. We finally choose a good estimate to compute the convex hull of a set of n planar convex objects of fixed type m in $O(n\beta(h, m) \log h)$ time where $\beta(h, m)$ is an extremely slowly growing function. We can follow the same scheme for computing the upper envelope of possibly partially defined functions. In that case, the bridge at a given oriented line is the maximal piece of the lower envelope intersected by that line and can be computed trivially in linear time. Therefore, when computing the convex hull of objects, one might first compute the dual functions and apply the upper envelope algorithm. However, computing directly the convex hull remains interesting in the case of k-colored partitions because, in that case, one does not need to apply the grouping scheme.

All these algorithms can be easily parallelized onto EREW PRAM multi-computers, following the algorithm of S. Akl [Akl84, Akl85a]. D.G. Kirkpatrick and R. Seidel [KS86] proved that $\Omega(n \log h)$ is a lower bound for computing the convex hull of a set of n points where h is the number of hull vertices. Can we improve that lower bound in the case of convex objects of fixed type m? It would also be interesting to find other applications of this method and to generalize it to higher dimensions.

Chapitre 3

Output-sensitive Peeling of Convex and Maximal Layers

(Les résultats de ce chapitre seront disponibles dans une version similaire au journal Information Processing Letters [Nie96e]. Nous présentons quelques remarques supplémentaires dans cette version.)

Abstract

We give an output-sensitive algorithm to compute the first k convex or maximal layers in $O(n \log H_k)$-time where H_k is the number of points participating in the first k layers. Computing only the first k layers is interesting in various problems that arise in computational geometry (k-sets and dually k-levels, k-hulls and dually k-belts), pattern recognition, statistics, operations research, etc.

3.1 Introduction and notations

Let $\mathcal{S}$ be a set of n planar points and $\mathcal{L}(\mathcal{S})$ its first layer, i.e. a subset of $\mathcal{S}$. Let $\mathcal{S}_1 = \mathcal{S}$ and $\mathcal{S}_{i+1} = \mathcal{S}_i \backslash \mathcal{L}(\mathcal{S}_i)$ for integer $i \geq 1$. The layers of $\mathcal{S}$ are defined by the following ordered sequence: $\mathcal{L}(\mathcal{S}_1)$, ..., $\mathcal{L}(\mathcal{S}_l)$ where l is the first integer such that $\mathcal{S}_{l+1} = \emptyset$. We successively investigate the case where $\mathcal{L}(\mathcal{S})$ is the set $\mathcal{V}(\mathcal{S})$ of points of $\mathcal{S}$ on the boundary of the convex hull $\mathcal{CH}(\mathcal{S})$ (Section 2, Figure 3.1) and the maxima $\mathcal{M}(\mathcal{S})$ of $\mathcal{S}$ (Section 3). Let $h_i = |\mathcal{L}_i|$ and $H_i = \sum_{j=1}^{i} h_j$ denote respectively the number of points of $\mathcal{L}(\mathcal{S}_i)$ and the number of points participating in the $\leq i$-layers. Let $H_0 = 0$. We have $H_i = H_{i-1} + h_i$ for $i \geq 1$. We assume in the sequel that $\mathcal{S}$ is in general position,

i.e., all the points are distinct and no three points are colinear. Notice that in that case $H_k \geq \min\{3k, n\}$. For a set of n points in the euclidean plane $\mathbb{E}^2$, the algorithms have running time $O(n \log H_k)$ using linear storage.

Computing the first k convex layers of a planar point set has numerous applications [PS85] and is also used as a first step when one computes k-hulls [CSY87] (or dually k-belts [EW86]) or k-sets (or dually k-levels [ERvK93]). A k-set of $\mathcal{S}$ is a set $\mathcal{S} \cap H$ of size k for some halfspace H. For any set $\mathcal{S}$ of points, the k-hull of $\mathcal{S}$ is the set of points p such that for any line passing through p there are at least $k \leq \lfloor \frac{n}{2} \rfloor$ points in each closed halfspace. It follows from the definition that the k-hull contains the set of points $\mathcal{S}_k$. k-hulls are also used in statistics to measure the *interiorness* of a given set. Using our output-sensitive algorithm (Section 2) for computing the first k layers, we obtain an $O(n \log H_k + (H_k + f_k) \log^2 k)$-time algorithm for computing the k-hull of a n-point set where f_k is the size of the k-hull and H_k is the size of the first k convex layers (the previous algorithm of [CSY87] has running time $O(n \log n + (n + f_k) \log^2 k)$).

As already noticed by H. Everett et al. [ERvK93] in their algorithm for computing the $\leq k$-levels: *"... we can remove all layers deeper than k from consideration..."*, so that we can replace their $O(n \log n)$ preprocessing time by a $O(n \log H_k)$ preprocessing time to compute the $(\leq k)$-sets ($\leq k$-levels) of a set of n points (respectively lines) in the euclidean plane. This is to be compared with the result of T. Chan [Cha95b]: a $O(n \log h + h \log^2 n)$-time algorithm is given to compute the h faces of the k-th level of n lines using the grouping scheme with ray shooting procedures based on parametric search [AM93].

We first begin with an overview of the previous algorithms that compute the convex layer decomposition, i.e., all the convex layers of a given set. One can trivially compute the convex layer decomposition in $O(n^2 \log n)$-time by successively peeling $\mathcal{S}$ using any optimal planar convex hull algorithm. Another way to proceed is to first create an x-monotone polygonal chain and then to peel it in $O(n^2)$-time by successively applying any linear-time convex hull algorithm on simple polygons. M. Overmars and J. van Leeuwen [OvL80, OvL81] gave a dynamic algorithm to maintain in $O(\log^2 n)$ time per insertion/deletion the convex hull of n points. Their method leads straightforwardly to an $O(n \log^2 n)$-time convex layers algorithm. B. Chazelle [Cha85] gave an optimal $\Theta(n \log n)$ time algorithm to compute the convex layer decomposition using the hull tree of M. Overmars and J. van Leeuwen. His solution relies upon an $\bar{O}(\log n)$ amortized cost procedure that maintains dynamically the convex hull of a set of n points (although the insertion/deletion of a simple point can cost linear time).

We consider the problem of computing only the first k layers of the convex

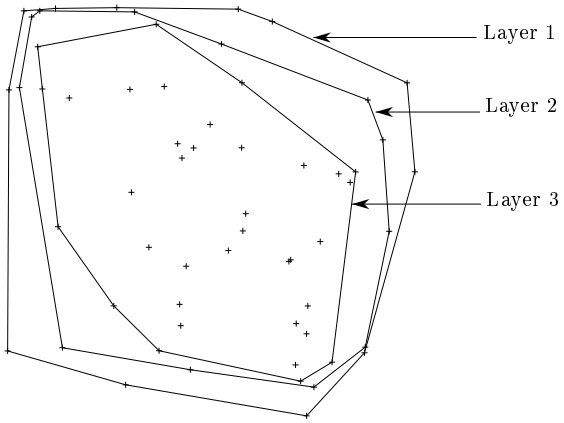


FIG. 3.1 – *The first three layers of a set of* 60 *planar points.*

or maximal layers of a planar point set in an output-sensitive manner.

We can apply round after round Jarvis's algorithm [Jar73] to get an output-sensitive algorithm with running time $O(nH_k)$. D.G. Kirkpatrick and R. Seidel [KS86] gave an optimal output-sensitive algorithm to compute the convex hull of a set of n points in $\Theta(n \log h_1)$-time. We can use successively that algorithm to compute the first k layers in time $O(\sum_{i=1}^{k}(n - H_{i-1}) \log h_i) = O(nk \log \frac{H_k}{k})$ using Hölder's inequality [1]. This algorithm is therefore optimal if $k = O(1)$. However, in pattern recognition [KR85], one wants to remove some of the first layers in order to eliminate noise of the input set and to get the real shape of the set of points. The number of layers we want to remove depends on the size of the set (the number of available data) so that generally k is a function of n.

The heretofore best known solution for computing the convex layer decomposition (i.e. all the layers) of a set of n points in $\mathbb{E}^3$ is an algorithm whose running time is $O(n^{1+\epsilon})$-time [AM93, Cha95a] (the previous solution was $O(n^{\frac{3}{2}} \log n)$ – see [Ede87]). Recently, T. Chan [Cha95b] gave a worst-case time optimal algorithm to compute the convex layer decomposition in time$\Theta(n^{\lfloor \frac{d}{2} \rfloor})$, for $d \geq 4$, by computing the depth of a n-point set in $O(n^{2-\gamma}) = o(n^2)$ with $\gamma < \frac{2}{\lfloor \frac{d}{2} \rfloor^2 + 1}$ and applying for each depth the optimal convex hull algorithm of B. Chazelle [Cha91]. In higher dimensions ($d > 2$), we can compute the h points participating to the first $\leq k$- layers in time $O(n \log^{d+1} h + (nh)^{1 - \frac{1}{\lfloor \frac{d}{2} \rfloor + 1} + \epsilon})$ using $O(n + (nh)^{1 - \frac{1}{\lfloor \frac{d}{2} \rfloor + 1} + \epsilon})$ storage or in time $O(n \log^{d+2} h + (nh)^{1 - \frac{1}{\lfloor \frac{d}{2} \rfloor + 1}} \log^{O(1)} n)$ using $O(n + (nh)^{1 - \frac{1}{\lfloor \frac{d}{2} \rfloor + 1}} \log^{O(1)} n)$

1. We can state Hölder's inequality as follows: $\max_{p_1 + p_2 = p} \{\log p_1 + \log p_2\} \leq 2 \log \frac{p}{2}$.

storage [Cha95b, Cha96]. (These results might be slightly improved [AR96]). Once we have computed the depth of each point, we can apply any non-output-sensitive algorithms for each set of points having the same depth. In dimension 3, we thus obtain an $O(h \log h + n \log^4 h + (nh)^{\frac{1}{2}+\epsilon}) = O(n \log^4 h)$ whenever $h < n^{1-\epsilon}$ and an $O(n^{1+\epsilon})$-time algorithm otherwise. In dimension $d > 3$, we have also to build each layer using the gift-wrapping algorithm [CK70, Swa85]; the number of faces of a convex hull of h_i points in $\mathbb{E}^d$ is $\Theta(h_i^{\lfloor \frac{d}{2} \rfloor})$.

We present below an $O(n \log H_k)$-time algorithm to compute the $\leq k$-maximal or convex layers.

3.2 Computing the first k convex layers

3.2.1 The power of grouping

3.2.2 Principles

Our algorithm basically uses the grouping scheme, i.e., a general paradigm to obtain output-sensitive algorithms, independently developed by T. Chan [Cha95b] and F. Nielsen [NY95]. It consists in grouping the points into balanced-size sets and in preprocessing each group in order to answer queries efficiently. Finally, we build the solution stepwise by querying these groups. For each query, we first determine in each group the local result of that query and then deduce the global result of the query by merging these local results (e.g., selecting one of them).

3.2.3 Answering queries

What is a *query* for the convex hull problem? We consider a group $\mathcal{P}$ of p points. Given a query point Q, with the property that $Q \notin \mathcal{CH}(\mathcal{P})$, we want to determine the two segments issued from Q and tangent to $\partial\mathcal{CH}(\mathcal{P})$, the boundary of the convex hull $\mathcal{CH}(\mathcal{P})$.

We preprocess $\mathcal{P}$ into a hull tree [OvL80, PS85] in time $O(p \log p)$ (see Figure 3.2), so that given a query point Q we can determine in $O(\log p)$ the two tangent segments linking Q to $\partial\mathcal{CH}(\mathcal{P})$ (see Figure 3.3). Maintaining dynamically under deletion the group $\mathcal{P}$ requires $O(\log^2 p)$ time per insertion/deletion. (Note that once the data-structures are built, we will only delete points from them in our algorithm).

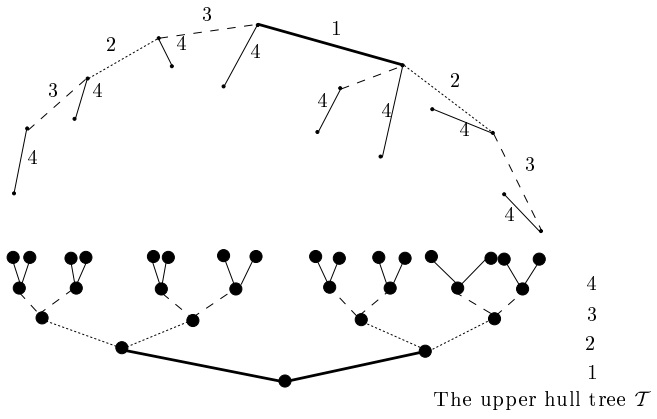


FIG. 3.2 – *The upper hull tree of a set of* 16 *points.*

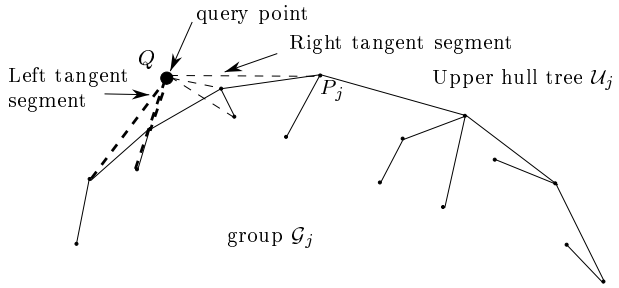


FIG. 3.3 – *A query (in one of the groups): given a point* Q *outside the convex hull, determine its two tangent segments.*

3.2.4 The algorithm given an estimate of the output-size

Assume that we know a good estimate $h_e \leq n^{\frac{1}{4}}$ of the output-size of the $\leq k$-layers. Let $\mathsf{Layers}(k, h_e)$ be the algorithm described below:

Preprocessing. Group the data points in $\lceil \frac{n}{h_e} \rceil$ balanced-size groups of size at most h_e. For each group $\mathcal{G}_i$, build the upper and lower hull trees. Let $\mathcal{U}_i$ and $\mathcal{L}_i$ be respectively the upper and lower hull trees of the points of $\mathcal{G}_i$.

Computing a layer. If $k = 0$ then HALT. Otherwise, find the two extremal points P_1 and P_2 that have respectively the smallest and largest x-coordinate of the current set of points ($x(P_1) \leq x(P_2)$). Let P be initially point P_1. Let $\overrightarrow{v}$ be any vertical vector.

Upper Hull. For each group $\mathcal{G}_j$, find the right tangent segment PP_j given the query point P by querying the upper hull tree $\mathcal{U}_j$. Among the $\lceil \frac{n}{h_e} \rceil$ groups, select the right tangent segment $[PP_i]$ whose angle with the vector $\overrightarrow{v}$ is minimized. Add P_i to the current layer and remove P_i from both the upper and the lower hull trees, $\mathcal{U}_i$ and $\mathcal{L}_i$, of its group $\mathcal{G}_i$. Let $\overrightarrow{v} = \overrightarrow{PP_i}$ and P be P_i. If $P \neq P_2$ then go to step *Upper Hull*.

Lower Hull. symmetric case.

Next Layer. $k \leftarrow k - 1$. Go to step *computing a layer*.

Let $c(n, h)$ denote the time complexity of the above algorithm. Grouping the points into $\lceil \frac{n}{h_e} \rceil$ balanced groups of size at most h_e and computing their upper and lower hull trees cost $O(\lceil \frac{n}{h_e} \rceil h_e \log h_e) = O(n \log h_e)$. (We may assume, w.l.o.g., that the k-th layer is not empty, i.e., $h_k > 0$). Consider the cost of computing the i-th layer:

- we first find the two extremal points P_1 and P_2 by computing in each group $\mathcal{G}_i$ the extremal points $P_{1,i}$ and $P_{2,i}$ with $x(P_{1,i}) \leq x(P_{2,i})$. Then, we select among those candidates, the extremal points in $O(\lceil \frac{n}{h_e} \rceil)$-time. Finding the $P_{1,i}$'s and $P_{2,i}$'s costs $O(\lceil \frac{n}{h_e} \rceil \log h_e)$ time by querying, for example, the upper hull trees.
- Computing the upper and lower i-th layers costs $O(\lceil \frac{n}{h_e} \rceil h_i \log^2 h_e)$. Indeed, each time we find a new point belonging to the i-th layer we answer $\lceil \frac{n}{h_e} \rceil$ queries and remove a point from one of the groups (which has size less than h_e). (A more careful inspection of the complexity analysis shows that the i-th layer can be computed in $O(\lceil \frac{n}{h_e} \rceil h_i \log h_e + h_i \log^2 h_e)$.)

Thus, computing the $\leq k$-layers costs:

$$O(\lceil \frac{n}{h_e} \rceil h_e \log h_e) + O(\lceil \frac{n}{h_e} \rceil (k + \sum_{i=1}^{k} h_i) \log^2 h_e) = O(n \log h_e) + O(\frac{nH_k}{h_e} \log^2 h_e)$$

In the sequel, we will choose h_e ($h_e \leq n$) such that $H_k^2 \leq h_e$. Thus, the algorithm runs in time $O(n \log H_k)$ since in that case $O(n \frac{H_k}{h_e} \log h_e) = O(n)$. (Notice that if we only choose $h_e \geq H_k$ then our algorithm runs in time $O(n \log^2 H_k)$.)

Another interesting feature of the algorithm is that we can determine whether $H_k > p$ or not for a given integer p in $O(n \log p)$. Indeed, we choose $h_e = p^2$ and we stop the queries as soon as we have computed $\min\{p, H_k\}$ points belonging to the first k convex layers. Denote by $\mathsf{Test}(p)$ that algorithm.

3.2.5 The final algorithm

We put together the basic procedures seen so far. The goal is to determine quickly a good estimate h_e of H_k, that is to find h_e such that $H_k^2 \leq h_e \leq n$. Then, we run the $\mathsf{Layers}(h_e, k)$ algorithm. The final algorithm is described below:

Initializing. Let $i = 0$ and $h_e = (2^{2^0})^2 = 4$.

Computing a good estimate. While $(\sqrt{h_e} < H_k)$ do $i \leftarrow i + 1$ and $h_e = (2^{2^i})^2$. If $h_e > n$ then run Chazelle's algorithm [Cha85].

Computing the $\leq k$-layers. Call algorithm $\mathsf{Layers}(k, h_e)$.

Note that the test $(\sqrt{h_e} < H_k)$ is performed using algorithm $\mathsf{Test}(\sqrt{h_e})$ described above in $O(n \log h_e)$-time. When we get out of the while-loop, we have the desired inequalities: $H_k^2 \leq h_e$. We have to assert that $h_e \leq n$. This means that in the worst-case when we exit the while-loop, we have $h_e = (H_k^2 - 1)^2 \leq H_k^4$. Therefore, if $H_k \leq n^{\frac{1}{4}}$ then we get $h_e \leq n$.

Let $c(n, H_k)$ denote the running time of the previous algorithm. Clearly, we have:

$$c(n, H_k) = O(1) + O(\sum_{i=0}^{\lceil \log \log H_k \rceil} n \log 2^{2^i}) + O(n \log h_e),$$

$$c(n, H_k) = O(n \log H_k), \text{ if } H_k \leq n^{\frac{1}{4}}.$$

If $H_k > n^{\frac{1}{4}}$ we run Chazelle's algorithm [Cha85] and find the $\leq k$-layers in $O(n \log n) = O(n \log H_k)$.

3.3 Computing the first k maximal layers

In this section, we just re-phrase the previous algorithm in a context of maximal layers. More precisely, we change the query problem of the last problem. Notice that in the case of the $\leq k$-maximal layers, we have $\mathcal{L}(\mathcal{S}) = \mathcal{M}(\mathcal{S})$ where $\mathcal{M}(\mathcal{S})$ is the set of maxima points of $\mathcal{S}$, that is all the points of $\mathcal{S}$ that are not dominated by any other, where a point P dominates a point Q if $x(P) > x(Q)$ and $y(P) > y(Q)$ (see [Yao74, KLP75, KS85, FR90, Jan91, Kap94]). The maxima $\mathcal{M}(\mathcal{S}) = \{M_1 = (x_1, y_1), ..., M_i = (x_i, y_i)\}$ of a point set $\mathcal{S}$ can be arranged into a staircase structure as follows: we join two consecutive maxima points $M_k = (x_k, y_k)$ and $M_l = (x_l, y_l)$, with $x_k \leq x_j$ or $x_j \geq x_l$ by the two iso-oriented segments $[(x_k, y_k), (x_k, y_l)]$ and $[(x_k, y_l), (x_l, y_l)]$ (see Figure 3.4).

We group the initial set $\mathcal{S}$ of n points into $\lceil \frac{n}{p} \rceil$ balanced-size groups of size at most p. We compute for each group $\mathcal{G}_i$, $1 \leq i \leq \lceil \frac{n}{p} \rceil$, a data-structure that allows to maintain dynamically its first maximal layer [Kap94]. The total cost of the preprocessing step is $O(\lceil \frac{n}{p} \rceil p \log p) = O(n \log p)$. The maximal layer of a set of p points can be maintained dynamically both under insertion/deletion in $O(\log p)$ [Kap94]. (Note that we only delete points from our data-structures in the grouping strategy described below). We describe the algorithm MaximaLayers(k, p) below:

Preprocessing. Group the points into $\lceil \frac{n}{p} \rceil$ groups of size at most p. For each group $\mathcal{G}_i$, compute the data-structure that allow to maintain dynamically the first maximal layer of $\mathcal{G}_i$ using the algorithm of S. Kapoor [Kap94].

Computing a maximal layer. if $k = 0$ then HALT.

Initializing. Find the rightmost abscissa point P_{init} of the points clustered into groups by querying each group and selecting the overall rightmost abscissa point. Set P to P_{init}.

Querying. For each group $\mathcal{G}_i$, 'ray shoot' from P horizontally to the left until it hits a staircase of the first layer of maximas of $\mathcal{G}_i$. Let P_i be the point of $\mathcal{G}_i$ at the top of this staircase (if it does not hit any staircase then set P_i to $(-\infty, +\infty)$). Let P' be the point which has the highest x-coordinate among the P_i's. If $P' = (-\infty, +\infty)$ then skip to step *next layer*. Otherwise, add P' to the current maximal layer and remove P' from the data-structure of its group. Set $P \leftarrow P'$. Go to *querying*.

Next layer. $k \leftarrow k - 1$. Go to *computing a maximal layer*.

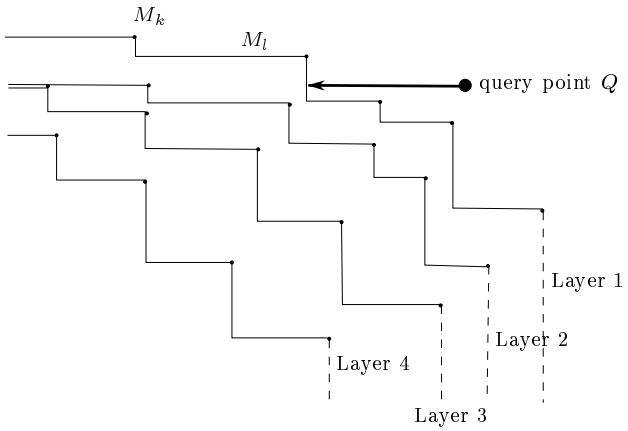


FIG. 3.4 – *A query (in one of the groups): given a point Q above the first layer of the maxima points, determine the first (rightmost abscissa) point of the first layer that is to the left of Q.*

Using the structure of S. Kapoor [Kap94], we can delete a point in a group $\mathcal{G}_i$ in $O(\log p)$ time. The 'ray shooting' operations (one per group) used in the querying stage can easily be done in $O(\frac{n}{p}\log p)$-time. Indeed, for a group $\mathcal{G}_i$ and a query point Q we want to find among the points of the first maximal layer, the one that is above $y(Q)$ and has the highest abscissa. This may be easily implemented using a binary search. (Alternatively, we can also simulate an insertion of Q into the data-structure of $\mathcal{G}_i$ in order to find its left neighbor).

As before, we can study the complexity of the previous algorithm. Its running time is $O(\frac{n}{p}H_k \log p) + O(n \log p)$. We then find a good estimate p such that $H_k \leq p < H_k^2$ and run the algorithm with that estimate. We finally end up with an $O(n \log H_k)$-time algorithm.

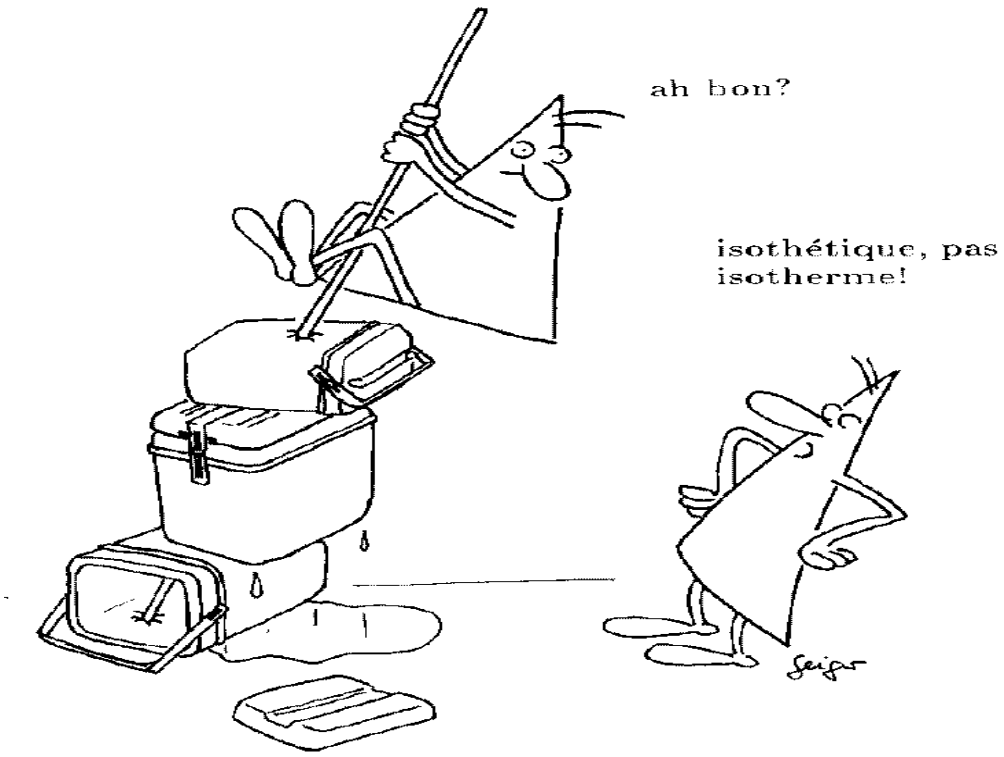
Acknowledgments. I am very thankful to my supervisors, Jean-Daniel Boissonnat and Mariette Yvinec, whose valuable comments on this paper led to a much clearer presentation. I also greatly appreciated the comments of an anonymous referee.

Troisième partie

Perçer un ensemble d'objets

Chapitre 1

Étude de la perçabilité d'objets: algorithmes et propriétés combinatoires



1.1 Introduction

Dans cette partie, nous considérons le problème suivant :

Problème 6 (Percer) *Étant donné un ensemble $\mathcal{S}$ d'objets géométriques, trouver un ensemble $\mathcal{P}$ de points, de taille minimale, tel que $\mathcal{P}$ perce $\mathcal{S}$, c'est-à-dire pour tout objet $O \in \mathcal{S}$, il existe un point $p \in \mathcal{P}$ tel que $p \in O$.*

$\mathcal{S}$ est perçable par k points s'il existe un ensemble $\mathcal{P}$ de k points (c'est-à-dire $|\mathcal{P}| = k$) perçant $\mathcal{S}$. Autrement dit, $\mathcal{S}$ est perçable par k points si on peut piquer $\mathcal{S}$ à l'aide de k aiguilles telles que $\mathcal{S}$ soit immobilisé en translation. Considérons un ensemble $\mathcal{S}$ de n objets convexes.

Trouver l'entier minimal k tel que $\mathcal{S}$ est k-perçable est NP-complet dès que nous considérons des objets bidimensionnels[1] [FPT81]. Notons que k ne dépend pas de n, et qu'un ensemble perçable par deux points aura au minimum deux éléments et éventuellement une infinité.

Ce problème est intractable, en pratique, pour de faibles valeurs de n (par exemple $n = 20$, cf. chapitre 1.2).

Jusqu'à présent, on transformait ce problème de couverture (set covering problem) ou dualement ce problème d'appartenance (hitting set problem) en un problème d'optimisation grâce à une formulation matricielle. Soit $\mathcal{V} = \{S_i | i \in I\}$ un ensemble de $v = |\mathcal{V}| = |I|$ sous-ensembles éléments de $2^{\mathcal{S}}$, l'ensemble des parties de $\mathcal{S}$. Nous désirons trouver une couverture de taille minimale, c'est-à-dire un sous-ensemble $I' \subseteq I$ d'indices tel que $\mathcal{S} = \bigcup_{i \in I'} S_i$ en minimisant le cardinal $|I'|$ de I'. En d'autres termes, nous voulons minimiser $e^T \times x = |I'|$ tel que $Ax \geq e$ où x est un vecteur binaire de dimension v ($\{0,1\}^v$-vecteur), $e = (1, ..., 1)$ de taille v et A est une matrice binaire de taille $(v \times n)$, dont chaque colonne est le vecteur d'incidence d'un des ensembles I_i, $1 \leq i \leq v$.

Les informaticiens confrontés à résoudre le problème de manière efficace pour de grandes valeurs de n ($n > 20$), ont construit quelques heuristiques donnant un sous-ensemble d'indices I'' dont le cardinal $|I''|$ soit contrôlé par les paramètres du problème : n et $k = |I|$. Chvátal [Chv79] donna un algorithme glouton de complexité cubique tel que $|I''| \leq |I| \log a$ où $a \leq n$ est la somme maximale des éléments des colonnes de la matrice d'incidence A. Récemment, Slavík [Sla96] montra que la performance de l'algorithme glouton est telle que $|I''| \leq |I|(\log a - \log\log a + \Theta(1))$. Hochbaum [Hoc82] proposa un autre algorithme de complexité cubique garantissant $|I''| \leq |I| \times f$

1. En effet, Fowler *et al.* ont montré que couvrir un ensemble de points par un nombre minimal de carrés isothétiques est NP-complet dès que la dimension est supérieure ou égale à deux (ce problème est nommé dans la littérature par BOX-COVER problem).

où f est la somme maximale des éléments des lignes de A. Bellare *et al.* [BGLR93] montrèrent qu'il n'existe pas d'algorithmes polynomiaux qui puissent approximer la taille de la solution optimale en $(\frac{1}{8} - \epsilon) \log |\mathcal{S}|$à moins que $NP \subseteq DTIME[n^{\log \log n}]$ où ϵ est n'importe quel réel strictement positif. Ce premier résultat négatif fut amélioré par Feige [Fei96] qui prouva la meilleure borne inférieure, c'est-à-dire $(1 - \epsilon) \log |\mathcal{S}|$ pour n'importe quel $\epsilon > 0$.

