UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

PhD THESIS

High-dimensional polytopes defined by oracles: algorithms, computations and applications

Vissarion G.E. Fisikopoulos

Athens

April 2014

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Πολύτοπα μεγάλης διάστασης ορισμένα μέσω μαντείων: αλγόριθμοι, υπολογισμοί και εφαρμογές

Βησσαρίων Γ.Ε. Φυσικόπουλος

ΑΘΗΝΑ

ΑΠΡΙΛΙΟΣ 2014

PhD THESIS

High-dimensional polytopes defined by oracles:
algorithms, computations and applications

Vissarion G.E. Fisikopoulos

Examination Date: 24/04/2014

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Πολύτοπα μεγάλης διάστασης ορισμένα μέσω μαντείων:
αλγόριθμοι, υπολογισμοί και εφαρμογές

Βησσαρίων Γ.Ε. Φυσικόπουλος

Ημερομηνία εξέτασης: 24/04/2014

ABSTRACT

The processing and analysis of high dimensional geometric data plays a fundamental role in disciplines of science and engineering. The last decades many successful geometric algorithms has been developed in 2 and 3 dimensions. However, in most cases their performance in higher dimensions is poor. This behavior is commonly called *the curse of dimensionality*. A solution framework adopted for the healing of the curse of dimensionality is the exploitation of the special structure of the data, such as sparsity or low intrinsic dimension and the design of approximation algorithms. This thesis studies problems inside this framework.

The main research area of this thesis is discrete and computational geometry and its connections to branches of computer science and applied mathematics like polytope theory, algorithm engineering, randomized geometric algorithms, computational algebraic geometry and optimization. The fundamental geometric objects of the study are *polytopes*, with main properties of being *convex* and defined by an *oracle* in a *high dimensional* space.

The contribution of this thesis is threefold. First, the design and analysis of geometric algorithms for problems concerning high-dimensional, convex polytopes, such as convex hull and volume computation and their applications to computational algebraic geometry and optimization. Second, the establishment of combinatorial characterization results for essential polytope families. Third, the implementation and experimental analysis of the proposed algorithms and methods. The developed software is open-source, publicly available and builds on, extends and is competitive with state-of-the-art geometric and algebraic software libraries such as `CGAL` and `polymake`.

SUBJECT AREA: Discrete and Computational Geometry

KEYWORDS: convex polytopes, general dimension, polytope oracle, edge-skeleton, volume computation, Newton polytope of sparse resultant, secondary polytope, regular triangulations, mixed subdivision, geometric predicates, algorithm engineering, experimental analysis

ΠΕΡΙΛΗΨΗ

Η επεξεργασία και ανάλυση γεωμετρικών δεδομένων σε υψηλές διαστάσεις διαδραματίζει ένα θεμελιώδη ρόλο σε διάφορους κλάδους της επιστήμης και της μηχανικής. Τις τελευταίες δεκαετίες έχουν αναπτυχθεί πολλοί επιτυχημένοι γεωμετρικοί αλγόριθμοι σε 2 και 3 διαστάσεις. Ωστόσο, στις περισσότερες περιπτώσεις, οι επιδόσεις τους σε υψηλότερες διαστάσεις δεν είναι ικανοποιητικές. Αυτή η συμπεριφορά είναι ευρέως γνωστή ως *κατάρα των μεγάλων διαστάσεων* (curse of dimensionality). Δυο πλαίσια λύσης που έχουν υιοθετηθεί για να ξεπεραστεί αυτή η δυσκολία είναι η εκμετάλλευση της ειδικής δομής των δεδομένων, όπως σε περιπτώσεις αραιών (sparse) δεδομένων ή στην περίπτωση που τα δεδομένα βρίσκονται σε χώρο χαμηλότερης διάστασης, και ο σχεδιασμός προσεγγιστικών αλγορίθμων. Στη διατριβή αυτή μελετάμε προβλήματα μέσα σε αυτά τα πλαίσια.

Το κύριο ερευνητικό πεδίο της παρούσας εργασίας είναι η διακριτή και υπολογιστικής γεωμετρία και οι σχέσεις της με τους κλάδους της επιστήμης των υπολογιστών και τα εφαρμοσμένα μαθηματικά, όπως είναι η θεωρία πολυτόπων, οι υλοποιήσεις αλγορίθμων, οι πιθανοθεωρητικοί γεωμετρικοί αλγόριθμοι, η υπολογιστική αλγεβρική γεωμετρία και η βελτιστοποίηση. Τα θεμελιώδη γεωμετρικά αντικείμενα της μελέτης μας είναι τα *πολύτοπα*, και οι βασικές τους ιδιότητες είναι η *κυρτότητα* και ότι ορίζονται από ένα *μαντείο* (oracle) σε ένα χώρο *υψηλής διάστασης*.