Un inconvénient majeur du point de vue de la géométrie algorithmique est que ces méthodes ne considèrent pas des objets géométriques mais plutôt des sous-ensembles d'un ensemble initial $\mathcal{S}$. Toutefois, il fut démontré par Wegner [Weg67] que le graphe d'intersection[2] d'objets convexes en dimension d peut être arbritraire aussitôt que $d \geq 3$. Il faut donc pour pouvoir appliquer les algorithmes précédents fournir la matrice d'incidence A. Si les objets dans $\mathcal{S}$ ont une taille descriptive constante, nous pouvons calculer tous les points d'intersection entre tout d-uplets d'objets. On considère pour ces $O(n^d)$ points d'intersection les sous-ensembles définis comme suit: à un point d'intersection p on associe le sous-ensemble constitué des objets de $\mathcal{S}$ contenant ce point, c'est-à-dire $\{S \in \mathcal{S} | p \in \mathcal{S}\}$. Ainsi, la taille de la matrice est $O(n^{d+1})$ et ces heuristiques demandent un temps de calcul en $O(n^{d+2})$ et un espace mémoire en $O(n^{d+1})$. Les programmes implantant ces algorithmes ne sont donc efficaces ni en temps de calcul ni en espace mémoire. Dans la plupart des applications provenant du prétraitement d'images, conception de circuits (VLSI design) et de la localisation de points, nous avons à considérer un espace de grande dimension et un nombre de données important [TF80].

Récemment, Brönnimann et Goodrich [BG94] regardèrent le problème de couverture en utilisant la dimension de Vapnik-Chervonenkis [LMR88] et en utilisant les concepts ϵ-nets [Mat91a]. Ils obtiennent des couvertures dont la taille dépend de la taille d'une couverture optimale (precision-sensitive) si la VC-dimension est bornée, comme dans beaucoup de systèmes d'ensembles (set systems) concernant les objets de nature géométrique. Toutefois, leur algorithme repose encore sur la sélection d'un sous-ensemble (la couverture). Il faut donc se donner la matrice A d'incidence.

Hochbaum et Maass [HM84] étudièrent le cas d'objets géométriques et donnèrent un schéma d'approximation polynomial (PTAS: Polynomial-Time Approximation Scheme, voir l'ouvrage de Garey et Johnson [GJ79]). À l'inverse, leur méthode ne s'applique que pour des objets géométriques et considère le problème de couverture.

2. Le graphe d'intersection d'un ensemble d'objets $\mathcal{S}$ est défini comme suit : à chaque objet on associe un nœud et deux nœuds sont connectés par une arête si et seulement si les objets correspondants se coupent.

Dans la deuxième partie (cf. chapitre 1.2), nous considérons le problème de percer un ensemble de boîtes isothétiques. Nous montrons comment il est possible de s'affranchir du calcul de la matrice A et d'obtenir ainsi une solution efficace. Une implantation corrobore les résultats théoriques.

La troisième partie (cf. 1.3) s'intéresse aux propriétés combinatoires de la géométrie des convexes. Après un court rappel de quelques propriétés combinatoires d'objets géométriques, nous considérons les nombres de Helly associés à la propriété de perçabilité. Nous donnons une batterie d'algorithmes qui constituent les preuves de nos propriétés combinatoires.

Finalement, dans la partie 1.4, nous concluons sur ces deux aspects en décrivant quelques problèmes ouverts liés aux deux parties précédentes.

1.2 Percer un ensemble de boîtes isothétiques

Nous présentons dans le chapitre 2 un algorithme simple et efficace pour percer un ensemble $\mathcal{S}$ de n boîtes isothétiques de dimension d. Nous commençons par montrer que les performances de l'algorithme glouton de Chvátal dans le cas de boîtes isothétiques sont celles de l'algorithme glouton en général dès que la dimension d est supérieure ou égale à deux.

Notre algorithme calcule un ensemble de c points perçant $\mathcal{S}$ en temps sensible à la sortie $O(dn + n \log c)$ et espace linéaire. Si c^* est le nombre minimal de points requis pour percer $\mathcal{S}$, alors nous montrons que

$$c \leq \min \left\{ \frac{{c^*}^{\overline{d}}}{d!} + \frac{{c^*}^{\overline{d-1}}}{(d-1)!} - 1, c^* \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!} \right\},$$

où $x^{\overline{m}}$ est la puissance factorielle croissante: $x^{\overline{m}} = \prod_{i=0}^{m-1}(x+i) = m!\binom{x+m-1}{m}$. Puisque trouver un ensemble minimal de c^* points est un problème NP-complet dès que $d > 1$, nous obtenons une heuristique adaptative efficace pour percer $\mathcal{S}$ en temps sensible à la sortie et espace linéaire. Dans le cas de boîtes isothétiques congruentes ou de boîtes isothétiques contraintes, notre algorithme renvoie respectivement au plus $c \leq 2^{d-1}c^*$ et $c = O(c^*)$ points. De plus, nous prouvons que les bornes obtenues sur c sont précises et corroborons nos résultats théoriques par des performances pratiques. Nous décrivons également un algorithme adaptatif optimal qui calcule un ensemble de points de taille minimale perçant une famille d'intervalles.

1.3 Percer un ensemble d'objets : résultats combinatoires

Dans le chapitre 3, nous nous intéressons aux propriétés combinatoires d'objets convexes. Soit $\mathcal{S}$ un ensemble d'objets convexes dans l'espace euclidien $\mathbb{E}^d$ de dimension d. Force nous est de constater qu'un des plus réputés théorème en géométrie convexe est le théorème de Helly [DGK63, GW93a] qui établit le fait suivant :

> $\mathcal{S}$ est perçable par un point si et seulement si chaque sous-ensemble de $\mathcal{S}$ de cardinalité au plus $d+1$ est perçable par un point.

La preuve de ce théorème repose sur le lemme de Radon et ne donne pas directement naissance à un algorithme. Avis et Houle [AH91] décrivent les aspects algorithmiques du théorème de Helly. Ils donnent un algorithme en temps $O(n^{d+1})$ et espace mémoire $O(n^d)$ en utilisant les partitions de Radon. L'espace mémoire peut être rendu linéaire en multipliant par $O_d(1)$ le temps de calcul.

Le nombre de Helly $h = h(\mathcal{C}, P)$ associé à une classe d'objets $\mathcal{C}$ et une propriété P est le plus petit entier (s'il existe) tel que pour tout ensemble $\mathcal{S} \subseteq \mathcal{C}$ nous ayons: si chaque sous-ensemble $\mathcal{S}$ de cardinal h a la propriété P alors $\mathcal{S}$ a aussi la propriété P. Si un tel entier n'existe pas, nous fixons h à ∞.

Soit $\prod^k$ la propriété d'être perçable par k points et $\mathcal{C}^d$ la classe des objets convexes d-dimensionnels. Alors, le théorème de Helly se réécrit $h(\mathcal{C}^d, \prod^1) = d+1$.

Il existe de nombreux théorèmes de Helly (cf.[GW93a] pour un état de l'art dernièrement actualisé en 1993) en géométrie convexe.

Danzer et Grünbaum [DG82] étudièrent les boîtes isothétiques $\mathcal{B}^d$ (parallélotopes) dans l'espace euclidien d-dimensionnel. Ils établirent les théorèmes de type Helly pour $\mathcal{B}^d$. Ils prouvèrent en particulier que $h(\mathcal{B}^d, \prod^k) = \infty$, pour $d, k \geq 3$. Leur papier mentionne la conjecture suivante:

> $h(\mathcal{T}^d(K), \prod^2) < \infty$ si et seulement si K est un polytope d-dimensionnel, où $\mathcal{T}^d(K)$ est la classe de tous les translatés de K.

Cette conjecture a été récemment réfutée par Katchalski and Nashtir [KNar], qui prouvèrent que $h(\mathcal{T}^2(K), \prod^2) = \infty$, pour un hexagone K.

Les problèmes étudiés dans la partie 3 sont en relation directe avec un ensemble de problèmes d'optimisation :

Soit $\mathcal{P}$ un ensemble de n points de l'espace euclidien $\mathbb{E}^d$ et K un objet convexe à symétrie centrale. Soit o le centre de symétrie et soit K_λ un homothète de K ayant comme facteur d'homothétie λ. K définit une fonction de semi-distance comme suit: pour deux points a et b, $d_K(a,b) = \lambda$, si K_λ est le plus petit homothète de K qui contient b lorsqu'il est centré en a.

Dans ces problèmes d'optimisation, nous cherchons le plus petit réel λ tel que $\mathcal{P}$ puisse être couvert par k translatés de K_λ. Ces problèmes admettent toujours une solution en fixant λ à une valeur arbitrairement grande. Le problème décisionnel associé à ces problèmes d'optimisation est le suivant : étant donné une valeur λ, déterminer si $\mathcal{P}$ peut être couvert par k translatés de K_λ, c'est-à-dire essayer de trouver k emplacements des copies de K_λ. Un point $p \in \mathcal{P}$ est contenu dans un translaté de K_λ centré en o si et seulement si $d_K(o,p) \leq \lambda$, et, puisque $d_K(\cdot,\cdot)$ est une fonction symétrique, si et seulement si $d_K(p,o) \leq \lambda$.

Soit $\mathcal{P}_{K_\lambda}$ l'ensemble obtenu à partir de $\mathcal{P}$ en remplaçant chaque point $p \in \mathcal{P}$ par le translaté K_λ qui est centré en p. Alors, $\mathcal{P}$ peut être couvert par k translatés de K_λ si et seulement si $\mathcal{P}_{K_\lambda}$ est perçable par k points.

Généralement, une fois que le problème décisionnel est résolu de manière algorithmique efficace, nous l'incorporons dans un algorithme de recherche paramétrique afin d'obtenir une solution efficace pour le problème d'optimisation originel (cf. la recherche paramétrique de Megiddo [Meg83a]). Par exemple, considérons le problème des "deux centres" (2-DISK PROBLEM). Dans ce problème, nous souhaitons couvrir $\mathcal{P}$ par deux disques congruents de rayon minimum. Le problème décisionnel correspondant est : étant donné une valeur du rayon, disons $r = 1$, déterminer si $\mathcal{P}$ peut être couvert par deux disques de rayon unité. C'est encore équivalent au problème suivant:

Soit $\mathcal{P}_K$ l'ensemble des disques unités centrés aux points de $\mathcal{P}$. Il faut déterminer si $\mathcal{P}_K$ est perçable par deux points (cf. figure 1.1). Récemment, Sharir [Sha96] a présenté un algorithme quasi-linéaire (c'est à dire $O(n \log^3 n)$) pour résoudre le problème décisionnel. Il en découle à l'aide de la recherche paramétrique un algorithme pour le problème d'optimisation correspondant en temps $O(n \log^9 n)$. Ce dernier résultat fut amélioré par Eppstein qui proposa un algorithme randomisé en temps $O(n \log^2 n)$.

Hadwiger [HD60b] montra que si K est un objet strictement convexe à symétrie centrale (par exemple un disque) alors $h(\mathcal{T}^2(K), \prod^2) = \infty$. Ceci implique immédiatement que le problème des "deux disques" n'est pas un problème de type PL, c'est-à-dire programmation linéaire (LP-type), puisque chaque problème de type PL implique un théorème de Helly [Ame94].

Amenta [Ame94] étudia les relations entre les problèmes de type PL et les théorèmes de Helly. On renvoie le lecteur à l'article fondateur de Matoušek *et al.* [MSW92] pour une définition des problèmes de type PL et un algorithme

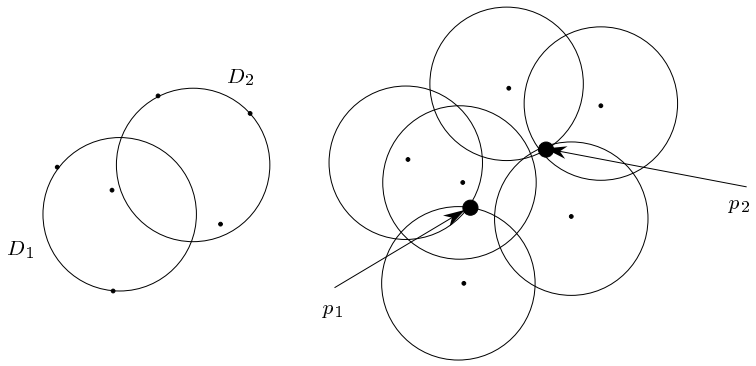


FIG. 1.1 – *Correspondance entre la couverture d'un ensemble de n points par deux disques unités et la perçabilité d'un ensemble de n disques par deux points*

randomisé en temps linéaire. Amenta montra que chaque problème de type PL a nécessairement un nombre de Helly mais pas forcément le contraire. Cependant, elle donna deux paradigmes pour obtenir des problèmes de type PL à partir de théorèmes de Helly (on obtient des systèmes de programmation linéaire paramétrés). Dans le chapitre 3, les théorèmes de Helly donneront directement naissance aux problèmes de type PL. Ainsi, nous obtenons des algorithmes randomisés en temps linéaire pour les problèmes de couverture correspondants. Ces algorithmes randomisés peuvent être également dérandomisés, à leur tour, en utilisant la technique de dérandomisation décrite dans l'article de Chazelle et Matoušek [CM93]. Cependant, ces algorithmes sont beaucoup plus compliqués que les nôtres. De plus, nos algorithmes sont les preuves même de l'existence des théorèmes de Helly que nous obtenons.

La table 1.1 résume nos principaux résultats. Nous montrons également:

(i) un algorithme simple en $O(n \log^2 n)$ pour le problème décisionnel des "deux disques" quand le diamètre de l'ensemble de points est plus grand que $3r$, où r est le rayon des disques couvrants.

(ii) le fait que $h(\mathcal{C}, \prod^2) = \infty$, où $\mathcal{C}$ est la classe des polygones convexes à 4 côtés.

Tous ces résultats améliorent de façon significative la solution naïve qui consiste à calculer les sommets de l'arrangement de $\mathcal{S}$ qui est de taille $O(n^d)$, puis de vérifier pour chaque sous-ensemble de k sommets si $\mathcal{S}$ est perçable par ces points. Ainsi, la solution naïve demande un temps de calcul en $O(n^{dk+1})$ pour k une constante fixée. Cette solution peut être légèrement améliorée: chaque sommet de l'arrangement est étiqueté par le sous-ensemble d'objets de

Objets	k	Temps
triangles homothétiques	2	$O(n)$ (Helly)
polygones 4, 5-orientés	2	$O(n \log n)$
polytopes d-dim. c-orientés	2	$O(n^{\min\{\lfloor \frac{c}{2} \rfloor, d\}} \log n)$
simplexes $(d+1)$-orientés	2	$O(n^{\lceil \frac{d}{2} \rceil} \log n)$
boîtes d-dim.	2	$O(n)$ (Helly)
triangles homothétiques	3	$O(n \log n)$

TAB. 1.1 – *Résumés des principaux résultats présentés dans le chapitre 3.*

$\mathcal{S}$ qu'il perce. Un sommet est dit *maximal* si son sous-ensemble correspondant n'est pas contenu dans le sous-ensemble défini par un autre sommet. Clairement, s'il existe une solution, alors il existe une solution qui consiste en k points provenant des sommets maximaux de l'arrangement de $\mathcal{S}$. Cependant, le nombre de cellules maximales peut être aussi grand que $\Omega(n^d)$. En effet, considérez l'ensemble suivant constitué de n objets convexes: pour chaque direction x_i, $i = 1, \ldots, d$, nous prenons les n/d objets $[0, n/d]^d \cap (j < x_i < j+1)$, $j = 0, \ldots, n/d - 1$. Chaque cellule d-dimensionelle est maximale. Nous pouvons raffiner la solution naïve comme suit: pour chaque sous-ensemble de $k-1$ cellules, nous vérifions si l'ensemble des objets non perçés par ces cellules maximales est perçable par un point, c'est-à-dire a une intersection non vide (c'est un problème de type PL). Ainsi, nous obtenons un algorithme en temps de calcul moyen $\tilde{O}(n^{d(k-1)+1})$.

1.4 Conclusion et problèmes ouverts

Dans les deux prochains chapitres, nous montrons deux visions différentes de l'étude de la perçabilité. L'une d'elle consiste à construire des heuristiques qui évitent de calculer la matrice d'incidence A tout en garantissant un facteur raisonnable de l'optimal. Nous obtenons un algorithme adaptatif dont la précision dépend elle-même de la valeur de l'optimal. Les problèmes soulevés lors de l'étude d'heuristiques sont les suivants:

– la relation exacte entre c^* et b^*, o ù b^* est la taille maximale d'un ensemble de boîtes mutuellement disjointes (nous obtenons à partir des nombres $N(p, q; d)$ l'inégalité $c^* \leq \binom{b^*-1+d}{d}$).

- la difficulté d'approximation des boîtes contraintes comparée aux boîtes non contraintes dans la hiérarchie polynomiale des problèmes.
- Pouvons nous trouver un algorithme garantissant un certain facteur entre b, le nombre d'un ensemble maximal de boîtes mutuellement disjointes, et b^* dans le cas de boîtes non contraintes ?
- Dans le papier de Imai et Asano [IA83], un algorithme est donné pour calculer une cellule maximale d'un arrangement de n boîtes isothétiques en temps $O(n^{d-1} \log n)$. Pouvons nous améliorer cet algorithme ? Ceci permettrait d'augmenter l'efficacité de l'algorithme glouton.

Une autre approche de l'étude de la perçabilité est basée sur l'étude de propriétés combinatoires de classes d'objets convexes pour des valeurs données de k. On peut voir les algorithmes donnés dans le chapitre 3 comme des algorithmes adaptatifs même si dans la complexité de ces algorithmes n'apparait qu'un seul paramétre. En effet, pour savoir si un ensemble est perçable par k points, on testera successivement s'il est perçable par 1, ..., $k-1$ et k points par des algorithmes de complexités respectives $T_1(n)$, ..., $T_{k-1}(n)$ et $T_k(n)$. Soit k_0 le nombre minimal de points requis pour percer $\mathcal{S}$, alors notre algorithme aura une complexité en $\sum_{i=1}^{k_0} T_i(n)$. Nos algorithmes sont donc en ce sens adaptatif puisqu'ils dépendent de k_0. Parmi les problèmes connectés à cette étude, il serait intéressant d'obtenir des résultats similaires pour les nombres de Galai et les nombres $C(p, q)$ [GW93a].

Chapitre 2

Fast Stabbing of Boxes in High Dimensions

(Cette partie est parue en rapport de recherche INRIA Numéro 2854 [Nie96b] et fût présentée sous forme résumée au 12$^{\text{e}}$ atelier de travail européen de géométrie algorithmique [Nie96c] ainsi qu'à la 8$^{\text{e}}$ conférence Canadienne de géométrie algorithmique [Nie96d]. Une version longue est actuellement soumise au journal Theoretical Computer Science. Nous présentons ci-dessous une version incluant certaines preuves et extensions omises dans [Nie96b].)

Abstract

We present in this technical report a simple yet efficient algorithm for stabbing a set $\mathcal{S}$ of n axis-parallel boxes in d-dimensional space with c points in output-sensitive time $O(dn + n \log c)$ and linear space. Let c^* be the minimum number of points required to stab $\mathcal{S}$, then we prove that

$$c \leq \min\{\frac{{c^*}^{\overline{d}}}{d!} + \frac{{c^*}^{\overline{d-1}}}{(d-1)!} - 1, c^* \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!}\},$$

where $x^{\overline{m}}$ is the rising factorial power: $x^{\overline{m}} = \prod_{i=0}^{m-1}(x+i) = m!\binom{x+m-1}{m}$. Since finding a minimal set of c^* points is NP-complete as soon as $d > 1$, we obtain a fast precision-sensitive heuristic for stabbing $\mathcal{S}$ in output-sensitive time and linear space. In the case of congruent or 'constrained' isothetic boxes, our algorithm reports respectively $c \leq 2^{d-1}c^*$ and $c = O(c^*)$ stabbing points. Moreover, we show that the bounds we get on c are tight and corroborate

our results with some experiments. We also describe an optimal output-sensitive algorithm for finding a minimal-size optimal stabbing point-set of intervals. Finally, we conclude with insights for further research.

Keywords: Computational geometry, Output-sensitive algorithms.

2.1 Setting the problem

Let $\mathcal{S}$ be a set of n d-dimensional geometric objects of constant descriptive-size. We say that $\mathcal{S}$ is stabbed by k points if there exist k points so that each object of $\mathcal{S}$ contains at least one of these points. Thus, $\mathcal{S}$ can be immobilized under translation with k points. Given a set $\mathcal{S}$ as above, finding the minimum k so that $\mathcal{S}$ can be stabbed by k points has been shown to be NP-complete [FPT81] as soon as $d \geq 2$[1]. Therefore this problem is untractable for small values of n (say $n \simeq 20$ and $d = 2$ – See section 2.3.4). This problem is also referenced in the literature as the *set covering problem* (or dually as the *hitting set problem*) where it is transformed into an optimization problem by means of matrix formulations. Let $\mathcal{V} = \{S_i | i \in I\}$ be a collection of $v = |\mathcal{V}| = |I|$ subsets of $2^{\mathcal{S}}$ for a set $\mathcal{S}$ of n elements. We want to find a minimal covering collection, i.e. a sub-set $I' \subseteq I$ of indices so that $\mathcal{S} = \bigcup_{i \in I'} S_i$ with $|I'|$ as small as possible. In other words, we want to minimize $e^T \times x = |I'|$ subject to $Ax \geq e$ for x a $\{0,1\}^n$-vector, $e = (1, ..., 1)$ and A a $(v \times n)$-binary matrix, each column of which is the incidence vector of one of the sets I_i, $1 \leq i \leq v$.

Some heuristics that give approximation of the minimum stabbing number c^* have been given. V. Chvátal [Chv79] gave a polynomial time (cubic) greedy algorithm to find a cover set of size c such that $c \leq c^*(1+\log k)$ where k is the maximum column sum ($k \leq n$). Very recently, this bound was improved to $c \leq c^*(\log k - \log\log k + \Theta(1))$ by P. Slavík [Sla96]. D.S. Hochbaum [Hoc82] proposed another cubic algorithm with a cover set of size at most $c^* f$, where f is the maximum row sum. Interestingly, Bellare et al. [BGLR93] showed that no polynomial time algorithm can approximate the optimal solution within a factor of $(\frac{1}{8} - \epsilon)\log|\mathcal{S}|$, unless $NP \subseteq DTIME[n^{\log\log n}]$, where $\epsilon > 0$. This result has been recently succesfully extended to the best possible $\log n$ bound by U. Feige [Fei96].

One major drawback from the computational geometry point of view is that these methods do not consider geometrical objects nor their shapes.

1. More precisely, Fowler et al. [FPT81] showed that covering a set of points with fixed-size squares (the so-called BOX-COVER problem) is NP-complete as soon as $d > 1$.

(Although it has been shown that the intersection graph[2] of d dimensional convex objects can be arbitrary as soon as $d \geq 3$ [Weg67]). This means that we have to supply matrix A. One way to proceed is to consider from the whole arrangement of the objects all the sets defined by vertices. More precisely, to each vertex we associate the set of objects containing it (thus, the size of the matrix is $O(n^d) \times n$ and these algorithms require $O(n^{d+2})$ time and $O(n^{d+1})$ space).

D.S. Hochbaum and W. Maass [HM84] considered the case of geometrical objects and give polynomial approximation scheme (note that no fully approximation scheme exists unless P=NP — we refer the reader to the comprehensive text book [GJ79] for a complete explanation). Their method is innovative since it is general and takes into account the nature of the objects. It should be noted that their method applies only to geometric objects. Unfortunately, the running time of these algorithms are at least cubic and thus cannot handle a huge amount of data. Moreover, their algorithm considers sets of identical convex objects T, or dually covering sets of points with convex translates T^*. (T^* is the centrally symmetric convex object of T). Many applications coming from VLSI design, image processing and point location have to deal with large inputs [TF80]. Recently, H. Brönninman and M.T. Goodrich [BG94] investigate these problems using the Vapnik-Chervonenkis dimension (VC-dimension). They obtain precision-sensitive set covers if the VC-dimension[3] is bounded as it is generally the case when considering geometric objects. Their algorithm uses nontrivial concepts (and subroutines) such as set systems, ϵ-net, net finder, ... (see also [Mat91a]) and still relies on the fact that matrix A is computed beforehand.

In this paper, we are even more restrictive by considering the case of axis-parallel boxes in high dimensions (that are often considered in VLSI design, image processing and point location in d-dimensional euclidean space); for example, we are given a set of points in $\mathbb{E}^d$ and some hypercube H_d. We want to associate to each point a hypercube that contains it so that we minimize the number of hypercubes. In other words, we want to cover the point set with a minimum number of patches, i.e. translates of H_d. Throughout the paper, the boxes are considered to be closed, i.e. points on the boundary of box B stab B. Our main algorithm, described in section 2.3, will not require to compute the arrangement of the isothetic boxes[4]. Note that, we do not

2. The intersection graph of a set of objects is defined as follows: we associate to each object a node and there exists an edge between two nodes iff the corresponding objects intersect.

3. The VC-dimension of d-dimensional isothetic boxes is 2.

4. Computing the arrangement of a set of n isothetic boxes cost $O(n^d)$ time and space [PS85].

consider d as a constant in the sequel.

We give in this paper a simple algorithm and study its approximation factor[5].

This fast algorithm may be useful in many applications. More precisely, we give a truly output-sensitive $O(dn + n \log c)$-time algorithm that computes a set of c points stabbing the set of n d-dimensional boxes. Interestingly, we show that $c \leq \min\{\frac{c^{*\overline{d}}}{d!} + \frac{c^{*\overline{d-1}}}{(d-1)!} - 1, c^* \frac{(\log n+1)^{\overline{d-1}}}{(d-1)!}\}$ where c^* is the optimal value and $x^{\overline{m}}$ is the rising factorial power: $x^{\overline{m}} = \prod_{i=0}^{m-1}(x+i) = m!\binom{x+m-1}{m}$ and $x^{\overline{0}} = 1$. Moreover, we exhibit a generic example where this bound is matched. We can refine the complexity analysis to show that $c \leq 2^{d-1}c^*$ and $c = O(c^*)$ when dealing respectively with congruent isothetic boxes and 'constrained' boxes.

The paper is organized as follows:

In section 2.2, we consider the case of a family of n intervals and give an optimal $\Theta(n(\log c^* + 1))$-time algorithm that gives an optimal stabbing set of c^* points. We use this basic case in order to devise another algorithm and analyze its behavior.

In section 2.3, we enhance the algorithm in higher dimensional space and study both its running time and its approximation factor. We show that the given bounds are tight. We refine the analysis for sets of congruent isothetic and constrained isothetic boxes. We corroborate our theoritical results with experiments.

Finally, in section 2.4, we conclude and give several guidelines for future research.

2.2 An optimal algorithm for stabbing intervals

In this section, we consider the case of intervals, i.e. 1-dimensional boxes. Let $\mathcal{S}$ be a set of n intervals.

2.2.1 Principle

Finding the minimum value c^* so that $\mathcal{S}$ can be stabbed with c^* points is easy and already known in [DG82, HM84]. Consider the interval I that has the rightmost left endpoint p. I must be stabbed by a point and clearly, the best place to stab it is on the left endpoint p. We then remove all the intervals

5. α is an *approximation factor* of an algorithm A if $c \leq \alpha c^*$ where c^* is the optimal value and c is the value delivered by algorithm A.

stabbed by p and loop until all the intervals are stabbed. We thus obtain a minimal-size set of c^* points that stab $\mathcal{S}$. A straightforward algorithm based on these facts has running time $O(nc^*)$ with linear space. We show below how an adequate preprocessing can yield an optimal output-sensitive algorithm in time $\Theta(n(\log c^* + 1))$.

Remark 1. In dimension 2 (and therefore in higher dimensions), the rectangle R with the rightmost left edge is not necessary the one that has the topmost bottom edge so that we cannot exhibit a rectangle R where we can easily *a priori* compute the best place to pierce it.

Remark 2. Let $b^*(\mathcal{S})$ be the maximum size of any subset of pairwise disjoint boxes of $\mathcal{S}$. Clearly, $c^*(\mathcal{S}) \geq b^*(\mathcal{S})$. In the 1-dimensional case, the above algorithm shows that $c^*(\mathcal{S}) = b^*(\mathcal{S})$ (which is not the case in higher dimensions). This property is no longer true in higher dimensions. (For example, in dimension 2, we may have $c^*(\mathcal{S}) \geq \frac{3}{2}b^*(\mathcal{S})$.)

2.2.2 Getting an output-sensitive algorithm

The methodology consists in grouping the intervals into groups and to preprocess each group in order to answer efficiently queries [Cha95b, NY95]. Typically, our queries are of two kinds: "what are the intervals stabbed by a point p?" and "which interval has the rightmost left endpoint?". Moreover, we must be able to remove some of these intervals at some steps. We use the interval tree of McCreight [McC80, PS85] as the data structure for answering these queries. Assume we know an estimate $\tilde{c^*}$ of c^*. Then, we group the n intervals into $\lceil \frac{n}{\tilde{c^*}} \rceil$ groups of size at most $\tilde{c^*}$ and preprocess each group into a static interval tree[6] for a total cost of $O(\lceil \frac{n}{\tilde{c^*}} \rceil \tilde{c^*} \log \tilde{c^*}) = O(n \log \tilde{c^*})$. At some step i, we find the rightmost left endpoint p_i of the remaining set of intervals and remove the n_i intervals stabbed by p_i from their corresponding groups. Thus the total cost of this step is $O(n_i \log \tilde{c^*} + \frac{n}{\tilde{c^*}} \log \tilde{c^*})$ (see [PS85] pp 352–355). Therefore, the total cost of these c^* steps is $O(n \log \tilde{c^*} + \frac{n}{\tilde{c^*}} c^* \log \tilde{c^*})$ time since $\sum_{i=1}^{c^*} n_i = n$. If we only want to know if $c^* > p$ we can derive a $O(n \log p)$-time algorithm by choosing $\tilde{c^*} = p$ and stopping the iterative process as soon as we have computed $i = \min\{c^*, p\}$ stabbing points. Note that our algorithm works in time $O(n(\log c^* + 1))$ if $c^* \leq \tilde{c^*} < {c^*}^2$. Since we do not know c^* beforehand we iteratively estimate it by squaring our current estimate. We start with any arbitrary value for $\tilde{c^*}$, say $\tilde{c^*} = 2$. Thus, we obtain an $O(\sum_{i=0}^{\lceil \log \log c^* \rceil} n 2^i) = O(n \log c^*)$ time algorithm.

6. In this context, static means that we know beforehand the $2\tilde{c^*}$ endpoints of each group. We only remove intervals and not add new ones to that data-structure.

TAB. 2.1 – *The table below shows the value of i as a function of c^*. We estimate c^* by $\tilde{c^*} = 2^{2^i}$. For example, for $2^{16} = 65536 \leq c^* < 2^{32}$, we set $\tilde{c^*}$ to 2^{2^4}.*

$c^* =$	4	16	256	65536	2^{32} (10 digit-number)
i ($\tilde{c^*} = 2^{2^i}$) $= 0$	1	2	3	4	

Since verifying if among n numbers k are distinct requires $\Omega(n \log k)$ time on the real RAM [KS86], it follows that this lower bound also holds for the stabbing problem by reduction in linear time. Therefore, we obtain the following theorem:

Theorem 8 *Given a set $\mathcal{S}$ of n intervals, there exists an optimal output-sensitive algorithm that reports an optimal stabbing point set of size c^* in optimal $\Theta(n \log c^*)$ time with linear space.*

Remark. As a direct consequence, we obtain a $\Theta(n \log c^*)$-time algorithm for computing the union of a n-interval set $\mathcal{S}$, where c^* is the minimal number of points required to stab $\mathcal{S}$. Note that it is not possible to get an $O(n \log C(\mathcal{S}))$-time algorithm for computing the union of intervals, where $C(\mathcal{S})$ is the number of connected components of $\mathcal{S}$. (We may have $c^*(\mathcal{S}) = \lceil \frac{n}{2} \rceil$ but $C(\mathcal{S}) = 1$.)

2.2.3 A fast algorithm and its analysis

One major drawback when implementing the previous algorithm is the preprocessing step (or *guessing step* when we find a good estimate $\tilde{c^*}$ of c^*) that takes into account only a few critical values, namely the values in $\{2^{2^i} | i \in \mathbb{N}\}$. (That is the width $2^{2^i}(2^{2^i} - 1)$ of the i-th range, where the group size does not change, is rapidly growing and therefore the preprocessing step becomes inefficient in practice). We only make at most 4 guessing steps in practice — See Table 2.1.

Moreover, we have $c^* \leq \tilde{c^*} < {c^*}^2$ so that in the worst-case, we choose $\tilde{c^*} = {c^*}^2 - 1$ in order to adapt our preprocessing in time $O(n) \log {c^*}^2 = 2O(n) \log c^*$. This yields a factor of 2 in the hidden constant. By drawing the graph of the complexity function, we see that it is always under the graph of a function $an \log c^*$ for some real a and that the function is linear by intervals. The size of these intervals however grows quadratically and explains the poor behavior in practice of that algorithm.

We propose below a very simple "divide & conquer" algorithm :

Basic case. If $n = 1$ then choose the left endpoint of $I \in \mathcal{I} = \{I\}$ for piercing $\mathcal{I}$.

Recurse. Find the median m of the $2n$ endpoints and partition $\mathcal{I}$ depending on whether the intervals contain m ($\mathcal{I}_m$), are located to the left of m ($\mathcal{I}_1$) or to the right of m ($\mathcal{I}_2$). Choose point m to stab the intervals of $\mathcal{I}_m$ and recurse on $\mathcal{I}_1$ and $\mathcal{I}_2$.

Let $c^*(\mathcal{I})$ denote the minimum number of points required for stabbing $\mathcal{I}$ and denote by $c(\mathcal{I})$ the number of points returned by the algorithm. Observe that $\mathcal{I}$ can be stabbed with a single point iff there are n left endpoints followed by n right endpoints. Let x_n and x_{n+1} be respectively the n-th and $(n+1)$-th smallest endpoints of the $2n$ endpoints. Then, we compute our median $m = \frac{x_n + x_{n+1}}{2}$ in linear time [BFP+72].

Let us prove that $c(\mathcal{I}) \leq 2c^*(\mathcal{I}) - 1$.

Proof. We have:

$$c(\mathcal{I}) = \begin{cases} c(\mathcal{I}_m) = c^*(\mathcal{I}) = 1 & \text{if both } \mathcal{I}_1 \text{ and } \mathcal{I}_2 \text{ are empty,} \\ 1 + c(\mathcal{I}_1) + c(\mathcal{I}_2) & \text{otherwise.} \end{cases}$$

We can modelize the running of the algorithm with a binary tree. Each node has a subset of intervals $\mathcal{I}_m$ and a point stabbing $\mathcal{I}_m$. Its left (right) child is $\mathcal{I}_1$ (resp. $\mathcal{I}_2$). Now, if $\mathcal{I}_1 \neq \emptyset$ and $\mathcal{I}_2 \neq \emptyset$ then we have at least two disjoint intervals. A leaf of this tree is a set of intervals that are stabbed by a single point. Thus, the number of stabbing points reported by our algorithm is the number of nodes of that tree. The number of leaves of the tree is bounded by the maximum number b^* of pairwise disjoint intervals. Clearly, $c^*(\mathcal{I}) \geq b^*$. The tree is a strictly binary tree, i.e. each internal node has exactly two children. Thus, there are $b^* - 1$ internal nodes and $c(\mathcal{I}) \leq 2b^* - 1 \leq 2c^*(\mathcal{I}) - 1$.

The trick is to ensure that whenever $\mathcal{I}_1 \neq \emptyset$ (resp. $\mathcal{I}_2 \neq \emptyset$) then $\mathcal{I}_2 \neq \emptyset$ (resp. $\mathcal{I}_1 \neq \emptyset$). This comes from the fact that we have $|\mathcal{I}_1| = |\mathcal{I}_2|$.

Note that we can take $m = x_n$ instead of $m = \frac{x_n + x_{n+1}}{2}$. We obtain the same result.

□

One may ask whether this bound is tight or not. We can prove the tightness of this result by building a family $\mathcal{I}$ of intervals such that $c(\mathcal{I}) = 2c^*(\mathcal{I}) - 1$: Let $\mathcal{I}_1 = \{[0,1], [\frac{1}{2}, \frac{5}{2}], [2,3]\}$. We have $c(\mathcal{I}_1) = 3 = 2 * c^*(\mathcal{I}_1) - 1$. Then, we can recursively set in the intervals $[\frac{1}{2}, 1]$ and $[2, \frac{5}{2}]$ another 'scaled' copy of $\mathcal{I}_1$ and so on, ... Thus, we are able to build a family $\mathcal{I}$ of $3 \times (2^i - 1)$ intervals (for any $i \geq 1$) so that $c(\mathcal{I}) = 2c^*(\mathcal{I}) - 1 = 2^{i+1} - 1$. Indeed, we have by construction $c^*(\mathcal{I}_{i+1}) = c^*(\mathcal{I}_i) + c^*(\mathcal{I}'_i)$, $c^*(\mathcal{I}_i) = c^*(\mathcal{I}'_i)$, $c(\mathcal{I}_i) = c(\mathcal{I}'_i)$ and $c(\mathcal{I}_{i+1}) = c(\mathcal{I}_i) + c(\mathcal{I}'_i) + 1 = 2c(\mathcal{I}_i)$ where $\mathcal{I}'_i$ is a translated copy of $\mathcal{I}_i$. The proof follows by induction on i.

Let $t(\mathcal{I})$ denote the running time of the algorithm ($n = |\mathcal{I}|$). Then, we have:

$$t(\mathcal{I}) = \begin{cases} An & \text{if } c(\mathcal{I}) = 1, \\ Bn + t(\mathcal{I}_1) + t(\mathcal{I}_2) & \text{otherwise.} \end{cases}$$

with A, B some constants. We can prove by induction on n that $t(\mathcal{I}) \leq Cn \log c + Cn$ where $c = c(\mathcal{I})$ and $C = \max\{A, B\}$.

Proof. Clearly, if $n = 1$ then $t(\mathcal{I}) = A \leq C \log c + C$ (note that $c = 1$). Otherwise, either $t(\mathcal{I}) = An \leq Cn \log c + Cn$ if $c(\mathcal{I}) = c^*(\mathcal{I}) = 1$ or

$$t(\mathcal{I}) \leq Bn + Cn' \log c_1 + Cn' \log c_2 + 2Cn',$$

with $n' \leq \frac{n}{2}$ and $c_1 + c_2 + 1 = c$. Thus,

$$t(\mathcal{I}) \leq Bn + C\frac{n}{2} \max_{c_1+c_2=c} \{\log c_1 + \log c_2\} + Cn.$$

The last inequality is maximized for $c_1 = c_2 \leq \frac{c}{2}$ since the logarithmic function is concave[7]. Therefore, we have:

$$t(\mathcal{I}) \leq Bn + Cn \log c - Cn + Cn \leq Cn \log c + Cn.$$

□

Theorem 9 *Let $\mathcal{S}$ be a family of n intervals and denote by c^* the minimal number of points required to stab $\mathcal{S}$. Then, there exists a simple output-sensitive algorithm that computes a set of c points stabbing $\mathcal{S}$ in time $O(n \log c)$ and linear space, with $c \leq 2c^* - 1$.*

Remark 1. If we pick a randomly chosen endpoint x of $\mathcal{I}$ and set $m = x$ then we obtain $c(\mathcal{I}) \leq (1 + \lceil \log n \rceil)c^*(\mathcal{I})$ instead of the output-sensitive bound $c(\mathcal{I}) \leq 2c^*(\mathcal{I}) - 1$.

Remark 2. We may get a (truly) fast algorithm that reports an optimal stabbing point set by first running the above approximate algorithm and getting a set of c points so that $c^* \leq c \leq 2c^* - 1$. In a second step, we choose $\tilde{c^*} = c$ and run the optimal algorithm described in section 2.2.2. Thus, the graph of the complexity function of this new algorithm is not anymore linear by intervals.

Remark 3. Although the greedy algorithm [Chv79] performs generally a H_n factor from the optimum ($H_n = \sum_{i=1}^{n} \frac{1}{i}$ and $H_n \leq 1 + \log n$), for the

7. Clearly, if f is a concave function then we have $f(a) + f(b) \leq 2f(\frac{a+b}{2})$.

case of intervals we show below that it will return at most $2c^* - 1$ points. Moreover, there is a family of intervals where the greedy algorithm attains this worst-case bound.

Proof. Let $\mathcal{I}$ be a set of intervals and $\mathcal{Q}$ an optimal set of points piercing $\mathcal{I}$. Consider two adjacent points (abscissæ) Q_1 and Q_2 of $\mathcal{Q}$. Then, the greedy algorithm will pick at most one point in the range $]Q_1, Q_2[$. Indeed, once it has picked one point Q in $]Q_1, Q_2[$ and removed all the intervals stabbed by Q, all the intervals intersecting $]Q_1, Q_2[$ intersect either Q_1 or Q_2 and therefore the maximal cells are located at the extremities. Therefore, the greedy algorithm will return at most $(c^*(\mathcal{I}) - 1) + c^*(\mathcal{I})$ points, i.e. $2c^*(\mathcal{I}) - 1$ points. □

2.3 The algorithm in higher dimensions

2.3.1 Principle

The "divide-and-conquer" strategy holds in any dimension and for any kind of objects. We show in that section how we can get results on the approximation factor when dealing with axis-parallel boxes. Let $\mathcal{S}$ be a set of n d-dimensional boxes. Each box can be viewed as the intersection of $2d$ halfspaces. A facet f of a box B is a $(d-1)$-dimensional box of the boundary ∂B supported by a hyperplane H_f. If H_f is defined by an equation of type $x_i = l$ for some real l then we say that f is a facet of *type* i. In other words, a facet of type i is perpendicular to the i-th axis. A box B can be viewed as the ordered cartesian product $\prod_{i=1}^{d}[r_i^-(B), r_i^+(B)]$ where $[r_i^-(B), r_i^+(B)]$ is the range of B along the i-th dimension. We say that box B is to the *left* (*right*) of $x_i = l$ if $r_i^+(B) < l$ (resp. $r_i^-(B) > l$). Let $X^{(d)}$ be the set of values defining the facets of type d, i.e. $X^{(d)} = \{x | \exists\ B \in \mathcal{S}$ such that $r_d^-(B) = x$ or $r_d^+(B) = x\}$.

We describe below the algorithm (see also Figure 2.1):

Intervals (Basic case). If $\mathcal{S}$ is one-dimensional then apply the optimal algorithm of section 2.2.2 for piercing this set of intervals.

Partition. Let $x_n^{(d)}$ and $x_{n+1}^{(d)}$ be respectively the n-th and $(n+1)$-th greatest elements of $X^{(d)}$. Compute the 'median' $m = \frac{x_n^{(d)} + x_{n+1}^{(d)}}{2}$ of $X^{(d)}$ in linear time [BFP+72]. Partition $\mathcal{S}$ according to the hyperplane $H_m : (x_d = m)$;

- Let $\mathcal{S}_1$ be the set of boxes that do not cross H_m and are to the left of H_m.
- Let $\mathcal{S}_2$ be the set of boxes that do not cross H_m and are to the right of H_m.

– Let $\mathcal{S}_m$ be the set of boxes intersecting H_m.

Marriage. Stab the boxes of $\mathcal{S}_m$ by piercing the set of $(d-1)$-dimensional boxes: $\mathcal{S}'_m = \{B \cap H_m | B \in \mathcal{S}_m\}$.

Conquest. Stab recursively $\mathcal{S}_1$ and $\mathcal{S}_2$.

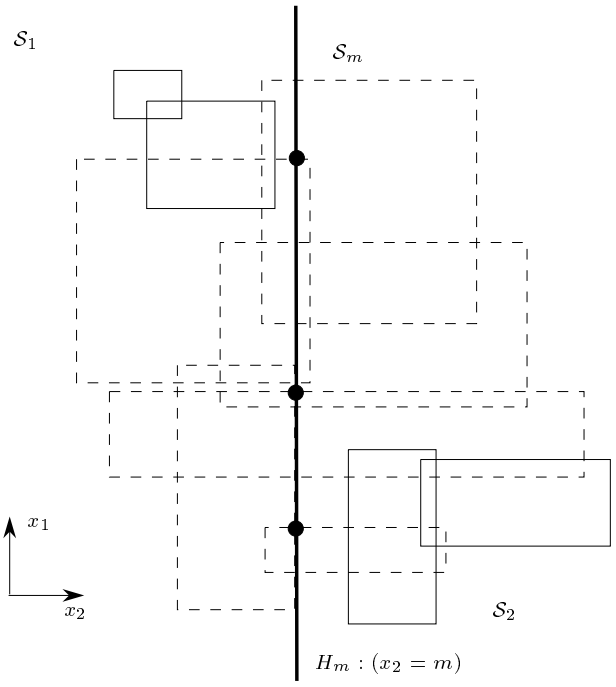


FIG. 2.1 – *Partition of $\mathcal{S}$ into three subsets depending on their location with respect to the hyperplane $H_m : (x_d = m)$. We denote by $\mathcal{S}'_m$ the set of $(d-1)$-dimensional boxes $\mathcal{S}_m \cap H_m$.*

Let $t(\mathcal{S})$ and $c(\mathcal{S})$ be respectively the running time of the algorithm and the number of stabbing points delivered by this algorithm. Sometimes, when we want to specify the dimension d of $\mathcal{S}$, we put in subscript of these notations a d. Thus, $t_d(\mathcal{S})$ and $c_d(\mathcal{S})$ denote respectively the running time and the output size of our algorithm for a set of d-dimensional isothetic boxes S. Denote by $c^*(\mathcal{S})$ the minimum number of stabbing points of set $\mathcal{S}$. We study both the approximation factor and the running time of the algorithm. In the sequel, d is not assumed to be a constant.

Our algorithm relies on the following simple facts:

Monotonicity. For any O, $c^*(\{O\} \cup \mathcal{S}) \geq c^*(\mathcal{S})$.

Additive rule. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two subsets so that $\forall I_1 \in \mathcal{I}_1$, $\forall I_2 \in \mathcal{I}_2, I_1 \cap I_2 = \emptyset$ then $c^*(\mathcal{I}_1 \cup \mathcal{I}_2) = c^*(\mathcal{I}_1) + c^*(\mathcal{I}_2)$.

Cutting rule. Let $\mathcal{S}$ be a set of boxes and H a hyperplane of type i, with $1 \leq i \leq d$. Then, $c^*(\mathcal{S}_H) = c^*(\mathcal{S}'_H)$ where $\mathcal{S}'_H = \{B \cap H | B \in \mathcal{S}\}$ and $\mathcal{S}_H = \{B | B \cap H \neq \emptyset\}$. This rule seems to be appropriate only for isothetic d-boxes.

We have:

$$c_d(\mathcal{S}) = \begin{cases} c_{d-1}(\mathcal{S}_m \cap H_m) & \text{if } \mathcal{S}_1 = \mathcal{S}_2 = \emptyset, \\ c_{d-1}(\mathcal{S}_m \cap H_m) + c_d(\mathcal{S}_1) + c_d(\mathcal{S}_2) & \text{otherwise.} \end{cases}$$

Let us prove by induction on the lexicographically ordered vector (d, n) that $c \leq \frac{c^*(\mathcal{S})^{\overline{d}}}{d!} + \frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} - 1$.

Proof.

For $d = 1$, section 2.2.2 describes an optimal algorithm so that $c(\mathcal{S}) = c^*(\mathcal{S}) \leq c^*(\mathcal{S}) + 1 - 1$ since $x^{\overline{0}} = 1$ by convention (finite calculus rules may be found in [GKP94]).

If $|\mathcal{S}| = n = 1$ then $c(\mathcal{S}) = c^*(\mathcal{S}) = 1 \leq \frac{1^{\overline{d}}}{d!} + \frac{1^{\overline{d-1}}}{(d-1)!} - 1$.

Otherwise ($d > 1$ and $n > 1$), there are two cases depending on whether $\mathcal{S}_1, \mathcal{S}_2 = \emptyset$ or not (recall that $|\mathcal{S}_1| = |\mathcal{S}_2|$).

If $\mathcal{S}_1 = \mathcal{S}_2 = \emptyset$ then we have:

$$c(\mathcal{S}) \leq \frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} + \frac{c^*(\mathcal{S})^{\overline{d-2}}}{(d-2)!} - 1 \leq \frac{c^*(\mathcal{S})^{\overline{d}}}{d!} + \frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} - 1,$$

since $\frac{(c^*(\mathcal{S})+d-1)(c^*(\mathcal{S})+d-2)}{d(d-1)} \geq 1$ (recall that $c^*(\mathcal{S}) \geq 1$).

If $\mathcal{S}_1, \mathcal{S}_2 \neq \emptyset$ then we have:

$$c_d(\mathcal{S}) = c_d(\mathcal{S}_1) + c_d(\mathcal{S}_2) + c_{d-1}(\mathcal{S}'_m),$$

$$c(\mathcal{S}) \leq \frac{c^*(\mathcal{S}_1)^{\overline{d}} + c^*(\mathcal{S}_2)^{\overline{d}}}{d!} + \frac{c^*(\mathcal{S}_1)^{\overline{d-1}} + c^*(\mathcal{S}_2)^{\overline{d-1}}}{(d-1)!} + \left(\frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} + \frac{c^*(\mathcal{S})^{\overline{d-2}}}{(d-2)!}\right) - 3,$$

with $1 \leq c^*(\mathcal{S}_1), c^*(\mathcal{S}_2)$ and $c^*(\mathcal{S}_1) + c^*(\mathcal{S}_2) \leq c^*(\mathcal{S})$ since $c^*(\mathcal{S}'_m) = c^*(\mathcal{S}_m) \leq c^*(\mathcal{S})$. Since the rising factorial power is convex, the right hand side of the

last inequality is maximized[8] for $c^*(\mathcal{S}_1) = 1$ and $c^*(\mathcal{S}_2) = c^*(\mathcal{S}) - 1$ (or the other way around). It follows that:

$$c(\mathcal{S}) \leq \frac{(c^*(\mathcal{S})-1)^{\overline{d}}}{d!} + \frac{(c^*(\mathcal{S})-1)^{\overline{d-1}}}{(d-1)!} + \left(\frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} + \frac{c^*(\mathcal{S})^{\overline{d-2}}}{(d-2)!}\right) - 3 + \frac{1^{\overline{d}}}{d!} + \frac{1^{\overline{d-1}}}{(d-1)!},$$

$$c(\mathcal{S}) \leq \frac{c^*(\mathcal{S})^{\overline{d-1}}}{d!}(c^*(\mathcal{S}) - 1 + d) + \frac{c^*(\mathcal{S})^{\overline{d-2}}}{(d-1)!}(c^*(\mathcal{S}) - 1 + d - 1) - 1,$$

and therefore

$$c(\mathcal{S}) \leq \frac{c^*(\mathcal{S})^{\overline{d}}}{d!} + \frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} - 1.$$

□

The tightness of the analysis will allow us to derive a collection of examples in section 2.3.2 such that $c(\mathcal{S}) = \frac{c^*(\mathcal{S})^{\overline{d}}}{d!} + \frac{c^*(\mathcal{S})^{\overline{d-1}}}{(d-1)!} - 1$.

Remark. For small values of $c^*(\mathcal{S})$ we get an approximation factor which is polynomial in d. As an example, consider $c^*(\mathcal{S}) = 3$ then we have $c(\mathcal{S}) \leq d(d+2)$ (the approximation scheme of Hochbaum and Maass [HM84] has an exponential dependence in d). However, when $c^*(\mathcal{S})$ is large (say $c^*(\mathcal{S}) > dn^{\frac{1}{d}}$) we have the trivial bound $c(\mathcal{S}) \leq n$. It does not reflect the dichotomy process. Therefore, we study the relationships between $c(\mathcal{S})$ and $|\mathcal{S}| = n$.

Let us prove that:

$$c(\mathcal{S}) \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!}. \qquad (\star)$$

Proof.

Recall that we denote by $c_d(\mathcal{S})$ the number $c(\mathcal{S})$ of stabbing points returned by our algorithm for a set of d-dimensional isothetic boxes $\mathcal{S}$ (this is an abuse of notation that might however help the reader when checking the proofs).

We do the induction on the lexicographically ordered vector (d, n). If $d = 1$ then section 2.2.2 gave an optimal algorithm, i.e. $c(\mathcal{S}) = c_1(\mathcal{S}) = c^*(\mathcal{S})$

8. Let $f(\cdot)$ be a convex function defined on range $[a, b]$ then $\max_{a \leq c \leq b}\{f(c)\} \leq \max\{f(a), f(b)\}$. Let $g_m(x) = x^{\overline{m}}$. $g_m(x)$ is convex on $[0, +\infty)$ for $m \geq 0$. Define $f(x) = g_m(x) + g_m(c-x) + g_{m-1}(x) + g_{m-1}(c-x)$ for $1 \leq x \leq c-1$. $f(x)$ is convex on $[1, c-1]$ and therefore $\max_{1 \leq x \leq c-1}\{f(x)\} \leq f(1)$.

and ($\star$) holds trivially since $x^{\overline{0}} = 1$ and $0! = 1$. If $n = 1$ then $c(\mathcal{S}) = c^*(\mathcal{S}) = 1 \leq 1 \times \frac{1^{\overline{d-1}}}{(d-1)!}$.

Otherwise ($d > 1$ and $n > 1$), we distinguish on whether $\mathcal{S}_1, \mathcal{S}_2 = \emptyset$ or not. In the former case, we have:

$$c(\mathcal{S}) = c_{d-1}(\mathcal{S}) \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-2}}}{(d-2)!} \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!},$$

since $\frac{\log n + 1 + d - 2}{d-1} \geq 1$ with equality for $n = 1$.

In the latter case, we have:

$$c_d(\mathcal{S}) = c_d(\mathcal{S}_1) + c_d(\mathcal{S}_2) + c_{d-1}(\mathcal{S}'_m),$$

with $|\mathcal{S}_1| = |\mathcal{S}_2| \leq \frac{n}{2}$ and $c_{d-1}(\mathcal{S}'_m) = c_d(\mathcal{S}_m) \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-2}}}{(d-2)!}$. Thus, we get

$$c_d(\mathcal{S}) \leq \Big(c^*(\mathcal{S}_1) + c^*(\mathcal{S}_2)\Big) \frac{(\log \frac{n}{2} + 1)^{\overline{d-1}}}{(d-1)!} + c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-2}}}{(d-2)!},$$

$$c_d(\mathcal{S}) \leq c^*(\mathcal{S}) \Big(\frac{(\log n)^{\overline{d-1}}}{(d-1)!} + \frac{(\log n + 1)^{\overline{d-2}}}{(d-2)!} \Big),$$

since $c^*(\mathcal{S}_1) + c^*(\mathcal{S}_2) = c^*(\mathcal{S}_1 \cup \mathcal{S}_2) \leq c^*(\mathcal{S})$.

$$c(\mathcal{S}) \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-2}}}{(d-1)!} (\log n + d - 1),$$

$$c(\mathcal{S}) \leq c^*(\mathcal{S}) \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!}.$$

□

Let us now analyze the time spent by this algorithm for reporting the $c_d(\mathcal{S})$ stabbing points.

We have:

$$t_d(\mathcal{S}) = \begin{cases} 0 & \text{if } \mathcal{S} = \emptyset, \\ An + t_{d-1}(\mathcal{S}'_m) + t_d(\mathcal{S}_1) + t_d(\mathcal{S}_2) & \text{otherwise.} \end{cases}$$

with A some constant related to the implementation of the partition scheme [BFP+72] (e.g., a typical value for A is 4 – see [BFP+72]).

Clearly, a trivial bound we can get is $t_d(\mathcal{S}) \leq Adn(2c(\mathcal{S})-1)$:

We prove below by induction on the lexicographically ordered vector (d,n) that $t_d(\mathcal{S}) \leq A(n\log c(\mathcal{S}) + dn)$ $(\star)$.

Proof. If $d=1$ then we proved in section 2.2.2 an $O(n\log c(\mathcal{S})+n)$-time algorithm. Therefore, $t_1(S) \leq Bn(\log c(\mathcal{S})+1) \leq An(\log c(\mathcal{S})+1)$ for $A \geq B$. If $n=1$ then $t_d(\mathcal{S}) \leq Ad$ and $(\star)$ holds trivially.

Otherwise ($d>1$ and $n>1$), consider the two cases depending on whether $\mathcal{S}_1, \mathcal{S}_2 = \emptyset$ or not: In the former case, we have $t_d(\mathcal{S}) = t_{d-1}(\mathcal{S}) + An$. Thus, we get

$$t_d(\mathcal{S}) \leq An\Big((d-1)+\log c(\mathcal{S})\Big) + An \leq An(d+\log c(\mathcal{S})).$$

In the latter case $(\mathcal{S}_1, \mathcal{S}_2 \neq \emptyset)$, we have:

$$t_d(\mathcal{S}) = An + t_d(\mathcal{S}_1) + t_d(\mathcal{S}_2) + t_{d-1}(\mathcal{S}'_m),$$

with $1 \leq |\mathcal{S}_1|, |\mathcal{S}_2| \leq \frac{n}{2}$, $|\mathcal{S}_1|+|\mathcal{S}_2|+|\mathcal{S}'_m| = n$ and $c(\mathcal{S}) = c(\mathcal{S}_1)+c(\mathcal{S}_2)+c(\mathcal{S}'_m)$. Let $n' = |\mathcal{S}_1| = |\mathcal{S}_2|$ $(|\mathcal{S}'_m| = n-2n')$. Hence,

$$t_d(\mathcal{S}) \leq A\Big(n + n'\log c(\mathcal{S}_1) + n'd \qquad +n'\log c(\mathcal{S}_2) + n'd \\ +(n-2n')\log c(\mathcal{S}'_m) + (n-2n')(d-1)\Big),$$

$$t_d(\mathcal{S}) \leq A\Big(nd + 2n' + n'(\log c(\mathcal{S}_1) + \log c(\mathcal{S}_2)) + (n-2n')\log c(\mathcal{S}'_m)\Big).$$

But $\log c(\mathcal{S}_1) + \log c(\mathcal{S}_2)$ is maximized when $c(\mathcal{S}_1) = c(\mathcal{S}_2) \leq \frac{c(\mathcal{S})}{2}$.
Therefore, we get

$$t_d(\mathcal{S}) \leq A\Big(dn + 2n'\log c(\mathcal{S}) + (n-2n')\log c(\mathcal{S}'_m)\Big),$$

with $c(\mathcal{S}'_m) \leq c(\mathcal{S})$.
Finally, we get $t_d(\mathcal{S}) \leq An(d+\log c(\mathcal{S}))$.
□

Note that when $c(\mathcal{S}) = c^*(\mathcal{S}) = 1$ then our algorithm requires $O(nd)$-time (this is easily checked). Let $N = O(nd)$ denote the input size of our set $\mathcal{S}$

of n d-dimensional boxes. Then, our algorithm runs in $O(N + \frac{N}{d}\log c)$-time and linear space.

Theorem 10 *Let $\mathcal{S}$ be a set of n d-dimensional boxes. Denote by c^* the minimum number of stabbing points of $\mathcal{S}$. Then, there exists an output-sensitive algorithm that reports a set of c stabbing points of $\mathcal{S}$ in time $O(dn + n\log c)$ with linear space whose approximation c is bounded by*

$$\min\{\frac{{c^*}^{\overline{d}}}{d!} + \frac{{c^*}^{\overline{d-1}}}{(d-1)!} - 1, c^*\frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!}\}.$$

Remark 1. It would be interesting to analyze probabilistically the algorithm under the uniform distribution of the isothetic d-dimensional boxes [Cof88]. Note that if the stabbing points are split into two balanced groups with respect to the hyperplane H_{m_s} at each stage s of the algorithm, then we get $c(\mathcal{S}) \leq c^*(\mathcal{S})\frac{(\log c^*(\mathcal{S})+1)^{\overline{d-1}}}{(d-1)!}$. We expect in practice far better results. This is mainly due to the fact that we maximize all the terms in the complexity analysis (e.g., $c(\mathcal{S}'_m), c(\mathcal{S}_1), c(\mathcal{S}_2)$) – See section 2.3.4. Indeed, it is unlikely that $c^*(\mathcal{S}'_m) = c^*(\mathcal{S})$.

Remark 2. As soon as we consider $d-boxes$ with $d \geq 2$, greedy algorithm may yield a H_n factor from the optimum.

Proof. We build a family $\mathcal{A}$ such that $c^*(\mathcal{A}) = 2$ of $n = 2^{k+2} - 2$ squares that is decomposed into two sub-families $\mathcal{A}_1$ and $\mathcal{A}_2$ of $2^{k+1} - 1$ squares each such that both $c^*(\mathcal{A}_1) = 1$ and $c^*(\mathcal{A}_2) = 1$. A planar box $B = [a, b]$ is defined by its leftmost bottomost corner $a = (x_a, y_a)$ and its rightmost uppermost corner $b = (x_b, y_b)$.

$\mathcal{A}_1$ is defined by 2^0 square $[(-k, 0), (0, k)]$, ..., 2^i squares $[(-k-i, -i), (-i, k-i)]$, ..., and 2^k squares $[(-2k, -k), (-k, 0)]$. Similarly, $\mathcal{A}_2$ is defined by 2^0 square $[(0, -k), (k, 0)]$, ..., 2^i squares $[(-i, -k - i), (k - i, -i)]$, ..., and 2^k squares $[(-k, -2k), (0, -k)]$ (see Figure 2.2). Clearly, the greedy algorithm will return k points $(-k, -k)$, ..., $(-k + i, -k + i)$, ... and $(0, 0)$ stabbing $\mathcal{A}$ instead of an optimal solution of two points (e.g., p_1 and p_2). □

2.3.2 A bad example

Let us now analyze the tightness of the upper bounds. In that section we assume w.l.o.g. that $m = x_n^{(d)}$ (see the algorithm in section 2.3.1).

Consider first the planar case. We want to exhibit an example showing that our algorithm reports $\Omega(\min\{c^{*2}, 1 + c^*\log n\})$ stabbing points for a family of planar boxes (we proved, for the planar case, that

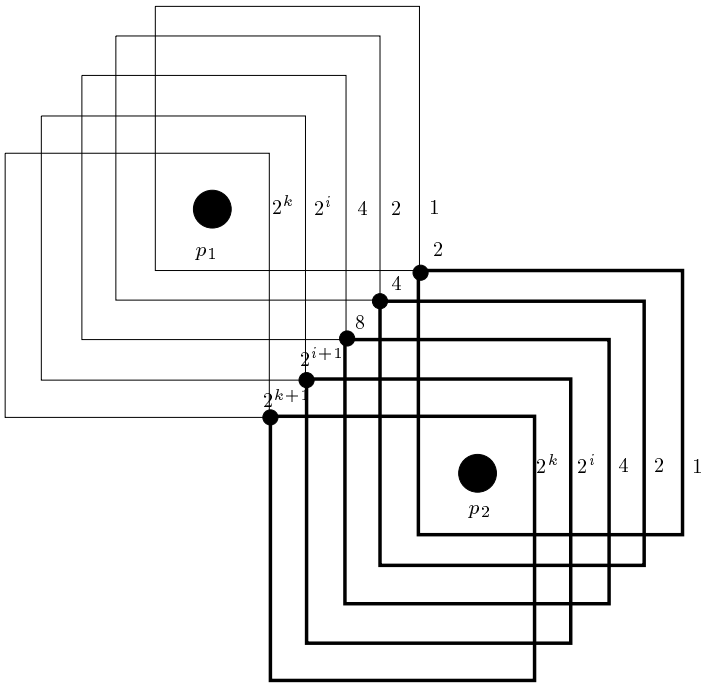


FIG. 2.2 – *Building a family $\mathcal{A}$ such that the greedy algorithm returns $\Omega(\log n)c^*(\mathcal{A})$ points.*

$c(\mathcal{S}) \leq \min\{c^*(\mathcal{S})\frac{c^*(\mathcal{S})+3}{2} - 1, c^*(\mathcal{S})(1+\log n)\})$. In the following, we define a box by its S.W. (leftmost bottommost) and N.E. (rightmost uppermost) corners.

Let E_{c^*} be a family of boxes requiring exactly c^* points to be stabbed. Denote by $|E_{c^*}|$ its cardinality, i.e. the number of boxes of E_{c^*}. We use a parameter b in order to build recursively E_{c^*} (initially, $b = 1$ and $\forall b, b',\ |E_{c^*}(b)| = |E_{c^*}(b')|$, in other words $|E_{c^*}(b)|$ does not depend on b).

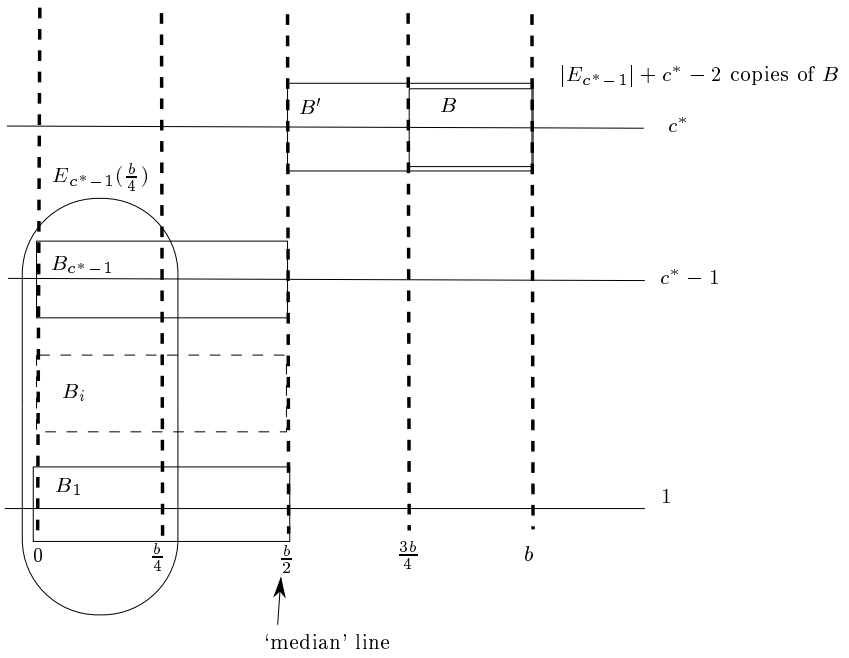


FIG. 2.3 – *Building a family $E_{c^*}(b)$ of 2-boxes so that $c(E_{c^*}(b)) = \Omega(c^*(E_{c^*}(b))^2)$.*

Define $E_{c^*}(b)$ recursively as follows (Figure 2.3 depicts the construction of $E_{c^*}(b)$):

- $E_1(b) = [(0, \frac{3}{4}), (b, \frac{5}{4})]$.
- $|E_{c^*-1}| + c^* - 2$ copies of box $B = [(\frac{3b}{4}, c^* - \frac{1}{4}), (b, c^* + \frac{1}{4})]$ union $\{B_i = [(0, i - \frac{1}{4}), (\frac{b}{2}, i + \frac{1}{4})] | 1 \leq i < c^*\}$ union box $B' = [(\frac{b}{2}, c^* - \frac{1}{4}), (b, c^* + \frac{1}{4})]$ union the boxes resulting from $E_{c^*-1}(\frac{b}{4})$ (note that b becomes $\frac{b}{4}$ at this stage).

Clearly, $E_{c^*}(b)$ can be stabbed with a minimal set of c^* points:

$$\left\{(i, \frac{3b}{4^{c^*-i+1}}) | 1 \leq i \leq c^*\right\}.$$

But our algorithm will report exactly $(\sum_{i=1}^{c^*} i)+c^*-1 = \frac{c^*(c^*+1)}{2}+c^*-1 = \Omega(c^{*2})$ points.

Note that in order to design this example, we need at least $n = 2^{c^*-1}$ boxes. (Indeed, we have $|E_{c^*}| = 2|E_{c^*-1}| + 2(c^*-1)$ and $E_1 = 1$). So that we also check that $c \leq c^*(1+\log n)$.

We can also build in higher dimensions, using the same principle, a family $\mathcal{S}$ of boxes so that $c(\mathcal{S}) = \Omega(\frac{c^*(\mathcal{S})^d}{d!})$. Recall that a d-dimensional box B can be viewed as an ordered product of d ranges: $B = [a_1, b_1] \times ... \times [a_d, b_d]$. Given a $(d-1)$-dimensional box B', we can extend it to a d-dimensional box $B = [a_1, b_1] \times B'$.

Define the set $E_{c^*}^{(d)}(b)$ of boxes as follows:

- $E_1^{(d)}(b) = [(0, \frac{3}{4}, ..., \frac{3}{4}), (b, \frac{5}{4}, ..., \frac{5}{4})]$ for any d.
- $|E_{c^*-1}^{(d)}| + |E_{c^*}^{(d-1)}| - 1$ copies of box $B = [(\frac{3b}{4}, ..., \frac{3b}{4}, c^* - \frac{1}{4}), (b, ..., b, c^* + \frac{1}{4})]$ union $[0, \frac{b}{2}] \times E_{c^*}^{(d-1)}(\frac{b}{2})$ union the box $B' = [(\frac{b}{2}, \ldots, \frac{b}{2}, c^* - \frac{1}{4}), (b, \ldots, b, c^* + \frac{1}{4})]$ union $E_{c^*-1}^{(d)}(\frac{b}{4})$.

Clearly, we have $|E_{c^*}^{(d)}| = 2(|E_{c^*}^{(d-1)}| + |E_{c^*-1}^{(d)}|)$.

$E_{c^*}^{(d)}(b)$ can be pierced with an optimal set of c^* points:

$$\left\{(i, \frac{3b}{4^{c^*-i+1}}, ..., \frac{3b}{4^{c^*-i+1}}) | 1 \leq i \leq c^*\right\}.$$

However, our algorithm will return $\Omega(\frac{c^{*\overline{d}}}{d!})$ stabbing points (using Euler's summation formula). More precisely, we can prove by induction on d that our algorithm will return exactly $c = \frac{c^{*\overline{d}}}{d!} + \frac{c^{*\overline{d-1}}}{(d-1)!} - 1$ stabbing points.

Proof. We do the proof by induction on the lexicographically ordered vector (d, n). If $d = 1$, cleary $c(\mathcal{S}) = c^*(\mathcal{S})$. If $|\mathcal{S}| = n = 1$ then $c(\mathcal{S}) = c^*(\mathcal{S}) = 1$. Otherwise, following our algorithm we have:

$$c(E_{c^*}^{(d)}) = 1 + c(E_{c^*-1}^{(d)}) + c(E_{c^*}^{(d-1)}).$$

Plugging our inductive hypothesis, we get:

$$c(E_{c^*}^{(d)}) = 1 + (\frac{c^*(E_{c^*-1}^{(d)})^{\overline{d}}}{d!} + \frac{c^*(E_{c^*-1}^{(d)})^{\overline{d-1}}}{(d-1)!} - 1) + (\frac{c^*(E_{c^*}^{(d-1)})^{\overline{d-1}}}{(d-1)!} + \frac{c^*(E_{c^*}^{(d-1)})^{\overline{c-2}}}{(d-2)!} - 1),$$

using $c^*(E_{c^*}^{(k)}) = c^*$ for any $k \geq 1$, we get

$$c(E_{c^*}^{(d)}) = \frac{c^{*\overline{d-1}}}{d!}(c+d-1) + \frac{c^{*\overline{d-2}}}{(d-1)!}(c+d-2) - 1.$$

Setting back $c^* = c^*(E_{c^*}^{(d)})$, we finally obtain the desired result:

$$c(E_{c^*}^{(d)}) = \frac{c^*(E_{c^*}^{(d)})^{\overline{d}}}{d!} + \frac{c^*(E_{c^*}^{(d)})^{\overline{d-1}}}{(d-1)!} - 1.$$

□

We can also check that $c \leq c^* \frac{(\log n+1)^{\overline{d-1}}}{(d-1)!}$ since $2^{c^*-1} \leq |E_{c^*}^{(d)}| \leq 4^{c^*}$ (more precisely $|E_{c^*}^{(d)}| = O(2^{c^*} c^{*d-1})$).

Thus, when the dimension d is fixed, we have exhibited a family of boxes such that the algorithm gives an approximation that matches the upper bound $\frac{c^{*\overline{d}}}{d!} + \frac{c^{*\overline{d-1}}}{(d-1)!} - 1$. We can also provide the same kind of scheme to build a family $\mathcal{S}$ of unit hypercubes so that $c(\mathcal{S}) = \Omega(\frac{c^*(\mathcal{S})^{\overline{d}}}{d!})$.

2.3.3 Congruent or Constrained boxes

The previous section exhibits an example where our algorithm reaches its worst-case performance. However, in order to build it, we did consider stretched boxes, i.e. non-constrained boxes. We define the aspect ratio of a d-dimensional box $B = [(a_1, ..., a_d), (b_1, ..., b_d)]$ to be $\max_{i,j=1..d}\{\frac{b_i - a_i}{b_j - a_j}\}$. Loosely speaking, we will call aspect ratio of $\mathcal{S}$ the maximum of the aspect ratio of the boxes in $\mathcal{S}$. Note that if the boxes have bounded aspect ratio then also have bounded-size (volume) but not the converse. Thus, hypercubes have aspect ratio equal to 1. We use nonbounded aspect ratio in order to build our bad example. In [HM84], the polynomial-time approximation scheme depends on the aspect ratio of the congruent isothetic boxes.

In this section, we refine the analysis of the approximation factor whenever the projections of the d-boxes onto the d axis have the bounded aspect ratio property. Note that the boxes may have a nonbounded aspect ratio but their projections (sets of intervals) may have their aspect ratio bounded.

Lemma 11 *Let $\mathcal{S}$ be a set of n congruent isothetic d-dimensional boxes. Then, our algorithm guarantees that $c(\mathcal{S}) \leq 2^{d-1} c^*(\mathcal{S})$.*

Proof. For sake of simplicity, let us first consider the case of (unit) hypercubes. $\mathcal{S}$ is a collection of n congruent hypercubes. We prove below by

induction on the dimension that $c(\mathcal{S}) \leq 2^{d-1}c^*(\mathcal{S})$. Section 2.2.2 shows that the algorithm ensures $c_1(\mathcal{S}) = c^*(\mathcal{S})$ for (unit) intervals.

Otherwise, let $\mathcal{S}$ be a set of n d-boxes. Consider the ordered sequence (left to right) of cutting hyperplanes of type d: $(H_m(1) : x_d = a_1), ..., (H_m(k) : x_d = a_k)$ with the associated partition of the hypercubes $\mathcal{S}'_m(1), ..., \mathcal{S}'_m(k)$. Clearly, we have $a_{i+1} - a_i > \frac{1}{2}, 1 \leq i \leq k-1$ for the case of unit hypercubes. Therefore $\mathcal{S}'_m(i) \cap \mathcal{S}'_m(j) = \emptyset$ if $|i-j| \geq 2$. We have:

$$c_d(\mathcal{S}) = \sum_{i=1}^{k} c_{d-1}(\mathcal{S}'_m(i)),$$

$$c_d(\mathcal{S}) \leq 2^{d-2} \sum_{i=1}^{k} c^*(\mathcal{S}'_m(i)).$$

We can decompose the last sum taking into account the parity of i:

$$c_d(\mathcal{S}) \leq 2^{d-2} \Big(\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} c^*(\mathcal{S}'_m(2i)) + \sum_{i=0}^{\lceil \frac{k}{2} \rceil - 1} c^*(\mathcal{S}'_m(2i+1)) \Big).$$

But $\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} c^*(\mathcal{S}'_m(2i)) = c^*(\cup_{i=1..\lfloor \frac{k}{2} \rfloor} \mathcal{S}'_m(2i)) \leq c^*(\mathcal{S})$ and $\sum_{i=0}^{\lceil \frac{k}{2} \rceil - 1} c^*(\mathcal{S}'_m(2i+1)) = c^*(\cup_{i=0..\lceil \frac{k}{2} \rceil - 1} \mathcal{S}'_m(2i+1)) \leq c^*(\mathcal{S})$ since both $S'_m(2i) \cap S'_m(2j) = \emptyset$ and $S'_m(2i+1) \cap S'_m(2j+1) = \emptyset$ as soon as $i \neq j$.

Therefore, we get:

$$c_d(\mathcal{S}) \leq 2^{d-2} \times 2c^*(\mathcal{S}),$$

$$c_d(\mathcal{S}) \leq 2^{d-1}c^*(\mathcal{S}).$$

We only use the fact that all the boxes have the same i-width (width along the i-th dimension). Therefore, the result applies for congruent boxes. □

In [HM84], Hochbaum and Maass also consider this problem (in its dual form however) and gave an $O(l^d n^{2l^d+1})$-time algorithm (a polynomial time approximation scheme) which ensures that $c(\mathcal{S}) \leq (1 + \frac{1}{l})^d c^*(\mathcal{S})$ for a given integer $l \geq 1$. Thus, for $l = 1$ it yields an $O(n^3)$-time algorithm with performance ratio 2^d.

Since our algorithm proceeds dimension by dimension, we only need to have the bounded aspect ratio for the projected boxes (along the d axis). Let $\mathcal{S}$ be a collection of n constrained boxes:

$$\mathcal{S} = \{[(b_{1,1}, ..., b_{1,d}), (u_{1,1}, ..., u_{1,d})], ..., [(b_{n,1}, ..., b_{n,d}), (u_{n,1}, ..., u_{n,d})]\},$$

with $\max_{i,j=1..n}\{\frac{u_{i,j}-b_{i,1}}{u_{j,1}-b_{j,1}}\} \leq B_1$, ..., $\max_{i,j=1..n}\{\frac{u_{i,d}-b_{i,d}}{u_{j,d}-b_{j,d}}\} \leq B_d$, for some constants $B_i \geq 1$, $1 \leq 1 \leq d$.

Using the same technic as above, we get the following lemma:

Lemma 12 *Let $\mathcal{S}$ be a collection of n d-dimensional constrained boxes with $B_1, ..., B_d$ defined as above. Then, our algorithm will return $c(\mathcal{S})$ stabbing points so that $c(\mathcal{S}) \leq (\prod_{i=2}^{d} \lceil 2B_i \rceil) c^*(\mathcal{S})$.*

We may assume w.l.o.g. that $B_1 = \max_{i=1..d}\{B_i\}$. Otherwise, we make a simple rotation of the orthogonal frame in linear time. This also means that we may have a direction where the projected boxes are not constrained since we are able to solve exactly the problem in one dimension (see Section 2.2.2).

We can also mix up our algorithm with the PTAS (polynomial-time approximation scheme) of [HM84] in order to obtain tradeoffs both for the running time and the performance ratio. One might wonder whether an additive constant is possible instead of a multiplicative constant in all our bounds concerning $c(\mathcal{S})$. It is very unlikely! Indeed, unless P=NP is it not possible to achieve the absolute performance $c(\mathcal{S}) \leq c^*(S) + k$ for some constant integer k.

Proof. We do the proof by contradiction (see also [GJ79] for this kind of technic). Consider that we have a deterministic polynomial time algorithm ALG that guarantees $c(\mathcal{S}) \leq c^*(\mathcal{S}) + k$. Then we consider $k+1$ non-overlapping copies of our boxes. Let $\mathcal{S}_{k+1}$ be this data set. Algorithm ALG will return $c(\mathcal{S}_{k+1})$ stabbing points so that $c(\mathcal{S}_{k+1}) \leq c^*(\mathcal{S}_{k+1}) + k$. One of the copies of $\mathcal{S}$ in $\mathcal{S}_{k+1}$ requires at most $\lfloor \frac{c(\mathcal{S}_{k+1})}{k+1} \rfloor = c(\mathcal{S})$ stabbing points and $c^*(\mathcal{S}_{k+1}) = (k+1)c^*(\mathcal{S})$. Thus, we will be able to have a deterministic polynomial-time algorithm ALG-B such that $c^*(\mathcal{S}) \leq c(\mathcal{S}) \leq c^*(\mathcal{S}) + \frac{k}{k+1}$. Since finding the value of $c^*(\mathcal{S})$ is NP-complete, we get the contradiction unless $P = NP$. □

Let $\mathcal{G}$ be the intersection graph of a set of n d-dimensional isothetic boxes $\mathcal{S}$, i.e. to each box corresponds a node and there is an edge between two nodes iff their corresponding boxes intersect. Isothetic boxes have nice combinatorial properties. For example, a set of boxes have a nonempty intersection iff they intersect pairwise (this is an Helly-type theorem [DG82, HD60b]). Therefore, finding a minimum-size set of stabbing points can be done by first computing $\mathcal{G}$ in quadratic time and then, finding a minimum clique partition

of $\mathcal{G}$, i.e. a set of cliques (complete subsets of $\mathcal{G}$) whose union covers the vertices of $\mathcal{G}$. There is a one-to-one correspondence between these two problems. As a direct corollary, it implies that the mimimum clique partition is NP-complete (as proved in [GJ79]), even for intersection graphs of isothetic boxes.

Note that if every 3-subset of $\mathcal{S}$ (i.e., a subset $S \in \binom{\mathcal{S}}{3}$) has an empty intersection ($\cap S = \emptyset$), then the stabbing problem can be solved in *polynomial time* by reduction to the maximum matching problem.

The stabbing problem is related somehow to $c(p,q)$-numbers and $N(p,q;d)$ numbers [HD60b, GW93a], i.e. $c(p,q)$-numbers of d-dimensional isothetic boxes. $N(p,q;d)$ numbers are defined for $2 \leq q \leq p$ for the class $\mathcal{P}$ of isothetic d-boxes as follows: it is the smallest integer so that for every set $\mathcal{S} \in \mathcal{P}$ of parallelotopes, we have: if every p-subset $S \in \binom{\mathcal{S}}{p}$ contains at least one q-subset $Q \in \binom{S}{q}$ that has a nonempty intersection ($\cap Q \neq \emptyset$), then $\mathcal{S}$ can be stabbed by $N(p,q;d)$ points. Debrunner et al. [HD60b, GW93a] proved that $N(p,q;d) \leq \binom{p-q+d}{d}$ if $2 \leq q \leq p$.

Therefore, we have:

$$c^* \leq N(b^*+1, 2; d) = \binom{b^* - 1 + d}{d},$$

since every (b^*+1)-subset of $\mathcal{S}$ contains at least a pair of intersecting boxes.

There is much literature concerning $c(p,q)$ numbers. Perhaps, one of the most challenging conjecture that remains unsettled is the following:
Conjecture (Wegner 67). If $\mathcal{K}$ is a family of parallel rectangles in the plane, no p of which are pairwise disjoint, then $\mathcal{K}$ can be stabbed by $2p-3$ points.

This conjecture has been confirmed for squares and $p \leq 4$ but remain opened for 'nonconstrained' boxes, i.e. non-bounded size boxes.

2.3.4 Experimental results

We did the implementation in C++ using the LEDA [9] and CGAL [10] librairies. The code length is about 1000 lines. It should be noted that the algorithm and therefore its implementation are robust. Indeed, we only compare our standard input values without creating intermediate values (even when our algorithm reports intersection points). Since the problem of computing the minimum value c^* so that our set is c^*-pierceable is NP-complete,

9. Library for Efficient DataStructure and Algorithms. Max-Planck Institut für Informatik, Im Stadtwald – 66123 Saarbrücken – Germany.
10. see "The CGAL Kernel User Manual" – INRIA Sophia-Antipolis (France).

$\lvert\mathcal{S}\rvert =$	16	32	64	128	256	512	1024	2048	4096	8192	16364	32728	65536	130000	260000	520000
$c(\mathcal{S}) =$	9	14	23	30	56	78	123	188	272	417	648	928	1413	2093	3122	4486

TAB. 2.2 – *Considering uniformly distributed rectangles in the unit square.*

we could not compare in the experiments the precise relationships between c and c^*. Note that if instead of implementing the optimal stabbing algorithm of intervals of section 2.2.2, we implement the heuristic of section 2.2.3 then we get $c \leq O(\binom{c^*+d-1}{d})$. We did choose that solution in our experiments (see discussion of section 2.2.3).

However, loosely speaking, if we admit that in average $c^*(\mathcal{S}'_m) = c^*(\mathcal{S})^{\frac{d-1}{d}}$ and $c^*(\mathcal{S}_1) = c^*(\mathcal{S}_2)$ (which might be the case, for example, when considering the uniform distribution of congruent boxes) then we expect $c(\mathcal{S})$ to be an $O(1)$-approximate of $c^*(\mathcal{S})$ for fixed dimension d. Indeed, let K_d with $K_1 = 1$ be the multiplicative constant. We have:

$$c(\mathcal{S}) \leq K_d(c^*(\mathcal{S})) \leq c^* + \sum_{i=0}^{\lceil \log c^*(\mathcal{S}) \rceil} 2^i K_{d-1}((\frac{c^*(\mathcal{S})}{2^i})^{\frac{d-1}{d}}).$$

Therefore, we find $K_d \leq 1 + \frac{1}{2^{\frac{1}{d}}-1} K_{d-1}$ with $K_1 = 1$. Thus, in the planar case we get under these hypothesis that $c(\mathcal{S}) \leq 3.42c^*(\mathcal{S})$. (This result is corroborated in the experiments (using a good lower bound for $c^*(\mathcal{S})$, e.g. $b(\mathcal{S})$).

We report the value of $c(\mathcal{S})$ in Table 2.2 for a set $\mathcal{S}$ of uniformly distributed isothetic planar boxes. We have also implemented an exhaustive search procedure for finding a minimal set of piercing points. This algorithm could not handle input size greater than 25 (although some tricks have been plugged[11] in). Table 2.3 shows the number of configurations explored by the exhaustive algorithm.

As an application, we considered the following problem (already mentionned in the introduction): given a set of n cities $\mathcal{C} = \{C_1 = (x_1, y_1), ..., C_n = (x_n, y_n)\}$, we wish to cover them by a minimum number of a given 'unit' square $S = [-r, r] \times [-r, r]$ of side length $2r$. We associate to each city $C_i = (x_i, y_i)$ the square $S_i = [x_i - r, x_i + r] \times [y_i - r, y_i + r]$ for $1 \leq i \leq n$. Let $\mathcal{S} = \{S_1, ..., S_n\}$. Now, we use the fact that $\mathcal{C}$ can be covered with k translated copies of S if and only if $\mathcal{S}$ can be stabbed with k points. We ran

11. Finding good cuts for an exhaustive algorithm is interesting in itself since it allows to handle large input sizes for 'special' down-to-earth tailored instances (see for example the well-known Traveling Salesman Problem [RSL74]).

$\lvert\mathcal{S}\rvert$ =	14	15	16	17	18
$\mathrm{conf}(\mathcal{S})$ =	15120	29400	99120	241272	672964

TAB. 2.3 – *Number of different configurations conf($\mathcal{S}$) for some input set $\mathcal{S}$, i.e. number of distinct piercing point sets of size ranging over $[c^*, n]$ for small values of n. Note that for d-dimensional set $\mathcal{S}$ of boxes, the complexity of the arrangement of $\mathcal{S}$ is $O(|\mathcal{S}|^d)$. Thus, we have conf($\mathcal{S}$) $\leq \sum_{i=c^*(\mathcal{S})}^{|\mathcal{S}|} \binom{O(|\mathcal{S}|^d)}{i} \leq 2^{O(|\mathcal{S}|^d)}$.*

our program on 120 cities of the United States of America (the input file is freely distributed and may be found in the *Stanford GraphBase* [Knu93]). The results are depicted in figure 2.4.

Figure 2.5 shows some experiments for sets $\mathcal{S}$ that are 20-pierceable, i.e. $c^*(\mathcal{S}) = 20$. The left chart shows the number of stabbing points reported by our algorithm in case of congruent/nonconstrained boxes. The right chart depicts the running time of our algorithm. (We took the average over 20 trials).

It may be nice to study the average stabbing number $a_n(d)$ of a fixed-size randomly chosen set of n d-boxes as a function of d.

2.4 Concluding remarks

We have investigated in this paper the *stabbing problem* for a set of d-dimensional isothetic boxes which consists in finding a set of points so that each box contains at least one of these points.

Finding a minimum-size set of stabbing points has been shown to be NP-complete as soon as $d > 1$, even when considering congruent isothetic boxes. Therefore that problem is untractable in practice. We gave in this technical report a heuristic that computes c points stabbing a set of n d-dimensional axis-parallel boxes in output-sensitive time $O(dn + n \log c)$ using linear space. Moreover, we proved that:

$$c \leq \min\{\frac{{c^*}^{\overline{d}}}{d!} + \frac{{c^*}^{\overline{d-1}}}{(d-1)!} - 1, c^* \frac{(\log n + 1)^{\overline{d-1}}}{(d-1)!}\},$$

where $x^{\overline{m}}$ is the rising factorial power: $x^{\overline{m}} = \prod_{i=0}^{m-1}(x+i)$. We showed the tightness of the bounds by building a generic family $\mathcal{S}$ of d-dimensional boxes so that $c(\mathcal{S}) = \Omega(\frac{c^*(\mathcal{S})^d}{d!})$. We proved in the case of congruent boxes and 'constrained' boxes that $c \leq 2^{d-1}$ and $c = O(c^*)$ respectively. Our algorithm

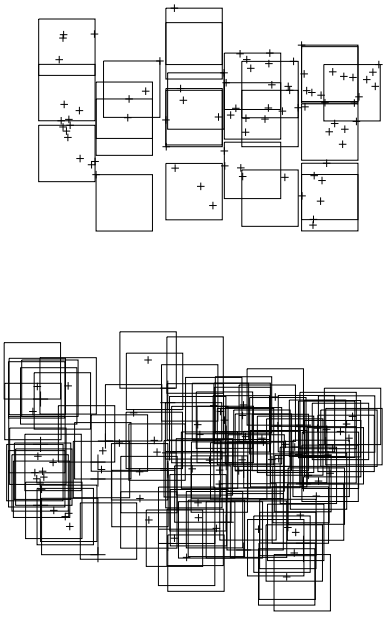


FIG. 2.4 – *Finding a covering of 120 cities of the U.S.A. Our algorithm reports a set of* 25 *unit squares covering the* 120 *cities. Any solution requires at least* 11 *squares. (Considering the corresponding piercing problem, one may find* 11 *disjoint squares.)*

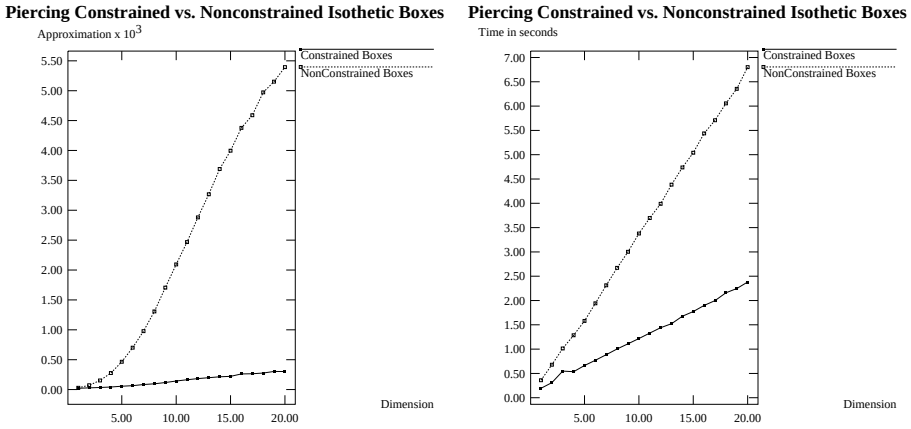


FIG. 2.5 – *Impact of the dimension over a set* $\mathcal{S}$ *(*$c^*(\mathcal{S}) = 20$ *and* $|\mathcal{S}| = 20000$*) of constrained/nonconstrained* d*-boxes for* $1 \leq d \leq 20$*. The right chart exhibits the running time of our implementation (the slope of the lines for the constrained/nonconstrained cases are* $O(d + \log c^*)$ *and* $O(d \log c^*)$ *respectively.*

can be easily parallelized onto PRAM computers in order to gain efficiency (see [AL93]).

We plan to investigate in a nearby future the tradeoff between the running time of any stabbing algorithm for sets of boxes and its relative performance.

We also focus on the case of c-oriented objects and general convex/non convex objects.

This technical report raises some open problems:

- the exact relationships betwen c^* and b^* (from $N(p,q;d)$ numbers, we get $c^* \leq \binom{b^*-1+d}{d}$).
- the hardness of approximation of constrained boxes compared with general boxes inside the polynomial hierarchy of problems.
- Can we find for 'nonconstrained' isothetic boxes an algorithm which guarantees some ratio between b and b^*?
- Can we obtain better fast approximation algorithms by applying to our set of c stabbing points other (time-costly) algorithms?
- In [IA83], H. Imai and Ta. Asano gave an algorithm to compute a maximal cell in an arrangement of n isothetic boxes in $O(n^{d-1} \log n)$. Can we compute a maximal cell in a better running time? This would improve the running time of the greedy algorithm.

Another aspect of this problem that is currently being investigated is to give efficient algorithms to detect whether a set of objects is k-pierceable or not for small values of k [KN96b, KN96a].

Acknowledgements. I would like to thank Jean-Daniel Boissonnat, Hervé Brönnimann and Mariette Yvinec for helpful discussions.

Chapitre 3

On Piercing Sets of Objects

(Ce travail en collaboration avec Matthew J. Katz[1] est disponible en rapport de recherche INRIA Numéro 2874 [KN96b] et dans les actes de conférence de la 12^{e} conférence ACM Computational Geometry [KN96a]. Quelques détails supplémentaires ont été ajouté dans cette version.)

Abstract

A set of objects is k-pierceable if there exists a set of k points such that each object is pierced by (contains) at least one of these points. Finding the smallest integer k such that a set is k-pierceable is NP-complete. In this technical report, we present efficient algorithms for finding a piercing set (i.e., a set of k points as above) for several classes of convex objects and small values of k. In some of the cases, our algorithms imply known as well as new Helly-type theorems, thus adding to previous results of Danzer and Grünbaum who studied the case of axis-parallel boxes. The problems studied here are related to the collection of optimization problems in which one seeks the smallest scaling factor of a centrally symmetric convex object K, so that a set of points can be covered by k congruent homothets of K.

Keyword: Computational Geometry, Pierceability, Helly numbers.

3.1 Introduction

Let $\mathcal{S}$ be a set of n d-dimensional convex objects. $\mathcal{S}$ is *k-pierceable* if there exists a set $\mathcal{P}$ of k points in $\mathbb{E}^d$ such that each object in $\mathcal{S}$ is pierced by

1. Department of Computer Science, Utrecht University, E-mail: matya@cs.ruu.nl.

(contains) at least one point in $\mathcal{P}$. Perhaps one of the most famous theorems in convex geometry is Helly's theorem [DGK63, GW93a] that states that $\mathcal{S}$ is 1-pierceable (i.e., has a non-empty intersection) if and only if every subset of $\mathcal{S}$ of cardinality $d+1$ is 1-pierceable. The *Helly-number* $h = h(\mathcal{C}, P)$ associated with a class of objects $\mathcal{C}$ and a property P is the smallest integer (if such exists) so that for any set $\mathcal{S} \subseteq \mathcal{C}$ we have: if every subset of $\mathcal{S}$ of cardinality h has property P, then $\mathcal{S}$ also has property P. If such an integer does not exist, we set h by convention to ∞ (infinity). Let $\prod^k$ denote the k-pierceability property and $\mathcal{C}^d$ be the class of d-dimensional convex objects. Then Helly's theorem states that $h(\mathcal{C}^d, \prod^1) = d+1$. There are many Helly-type theorems in convex geometry (see [GW93a] for an up-to-date survey).

Danzer and Grünbaum [DG82] studied the case of d-dimensional (axis-parallel) boxes $\mathcal{B}^d$ and obtained Helly-type theorems whenever they exist. They prove in particular that $h(\mathcal{B}^d, \prod^k) = \infty$, for $d, k \geq 3$. They conclude their paper with the following conjecture:

> $h(\mathcal{T}^d(K), \prod^2) < \infty$ if and only if K is a convex d-dimensional polytope, where $\mathcal{T}^d(K)$ is the class of all translates of K.

This conjecture has been refuted very recently by Katchalski and Nashtir [KNar], who proved that $h(\mathcal{T}^2(K), \prod^2) = \infty$, for a centrally symmetric convex hexagon K. The problems studied in this paper are related to a collection of optimization problems. Let $\mathcal{P}$ be a set of n points in $\mathbb{E}^d$ and K a centrally symmetric convex object. Let o be the center point of K and let K_λ be a homothet of K with scaling factor λ. K defines a semi-distance function as follows: For two points a and b, $d_K(a,b) = \lambda$, if K_λ is the smallest homothet of K that contains b when it is centered at a.

In these optimization problems we seek the smallest real λ such that $\mathcal{P}$ can be covered by k translates of K_λ. Note that these problems are always solvable by setting λ to an arbitrary large real. The decision problem associated with these optimization problems is the following. Given a value λ determine whether $\mathcal{P}$ can be covered by k translates of K_λ, i.e., try to find k appropriate loci for the (center points of the) k copies of K_λ. A point $p \in \mathcal{P}$ lies in a translate of K_λ centered at o iff $d_K(o,p) \leq \lambda$, and, since $d_K(\cdot,\cdot)$ is a symmetric function, iff $d_K(p,o) \leq \lambda$. Let $\mathcal{P}_{K_\lambda}$ be the set that is obtained from $\mathcal{P}$ by replacing each point $p \in \mathcal{P}$ with the translate of K_λ that is centered at p. Then, $\mathcal{P}$ can be covered by k translates of K_λ iff $\mathcal{P}_{K_\lambda}$ is k-pierceable. Generally, once the decision problem is solved efficiently, we link it with the parametric searching technique of Megiddo [Meg83a], in order to obtain an efficient solution for the original optimization problem. For example, consider the 2-center problem. In this problem we wish to cover $\mathcal{P}$ by two congruent disks of minimum radius. The corresponding decision problem is: Given a

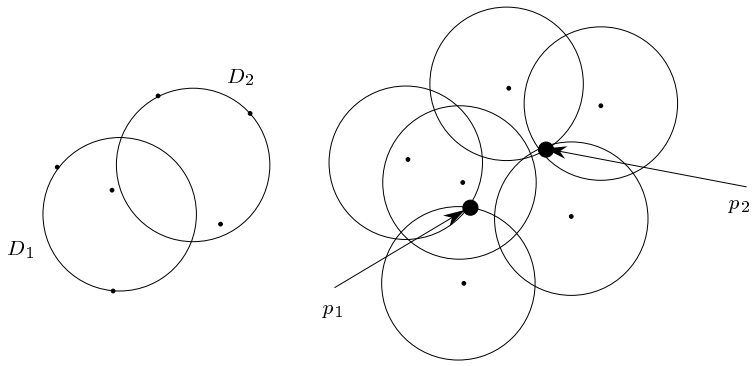


FIG. 3.1 – *Correspondence between covering a set of points with two (unit) disks and piercing a set of (unit) disks with two points.*

fixed value for the radius, say, 1, determine whether $\mathcal{P}$ can be covered by two disks of radius 1. This is equivalent to the following problem:

Let $\mathcal{P}_K$ be the set of unit disks centered at the points of $\mathcal{P}$, determine whether $\mathcal{P}_K$ is 2-pierceable (see Figure 3.1). Recently, Sharir [Sha96] has presented a nearly linear algorithm for the 2-center decision problem, which immediately implies a nearly linear solution to the equivalent piercing problem. It is well known that if K is a centrally symmetric strictly convex object in the plane (e.g., a disk), then $h(\mathcal{T}^2(K), \prod^2) = \infty$ (see [HD60b] for an elegant proof). This immediately implies that the 2-center decision problem is not an LP-type problem, since every LP-type problem implies a Helly-type theorem [Ame94] (see discussion below).

Amenta [Ame94] studied the relationships between LP-type problems and Helly-type theorems. (See [MSW92] for the definition of LP-type problems and an efficient linear-time randomized algorithm.) She showed that every LP-type problem has a corresponding Helly-type theorem but that the converse is not necessarily true. Nevertheless, she gave two paradigms for obtaining a LP-type problem from a Helly-type theorem. These paradigms fit the Helly-type theorems that are derived in this paper. We thus can obtain a randomized linear-time algorithm for the corresponding piercing problems (which can be derandomized, in turn, using the algorithm of Chazelle and Matoušek [CM93]). However, these algorithms are much more complicated than ours. Moreover, our algorithms consist of the proofs for the Helly-type theorems that we obtain. Table 3.1 summarizes our results. We also:

(i) give a simple $O(n \log^2 n)$ time algorithm for the 2-center decision problem, when the diameter of the point set is greater than $3r$, where r is the radius of the covering disks, and

Objects	k	Time
homothetic triangles	2	$O(n)$ (Helly-type)
$4,5$-oriented polygons	2	$O(n \log n)$
d-dim. c-oriented polytopes	2	$O(n^{\min\{\lfloor \frac{c}{2} \rfloor, d\}} \log n)$
$(d+1)$-oriented simplices	2	$O(n^{\lceil \frac{d}{2} \rceil} \log n)$
d-dim. boxes	2	$O(n)$ (Helly-type)
homothetic triangles	3	$O(n \log n)$

TAB. 3.1 – *Summary of the results presented in this paper.*

(ii) show that $h(\mathcal{C}, \prod^2) = \infty$, where $\mathcal{C}$ is the class of 4-sided convex polygons.

All these results are a significant improvement over the naïve solution that consists of computing the arrangement which is of size $O(n^d)$, and checking, for each subset of k cells, whether the underlying set of objects is pierced by this subset. Thus, the naïve method requires $O(n^{dk+1})$ time (assuming k is a constant). The naïve solution can be slightly improved. Each cell of the arrangement is labelled with the subset of objects containing it. A cell is *maximal* if its corresponding subset is not contained in the subset of another cell. Clearly, if there exists a solution, then there also exists a solution that consists of at most k points drawn from maximal cells. Note, however, that the number of maximal cells can be as large as $\Omega(n^d)$. (Consider the following set of n convex objects: For each direction x_i, $i = 1, \ldots, d$, we take the n/d objects $[0, n/d]^d \cap (j < x_i < j+1)$, $j = 0, \ldots, n/d - 1$. Clearly, each d-cell of the arrangement of this set is maximal.) We can refine the naïve algorithm as follows. For each subset of $k-1$ cells, check whether the remaining set of objects that are not pierced by these cells has a non-empty intersection (this is an LP-type problem). Thus, we obtain an $O(n^{d(k-1)+1})$ expected time algorithm.

Since the problem of finding the minimum integer k so that a set of objects $\mathcal{S}$ is k-pierceable was shown to be NP-complete [Kar72, FPT81], many authors have focused on the problem of approximating k. There exist some polynomial-time algorithms for the latter problem with bounded error ratio [Chv79, Hoc82]. Bellare et al. [BGLR93] show that no polynomial-time algorithm can approximate the optimal solution within a factor of $(\frac{1}{8} - \epsilon) \log |\mathcal{S}|$, unless $NP \subseteq DTIME[n^{\log \log n}]$, where $\epsilon > 0$. Very recently U. Feige [Fei96] proved the best possible $(1-\epsilon) \log n$ lower bound for any $\epsilon > 0$.

This problem (in its non-geometric formulation) is called in the literature the *set cover* problem.

In this paper, we essentially study a larger class of objects (in comparison with Danzer and Grünbaum's paper), namely, the class of c-oriented convex polytopes, which contains the special case of all homothets of some c-facet polytope. An object O is *c-oriented* if it can be defined as the intersection of at most c translates of members of a given set of c halfspaces. For example, d-dimensional (axis-parallel) boxes are $2d$-oriented.

This paper is organized as follows. We first demonstrate our general method (Section 3.2) on the case of boxes and for $k = 2$, and derive a Helly-type theorem of Danzer and Grünbaum. We then describe this method (Section 3.3) for the case of c-oriented convex polytopes. In Section 3.4, several special cases are considered in which we are able to obtain more efficient solutions than those that are implied by the general method. In this section we obtain a new Helly-type theorem (this theorem with the exact Helly-number was also discovered independently by Katchalski and Nashtir [KNar]). Finally, we conclude in Section 3.5.

3.2 Piercing Sets of Boxes with 2 Points

Let $h(d, k)$ be the Helly-number for d-dimensional (axis-parallel) boxes and k points, that is, $h(d, k)$ is the smallest integer (if such exists) such that, if every subset of cardinality $h(d, k)$ of a set $\mathcal{B}$ of d-boxes is k-pierceable, then $\mathcal{B}$ is k-pierceable. If such an integer does not exist, we set $h(d, k) = \infty$. Danzer and Grünbaum [DG82] have proven the following theorem, which we present as a table:

Theorem 13 (Danzer and Grünbaum)

$h(d,k)$	$k=1$	$k=2$	$k=3$	$k \geq 4$
$d=1$	2	3	4	$k+1$
$d=2$	2	5	16	∞
$d \geq 3$	2	$\begin{cases} 3d & \text{if } d \text{ is odd} \\ 3d-1 & \text{if } d \text{ is even} \end{cases}$	∞	∞

Their proof for the case of d-dimensional boxes, $d \geq 2$, and two points is constructive and induces a simple algorithm. We briefly describe this algorithm in order to compare it subsequently with ours. If there exists a

hyperplane $x_i = c$ that meets all the boxes of $\mathcal{B}$, then $\mathcal{B}$ is 2-pierceable if and only if the $(d-1)$-dimensional set of boxes $\mathcal{B}' = \{B \cap (x_i = c) | B \in \mathcal{B}\}$ is 2-pierceable. Hence, we may assume w.l.o.g. that the projection $\mathcal{P}_i$ of $\mathcal{B}$ onto the x_i axis yields at least one pair of disjoint segments, for $i = 1, \ldots, d$. Denote by a_i (resp. b_i) the lower bound (resp. upper bound) of the right (resp. left) endpoints of the segments of $\mathcal{P}_i$. W.l.o.g., assume that $a_i = 0$ and $b_i = 1$, for $i = 1, \ldots d$, and put $v = (1, 1, ..., 1)$. Then, $\mathcal{B}$ is 2-pierceable if and only if there are two opposite points $p \in \{0, 1\}^d$ and $v - p$ that pierce $\mathcal{B}$. This gives us an $O(2^d n)$ time algorithm. Note that if $\mathcal{B}$ is not 2-pierceable, then we can obtain a counterexample of size $O(2^d)$ from which we can select a subset of size $3d$ (for odd d) or $3d - 1$ (for even d) in $O(1)$ time, for fixed dimension d.

Remark. The same kind of considerations allows us to derive an $O(n \log n)$-time algorithm for piercing a set of d-dimensional boxes with three points, for $3 \leq d \leq 5$.

We describe in this section a more general and algorithmic approach to the k-piercing problem, and demonstrate it for the case of d-boxes with 2 points. (The algorithm above of Danzer and Grünbaum is very specialized and it cannot be generalized to handle other classes of objects.) Our approach does not yield the exact Helly-numbers but only upper bounds. In the subsequent sections, we will apply our approach to sets of more complex objects to obtain new results. We begin with the case of axis-parallel rectangles (2-dimensional boxes) in order to focus on the method.

3.2.1 2-dimensional boxes (rectangles)

Let $\mathcal{B} = \{B_1, ..., B_n\}$ be a set of n 2-boxes. We wish to find a piercing set for $\mathcal{B}$ which consists of 2 points, if such a set exists. In other words, we wish to solve the 2-piercing problem for $\mathcal{B}$. We first check whether $\mathcal{B}$ is 1-pierceable by computing the region $\cap_{k=1}^{n} B_k$ (which is of course also a box) incrementally. If at some stage, j, the current region becomes empty, then obviously $\mathcal{B}$ is not 1-pierceable. Denote the box $\cap_{k=1}^{j-1} B_k$ by R. Since $R \cap B_j = \emptyset$, there exists either a vertical line or a horizontal line that separates between R and B_j. Assume for example that the separating line is vertical and that B_j lies to the right of this line, then the box B_i of $\mathcal{B}$ whose right edge contains the right edge of R and B_j are disjoint. We have thus found in linear time two disjoint boxes B_i and B_j in $\mathcal{B}$ (and have proven the well known result: $h(2, 1) = 2$). Clearly, if $\mathcal{B}$ is 2-pierceable, then one of the two piercing points must lie in B_i and the other in B_j.

Consider the following partition of $\mathcal{B}$ into four subsets:

- $\mathcal{B}_1 = \{B \in \mathcal{B} | B \cap B_i \neq \emptyset \text{ and } B \cap B_j = \emptyset\}$

- $\mathcal{B}_2 = \{B \in \mathcal{B} | B \cap B_i = \emptyset \text{ and } B \cap B_j \neq \emptyset\}$
- $\mathcal{B}_3 = \{B \in \mathcal{B} | B \cap B_i = \emptyset \text{ and } B \cap B_j = \emptyset\}$
- $\mathcal{B}_4 = \{B \in \mathcal{B} | B \cap B_i \neq \emptyset \text{ and } B \cap B_j \neq \emptyset\}$

Clearly, if $\mathcal{B}_3 \neq \emptyset$ then $\mathcal{B}$ is not 2-pierceable, and we have a counterexample of size 3. Assume, therefore, that $\mathcal{B}_3 = \emptyset$. Also, all boxes in $\mathcal{B}_1$ must be pierced in B_i, and all boxes in $\mathcal{B}_2$ must be pierced in B_j. Put $B'_1 = \cap\mathcal{B}_1$ and $B'_2 = \cap\mathcal{B}_2$. If one of these boxes is empty, then $\mathcal{B}$ is not 2-pierceable (and we have a counterexample of size 3), and if both are non-empty and $\mathcal{B}_4 = \emptyset$ then we are done.

Now, choose the box $B \in \mathcal{B}_4$ with the highest bottom edge b. B must be pierced either in B'_1 or in B'_2. If B is pierced in B'_k, $1 \leq k \leq 2$, then clearly the best place to put a piercing point inside B'_k is at the intersection point formed by b and the vertical edge of B'_k that is closer to the separating vertical line. We first try to pierce B in B'_1 (if possible). We then check whether the set of all boxes in $\mathcal{B}$ that are still not pierced is 1-pierceable. If this does not yield a solution (i.e., this set is not 1-pierceable and we obtain two disjoint boxes as above), we try the analogue case. If this too does not yield a solution then we may conclude that $\mathcal{B}$ is not 2-pierceable, and we have obtained a counterexample of size at most 13. (The four boxes defining B'_1, plus the four boxes defining B'_2, plus B, plus the two disjoint boxes that are obtained when piercing B in B'_1, and the two disjoint boxes that are obtained when piercing B in B'_2.)

We have described a linear-time algorithm for the 2-piercing problem for a set of planar boxes. Our algorithm immediately implies a Helly-type theorem, since it always provides a counterexample of size at most 13 whenever the input set is not 2-pierceable. (A more careful inspection shows that the maximum size of a counterexample is only 5, see Theorem 13.)

3.2.2 *d*-dimensional boxes

In this section we describe our algorithm for the 2-piercing problem for a set $\mathcal{B}$ of n d-dimensional boxes. This algorithm will imply that $h(d, 2)$ is finite. As mentioned, the exact value for $h(d, 2)$ was given by Danzer and Grünbaum. However, our emphasis here is on the general method which will lead to new results in the subsequent sections.

As before, we first check in linear time whether $\mathcal{B}$ is 1-pierceable, and obtain two disjoint boxes B_i and B_j, if it is not. A box B is a special type of polytope; it is defined by $2d$ halfspaces of the form $x_i \geq a_i$ or $x_i \leq b_i$, for $1 \leq i \leq d$ and real values a_i and b_i. We will distinguish between the

'orientations' of opposite facets of B; thus B has $2d$ facets of $2d$ different orientations.

Our algorithm consists of two stages. In the first stage we obtain a collection of at most 2^{2d} pairs of disjoint regions (boxes) $\mathcal{C} = \{(E_1, F_1), \ldots, (E_m, F_m)\}$ with the following two properties:

(i) If there exists a solution, then there exists a pair $(E_k, F_k) \in \mathcal{C}$ and points $p \in E_k$ and $q \in F_k$ such that $\{p, q\}$ is a piercing set for $\mathcal{B}$.

(ii) Let $\mathcal{A}(B)$ denote the arrangement of $\mathcal{B}$ restricted to the (open) box B. Then the arrangements $\mathcal{A}(E_k)$ and $\mathcal{A}(F_k)$ do not have a common facet orientation. That is, if $\mathcal{A}(E_k)$ has a facet of some orientation o, then $\mathcal{A}(F_k)$ does not have a facet of this orientation (and vice versa).

In the second stage, for each of the pairs (E, F) in $\mathcal{C}$, we search for a solution in which one of the points lies in E and the other lies in F.

We begin the first stage with the disjoint regions B_i and B_j. If there exists a third box in $\mathcal{B}$ which is disjoint from both B_i and B_j, then clearly $\mathcal{B}$ is not 2-pierceable, and we are done. Let o denote a facet orientation that appears in both the clipped arrangements $\mathcal{A}(B_i)$ and $\mathcal{A}(B_j)$. We show below how to eliminate o from one of them. More precisely, we will replace the pair (B_i, B_j) by two pairs of regions such that the first region in both these pairs is contained in B_i, the second region is contained in B_j, and for each of these pairs the orientation o appears in only one of the corresponding clipped arrangements, and property (i) above holds for the collection consisting of these two pairs. Among all the facets of orientation o of the clipped arrangements $\mathcal{A}(B_i)$ and $\mathcal{A}(B_j)$, choose the 'topmost' facet f (i.e., if the halfspaces corresponding to these facets are of the form $x_i \geq a_i$, choose the facet with largest a_i, and if they are of the form $x_i \leq b_i$, choose the facet with smallest b_i). Let B be the box that is partially defined by f. In any solution, B is either pierced in B_i or in B_j. If it is pierced in B_k, $1 \leq k \leq 2$, then we can replace the region B_k by the smaller region $B_k \cap B$ for which the orientation o does not appear in its corresponding clipped arrangement. We thus replace the pair (B_i, B_j) by the (at most) two pairs $(B_i \cap B, B_j)$ and $(B_i, B_j \cap B)$. Note that it is necessary to choose a facet orientation o that appears in both clipped arrangements. We apply this procedure now to the (at most) two pairs that were obtained (thus obtaining (at most) four new pairs) and so on, until none of the current pairs can be further processed. At the end of this *tree process* we are left with a collection $\mathcal{C}$ of at most 2^{2d} pairs of regions so that properties (i) and (ii) hold.

In the second stage, for each of the pairs $(E, F) \in \mathcal{C}$, we search for a solution in which one of the points lies in E and the second in F. Consider a pair $(E, F) \in \mathcal{C}$. Our goal is to shrink the two regions E and F without

loosing all the solutions (if such exist), until eventually one of them becomes a cell of the arrangement of $\mathcal{B}$. We then can place a point anywhere inside this cell, and check whether the set of boxes in $\mathcal{B}$ that are not pierced by this point is 1-pierceable.

Assume w.l.o.g. that E and F are vertically separable. That is, there exists a vertical hyperplane h such that E lies on one of its sides and F lies on the other. As in the planar case, we partition the set $\mathcal{B}$ into the four subsets $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ (with respect to E and F), and compute the regions B_1' and B_2'. The problematic boxes are those boxes in $\mathcal{B}_4$ that still intersect both B_1' and B_2', since such boxes can be pierced both in B_1' and in B_2'. The hyperplane h intersects all but the two vertical facets of these boxes. Consider a pair of opposite facet orientations o and $\overline{o}$. By property (ii) above, o ($\overline{o}$) appears in at most one of the two corresponding clipped arrangements $\mathcal{A}(B_1')$ and $\mathcal{A}(B_2')$. If o and $\overline{o}$ do not appear in the same clipped arrangement, then we can eliminate both of them easily (see Figure 3.2 – Case 1). Assume, for example, that $\overline{o}$ appears in $\mathcal{A}(B_1')$. We can replace the region B_1' (resp. B_2') by the region $B_1' \cap \overline{s}$ (resp. $B_2' \cap s$), where $\overline{s}$ (resp. s) is the topmost halfspace (in the sense explained above) corresponding to a facet of orientation $\overline{o}$ (resp. o) in $\mathcal{A}(B_1')$ (resp. $\mathcal{A}(B_2')$), without loosing all solutions, since all the closing facets of orientation o (resp. $\overline{o}$) of boxes in $\mathcal{B}_4$ that intersect B_1' (resp. B_2') are necessarily 'below' the closing facet of B_1' (resp. B_2') of this orientation. So now assume that o and $\overline{o}$ appear in the same clipped arrangement, say, $\mathcal{A}(B_1')$. Let s (resp. $\overline{s}$) be the topmost halfspace corresponding to a facet of orientation o (resp. $\overline{o}$) in $\mathcal{A}(B_1')$. If $s \cap \overline{s} \neq \emptyset$, then again we can replace the region B_1' by the region $B_1' \cap (s \cap \overline{s})$ without loosing all the solutions (see Figure 3.2 – Case 2). If, however, $s \cap \overline{s} = \emptyset$, then, if B_2' is disjoint from both these halfspaces, then clearly there is no solution (see Figure 3.2 – Case 3), and, otherwise (B_2' is contained in one of these halfspaces), one of the two boxes that are partially defined by these halfspaces, say the one that is defined by $\overline{s}$, must be pierced in B_1' since it does not intersect B_2', so we replace B_1' by the region $B_1' \cap \overline{s}$ (see Figure 3.2 – Case 4) .

We now consider another pair of opposite orientations, and so on, until one of our two regions reduces to a cell of the arrangement of $\mathcal{B}$, and we can conclude as described above. To summarize, we have presented a linear-time algorithm for the 2-piercing problem for the set $\mathcal{B}$. Moreover, we have shown that $h(d, 2)$ is finite, since our algorithm will always provide a counterexample of bounded constant size.

Remark 1: The above tree process can be applied to any number of disjoint regions to remove an orientation that appears in all the corresponding clipped arrangements. If we deal with p regions and c-oriented objects, then after the

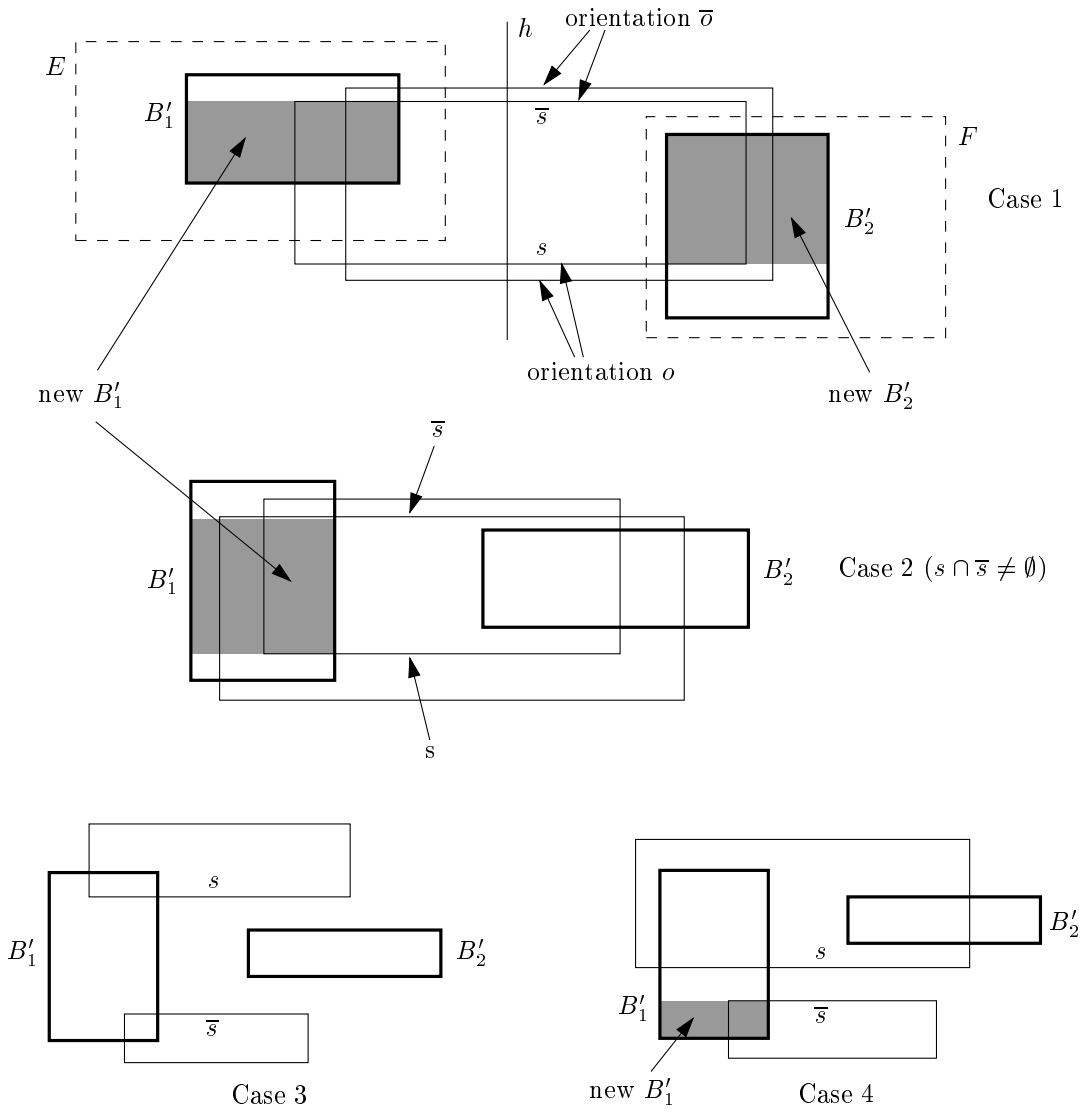


FIG. 3.2 – *Considering a pair of opposite orientations o and $\overline{o}$ in (B'_1, B'_2).*

tree process there exists one region which has at most $c - \lceil \frac{c}{p} \rceil$ orientations.

Remark 2: It is easy to transform the linear algorithm that is induced by the proof of Danzer and Grünbaum (described at the beginning of this section) into a linear algorithm for the 3-piercing problem for rectangles in the plane. Combining the approach of this algorithm with some dynamic data structures, Sharir and Welzl [SW96] obtain $O(n \text{ polylog}(n))$ solutions to the 4- and 5-piercing problems for rectangles in the plane.

3.3 Piercing Sets of c-Oriented Polytopes with 2 Points

Let $\mathcal{H}$ be a set of c halfspaces in $\mathbb{E}^d$, for some constant c. In this section, we consider the class $\mathcal{C}_\mathcal{H}$ of c-oriented convex polytopes above $\mathcal{H}$ (that is, every polytope in $\mathcal{C}_\mathcal{H}$ can be defined as the intersection of at most c translates of halfspaces in $\mathcal{H}$). Let $\mathcal{S}$ be a subset of $\mathcal{C}_\mathcal{H}$ of size n. The main result of this section is a general algorithm for the 2-piercing problem for $\mathcal{S}$, whose time complexity is roughly $O(n^{\min\{\lfloor \frac{c}{2} \rfloor, d\}})$. Thus, whenever $\lfloor \frac{c}{2} \rfloor < d$ (e.g., when $c = d+1$ and $\mathcal{S}$ is a set of simplices), this algorithm is much faster than the naïve algorithm described in the introduction.

Piercing $\mathcal{S}$ with a single point. As the intersection of any subset of $\mathcal{S}$ is either empty or a member of $\mathcal{C}_\mathcal{H}$ (and thus defined by at most c objects of $\mathcal{S}$), we can compute the intersection $R = \cap\mathcal{S}$ in linear time. More precisely, we process the objects of $\mathcal{S}$ one by one, spending $O(c)$ time per object. If $R \neq \emptyset$, then $\mathcal{S}$ is 1-pierceable, and we may pick any point in R. If $R = \emptyset$, then we obtain a subset $\mathcal{S}'$ of cardinality at most $c+1$ such that $\cap\mathcal{S}' = \emptyset$. This implies that $h(\mathcal{C}_\mathcal{H}, \prod^1) \leq \min\{c+1, d+1\}$. However, the case $c < d$ is not interesting, since, in this case, either $\cap\mathcal{S} \neq \emptyset$ or it is possible to decrease the dimension. We can obtain in constant time a minimal subset $\mathcal{S}'' \subseteq \mathcal{S}'$ of size at most $d+1$ such that $\cap\mathcal{S}'' = \emptyset$.

Piercing $\mathcal{S}$ with two points. We distinguish between two cases: $\lfloor \frac{c}{2} \rfloor < d$ and $\lfloor \frac{c}{2} \rfloor \geq d$. For the former case we present below a roughly $O(n^{\lfloor \frac{c}{2} \rfloor})$ time algorithm, while for the latter case we basically apply the naïve algorithm described in the introduction (exploiting the fact that the underlying objects are c-oriented), and obtain a roughly $O(n^d)$ time algorithm. We begin with the case $\lfloor \frac{c}{2} \rfloor < d$. We first check whether $R = \cap\mathcal{S}$ is empty. If $R \neq \emptyset$ then $\mathcal{S}$ is 1-pierceable. Otherwise, we obtain a subset $\mathcal{S}''$, as above, of size at most $d+1$ so that $\cap\mathcal{S}'' = \emptyset$. We now consider all the (constant number of) ways to partition $\mathcal{S}''$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $X = \cap\mathcal{S}_1 \neq \emptyset$ and $Y = \cap\mathcal{S}_2 \neq \emptyset$. Clearly, the regions X and Y are disjoint. For each such partition (producing the pair of disjoint regions X and Y) we proceed as

follows. Apply the 'tree process' of stage 1 of the algorithm of the previous section for the case of d-dimensional boxes. At the end of this stage, we are left with a collection of at most 2^c pairs of regions for which properties (i) and (ii) of the previous section hold. Consider each of these pairs (E, F) separately. As before let $\mathcal{A}(R)$ denote the arrangement of $\mathcal{S}$ within the region R, then property (ii) assures that the arrangements $\mathcal{A}(E)$ and $\mathcal{A}(F)$ do not have facets of the same orientation, and therefore at least one of these arrangements, say, $\mathcal{A}(E)$, does not have more than $\lfloor \frac{c}{2} \rfloor$ facet orientations. Thus the arrangement $\mathcal{A}(E)$ has only faces of dimension k for $d - \lfloor \frac{c}{2} \rfloor \leq k \leq d$, and its combinatorial complexity is therefore only $O(n^{\lfloor \frac{c}{2} \rfloor})$. We compute the arrangement $\mathcal{A}(E)$, and traverse its cells, moving from a cell to an adjacent cell, and maintaining the intersection of the remaining objects that are not pierced by the current cell. (This is done by maintaining c sorted lists, one per orientation; an insert/delete operation costs $O(\log n)$.) This gives us an algorithm of time complexity $O(n^{\lfloor \frac{c}{2} \rfloor} \log n)$.

Notice that if $\mathcal{S}$ consists of simplices of $d+1$ orientations, then this algorithm yields an $O(n^{\lceil d/2 \rceil} \log n)$ time algorithm. The following theorem summarizes the main result of this section.

Theorem 14 *Let $\mathcal{S}$ be a set of n c-oriented convex polytopes in $\mathbb{E}^d$, where $\lfloor \frac{c}{2} \rfloor < d$. It is possible to solve the 2-piercing problem for $\mathcal{S}$ in $O(n^{\lfloor \frac{c}{2} \rfloor} \log n)$ time. In particular, if $\mathcal{S}$ consists of simplices of $d+1$ orientations, then the running time is $O(n^{\lceil d/2 \rceil} \log n)$.*

For the case $\lfloor \frac{c}{2} \rfloor \geq d$, we apply the naïve solution that was described in the introduction, exploiting the additional property that the intersection of any subset of $\mathcal{S}$ is either empty or is a c-oriented polytope. That is, we compute the arrangement of $\mathcal{S}$ and traverse its cells, moving from one cell to an adjacent cell, and maintaining as above the intersection of the remaining set of objects that are not pierced by the current cell. The running time of this algorithm is $O(n^d \log n)$.

In the next section we mention a few cases for which we are able to obtain algorithms that are more efficient that those that are implied by the general algorithm described above.

3.4 Applications

This section is divided into three parts. In the first part we obtain a new Helly-type theorem as a corollary of the general method described in the previous section. In the second part we show that for the class $\mathcal{C}$ of 4-sided convex polygons $h(\mathcal{C}, \prod^2) = \infty$. Finally, in the third part, we mention

several cases for which better bounds than those that are implied by the general method are attainable, due to some 'boundary property' that holds in these cases.

3.4.1 A new Helly-type theorem for homothetic triangles

Consider the case of 3-oriented polygons in the plane, i.e, homothetic triangles, and the appropriate version of the 2-piercing problem. We apply the method of the previous section to obtain a collection of at most $3 \times 6 = 18$ pairs (A_i, B_i) of disjoint regions, such that, in each of them at least one of the regions, say, A_i, is crossed only by edges of a single orientation. Now, considering a pair (A, B), it is clear where to pick the piercing point in A, and we check whether the remaining set of triangles that are not pierced by this point is 1-pierceable. Thus, we have a linear-time algorithm for the 2-piercing problem for this case. Moreover, the algorithm implies a Helly-type theorem, since, if there is no solution, then it is always possible to find a counterexample of size at most $18 \times (3 + 4)$, so that the Helly-number h is at most 126.

Theorem 15 *Let $\mathcal{T}$ be a class consisting of all homothets of a fixed triangle in the plane. Then $h(\mathcal{T}, \prod^2) \leq 126$.*

Remark 1: As mentioned above, this theorem was also discovered independently by Katchalski and Nashtir [KNar], who proved that $h(\mathcal{T}, \prod^2) = 9$ in a direct way.

Remark 2: In order to prove a similar result for the case of homothetic quadrilaterals, i.e., all homothets (or even translates) of a fixed convex quadrilateral, we must specify a way to pick a 'best' point as above in one of the regions of a pair of regions that is obtained by the method of the previous section. However, this does not seem possible. In section 3.4.3 we consider a slightly more general case, the case of 5-oriented convex polygons, and obtain an $O(n \log n)$-time algorithm.

3.4.2 A non Helly-type theorem

Let $\mathcal{C}$ be the class of 4-sided (but not 4-oriented) convex polygons in the plane. We prove below that $h(\mathcal{C}, \prod^2) = \infty$. Consider any triangle Δ, and place $k = 2m + 1$ points $p_1, \ldots, p_k$ on the boundary of Δ; 3 points at the vertices of Δ and $\frac{k-3}{3}$ at each of its edges. Define the 4-sided convex polygon Q_i as the convex hull of $p_i, \ldots, p_{i+m-1}$, for $i = 1, \ldots, k$ (addition of

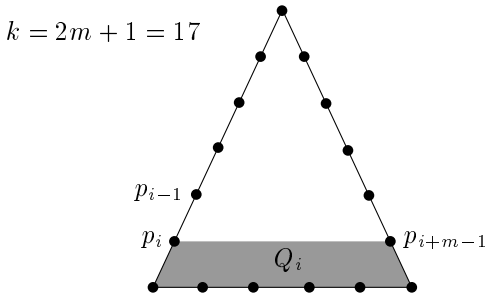


FIG. 3.3 – *The construction for $k = 17$. Every proper subset is 2-pierceable, while the whole set requires 3 points.*

subscripts is done modulo k). Put $\mathcal{S} = \{Q_1, \ldots, Q_k\}$ (see Figure 3.3). Clearly, $Q_i \cap Q_j \neq \emptyset$ iff $\{p_i, \ldots, p_{i+m-1}\} \cap \{p_j, ..., p_{j+m-1}\} \neq \emptyset$. Therefore p_l belongs to exactly m members of $\mathcal{S}$, $l = 1, \ldots, k$, and thus $\mathcal{S}$ is not 2-pierceable. On the other hand, $\mathcal{S} \backslash \{Q_j\}$ is 2-pierceable; take the points p_{j+m} and p_{j+2m}. Since $\lim_{m \to +\infty} k(m) = \infty$, the assertion $h(\mathcal{S}, \prod^2) = \infty$ follows.

Theorem 16 *Let $\mathcal{C}$ be the class of 4-sided convex polygons, then $h(\mathcal{C}, \prod^2) = \infty$.*

3.4.3 Looking at the boundary

In this section, we show that in some cases it is possible to obtain better bounds than those implied by the method of the previous section. In these cases we also apply the general method, but, in the last step, when considering the pairs of regions that were obtained, we are able to decrease the size of the search space. In other words, instead of having to consider all the cells in the clipped arrangement $\mathcal{A}(E)$, we observe that it is sufficient to consider only the arrangement $\mathcal{A}(\partial E)$, where ∂E is the boundary of E, since all the maximal cells of $\mathcal{A}(E)$ are adjacent to ∂E.

3.4.4 $\leq$ 5-oriented polygons with 2 points

Let us consider any class $\mathcal{C}_{\mathcal{H}}$, $|\mathcal{H}| \leq 5$, in the plane. That is, a class of at most 5-oriented convex polygons. In this case, let A be the region of a pair (A, B) for which at most $\lfloor \frac{5}{2} \rfloor = 2$ orientations appear in its clipped arrangement $\mathcal{A}(A)$. It is easy to see that all the maximal cells of $\mathcal{A}(A)$ are adjacent to the boundary ∂A of A. We therefore compute in $O(n \log n)$ time the intersection of ∂A with the n objects. Then, we traverse the $O(n)$ intervals along ∂A (each object may give rise to at most $O(c)$ intervals). The last step is done

by maintaining dynamically the intersection property (1-pierceability) of the complementary set of objects. We thus obtain an $O(n \log n)$ time algorithm.

Theorem 17 *Let $\mathcal{S}$ be a set of n 5-oriented convex polygons. Then it is possible to find a piercing pair for $\mathcal{S}$ (if such exists) in $O(n \log n)$ time.*

3.4.5 Homothetic triangles with 3 points

We can use the linear-time algorithm of section 3.4.1 in order to obtain an $O(n \log n)$ time algorithm for piercing a set $\mathcal{S}$ of n homothetic triangles in the plane with 3 points. Indeed, we first check in linear time whether $\mathcal{S}$ is $\leq$ 2-pierceable. If $\mathcal{S}$ is $\leq$ 2-pierceable then we are done, otherwise we consider the counterexample that was obtained which is of constant size. We make all the possible assumptions for piercing this set with three points. Each assumption gives rise to a set of three disjoint convex regions (homothetic triangles). We apply the 'tree process' method to each of these triplets, and end up with $O(1)$ triplets of convex disjoint regions. Consider one such triplet (A, B, C). One of its regions, say A, has at most 2 orientations appearing in its clipped arrangement $\mathcal{A}(A)$ (since, if all regions have 3 orientations, then we can remove at least one orientation by applying the 'tree process'). It is easy to see that A has the boundary property, i.e., all the maximal cells of $\mathcal{A}(A)$ are adjacent to the boundary ∂A of A. We may therefore restrict our attention to the $O(n)$ intervals along ∂A (each object intersecting A defines at most 2 intervals along ∂A). We traverse ∂A moving from one endpoint of an interval to the next. At the current endpoint p, we check whether the set of objects that are not pierced by p is 2-pierceable. We can maintain the 2-pierceability property of this complementary set under insertions and deletions of objects in logarithmic time, using a linear-size dynamic data structure that is reminiscent of the data structures constructed in [SW96] for the 4- and 5-piercing problems for rectangles in the plane. We describe the dynamic data structure in the proof of Lemma 19 below. We thus obtain an $O(n \log n)$ algorithm.

Theorem 18 *The 3-piercing problem for a set of n homothetic triangles can be solved in $O(n \log n)$ time.*

The previous theorem relies on the following lemma:

Lemma 19 *The 2-pierceability property of a set of n homothetic triangles can be maintained dynamically in $O(\log n)$-time per insertion/deletion.*

Proof. Let R be a set of n homothetic triangles in the plane. We need to show how to maintain the 2-pierceability property (i.e., whether R is 2-pierceable or not) when a homothetic triangle is either inserted or deleted from R. For each orientation o, draw a line through the 'topmost' edge of orientation o. Let L_R denote the set of these three lines. According to Sharir and Welzl [SW96], if R is 2-pierceable, then it can be pierced by two points lying on the lines of L_R and we may assume that one of the piercing points lies on a vertex of the arrangement of L_R.

Our data structure consists of three balanced binary trees, one per each orientation. Assume w.l.o.g. that the first orientation is the vertical orientation and that the triangles lie to the left of their vertical edge. Then the first tree is built as follows. Let $x_1 < x_2 < \cdots < x_n$ be the projections of the vertical edges onto the x-axis. Store these values in sorted order in the leaves of a balanced binary tree; x_1 in the leftmost leaf, etc. A node u of the tree represents the corresponding subset R_u of R (i.e. R_u consists of the triangles in R for which the projection of their vertical edge is stored in a leaf of the subtree rooted at u). We associate with u the region $T_u = \cap R_u$ (which is either empty or a homothetic triangle). The tree can be constructed in a bottom-up manner in $O(n)$ time, after sorting the $x_i's$ in $O(n \log n)$ time. The second and third trees are constructed in a similar way.

We need to determine in logarithmic time whether R is 2 pierceable or not. First notice that we can obtain L_R, the three lines forming the 'location domain' of R, in constant time, since they are the lines corresponding to the minimal (or maximal) values stored in the three trees. Now let v be a vertex of the arrangement of L_R. We need to determine whether the subset R_v of R consisting of the triangles in R that are not pierced by v has a non-empty intersection. We can obtain R_v (implicitly) as the union of a logarithmic number of represented subsets (see below), and since we have the intersection of each of these subsets, we can compute $\cap R_v$ in logarithmic time.

How do we obtain R_v as a collection of represented subsets? Notice that v lies in a triangle Δ iff it lies in the three half-planes defined by the edges of Δ. Let R_v^i, $i = 1, 2, 3$, denote the subset of R consisting of the triangles whose half-plane of the i'th orientation doesn't contain v. Clearly $R_v^1 \cup R_v^2 \cup R_v^3 = R_v$. We will perform a query in the i'th tree to obtain the subset R_v^i as the union of a logarithmic number of represented subsets. Consider for example the first tree, and let v_x be the projection of v onto the x-axis. We search with v_x in the first tree (beginning at the root) for the smallest value x that is stored in a leaf of the tree for which $v_x \leq x$. This search defines a root-leaf path in the tree. Consider the nodes of the tree that do not belong to this path and are the left children of nodes on this path. The number of these nodes is logarithmic and the union of their corresponding represented sets is

exactly R_v^1.

Using standard techniques [Meh84b] we can update this data structure when a triangle is either inserted or deleted from R in logarithmic time. Thus the 2-pierceability property can be maintained in logarithmic time.

□

Remark: We cannot obtain a similar theorem for the case of at most 5-oriented convex polygons considered in the previous subsection, since we do not have a Helly-type theorem for the 2-piercing problem in that case.

3.4.6 Unit disks

Let us finally consider the class $\mathcal{D}$ of unit disks in $\mathbb{E}^2$. Let $\mathcal{S} \subset \mathcal{D}$ be a set of n unit disks. We wish to pierce $\mathcal{S}$ with at most two points. We first check in linear time whether $\mathcal{S}$ can be pierced with a single point or not (using an LP-type program). If $\cap\mathcal{S} = \emptyset$, then we can proceed only if the diameter of the set of center points of the disks in $\mathcal{S}$ is at least 3. We can compute this diameter in $O(n \log n)$ time. Let A and B be the two disks corresponding to the diameter pair of centers. Observe that p disks intersecting both A and B have a non-empty intersection within A iff they have a non-empty intersection on the boundary of A. This means that we can restrict our attention to the boundary ∂A of A and check the at most n intervals on ∂A. We can maintain the intersection of the remaining set of objects dynamically under both insertion and deletion in $O(\log^2 n)$ (see [HS91]). Thus, we obtain an $O(n \log^2 n)$ time algorithm provided that there exist two disks in $\mathcal{S}$ such that their centers are at distance at least 3. Note that the 2-piercing problem of a set of unit disks is equivalent to the decision problem where one wants to determine whether a set of n points can be covered by two unit disks. So that our condition implies that the diameter of the set of points is at least 3.

Lemma 20 *Let $\mathcal{S}$ be a set of n points in the plane with diameter at least 3. Then there exists a simple algorithm that determines in $O(n \log^2 n)$ time whether $\mathcal{S}$ can be covered by two unit disks.*

3.5 Conclusion

The main result of this paper is a general method for solving the 2-piercing problem for a set $\mathcal{S}$ of d-dimensional c-oriented polytopes. This method is much more efficient than the naïve method whenever $\lfloor \frac{c}{2} \rfloor < d$ (e.g., when $c = d + 1$ and $\mathcal{S}$ is a set of simplices). It yields a linear-time algorithm and a

new Helly-type theorem for the 2-piercing problem for homothetic triangles in the plane. This Helly-type theorem allows us to apply the method also to the 3-piercing problem for homothetic triangles and obtain an $O(n \log n)$ algorithm, since, in general, the method is based on the existence of a Helly-type theorem for the corresponding $(k-1)$-piercing problem.

Our method applied to the 2-piercing problem for d-dimensional axis-parallel boxes yields an alternative linear-time algorithm to the one implied in a paper of Danzer and Grünbaum [DG82] and also implies the corresponding Helly-type theorems of [DG82].

Conclusion et perspectives

Les travaux effectués lors de cette thèse portent sur les algorithmes adaptatifs, une classe d'algorithmes dont la complexité en temps de calcul (ou en toute autre ressource critique comme l'espace mémoire, etc.) dépend à la fois de la taille des entrées mais aussi d'autres paramètres, dits *significatifs*. Lorsque ces paramètres sont liés à la sortie, on parle d'algorithme sensible à la sortie.

En considérant l'union d'un ensemble $\mathcal{S}$ de n intervalles, nous avons montré qu'il était possible d'obtenir un algorithme en temps $O(n \log c^*)$ où c^* est le nombre minimal de points requis pour percer $\mathcal{S}$. De plus, en considérant n et c^* comme paramètres du problème, nous avons montré l'optimalité de cet algorithme. Ceci met en évidence l'intérêt qu'il y a à analyser un même problème avec d'autres paramètres que la seule taille des entrées. En effet, la solution précédente demandait $\Omega(n \log n)$ opérations élémentaires sur le même modèle de machine RAM. Si nous considérons comme paramètre le nombre de composantes connexes C de l'union des intervalles de $\mathcal{S}$, nous ne pouvons pas obtenir d'algorithme adaptatif qui batte la borne $\Omega(n \log n)$ quelle que soit la valeur de C. Ainsi, la notion d'optimalité dépend à la fois du modèle de machine considéré mais aussi des paramètres choisis.

L'introduction d'un nouveau paramètre significatif pourrait permettre d'expliquer théoriquement le bon comportement expérimental d'un algorithme ayant une mauvaise complexité "théorique". Par exemple, on aimerait justifier les performances expérimentales de l'algorithme Quickhull [BDH93] en introduisant de nouveaux paramètres dans l'analyse de sa complexité. Il serait également intéressant de considérer de nouveaux paramètres significatifs pour le problème de calcul d'enveloppes convexes d'un ensemble $\mathcal{S}$ de points de l'espace euclidien $\mathbb{E}^d$. Deux difficultés majeures apparaissent dans

ce cadre : d'une part trouver ces paramètres à proprement parler et d'autres part arriver à construire un algorithme plus efficace vis-à-vis de ces paramètres.

Dans la seconde partie de cette thèse, nous nous sommes intéressés au calcul d'enveloppes dans le plan. Nous utilisons une approche *composite* afin d'obtenir un algorithme sensible à la sortie pour le calcul d'enveloppes convexes de n objets planaires. L'approche composite combine à la fois le paradigme *groupement par paquets* et le paradigme *fusion avant conquête*. Cette approche met en jeu un paramètre k, dit de *coloration*, qui est défini comme étant le nombre minimal de couleurs requis pour colorier les objets de façon que deux objets d'une même couleur ne se coupent pas. Lorsque nous donnons en entrée les objets coloriés, nous obtenons un algorithme de complexité $O(n \log h + kh)$ où h est la taille de la sortie. En groupant les objets par paquets, nous formons un nouvel ensemble de $O(n\beta(h))$ objets qui a la même enveloppe convexe et tel que $k \leq \lceil \frac{n}{h} \rceil$ où $\beta(\cdot)$ est une fonction qui croit extrêmement lentement, en rapport avec les longueurs maximales des suites de Davenport-Schinzel. Ainsi, pour tout ensemble d'objets planaires nous obtenons un algorithme de complexité $O(n\beta(h) \log h)$. Ces résultats se transposent aisément au calcul d'enveloppes supérieures de segments ou plus généralement de fonctions. Il serait intéressant de considérer deux extensions de ce travail:

- La première extension consisterait à tenir compte d'un nouveau paramètre significatif, v, pour le calcul de l'enveloppe supérieure d'un ensemble $\mathcal{S}$ de n segments. Ce paramètre v est défini comme étant le nombre d'extrémités distinctes de $\mathcal{S}$ ($n = O(v^2)$). En effet, Arkin *et al.* [AHK+91, AHK+94, AHK+95] ont montré que la taille de cette enveloppe est $O(v \log v)$. Il serait intéressant d'utiliser n, v, k et h comme paramètres significatifs et d'obtenir un algorithme plus efficace dans le cas où $n = \Omega(v^2)$.
- La seconde extension possible serait d'étudier le calcul de l'enveloppe supérieure d'hypersurfaces ou de simplexes en généralisant la méthode en dimensions supérieures. Actuellement, le meilleur algorithme [AM92,

AM93] connu pour calculer de façon adaptative l'enveloppe supérieure de n triangles a une complexité en $O(n^{1+\epsilon} + n^{\frac{2}{3}+\epsilon}h^{\frac{2}{3}})$ où h est la taille de la sortie en cellules élémentaires. Cet algorithme utilise des primitives, d'implantation difficile, de lancer de rayon, basées sur la recherche paramétrique.

Il serait également fort intéressant de connecter par le biais d'un nouveau modèle de machine les algorithmes adaptatifs aux algorithmes parallèles. Notons que dans l'approche *groupement par paquets*, nous pouvons visualiser chaque paquet comme un ensemble de données attribué à sur un processeur. Étudier les algorithmes adaptatifs permet également de mieux cerner les paramètres intrinsèques d'un problème et souvent, dans un second temps, d'obtenir des algorithmes efficaces en pratique.

Dans la troisième partie de cette thèse nous nous sommes intéressés au problème qui consiste à percer un ensemble d'objets $\mathcal{S}$, c'est-à-dire de calculer un ensemble de points $\mathcal{P}$ tel que chaque objet de $\mathcal{S}$ contienne au moins un point de $\mathcal{P}$. Si k est le cardinal de $\mathcal{P}$, on dit que $\mathcal{S}$ est perçable par k points. Trouver un ensemble de points de taille minimale perçant $\mathcal{S}$ est un problème NP-complet (même dans le cas de carrés). Nous avons attaqué ce problème sous deux aspects:

- Le premier aspect consiste à vérifier précisément si $\mathcal{S}$ est perçable par k points pour de faibles valeurs de k ($k \leq 4$). Nous obtenons une batterie d'algorithmes efficaces pour des classes variées d'objets. Certains de ces algorithmes constituent les preuves mêmes de l'existence de théorèmes de type Helly. Nous aimerions étendre la méthodologie aux nombres de Gallai et $C(p, q)$. Ces algorithmes permettent d'obtenir des algorithmes adaptatifs pour de faibles valeurs de k. En effet, étant donné $\mathcal{S}$, nous cherchons à savoir si $\mathcal{S}$ est perçable par $1, ..., k$ points itérativement. Si $\mathcal{S}$ est perçable par i points, nous ne testerons pas, bien évidemment, s'il est perçable par $i + 1, ..., k$ points.
- Le second aspect consiste à calculer un ensemble $\mathcal{P}$ de points perçant un ensemble $\mathcal{S}$ de n objets tel que $c = |\mathcal{P}|$ dépende de la taille c^* d'une

solution optimale. Nous nous sommes penchés sur le cas, plus simple, des boîtes isothétiques dans l'espace euclidien $\mathbb{E}^d$. Nous avons proposé une heuristique adaptative de complexité $O(n \log c + dn)$ qui donne un ensemble de c points perçant un ensemble de boîtes $\mathcal{B}$ tel que l'approximation obtenue soit compétitive, c'est-à-dire sensible à la solution optimale c^* ($c = O(c^*)$ dans le cas des boîtes contraintes et $O(c^{*d})$ sinon). Il serait judicieux de considérer la maintenance dynamique (on-line ou off-line) d'un ensemble de points $\mathcal{P}$ perçant $\mathcal{B}$ en temps polylogarithmique. Une difficulté majeure semble provenir du cas des boîtes dites non contraintes. Il serait également judicieux de considérer ce problème dans le contexte du parallélisme. Hochbaum et Maass [HM84] ont donné un schéma d'approximation polynomial (Polynomial-Time Approximation Scheme) dans le cas d'objets congruents. On aimerait étendre cet algorithme dans le cas d'objets quelconques. Une première tentative serait de considérer le cas des boîtes isothétiques. Nous pensons que l'obtention de réponses à ces questions permettrait de mieux comprendre les paramètres et difficultés intrinsèques du problème et ainsi d'obtenir des algorithmes adaptatifs plus généraux.

Il y a fort à parier que la théorie des algorithmes adaptatifs se developpera dans les prochaines années. Et ceci pour deux raisons : d'une part pour justifier le bon comportement expérimental de certains algorithmes et d'autre part pour considérer des algorithmes plus efficaces en tenant compte de nouveaux paramètres liés à la nature des données ou de leur(s) sortie(s).

"Longue vie aux algorithmes adaptatifs! "

18 octobre 1996,

Franck Nielsen.

Table des figures

Liste des Algorithmes

Annexe A

Notations et conventions utilisées

Afin de rendre cohérente la lecture de cette thèse, nous avons adopté les conventions d'écriture suivantes:

- Les noms des méthodologies sont désignées en *italique* comme par exemple *diviser pour régner*.
- L'équivalent anglais est noté entre parenthéses en fonte sans serif (divide-and-conquer).
- Les problèmes sont référencés en majuscule en utilisant la fonte sans-serif (exemple:UNIFORM GAP).

J'ai essayé d'utiliser le plus souvent possible la traduction française des mots anglais quand elle existait;

Ensemble – Partitionnement:

$\mathcal{A} < \mathcal{B}$	Comparaison ensembliste $\forall\ a \in \mathcal{A}, \forall\ b \in \mathcal{B},\ a < b$
$\mathcal{P} = \uplus_{i \in I} \mathcal{P}_i$	Partitionnement ordonné de $\mathcal{P}$
	$\forall\ i, j\ , i < j,\ \mathcal{P}_i < \mathcal{P}_j$

Complexité:

$\Omega(f(n))$	Borne inférieure en $f(n)$
$O(f(n))$	Borne supérieure en $f(n)$
$\Theta(f(n))$	Borne inférieure et supérieure en $f(n)$
$\tilde{\mathrm{O}}(f(n))$	Complexité d'un algorithme randomisé en $f(n)$
$\bar{O}(f(n))$	Coût amorti sur n opérations
$O_d(f(n))$	Complexité en considérant d fixé

Enveloppe Convexe:

$\text{conv}_{i\in I}(p_i)$	Enveloppe convexe des objets $p_i,\ i \in I$
$\mathcal{CH}(\mathcal{P})$	Enveloppe convexe de l'ensemble $\mathcal{P}$
$\mathcal{CH}^+(\mathcal{P})$	Enveloppe convexe supérieure $\mathcal{P}$
$\mathcal{CH}^-(\mathcal{P})$	Enveloppe convexe inférieure $\mathcal{P}$
ϕ_i	Nombre de i-faces d'un polytope
ϕ	Nombre total de faces : $\phi = \sum_{i=0}^{d-1} \phi_i$
$\cap H$	polytope intersection résultat de l'intersection des demi-espaces contenus dans H

Vecteurs maximaux:

$maxima(\mathcal{P})$	Ensemble des vecteurs maximaux de $\mathcal{P}$
$\mathcal{M}$	Ensemble des vecteurs maximaux: $\mathcal{M} = maxima(\mathcal{M})$

Décomposition:

$\Delta(R)$	Décomposition du polytope intersection $\cap R$ en triangulant ses facettes et en prenant la fermeture convexe par rapport à un point intérieur.
$\Delta^*(R)$	Ensemble des hyperplans coupant le polytope intersection $\cap R$.

Opérateur:

$\oplus$	Différence symétrique : $A \oplus B = (A \cup B) \backslash (A \cap B)$

Perçabilité:

$c^*(\mathcal{S})$	Nombre minimal de points perçant $\mathcal{S}$
$b^*(\mathcal{S})$	Nombre minimal de boîtes de $\mathcal{S}$ mutuellement disjointes
$c(\mathcal{S})$	Nombre de points perçant $\mathcal{S}$ par une heuristique
$C(p,q)$	Nombre de points perçant une famille telle que dans chaque sous-ensemble de p éléments q d'entre-eux se coupent mutuellement
$N(p,q;d)$	Nombre $C(p,q)$ pour les boîtes en dimension d

$\prod^d(\mathcal{S})$	Propriété d'être perçable par k points
$h(\mathcal{F}, P)$	Nombre de Helly de la famille $\mathcal{F}$ pour la propriété P

Annexe B

Le calcul de la médiane:

On présente dans cette annexe le calcul du k$^{\text{e}}$ élément d'un tableau de n élements ordonnés (mais non triés). Nous décrivons successivement l'algorithme séquentiel et parallèle pour un modéle de calcul EREW PRAM SIMD. Le lecteur intéressé trouvera le calcul de la médiane par un algorithme séquentiel dans [AHU74, AHU83] et l'algorithme parallèle dans [Akl84, Akl85a]. On citera également les références [ACG$^+$88, ACG$^+$85, AL93, Ata92, AG86, Cho80, Dat94, DFSV92, DS88, Goo87, RS87, Wag85] pour une introduction à la géométrie parallèle algorithmique, plus particulièrement aux techniques de parallèlisation des algorithmes géométriques séquentiels connus.

B.1 Description de l'algorithme séquentiel:

Tout au long de cette thèse, le partage des données s'est effectué grâce à l'algorithme de la médiane. Le calcul de la médiane rentre dans un cadre plus général: celui du calcul du k^{e} plus petit élément d'un ensemble $\mathcal{S}$ de n éléments. On peut trouver l'algorithme du calcul de la médiane dans des ouvrages généraux algorithmiques [AHU74, AHU83]. Nous détaillons ci-dessous la méthode (cf. B.1) et montrons sa complexité optimale linéaire $\Theta(n)$. La procédure décrite est appelée `Select`. Le calcul de la médiane se fait par l'appel initial `Select`$(\lfloor \frac{n}{2} \rfloor, \mathcal{S})$.

La méthodologie utilisée est celle de *bisection-décimation* (searching-and-pruning) dans laquelle on partitionne l'ensemble des données puis on localise la solution dans cette partition et on lance récursivement l'algorithme sur le paquet de la partition contenant la solution. Cette méthodologie est très connue et fût appliquée notamment pour la programmation linéaire en temps linéaire par Megiddo.

Algorithme B.1: *Calcul du k^e plus petit élément d'un ensemble de n éléments.*

Procedure *Select*(k, S);
début
si $|S| < 50$ alors début
TrierCroissant(S);
renvoyer $S[k]$;
fin;
sinon
début
$\mathcal{S} \leftarrow$*DivisePaquet5*(S);
(* on divise S par $\lceil \frac{|S|}{5} \rceil$ paquets d'au plus 5 éléments *)
TrierCroissant$(\mathcal{S})$;
(* on tri les paquets de 5 éléments *)
$\mathcal{M} \leftarrow$*Médiane5*$(\mathcal{S})$;
(* $\mathcal{M}$ est l'ensemble des médianes des paquets $\mathcal{S}$
Cela correspond a prendre le 3^e élément de chaque paquet *)
$m \leftarrow$*Select*$(\lceil \frac{|\mathcal{M}|}{2} \rceil, \mathcal{M})$;
(* on prend la médiane des médianes *)
$S_1 \leftarrow$*ChoixInf*(S,m);
(* on prend les éléments inférieurs à m de S *)
$S_2 \leftarrow$*ChoixEgal*(S,m);
(* on prend les éléments égaux à m de S *)
$S_3 \leftarrow$*ChoixSup*(S,m);
(* on prend les éléments supérieurs à m de S *)
si $|S_1| \geq k$ alors *renvoyer Select*(k,S_1);
sinon
si $(|S_1| + |S_2|) \geq k$ *renvoyer* m;
sinon *renvoyer Select*$(k - |S_1| - |S_2|, S_3)$;
fin;
fin

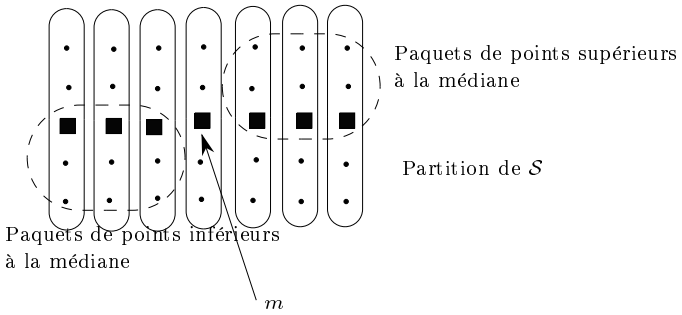


FIG. B.1 – *Partitionnement de S en $\mathcal{S}$ pour le calcul du k^e élément.*

B.2 Analyse de la complexité:

Les paquets S_1 et S_3 ont au plus $\frac{3n}{4}$ éléments de S. Cela se vérifie immédiatement en considérant la médiane des médianes m qui sépare les paquets $\mathcal{S}$ en deux zones: ceux qui ont une médiane qui est supérieure à m et ceux qui ont une médiane qui est inférieure à m. Parmi les paquets qui ont une médiane supérieure à m, on sait qu'il y a dans chaque paquet 3 éléments qui sont supérieurs ou égaux à m. On en déduit donc qu'il y au plus $n-3\lfloor\frac{n}{10}\rfloor$ éléments qui sont inférieurs strictement à m. En tenant compte que $n > 50$, on obtient $n - 3\lfloor\frac{n}{10}\rfloor \leq \frac{3n}{4}$. Le même raisonnement s'applique pour les éléments strictement supérieurs. Le partitionnement est illustré sur la figure B.1. La complexité de l'algorithme est donc donnée par:

$$c(n) = \begin{cases} O(1) & \text{si } n \leq 50, \\ c(\frac{n}{5}) + c(\frac{3n}{4}) + k \times n & \text{sinon.} \end{cases}$$

Supposons par hypothése de récurrence que $c(n) \leq A \times n$ où A est une constante et injectons ce résultat dans l'équation précedente. On obtient:

$$c(n) \leq A \times (\frac{n}{5} + \frac{3n}{4}) + k \times n$$

$$\leq A \times \frac{19n}{20} + k \times n \leq A \times n$$

dès que $A \geq 20k$. Une borne inférieure est trivialement $\Omega(n)$. L'algorithme est donc optimal.

Théorème 11 *On peut calculer le k^e élément d'un ensemble de n éléments par la méthodologie de bisection-décimation (*searching-and-pruning*) en temps optimal $\Theta(n)$.*

B.3 Description de l'algorithme parallèle:

Le calcul du k^{e} élément d'un ensemble totalement ordonné $\mathcal{S}$ mais non trié par un algorithme parallèle reprend les étapes de l'algorithme séquentiel `Select()` du paragraphe précédant en adaptant chaque étape au modéle EREW (Exclusive Read Exclusive Write) PRAM (Parallel Random Access Memory) SIMD (Single Instruction Multiple Data). Ce modèle de machine parallèle fût l'un des premiers étudiés. Même s'il ne reflète pas de façon satisfaisante les architectures et les caractéristiques des machines actuelles, il a l'avantage d'être simple et ressemblant le plus au modèle universel de la machine séquentielle de Van Neumann. L'algorithme est décrit dans l'encadré B.2 puis on étudie la complexité de celui-ci en raffinant les détails des étapes. On suppose que l'on dispose de $P(n) = n^{1-\epsilon},\ 0 < \epsilon < 1$ processeurs numérotés de 1 à $n^{1-\epsilon}$. L'algorithme détermine alors le k^{e} élément en temps $O(n^{\epsilon})$. Le travail de l'algorithme est alors $T(n) = C(n) \times P(n) = O(n^{\epsilon}) \times O(n^{1-\epsilon}) = O(n)$. L'algorithme atteint la borne optimale du travail.

On détaille et examine la complexité $c(n)$ de l'algorithme étape par étape ci-dessous:

Étape 1 On diffuse la valeur k et A du tableau S à tous les processeurs. Comme il y a $n^{1-\epsilon}$ processeurs, la diffusion coûte (équivalente à un prefixe parallèle) $O(\log n^{1-\epsilon})$. Le processeur P_i calcule son paquet correspondant en $O(1)$; en effet, P_i s'occupe des éléments $p_i : [A+(i-1).n^{\epsilon}...A+i.n^{\epsilon}-1]$.

Étape 2 On utilise la procédure séquentielle *Select()* pour calculer l'élément médian du paquet p_i. Chaque paquet a une taille n^{ϵ} et le calcul de la médiane coûte $\Theta(n^{\epsilon})$.

Étape 3 On calcule récursivement la médiane des médianes par appel de *Select-Par()*. Cette étape coûte $c(n^{1-\epsilon})$.

Étape 4 Chaque processeur P_i calcule sa contribution au trois paquets en calculant S_1^i, S_2^i, S_3^i en $O(n^{\epsilon})$. Il suffit alors de fusionner les $S_i^1, 1 \leq i \leq n^{1-\epsilon}$ afin d'obtenir S_1 (de même pour S_2 et S_3). On diffuse m au $n^{1-\epsilon}$ processeurs en temps $O(\log n^{1-\epsilon})$. L'étape de fusion des $S_i^1, 1 \leq i \leq n^{1-\epsilon}$ prend un temps $O(n^{\epsilon})$ en utilisant le procédé suivant: soit s_i la taille de S_i^1 ($s_i = |S_i^1|$), on calcule tout d'abord les sommes partielles: $z_i = \sum_{j=1}^{i} s_j,\ 1 \leq i \leq n^{1-\epsilon}$. Ces sommes partielles sont calculées en $O(\log n^{1-\epsilon})$. Par convention, on fixe $z_0 = 0$; chaque processeur P_i écrit alors sa liste S_i^1 à partir de la position $B + z_i$ où B est la case mémoire de début de S_1. Ainsi, le temps requit pour fusionner les sous-tableaux est $\max_{i=1..n^{1-\epsilon}} z_i$ qui est borné trivialement par n^{ϵ}.

Étape 5 Puisque m est la médiane des paquets p_i, $n^{1-\epsilon}/2$ éléments de S sont plus grands (ou plus petits). Or chaque élément m_i de M est plus petit

Algorithme B.2: *Algorithme parallèle de calcul du k^e plus petit élément d'un ensemble de n éléments.*

Procedure *SelectPar*(k, S);
début
si $|S| < 50$ alors début
(* 50 est une constante arbitraire qui assure le $O(1)$ de cette étape *)
TrierCroissant(S); (* sur le processeur P_1 *)
renvoyer $S[k]$;
fin;
sinon début
1 Diviser S en $n^{1-\epsilon}$ paquets de taille n^ϵ
à chaque processeur P_i, $1 \leq i \leq n^{1-\epsilon}$;
2 Faire en parallèle
Processeur P_i calcule m_i la médiane
du paquet $[i.n^\epsilon, (i+1).n^\epsilon - 1]$
par l'algorithme séquentiel *Select()*: $M[i] = m_i$;
fin;
3 Calculer la médiane m des $m_i, 1 \leq i \leq n^{1-\epsilon}$
en appelant *SelectPar*$(\lceil \frac{|M|}{2} \rceil, M)$;
4 Subdiviser S en trois paquets S_1, S_2, S_3 comme dans *Select()*;
(* S_1 éléments plus petits que m
(* S_2 éléments égaux à m
(* S_1 éléments plus grands que m *)
5 si $|S_1| \geq k$ alors *renvoyer SelectPar*(k,S_1);
sinon
si $(|S_1| + |S_2|) \geq k$ *renvoyer* m;
sinon *renvoyer SelectPar*$(k - |S_1| - |S_2|,S_3)$;
fin;
fin

(ou plus grand) de $n^\epsilon/2$ éléments de S. Il s'en suit que S_1 a une taille d'au plus $\lceil \frac{3n}{4} \rceil$ éléments de S. La même borne est trouvée pour S_3 (S_2 quand a lui ne pose pas de problèmes puisque si l'élément à chercher se trouve dans sa zone alors c'est m).

En rassemblant tous les éléments de complexité, on aboutit à la formule de récurrence suivante:

$$c(n) = \begin{cases} O(1) & \text{si } n \leq 50, \\ O(\log n) + n^\epsilon + c(n^{1-\epsilon}) + c(\lceil \frac{3n}{4} \rceil) & \text{sinon.} \end{cases}$$

Le temps d'execution $c(n)$ de l'algorithme est donc $O(n^\epsilon)$. Le travail est $O(n^\epsilon \times n^{1-\epsilon}) = O(n)$. L'algorithme est donc optimal au sens où son travail atteint la borne optimale séquentielle. Il permet en outre de construire des algorithmes de tri PRAM optimaux. On peut mettre en relief la question suivante:

> Existe-t-il un algorithme parallèle EREW PRAM SIMD calculant le k^{e} élément d'un tableau S en temps $O(\frac{n}{\log^u n})$ avec $O(\log^u n)$ processeurs? (**S.G. Akl 85 [Akl85a]**)

La difficulté soulevée est celle de la diffusion et du calcul de la somme partielle qui requiert $O(\log n)$ unité de temps et est donc négligeable par rapport à $O(n^{1-\epsilon})$, $0 < \epsilon < 1$ dans l'étude de la complexité.

Annexe C

Graphe d'incidence et effeuillage

Dans cette annexe, nous décrivons tout d'abord le graphe d'adjacence d'un polytope et en donnons quelques propriétés. Nous montrons dans la seconde partie comment on peut calculer les points extrêmes par programmation linéaire. Finalement, nous donnons le calcul du graphe d'incidence grâce à la méthodologie de l'effeuillage.

C.1 Propriétés du graphe d'adjacence

On donne ci-dessous quelques lemmes qui reflétent la structure du polytope (sous la condition de position générale des points de $\mathcal{P}$ sinon on considére encore une fois la sensibilité à la sortie forte ou faible).

Lemme 14 *Chaque sous-facette (*ridge*, $(d-2)$-face) d'un polytope est contenue dans exactement deux facettes ($(d-1)$-faces).*

Lemme 15 *Soit $\mathcal{P}$ un d-polytope et O un point à l'intérieur de $\mathcal{P}$ alors pour chaque facette F de $\mathcal{P}$ on désigne par b_F le vecteur tel que $< b_F, x >= 1$, $\forall\ x \in F$ et $< b_F, x > \leq 1$, $\forall x \in \mathcal{P}$. Et soit G une face (i-face, $i \leq d-2$) de $\mathcal{P}$ et Φ les* facettes *de $\mathcal{P}$ qui contiennent G alors l'ensemble $H = \{x| < b, x >= 1\}$ est un hyperplan support de $\mathcal{P}$ qui contient G si et seulement si $b = \sum_{F \in \Phi} \lambda_F b_F$ avec $0 \leq \lambda_F \leq 1$ pour toute face F et $\sum_{F \in \Phi} \lambda_F = 1$.*

Lemme 16 *Soit $\mathcal{P}$ un d-polytope de $\mathbb{R}^d$ et p un point qui n'est contenu dans aucun espace affine convexe de $\mathcal{P}$. On note $\mathcal{P}'$ le polytope enveloppe convexe de $\mathcal{P}$ et p: $\mathcal{P}' = conv(\mathcal{P}, p)$. On dit que G est une face de $\mathcal{P}'$ qui contient p si et seulement si G' est de la forme $conv(G, p)$ où G est une face de $\mathcal{P}$ qui est contenue dans une facette de $\mathcal{P}$ pour laquelle p est au-dessus et dans une facette pour laquelle il est en-dessous.*

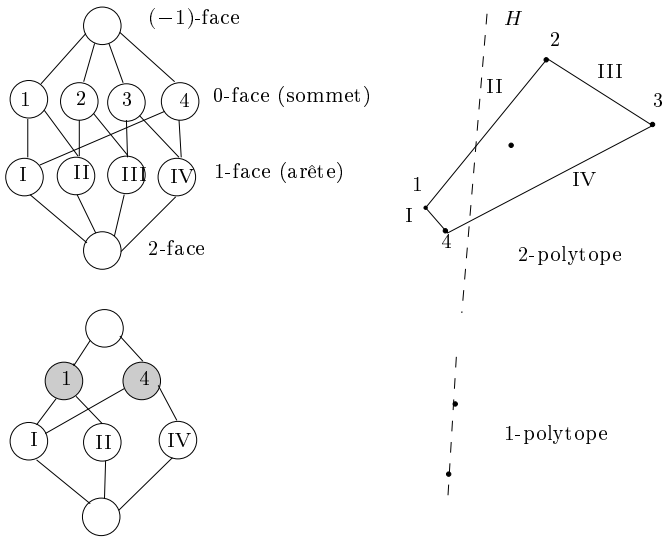


FIG. C.1 – *Isomorphisme du graphe d'incidence des faces du polytope contenant le sommet s avec le* $d-1$*-polytope intersection (lemme 17).*

Lemme 17 *Soit* s *un sommet d'un* d*-polytope* $\mathcal{P}$*, et* H *un hyperplan qui sépare* s *des autres sommets de* $\mathcal{P}$ *et désignons par* $\mathcal{P}'$ *le* $(d-1)$*-polytope* $\mathcal{P}' = \mathcal{P} \cap H$*, alors le graphe d'incidence de* $\mathcal{P}$ *qui contient* s *(sous-graphe d'incidence des faces contenant* s*) est isomorphe à la structure faciale de* $\mathcal{P}'$*.*

La figure C.1 illustre cet isomorphisme. Nous donnons maintenant la notion d'effeuillage d'un polytope.

C.2 Trouver les sommets de l'enveloppe convexe

Si p est un sommet de l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$ alors il existe un hyperplan H support de $\mathcal{P}$ qui contient p. On veut trouver celui qui minimise $x(t_p)$. L'hyperplan H_p est l'ensemble des points $z \in \mathbb{R}^d$ tel que $< p-z, n >= 0$ pour un vecteur $\vec{\mathbf{n}}$ que l'on doit déterminer. H_p supporte $\mathcal{P}$ si tous les points de $\mathcal{P}$ se trouvent dans le demi-espace inférieur (on a pris l'origine à l'intérieur de l'enveloppe convexe):

$$< p-q, n >\leq 0 \ \forall \ q \in \mathcal{P}. \tag{C.1}$$

L'équation C.1 fournit $(n-1)$ contraintes linéaires pour le vecteur $\vec{\mathbf{n}}$. Ce système est soluble si et seulement si un hyperplan support à $\mathcal{CH}(\mathcal{P})$ existe

et contient p. L'intersection avec la droite d'effeuillage Δ doit satisfaire:

$$< -\frac{1}{t_p}a - p, n >= 0. \tag{C.2}$$

On déduit de C.2 la valeur de t_p:

$$t_p = -\frac{< a, n >}{< p, n >}.$$

On doit déterminer la direction de $\vec{\mathbf{n}}$ afin de minimiser t_p. Afin de pourvoir résoudre ce problème par la programmation linéaire, on a besoin de codifier cette minimisation par une fonction linéaire. Pour réaliser ceci, on partitionne la direction de $\vec{\mathbf{n}}$ selon deux directions orthogonales. $n = y - t_p a$, ou $\vec{\mathbf{y}}$ est un vecteur à déterminer qui doit satisfaire la contrainte d'orthogonalité $< y, a >= 0$. En substituant dans les équations C.1 et C.2, on obtient le programme linéaire suivant:

Programme linéaire:

minimise t_p tel que
$< q - p, y > - < q - p, a > t_p \leq 0 \ \forall \ q \in \mathcal{P}, q \neq p$
$< p, y > - < p, a > t_p - < a, a >= 0$
$< a, y >= 0$

Ce programme linéaire est équivalent à un autre à $(n-1)$ contraintes et $(d-1)$ variables. Soit $H_p = \{z| < z - p, y - t_p a >= 0\}$ l'hyperplan solution du programme linéaire. Si on choisit la solution basique réalisable alors on a $(d-1)$ inéquations de contraintes qui sont nulles. On en déduit les $(d-1)$ sommets qui engendrent la face F_p du sommet p de l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$.

C.3 Construction du *graphe d'incidence* de $\mathcal{CH}(\mathcal{P})$

La méthode d'effeuillage peut être également utilisée pour construire le graphe d'incidence de l'enveloppe convexe de $\mathcal{P}$. Nous donnons ci-après les grandes lignes de l'algorithme. On ne fait plus l'hypothèse de position générale sur les points. Les faces propres (toutes les faces sauf la (-1)-face et la d-face) de $\mathcal{CH}(\mathcal{P})$ ne sont donc pas forcément des k-simplexes mais des k-polytopes arbitraires. L'esquisse de l'algorithme serait (cf. [Swa85]): *"détermine les points de $\mathcal{P}$ qui se trouvent sur une même facette; construit la facette en appliquant l'algorithme récursivement (on descend d'une dimension); combine les graphes d'incidences obtenus"*. Le principal problème avec

ce type d'algorithme est de guarantir que les faces propres ne sont trouvées qu'une fois et une seule alors qu'elles peuvent être partagées par beaucoup de faces (cf. les relations d' Euler et de Dehn-Sommerville, [BY95a]). On a donc besoin d'une structure de données dynamique afin de stocker les faces et de regarder rapidement si une face a déjà été calculée ou non. On détermine toujours de la même façon la prochaine face à calculer. Le lecteur trouvera tous les détails de cet algorithme dans la thèse de Seidel [Sei86b]. On ne donne ici que la complexité finale:

Théorème 12 *Le graphe d'incidence de l'enveloppe convexe $\mathcal{CH}(\mathcal{P})$ d'un ensemble de n points $\mathcal{P}$ de $\mathbb{E}^d$ peut être calculé par un algorithme adaptatif en temps $O(n^{2-\frac{2}{\lfloor \frac{d}{2} \rfloor + 1}+\epsilon} + L \log n)$ $\forall\ \epsilon > 0$ où L est le nombre total de faces ($L = \sum_{i=0}^{d-1} \phi_i$).*

Index

Index par mots clefs

Bibliographie

[ACG+85] Aggarwal (A.), Chazelle (B.), Guibas (L. J.), Ó'Dúnlaing (C.) et Yap (C. K.). – Parallel computational geometry. *In : Proc. 26th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 468–477.

[ACG+88] Aggarwal (A.), Chazelle (B.), Guibas (L.), Ó'Dúnlaing (C.) et Yap (C.). – Parallel computational geometry. *Algorithmica*, vol. 3, 1988, pp. 293–327.

[AF91a] Avis (D.) et Fukuda (K.). – A basis enumeration algorithm for linear systems with geometric applications. *Appl. Math. Lett.*, vol. 4, n5, 1991, pp. 39–42.

[AF91b] Avis (D.) et Fukuda (K.). – A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *In : Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pp. 98–104.

[AF92a] Avis (D.) et Fukuda (K.). – A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, vol. 8, 1992, pp. 295–313.

[AF92b] Avis (D.) et Fukuda (K.). – *Reverse Search for Enumeration.* – Rapport technique nNo. 92-5, Tsukuba University, Graduate School of Systems Management, Tokyo, Japan, avril 1992.

[AG86] Atallah (M. J.) et Goodrich (M. T.). – Efficient parallel solutions to some geometric problems. *J. Parallel Distrib. Comput.*, vol. 3, 1986, pp. 492–507.

[Aga91] Agarwal (Pankaj K.). – *Intersection and decomposition algorithms for planar arrangements.* – New York, NY, Cambridge University Press, 1991.

[AGR95] Amato (N.), Goodrich (M.T.) et Ramos (E.A.). – Optimal-work parallel algorithms for higher-dimensional convex hulls. *In : SIAM*, p. Proc. 6th ACM Symp. on Disc. Alg.

[AH91] Avis (D.) et Houle (M. E.). – Computational aspects of Helly's theorem and its relatives. *In : Proc. 3rd Canad. Conf. Comput. Geom.*, pp. 11–14.

[AHK+91] Arkin (E. M.), Halperin (D.), Kedem (K.), Mitchell (J. S. B.) et Naor (N.). – Arrangements of segments that share endpoints: Single face results. *In : Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pp. 324–333.

[AHK+94] Arkin (E. M.), Halperin (D.), Kedem (K.), Mitchell (J. S. B.) et Naor (N.). – *Arrangements of segments that share endpoints: Single face results.* – Rapport technique, SUNY Stony Brook, 1994.

[AHK+95] Arkin (E. M.), Halperin (D.), Kedem (K.), Mitchell (J. S. B.) et Naor (N.). – Arrangements of segments that share endpoints: Single face results. *Discrete Comput. Geom.*, vol. 13, 1995, pp. 257–270.

[AHU74] Aho (A. V.), Hopcroft (J. E.) et Ullman (J. D.). – *The Design and Analysis of Computer Algorithms.* – Reading, MA, Addison-Wesley, 1974.

[AHU83] Aho (A. V.), Hopcroft (J. E.) et Ullman (J. D.). – *Data Structures and Algorithms.* – Reading, MA, Addison-Wesley, 1983.

[Akl84] Akl (S. G.). – Optimal algorithms for computing convex hulls and for sorting. *Computing*, vol. 33, 1984, pp. 1–11.

[Akl85a] Akl (S. G.). – Optimal parallel algorithms for selection, sorting and computing convex hulls. *In : Computational Geometry*, éd. par Toussaint (G. T.), pp. 1–22. – Amsterdam, Netherlands, North-Holland, 1985.

[Akl85b] Akl (S. G.). – *Parallel Sorting Algorithms.* – Academic Press, 1985, xiii+229pp.

[AL93] Akl (S. G.) et Lyons (K. A.). – *Parallel Computational Geometry.* – Prentice-Hall, 1993.

[AM92] Agarwal (Pankaj K.) et Matoušek (J.). – Ray shooting and parametric search. *In : Proc. 24th Annu. ACM Sympos. Theory Comput.*, pp. 517–526.

[AM93] Agarwal (Pankaj K.) et Matoušek (J.). – Ray shooting and parametric search. *SIAM J. Comput.*, vol. 22, n 4, 1993, pp. 794–806.

[Ame93] Amenta (N.). – Helly theorems and generalized linear programming. *In : Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pp. 63–72.

[Ame94] Amenta (N.). – Helly-type theorems and generalized linear programming. *Discrete Comput. Geom.*, vol. 12, 1994, pp. 241–261.

[AR96] Amato (Nancy M.) et Ramos (Edgar A.). – On computing Voronoi diagrams by divide-prune-and-conquer. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 166–175.

[AS90] Agarwal (Pankaj K.) et Sharir (M.). – *Circle shooting in a simple polygon.* – Technical Report n 90-56, DIMACS, Rutgers Univ., 1990.

[AS93a] Agarwal (Pankaj K.) et Sharir (M.). – Circle shooting in a simple polygon. *J. Algorithms*, vol. 14, 1993, pp. 69–87.

[AS93b] Alon (N.) et Spencer (J.). – *The probabilistic method.* – J. Wiley & Sons, 1993.

[ASS89] Agarwal (Pankaj K.), Sharir (M.) et Shor (P.). – Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *J. Combin. Theory Ser. A*, vol. 52, n 2, 1989, pp. 228–274.

[AT78] Akl (S. G.) et Toussaint (G. T.). – A fast convex hull algorithm. *Inform. Process. Lett.*, vol. 7, n 5, 1978, pp. 219–222.

[Ata92] Atallah (M. J.). – Parallel techniques for computational geometry. *Proc. IEEE*, vol. 80, n 9, septembre 1992, pp. 1435–1448.

[ATB82] Avis (D.), Toussaint (G. T.) et Bhattacharya (B. K.). – On the multimodality of distances in convex polygons. *Comput. Math. Appl.*, vol. 8, 1982, pp. 153–156.

[AW91] Affentranger (F.) et Wieacker (J. A.). – On the convex hull of uniform random points in a simple d-polytope. *Discrete Comput. Geom.*, vol. 6, 1991, pp. 291–305.

[Bal95] Balaban (Ivan J.). – An optimal algorithm for finding segment intersections. *In : Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pp. 211–219.

[BCD+92] Boissonnat (J. D.), Cérézo (A.), Devillers (O.), Duquesne (J.) et Yvinec (M.). – An algorithm for constructing the convex hull of a set of spheres in dimension d. *In : Proc. 4th Canad. Conf. Comput. Geom.*, pp. 269–273.

[BCDT91a] Boissonnat (J.-D.), Cérézo (A.), Devillers (O.) et Teillaud (M.). – *Output-sensitive construction of the 3-d Delaunay triangulation of constrained sets of points.* – Research Report n 1415, Valbonne, France, INRIA Sophia-Antipolis, avril 1991.

[BCDT91b] Boissonnat (J.-D.), Cérézo (A.), Devillers (O.) et Teillaud (M.). – Output-sensitive construction of the 3-d Delaunay triangulation of constrained sets of points. *In : Proc. 3rd Canad. Conf. Comput. Geom.*, pp. 110–113.

[BCDT96] Boissonnat (J.-D.), Cérézo (A.), Devillers (O.) et Teillaud (M.). – Output-sensitive construction of the Delaunay triangulation of points lying in two planes. *Internat. J. Comput. Geom. Appl.*, vol. 6, n 1, 1996, pp. 1–14.

[BDH93] Barber (C. B.), Dobkin (D. P.) et Huhdanpaa (H.). – *The Quickhull algorithm for convex hull.* – Technical Report n GCG53, Geometry Center, Univ. of Minnesota, juillet 1993.

[BDS+90] Boissonnat (J.-D.), Devillers (O.), Schott (R.), Teillaud (M.) et Yvinec (M.). – *Applications of random sampling to on-line algorithms in computational geometry.* – Technical Report n1285, Valbonne, France, INRIA Sophia-Antipolis, 1990.

[BDS+92] Boissonnat (J.-D.), Devillers (O.), Schott (R.), Teillaud (M.) et Yvinec (M.). – Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, vol. 8, 1992, pp. 51–71.

[BEG84] Bhattacharya (B.) et El-Gindy (H.). – A New Linear Convex Hull Algorithm for Simple Polygons. *IEEE Transactions on Information Theory*, vol. 30, 84.

[Ber74] Berger (M.). – *Géométrie 3. convexes et polytopes, polyèdres réguliers, aires et volumes.* – Paris, Fernand Nathan, 1974.

[BFP+72] Blum (M.), Floyd (R.W.), Pratt (V.R.), Rivest (R.L.) et Tarjan. – Time bounds for selection. *J. Comput. Syst. Sci.*, vol. 7, 1972.

[BG94] Brönnimann (H.) et Goodrich (M. T.). – Almost optimal set covers in finite VC-dimension. *In : Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pp. 293–302.

[BGJR91] Borgwardt (K. H.), Gaffke (N.), Junger (M.) et Reinelt (G.). – Computing the convex hull in the Euclidean plane in linear expected time. *In : Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, éd. par Gritzman (P.) et Sturmfels (B.), pp. 91–107. – AMS Press, 1991.

[BGLR93] Bellare (M.), Goldwasser (S), Lund (C.) et Russel (A.). – Efficient probabistically checkable proofs and applications to approximation. *In : Proc. 25th Annu. Symp. Theory Comput.*, pp. 294 – 304.

[BK87] Bajaj (C.) et Kim (M.-S.). – Compliant motion planning with geometric models. *In : Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pp. 171–180.

[BK91] Bajaj (C.) et Kim (M. S.). – Convex hull of objects bounded by algebraic curves. *Algorithmica*, vol. 6, 1991, pp. 533–553.

[BKM+95] Burnikel (Christoph), Könnemann (Jochen), Mehlhorn (Kurt), Näher (Stefan), Schirra (Stefan) et Uhrig (Christian). – Exact geometric computation in LEDA. *In : Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pp. C18–C19.

[BL89] Boissonnat (J.-D.) et Laumond (J.-P.) (édité par). – *Geometry and Robotics*. – Springer-Verlag, 1989, *Lecture Notes in Computer Science*, volume 391.

[BM71] Bruggesser (H.) et Mani (P.). – Shellable decompositions of cells and spheres. *Math. Scand.*, vol. 29, 1971, pp. 197–205.

[BMS94] Burnikel (C.), Mehlhorn (K.) et Schirra (S.). – On degeneracy in geometric computations. *In : Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pp. 16–23.

[BO83] Ben-Or (M.). – Lower bounds for algebraic computation trees. *In : Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86.

[BOKR86] Ben-Or (M.), Kozen (D.) et Reif (J. H.). – The complexity of elementary algebra and geometry. *J. Comput. Syst. Sci.*, vol. 32, 1986, pp. 251–264.

[Boo89] Boots (B. N.). – *Voronoi (Thiessen) Polygons*. – Norwich, England, Geo Books, 1989.

[Bro80] Brown (K. Q.). – *Geometric transforms for fast geometric algorithms*. – Pittsburgh, PA, Ph.D. thesis, Dept. Comput. Sci., Carnegie-Mellon Univ., 1980. Report CMU-CS-80-101.

[Brö83] Brönsted (A.). – *An Introduction to Convex Polytopes*. – New York, NY, Springer-Verlag, 1983.

[Brö95] Brönnimann (H.). – *Derandomization of geometric algorithms*. – Ph.D. thesis, Princeton Univ., 1995.

[BY95a] Boissonnat (J.-D.) et Yvinec (M.). – *Géométrie algorithmique.* – Paris, Ediscience international, 1995.

[BY95b] Boissonnat (J-D.) et Yvinec (M.). – *Géométrie algorithmique.* – Paris, Ediscience international, 1995.

[CE88] Chazelle (B.) et Edelsbrunner (H.). – An optimal algorithm for intersecting line segments in the plane. *In : Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 590–600.

[CE92] Chazelle (B.) et Edelsbrunner (H.). – An optimal algorithm for intersecting line segments in the plane. *J. ACM*, vol. 39, 1992, pp. 1–54.

[CEG+91] Chazelle (B.), Edelsbrunner (H.), Grigni (M.), Guibas (L.), Hershberger (J.), Sharir (M.) et Snoeyink (J.). – Ray shooting in polygons using geodesic triangulations. *In : Proc. 18th Internat. Colloq. Automata Lang. Program.* pp. 661–673. – Springer-Verlag.

[CF88] Chazelle (B.) et Friedman (J.). – A deterministic view of random sampling and its use in geometry. *In : Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 539–549.

[CF90] Chazelle (B.) et Friedman (J.). – A deterministic view of random sampling and its use in geometry. *Combinatorica*, vol. 10, n 3, 1990, pp. 229–249.

[Cha85] Chazelle (B.). – On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, vol. IT-31, 1985, pp. 509–517.

[Cha91] Chazelle (B.). – *An optimal convex hull algorithm for point sets in any fixed dimension.* – Technical Report n CS-TR-336-91, Princeton, NJ, Dept. Comput. Sci., Princeton Univ., 1991.

[Cha95a] Chan (M. Y.). – Output-Sensitive Results on Convex Hulls, Extreme Points, and Related Problems. *Proc. of the A.C.M. Symp. on Computational Geometry*, 1995.

[Cha95b] Chan (Timothy M. Y.). – Output-sensitive results on convex hulls, extreme points, and related problems. *In : Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pp. 10–19.

[Cha96] Chan (Timothy M.). – Fixed-dimensional linear programming queries made easy. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 284–290.

[Cha96] Chazelle (B.). – *CG Impact Task Force Report.* – Report, University of Princeton, 96.

[Cho80] Chow (A. L.). – *Parallel algorithms for geometric problems.* – Urbana, IL, Ph.D. thesis, Dept. Comput. Sci., Univ. Illinois, 1980.

[Chv79] Chvátal (V.). – A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, vol. 4, 1979, pp. 233–235.

[CK70] Chand (D. R.) et Kapur (S. S.). – An algorithm for convex polytopes. *J. ACM*, vol. 17, 1970, pp. 78–86.

[Cla86] Clarkson (K. L.). – Further applications of random sampling to computational geometry. *In : Proc. 18th Annu. ACM Sympos. Theory Comput.*, pp. 414–423.

[Cla87] Clarkson (K. L.). – New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, vol. 2, 1987, pp. 195–222.

[Cla88a] Clarkson (K. L.). – Applications of random sampling in computational geometry, II. *In : Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 1–11.

[Cla88b] Clarkson (K. L.). – A Las Vegas algorithm for linear programming when the dimension is small. *In : Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 452–456.

[CLR90] Cormen (T. H.), Leiserson (C. E.) et Rivest (R. L.). – *Introduction to Algorithms.* – Cambridge, Mass., The MIT Press, 1990.

[CM92] Chazelle (B.) et Matoušek (J.). – *Derandomizing an Output-Sensitive Convex Hull Algorithm in Three Dimensions.* – Rapport technique, Dept. Comput. Sci., Princeton Univ., 1992.

[CM93] Chazelle (B.) et Matoušek (J.). – On linear-time deterministic algorithms for optimization problems in fixed dimension. *In : Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pp. 281–290.

[CM94] Chazelle (Bernard) et Matoušek (Jiří). – Derandomizing an output-sensitive convex hull algorithm. *Proceedings of CG-TA'94*, 94. – to appear.

[Cof88] Coffman (E.G.). – *Probabilistic Analysis of Packing and Partitionning Algorithms.* – S. Lueker, 1988.

[CS88] Clarkson (K. L.) et Shor (P. W.). – Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. *In : Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 12–17.

[CS89] Clarkson (K. L.) et Shor (P. W.). – Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, vol. 4, 1989, pp. 387–421.

[CSY87] Cole (R.), Sharir (M.) et Yap (C. K.). – On k-hulls and related problems. *SIAM J. Comput.*, vol. 16, 1987, pp. 61–77.

[CSY95] Chan (Timothy M.), Snoeyink (Jack) et Yap (Chee-Keng). – Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three. *In : Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pp. 282–291.

[Dat94] Datta (A.). – Efficient parallel algorithms for geometric k-clustering problems. *In : Proc. 11th Sympos. Theoret. Aspects Comput. Sci.* – Springer-Verlag.

[dB92] de Berg (M.). – *Efficient algorithms for ray shooting and hidden surface removal.* – Utrecht, Netherlands, Ph.D. dissertation, Dept. Comput. Sci., Utrecht Univ., 1992.

[Dev91] Devroye (L.). – On the oscillation of the expected number of points on a random convex hull. *Statistics and Probability Letters*, vol. 11, 1991, pp. 281–286.

[DF93] Devillers (O.) et Fabri (A.). – Scalable algorithms for bichromatic line segment intersection problems on coarse grained multicomputers. *In : Proc. 3rd Workshop Algorithms Data Struct.*, pp. 277–288.

[DFM84] Dyer (M. E.), Frieze (A. M.) et McDiarmid (C. J. H.). – Partitioning heuristics for two geometric maximization problems. *Oper. Res. Lett.*, vol. 3, 1984, pp. 267–270.

[DFRC93] Dehne (F.), Fabri (A.) et Rau-Chaplin (A.). – Scalable parallel geometric algorithms for coarse grained multicomputers. *In : Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pp. 298–307.

[DFSV92] Dehne (F.), Flach (B.), Sack (J.-R.) et Valiveti (N.). – Analog parallel computational geometry. *In : Proc. 4th Canad. Conf. Comput. Geom.*, pp. 143–153.

[DG82] Danzer (L.) et Grünbaum (B.). – Intersection Properties of Boxes in R^d. *Combinatorica*, vol. 2(3), 1982, pp. 237 – 246.

[DGK63] Danzer (Ludwig), Grünbaum (Branko) et Klee (Victor). – *Helly's Theorem and its Relatives.* – Providence, RI, Proceedings of the symposium on pure mathematics, 1963.

[DN96] Duquesne (J.) et Nielsen (F.). – *Docking two sets of spheres.* – Research Report, BP93, 06902 Sophia-Antipolis, France, INRIA, to appear, 96.

[DS88] Dehne (F.) et Sack (J.-R.). – A survey of parallel computational geometry algorithms. *Lecture Notes in Computer Science*, vol. 342, 1988, pp. 73–88. – Proc. International Workshop on Parallel Processing by Cellular Automata and Arrays (PARCELLA).

[DS90] Dobkin (D. P.) et Souvaine (D. L.). – Computational geometry in a curved world. *Algorithmica*, vol. 5, 1990, pp. 421–457.

[DS91] Dobkin (D. P.) et Souvaine (D. L.). – Detecting the intersection of convex objects in the plane. *Comput. Aided Geom. Design*, vol. 8, 1991, pp. 181–200.

[Dwy89] Dwyer (R. A.). – Convex hulls of points from spherically symmetric distributions. *In : Abstracts 1st Canad. Conf. Comput. Geom.*, p. 2.

[Dwy90] Dwyer (R. A.). – Las Vegas gift-wrapping is twice as fast. *In : Proc. 2nd Canad. Conf. Comput. Geom.*, pp. 261–264.

[Dye82] Dyer (M. E.). – *Two-variable linear programs are solvable in linear time.* – Report n??, Middlesbrough, England, Dept. Math. Statist., Teesside Polytech., 1982.

[Dye84] Dyer (M. E.). – Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.*, vol. 13, 1984, pp. 31–45.

[EC91] Emiris (I.) et Canny (J.). – A general approach to removing degeneracies. *In : Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 405–413.

[EC92] Emiris (I.) et Canny (J.). – An efficient approach to removing geometric degeneracies. *In : Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pp. 74–82.

[Ede87] Edelsbrunner (H.). – *Algorithms in Combinatorial Geometry.* – Heidelberg, West Germany, Springer-Verlag, 1987, *EATCS Monographs on Theoretical Computer Science*, volume 10.

[Ede88] Edelsbrunner (H.). – Geometric structures in computational geometry. *In : Proc. 15th Internat. Colloq. Automata Lang. Program.* pp. 201–213. – Springer-Verlag.

[Efr65] Efron (B.). – The convex hull of a random set of points. *Biometrika*, vol. 52, n3, 1965, pp. 331–343.

[EGS89] Edelsbrunner (H.), Guibas (L.) et Sharir (M.). – The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, vol. 4, 1989, pp. 311–336.

[EJ88] E. (Blelloch G.) et J. (Little J.). – Parallel solutions to some geometric problems on the scan model of computation. *Proceedings of the 1988 International Conference on Parallel Processing*, vol. 3, 88, pp. 218–222.

[EM88] Edelsbrunner (H.) et Mücke (E. P.). – Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *In : Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 118–133.

[EM90] Edelsbrunner (H.) et Mücke (E. P.). – Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, vol. 9, 1990, pp. 66–104.

[ERvK93] Everett (H.), Robert (J.-M.) et van Kreveld (M.). – An optimal algorithm for the $(\leq k)$-levels, with applications to separation and transversal problems. *In : Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pp. 38–46.

[ES74] Erdos (P.) et Spencer (J.). – *Probabilistic Methods in Combinatorics.* – Academic Press, 1974.

[ES91] Edelsbrunner (H.) et Shi (W.). – An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, vol. 20, 1991, pp. 259–277.

[ES92] Edelsbrunner (H.) et Shah (N. R.). – Incremental topological flipping works for regular triangulations. *In : Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pp. 43–52.

[Euc56] Euclid. – *Elements.* – Dover, 1956. Translated by Sir Thomas L. Heath.

[EW86] Edelsbrunner (H.) et Welzl (E.). – Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, vol. 15, 1986, pp. 271–284.

[Fei96] Feige (U.). – A Threshold of $\log n$ for Approximating Set Cover. *In : Proc. of the 28-th ACM Symposium on Theory of Computing.*

[FGS90] Froidevaux (C.), Gaudel (M. C.) et Soria (M.). – *Type de données et algorithmes.* – Paris, McGraw-Hill, 1990.

[FPT81] Fowler (R. J.), Paterson (M. S.) et Tanimoto (S. L.). – Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, vol. 12, n3, 1981, pp. 133–137.

[FR90] Frederickson (G. N.) et Rodger (S.). – A new approach to the dynamic maintenance of maximal points in a plane. *Discrete Comput. Geom.*, vol. 5, 1990, pp. 365–374.

[GG93] Ghouse (Mujtaba) et Goodrich (Michael T.). – Experimental evidence for the power of random sampling in practical parallel algorithms. pp. 549–556. – IEEE Computer Society.

[GJ79] Garey (M. R.) et Johnson (D. S.). – *Computers and Intractability: A Guide to the Theory of NP-Completeness.* – New York, NY, W. H. Freeman, 1979.

[GKP94] Graham (Ronald L.), Knuth (Donald E.) et Patashnik (Oren). – *Concrete Mathematics.* – New York, Addison-Wesley Publishing Company, 1994.

[Gol90] Golin (M. J.). – *Probabilistic analysis of geometric algorithms.* – Princeton, NY, Ph.D. thesis, Dept. Comput. Sci., Princeton Univ., 1990.

[Gon75] Gonzalez (T.). – *Algorithms on sets and related problems.* – Technical Report n ??, Norman, OK, Dept. Comput. Sci., Univ. Oklahoma, 1975.

[Goo87] Goodrich (M. T.). – *Efficient parallel techniques for computational geometry.* – West Lafayette, IN, Ph.D. thesis, Dept. Comput. Sci., Purdue Univ., 1987.

[Goo92] Goossens (P.). – Shelling pseudopolyhedra. *Discrete Comput. Geom.*, vol. 7, 1992, pp. 207–215.

[GRS80] Graham (R.), Rothschild (B.) et Spencer (J.). – *Ramsey Theory.* – New York, Wiley-Interscience, 1980.

[Grü67] Grünbaum (B.). – *Convex Polytopes.* – New York, NY, Wiley, 1967.

[GS88] Golin (M.) et Sedgewick (R.). – Analysis of a simple yet efficient convex hull algorithm. *In : Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 153–163.

[GS96] Gupta (N.) et Sen (S.). – Faster output-sensitive parallel convex hulls for $d \leq 3$: optimal sublogarithmic algorithms for small outputs. *In : Proc. 12th ACM Symp. on Computational Geometry.*

[Güt84] Güting (R. H.). – An optimal contour algorithm for iso-oriented rectangles. *J. Algorithms*, vol. 5, 1984, pp. 303–326.

[GW93a] Gruber (Peter M.) et Wills (J. M.) (édité par). – *Handbook of Convex Geometry*. – Amsterdam, Netherlands, North-Holland, 1993volume A.

[GW93b] Gruber (Peter M.) et Wills (J. M.) (édité par). – *Handbook of Convex Geometry*. – Amsterdam, Netherlands, North-Holland, 1993volume B.

[GY83] Graham (R. L.) et Yao (F. F.). – Finding the convex hull of a simple polygon. *J. Algorithms*, vol. 4, 1983, pp. 324–331.

[HD60a] Hadwiger (H.) et Debrunner (H.). – *Kombinatorische Geometrie in der Ebene*. – Geneva, Switzerland, Enseignement Mathematique, 1960, *Monograph. Enseign. Math.*, volume 2.

[HD60b] Hadwiger (H.) et Debrunner (H.). – *Kombinatorische Geometrie in der Ebene*. – Geneva, Monographies de l'Enseignement Mathématique, N2, 1960.

[HDK64] Hadwiger (H.), Debrunner (H.) et Klee (V.). – *Combinatorial Geometry in the Plane*. – New York, Holt, Rinehart and Winston, 1964.

[Her89] Hershberger (J.). – Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, vol. 33, 1989, pp. 169–174.

[HM84] Hochbaum (D. S.) et Maass (W.). – Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, vol. 31, 1984, pp. 130–136.

[HO94] Halperin (D.) et Overmars (M. H.). – Spheres, molecules, and hidden surface removal. *In : Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pp. 113–122.

[Hoc82] Hochbaum (D. S.). – Approximation Algorithms of the Set Covering and Vertex Cover Problems. *SIAM J. COMPUT.*, vol. 11, n3, August 1982, pp. 555 – 556.

[HR93] Hagauer (J.) et Rote (G.). – Three-clustering of points in the plane. *In : Proc. 1st European Symp. on Algorithms. LNCS 726, Springer-Verlag.*

[HS91] Hershberger (J.) et Suri (S.). – Finding tailored partitions. *J. Algorithms*, vol. 12, 1991, pp. 431–463.

[HS93] Hershberger (J.) et Suri (S.). – A pedestrian approach to ray shooting: Shoot a ray, take a walk. *In : Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pp. 54–63.

[IA83] Imai (H.) et Asano (Ta.). – Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, vol. 4, 1983, pp. 310–323.

[Jan91] Janardan (R.). – On the dynamic maintenance of maximal points in the plane. *Inform. Process. Lett.*, vol. 40, 1991, pp. 59–64.

[Jar73] Jarvis (R. A.). – On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, vol. 2, 1973, pp. 18–21.

[Jar77] Jarvis (R. A.). – Computing the shape hull of points in the plane. *In : Proc. IEEE Internat. Conf. Pattern Recogn. Image Process.*, pp. 231–241.

[Kap94] Kapoor (S.). – Dynamic maintenance of maximas of 2-d point sets. *In : Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pp. 140–149.

[Kar72] Karp (R.). – *Reducibility among combinatorial problems, Complexity of Computer Computations.* – E. Miller and J.W. Thatcher, New York 1972.

[(Ke85] (Kenyon) (Claire Mathieu). – *Géométrie Stochastique et Informatique.* – Rapports de Recherche lri, Bat. 490 91405 ORSAY, France, Université de Paris-Sud, Centre d'Orsay, 1985.

[Kir83] Kirkpatrick (D. G.). – Optimal search in planar subdivisions. *SIAM J. Comput.*, vol. 12, 1983, pp. 28–35.

[Kla95] Klazar (M.). – *Combinatorial Aspects of Davenport–Schinzel Sequences.* – Prague, Czech Republic, Ph.D. thesis, Department of Applied Mathematics, Charles University, 1995.

[KLP75] Kung (H. T.), Luccio (F.) et Preparata (F. P.). – On finding the maxima of a set of vectors. *J. ACM*, vol. 22, 1975, pp. 469–476.

[KLPS86] Kedem (K.), Livne (R.), Pach (J.) et Sharir (M.). – On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, vol. 1, 1986, pp. 59–71.

[KN96a] Katz (M.) et Nielsen (F.). – On Piercing Convex Sets. *In : Proc. of the 12-th international conference on Computational Geometry.*

[KN96b] Katz (M. J.) et Nielsen (F.). – *On Piercing Convex Sets.* – Research Report 2874 n 2874, BP93, 06902 Sophia-Antipolis, France, INRIA, 1996.

[KNar] Katchalski (M.) et Nashtir (D.). – On a conjecture of Danzer and Grünbaum. *In : Proc. AMS.*

[Knu73a] Knuth (D. E.). – *Fundamental Algorithms.* – Reading, MA, Addison-Wesley, 1973, 2nd édition, *The Art of Computer Programming*, volume 1.

[Knu73b] Knuth (D. E.). – *Sorting and Searching.* – Reading, MA, Addison-Wesley, 1973, *The Art of Computer Programming*, volume 3.

[Knu81] Knuth (D. E.). – *Seminumerical Algorithms.* – Reading, MA, Addison-Wesley, 1981, 2nd édition, *The Art of Computer Programming*, volume 2.

[Knu93] Knuth (D. E.). – *The Stanford Graphbase.* – Reading, MA, Addison-Wesley, 1993, *A Platform for Combinatorial Computing*, volume 1.

[KR85] Kirkpatrick (D. G.) et Radke (J. D.). – A framework for computational morphology. *In : Computational Geometry*, éd. par Toussaint (G. T.), pp. 217–248. – Amsterdam, Netherlands, North-Holland, 1985.

[KS82] Kirkpatrick (D. G.) et Seidel (R.). – The ultimate planar convex hull algorithm? *In : Proc. 20th Allerton Conf. Commun. Control Comput.*, pp. 35–42. – Monticello, Illinois, 1982.

[KS85] Kirkpatrick (D. G.) et Seidel (R.). – Output-size sensitive algorithms for finding maximal vectors. *In : Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pp. 89–96.

[KS86] Kirkpatrick (D. G.) et Seidel (R.). – The ultimate planar convex hull algorithm? *SIAM J. Comput.*, vol. 15, 1986, pp. 287–299.

[LMR88] Linial (N.), Mansour (Y.) et Rivest (R. L.). – Results on learnability and the Vapnik-Chervonekis dimension. *In : Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 120–129.

[LW86] Lee (D. T.) et Wu (Y. F.). – Geometric complexity of some location problems. *Algorithmica*, vol. 1, 1986, pp. 193–211.

[MA79] McCallum (D.) et Avis (D.). – A linear algorithm for finding the convex hull of a simple polygon. *Inform. Process. Lett.*, vol. 9, 1979, pp. 201–206.

[Mat91a] Matoušek (J.). – Approximations and optimal geometric divide-and-conquer. *In : Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pp. 505–511. – Also to appear in *J. Comput. Syst. Sci.*

[Mat91b] Matoušek (J.). – Efficient partition trees. *In : Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pp. 1–9.

[Mat93] Matoušek (J.). – Linear optimization queries. *J. Algorithms*, vol. 14, 1993, pp. 432–448.

[Mau84] Maus (A.). – Delaunay triangulation and the convex hull of n points in expected linear time. *BIT*, vol. 24, 1984, pp. 151–163.

[McC80] McCreight (E. M.). – *Efficient algorithms for enumerating intersecting intervals and rectangles.* – Report n CSL-80-9, Palo Alto, CA, Xerox Palo Alto Res. Center, 1980.

[McI93] McIlroy (P.). – Optimistic Sorting and Information Theoretic Complexity. *In : Proc. 4th ACM-SIAM Symp. on Disc. Alg.*, pp. 467–474.

[McM70] McMullen (P.). – The maximal number of faces of a convex polytope. *Mathematica*, vol. 17, 1970, pp. 179–184.

[Meg82] Megiddo (N.). – Linear-time algorithms for linear programming in R^3 and related problems. *In : Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 329–338.

[Meg83a] Megiddo (N.). – Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, vol. 30, 1983, pp. 852–865.

[Meg83b] Megiddo (N.). – Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, vol. 12, 1983, pp. 759–776.

[Meg84] Megiddo (N.). – Linear programming in linear time when the dimension is fixed. *J. ACM*, vol. 31, 1984, pp. 114–127.

[Meg85] Megiddo (N.). – Partitioning with two lines in the plane. *J. Algorithms*, vol. 6, 1985, pp. 430–433.

[Meh84a] Mehlhorn (K.). – *Multi-dimensional Searching and Computational Geometry.* – Heidelberg, West Germany, Springer-Verlag, 1984, *Data Structures and Algorithms*, volume 3.

[Meh84b] Mehlhorn (K.). – *Sorting and Searching.* – Heidelberg, West Germany, Springer-Verlag, 1984, *Data Structures and Algorithms*, volume 1.

[MII96] Masada (Tomonari), Imai (Hiroshi) et Imai (Keiko). – Enumeration of regular triangulations. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 224–233.

[MNS+96] Mehlhorn (Kurt), Näher (Stefan), Schilz (Thomas), Schirra (Stefan), Seel (Michael), Seidel (Raimund) et Uhrig (Christian). – Checking geometric programs or verification of geometric structures. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 159–165.

[MS71] McMullen (P.) et Shephard (G. C.). – *Convex Polytopes and the Upper Bound Conjecture.* – Cambridge, England, Cambridge University Press, 1971.

[MS92] Matoušek (J.) et Schwarzkopf (O.). – Linear optimization queries. *In : Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pp. 16–25.

[MS93] Matoušek (J.) et Schwarzkopf (O.). – On ray shooting in convex polytopes. *Discrete Comput. Geom.*, vol. 10, n2, 1993, pp. 215–232.

[MSW92] Matoušek (J.), Sharir (M.) et Welzl (E.). – A subexponential bound for linear programming. *In : Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pp. 1–8.

[MT85] McQueen (M. M.) et Toussaint (G. T.). – On the ultimate convex hull algorithm in practice. *Pattern Recogn. Lett.*, vol. 3, 1985, pp. 29–34.

[Mul94] Mulmuley (K.). – *Computational Geometry: An Introduction Through Randomized Algorithms.* – Englewood Cliffs, NJ, Prentice Hall, 1994.

[Nie94] Nielsen (F.). – *Un Algorithme Optimal Sensible à la Sortie pour le Calcul de l'Intersection de Demi-Espaces et d'un Convexe en Dimension 2.* – manuscrit, BP93, 06902 Sophia-Antipolis, France, INRIA, 1994.

[Nie96a] Nielsen (F.). – *Output-sensitive Planar Convex Hull Algorithms on Coarse Grained Multicomputers.* – manuscrit, BP93, 06902 Sophia-Antipolis, France, INRIA, 1993-1996.

[Nie96b] Nielsen (F.). – *Fast Stabbing of Boxes in High Dimensions.* – Research Report, BP93, 06902 Sophia-Antipolis, France, INRIA, 1996.

[Nie96c] Nielsen (F.). – Fast stabbing of boxes in high dimensions. *In : Proc. 12th European Workshop on Comput. Geom.(CG96)*, p. 121.

[Nie96d] Nielsen (F.). – Fast stabbing of boxes in high dimensions. *In : Proc. 8th Canad. Conf. Comput. Geom.* pp. 87–92. – Carleton University Press, Ottawa, Canada.

[Nie96e] Nielsen (F.). – Output-sensitive Peeling of Convex and Maximal Layers. *Inform. Process. Lett.*, vol. ??, 1996, p. to appear.

[NY95] Nielsen (F.) et Yvinec (M.). – *An Output-Sensitive Convex Hull Algorithm for Planar Objects.* – Research Report 2575 (Presented at the 4th Israelian Workshop on Computational and Combinatorial Geometry) n2575, BP93, 06902 Sophia-Antipolis, France, INRIA, 1995.

[NY96] Nielsen (F.) et Yvinec (M.). – An Output-Sensitive Convex Hull Algorithm for Planar Objects. *Internat. J. Comput. Geom. Appl.*, to appear, 1996.

[OBS92] Okabe (Atsuyuki), Boots (Barry) et Sugihara (Kokichki). – *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* – Chichester, England, John Wiley & Sons, 1992.

[Ove83] Overmars (M. H.). – *The Design of Dynamic Data Structures.* – Heidelberg, West Germany, Springer-Verlag, 1983, *Lecture Notes in Computer Science*, volume 156.

[OvL80] Overmars (M. H.) et van Leeuwen (J.). – Dynamically maintaining configurations in the plane. *In : Proc. 12th Annu. ACM Sympos. Theory Comput.*, pp. 135–145.

[OvL81] Overmars (M. H.) et van Leeuwen (J.). – Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, vol. 23, 1981, pp. 166–204.

[PH77] Preparata (F. P.) et Hong (S. J.). – Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, vol. 20, 1977, pp. 87–93.

[PS82] Papadimitriou (C. H.) et Steiglitz (K.). – *Combinatorial Optimization: Algorithms and Complexity.* – Englewood Cliffs, NJ, Prentice Hall, 1982.

[PS85] Preparata (F. P.) et Shamos (M. I.). – *Computational Geometry: An Introduction.* – New York, NY, Springer-Verlag, 1985.

[Rap92] Rappaport (D.). – A convex hull algorithm for discs, and applications. *Comput. Geom. Theory Appl.*, vol. 1, n3, 1992, pp. 171–181.

[Ray70] Raynaud (H.). – Sur l'enveloppe convex des nuages de points aleatoires dans R^n. *J. Appl. Probab.*, vol. 7, 1970, pp. 35–48.

[RS87] Reif (John H.) et Sen (Sandeep). – Optimal randomized parallel algorithms for computational geometry. *In : Proceedings of the 1987 International Conference on Parallel Processing.* pp. 270–277. – University Park, PA, 1987.

[RSL74] Rosenkrantz (D. J.), Stearns (R. E.) et Lewis (P. M.). – Approximate algorithms for the traveling salesperson problem. *In : Proc. 15th Annu. IEEE Sympos. Switching Automata Theory*, pp. 33–42.

[SA95] Sharir (M.) et Agarwal (P. K.). – *Davenport-Schinzel Sequences and Their Geometric Applications.* – New York, Cambridge University Press, 1995.

[SCK+86] Sharir (M.), Cole (R.), Kedem (K.), Leven (D.), Pollack (R.) et Sifrony (S.). – Geometric applications of Davenport-Schinzel sequences. *In : Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 77–86.

[Sed83] Sedgewick (R.). – *Algorithms.* – Reading, MA, Addison-Wesley, 1983.

[Sed92] Sedgewick (R.). – *Algorithms in C++.* – Reading, MA, Addison Wesley, 1992.

[Sei86a] Seidel (R.). – Constructing higher-dimensional convex hulls at logarithmic cost per face. *In : Proc. 18th Annu. ACM Sympos. Theory Comput.*, pp. 404–413.

[Sei86b] Seidel (R.). – *Output-size sensitive algorithms for constructive problems in computational geometry.* – Ithaca, NY, Ph.D. thesis, Dept. Comput. Sci., Cornell Univ., 1986. Technical Report TR 86-784.

[Sha87] Sharir (M.). – Almost linear upper bounds for the length of general Davenport-Schinzel sequences. *Combinatorica*, vol. 7, 1987, pp. 131–143.

[Sha88] Sharir (M.). – Improved lower bounds on the length of Davenport-Schinzel sequences. *Combinatorica*, vol. 8, 1988, pp. 117–124.

[Sha96] Sharir (Micha). – A near-linear algorithm for the planar 2-center problem. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 106–112.

[Sla96] Slavík (P.). – A Tight Analysis of the Greedy Algorithm for Set Cover. *In : Proc. of the* 28*-th ACM Symposium on Theory of Computing.*

[SS96] Schwarzkopf (Otfried) et Sharir (Micha). – Vertical decomposition of a single cell in a three-dimensional arrangement of surfaces and its applications. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 20–29.

[SV87] Schäffer (A. A.) et Van Wyk (C. J.). – Convex hulls of piecewise-smooth Jordan curves. *J. Algorithms*, vol. 8, 1987, pp. 66–94.

[SW86] Shin (S. Y.) et Woo (T. C.). – Finding the convex hull of a simple polygon in linear time. *Pattern Recogn.*, vol. 19, 1986, pp. 453–458.

[SW92] Sharir (M.) et Welzl (E.). – A combinatorial bound for linear programming and related problems. *In : Proc. 9th Sympos. Theoret. Aspects Comput. Sci.* pp. 569–579. – Springer-Verlag.

[SW96] Sharir (Micha) et Welzl (Emo). – Rectilinear and polygonal p-piercing and p-center problems. *In : Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 122–132.

[Swa85] Swart (G. F.). – Finding the convex hull facet by facet. *J. Algorithms*, vol. 6, 1985, pp. 17–48.

[Tar87] Tarjan (R. E.). – *Data Structures and Network Algorithms.* – Philadelphia, PA, Society for Industrial and Applied Mathematics, 1987.

[TF80] Tanimoto (S.L.) et Fowler (R.J.). – Covering image subsets with patches. *In : Proc. of the* 5*-th international conference on pattern recognition*, pp. 835–839.

[Wag85] Wagener (H.). – *Parallel Computational Geometry Using Polygonal Order.* – FRG, Thèse de PhD, Technical University of Berlin, 1985.

[Weg67] Wegner (G.). – *Eigenschaften der Nerven homologish-einfacher Familien im* $\mathbb{R}^n$. – Thèse de PhD, Göttingen, 1967.

[Wel91] Welzl (E.). – Smallest enclosing disks (balls and ellipsoids). *In : New Results and New Trends in Computer Science*, éd. par Maurer (H.), pp. 359–370. – Springer-Verlag, 1991.

[WS88] Wiernik (A.) et Sharir (M.). – Planar realizations of nonlinear Davenport-Schinzel sequences by segments. *Discrete Comput. Geom.*, vol. 3, 1988, pp. 15–47.

[Yao74] Yao (F. F.). – *On finding the maximal elements in a set of plane vectors.* – Report nR-667, Urbana, IL, Dept. Comput. Sci., Univ. Illinois, 1974.

[Yap87] Yap (C. K.). – Symbolic treatment of geometric degeneracies. *In : Proc. 13th IFIP Conf. System Modelling and Optimization*, pp. 348–358. – Tokyo, Japan, 1987.

[Yap90] Yap (C. K.). – Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.*, vol. 10, 1990, pp. 349–370.

[Zie94] Ziegler (G. M.). – *Lectures on Polytopes.* – Springer-Verlag, 1994, *Graduate Texts in Mathematics*, volume 152.

Adaptive Geometriske Algoritmer

Denne afhandling omhandler design af geometriske algoritmer, hvis køretid afhænger af størrelsen af uddata og som derfor kaldes adaptive. Vi beskriver først de væsentligste paradigmer der tillader frembringelse af adaptive algoritmer. Derefter præsenterer vi en nær-optimal adaptiv algoritme til beregning af det konvekse hylster af plane objekter for hvilken størrelsen af uddata for alle objektpar er begrænset. Algoritmen er baseret på en kombination af ægteskabsparadigmet og pakkegrupperingsmetoden. Vi beskæftiger os ligeledes med beregning af det øvre hylster af funktioner samt konveks partiel dekomposition af en punktmængde. Endelig betragter vi gennembrydelighedsproblemet for familier af konvekse objekter, der er bevist NP-hårdt. Vi ser først på isotetiske æsker og giver en adaptiv heuristik, hvis præcision selv er adaptiv. Derefter kigger vi på de kombinatoriske egenskaber af konvekse objekter ud fra et gennembrydelighedssynspunkt. Vi opnår en samling af algoritmer for forskellige klasser af objekter, af hvilke nogle medfører Helly-agtige teoremer.

Nøgleord: Geometriske Algoritmer, Adaptive Algoritmer, Hylstre, Heuristikker, Gennembrydelighed

Algorithmes Géométriques Adaptatifs

Les travaux effectués lors de cette thèse portent sur la construction d'algorithmes géométriques dit adaptatifs dans le sens où leur temps de calcul s'adapte à la solution construite. Nous décrivons tout d'abord les principaux paradigmes qui permettent d'obtenir des algorithmes adaptatifs. Puis , nous proposons un algorithme quasi-optimal adaptatif pour le calcul d'enveloppe convexe d'objets planaires dont la complexité de l'enveloppe convexe de toute paire soit bornée. L'algorithme est basé sur une approche composite combinant le paradigme mariage avant conquête et la méthodologie du groupement en paquets. Nous considérons également le calcul de l'enveloppe supérieure de fonctions et la décomposition convexe partielle d'un ensemble de points. Finalement, nous nous sommes intéressés aux problèmes de perçabilité d'objets qui ont été montré NP-difficiles. En premier lieu, nous avons étudié le cas de boîtes isothétiques en donnant une heuristique adaptative dont la précision soit elle-même adaptative. Ensuite, nous avons étudié les propriétés combinatoires des objets convexes pour la perçabilité. Nous obtenons une batterie d'algorithmes pour des classes variées d'objets dont certains prouvent l'existence de théorèmes de type Helly.

Mots clés : Géométrie algorithmique, Algorithmes adaptatifs, Enveloppes, Heuristiques, Perçabilité.

Output-sensitive Computational Geometry

This thesis deals with the design of algorithms in computational geometry whose complexity depends on the output-size, the so-called output-sensitive algorithms. We first describe the main paradigms that allow algorithms to be output-sensitive. Then, we give a near-optimal output-sensitive algorithm to compute the convex hull of general planar objects such that the output-size of the convex hull of any pair of objects is bounded. We extend the results to the case of envelopes and the partial decomposition of convex and maxima layers. Finally, we consider the pierceability problem for families of convex objects which has been proven NP-hard. We first study the case of isothetic boxes and give an output-sensitive heuristic that is precision-sensitive. Then, we consider the combinatorial properties of convex objects from the pierceability point of view. We obtain a collection of algorithms for various class of objects, some of them implying Helly-type theorems.

Keywords : Computational geometry, Output-sensitive algorithms, Envelopes, Heuristics, Pierceability.