Η συμβολή αυτής της διατριβής είναι τριπλή. Πρώτον, στο σχεδιασμό και την ανάλυση των γεωμετρικών αλγορίθμων για προβλήματα σε μεγάλες διαστάσεις. Δεύτερον, στην απόδειξη θεωρητικών αποτελεσμάτων σχετικά με το συνδυαστικό χαρακτηρισμό βασικών οικογενειών πολυτόπων. Τρίτον, στην εφαρμογή και πειραματική ανάλυση των προτεινόμενων αλγορίθμων και μεθόδων. Η ανάπτυξη του λογισμικού είναι ανοιχτού κώδικα, το λογισμικό είναι διαθέσιμο στο κοινό και βασίζεται και επεκτείνει διαδεδομένες γεωμετρικές και αλγεβρικές βιβλιοθήκες λογισμικού, όπως η `CGAL` και το `polymake`.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διακριτή και Υπολογιστική Γεωμετρία

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: κυρτά πολύτοπα, γενική διάσταση, μαντείο πολυτόπων, σκελετός ακμών, υπολογισμός όγκου, πολύτοπο της αραιής απαλείφουσας, δευτερεύον πολύτοπο, κανονικές τριγωνοποίησεις, μικτές υποδιαιρέσεις, γεωμετρικά κατηγορήματα, υλοποιήσεις αλγορίθμων, πειραματική ανάλυση

Acknowledgements

First and foremost I would like to thank my thesis advisor Ioannis Emiris for his inspired guidance. Our white-board discussions have helped me many times to overcome difficulties and deadlocks in my research. I truly enjoy working with him.

I would like to thank Monique Teillaud, member of my thesis three-person committee, for the fruitful discussions on several aspects of my thesis and for being a kind advisor while I was an intern at INRIA Sophia-Antipolis.

I would also like to thank the other member of my thesis three-person committee, Dimitrios Gunopulos, for his interest in my research and for our discussions on subjects that connect computational geometry with more applied topics such as data mining.

I am thankful for the other members of my thesis seven-person committee: Michael Joswig for our interesting discussions on my thesis topics in our few meetings the last two years in U.S. and Berlin; Apostolos Giannopoulos for accepting to participate in my committee and for his kind comments; Menelaos Karavelas for several enlightening discussions on various topics in computational geometry; Stavros Kolliopoulos for his useful comments and discussions on combinatorial optimization topics.

I would like to thank Alicia Dickenstein for her continuous support and encouragement, for our productive collaboration starting with her visit in Athens and for helping me to understand fundamental concepts in algebraic geometry.

I also like to thank Bernd Gaertner for hosting me in ETH Zurich the summer of 2012, for our collaboration and his advice in the topics of discrete geometry and optimization.

I am grateful to my colleague and friend Christos Konaxis for his support and guidance in my first steps of my PhD. I'd also like to thank my other colleague and friend Luis Penaranda for our fruitful and enjoyable collaboration while being in Athens.

I thank Manolis Koubarakis for giving me the opportunity to work in a research project in databases and for the financial support the first months as a PhD student in the University of Athens. Also I'd like to thank Babis Nikolaou for our enjoyable collaboration while working in that project.

I thank all my colleagues in the University of Athens for providing a wonderful working and social environment. I would like to thank my family for their support and their unconditional love. I also thank my friends for reminding me that the adventure is out there. Last but not least, I would like to thank my life partner, Sofia, for her love, her encouragement and her help to overcome my fears. This thesis is dedicated to her.

Part of this thesis was partially supported by the European Commission research project *Computational Geometric Learning*, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 255827. Part of this thesis was co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS - UOA (MIS 375891).

Vissarion Fisikopoulos
Athens, 24 April 2014.

Contents

List of Figures

List of Tables

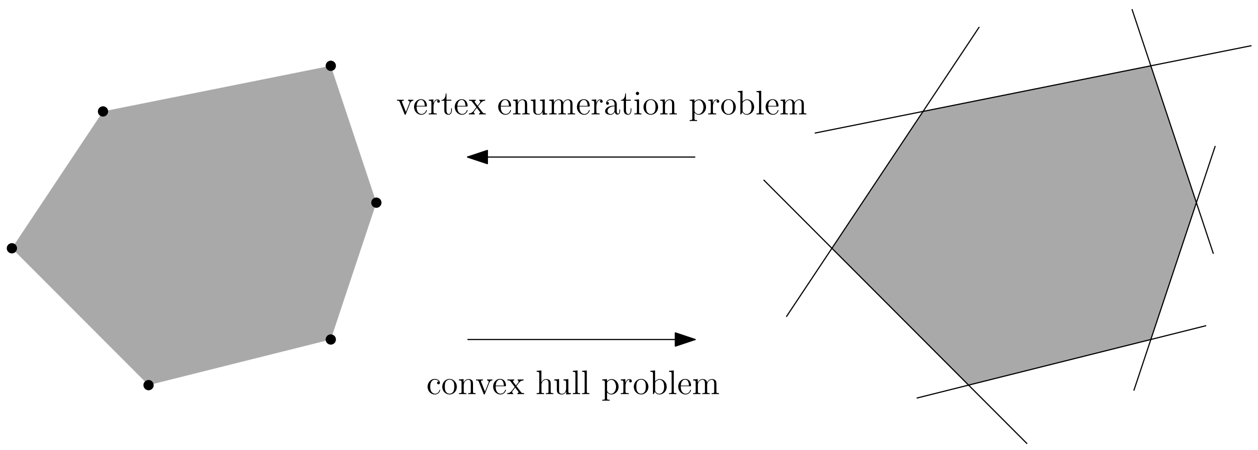
Συνοπτική Παρουσίαση

1 Εισαγωγή

Η επεξεργασία και ανάλυση γεωμετρικών δεδομένων σε υψηλές διαστάσεις διαδραματίζει ένα θεμελιώδη ρόλο σε διάφορους κλάδους της επιστήμης και της μηχανικής. Τις τελευταίες δεκαετίες έχουν αναπτυχθεί πολλοί επιτυχημένοι γεωμετρικοί αλγόριθμοι σε 2 και 3 διαστάσεις. Ωστόσο, στις περισσότερες περιπτώσεις, οι επιδόσεις τους σε υψηλότερες διαστάσεις δεν είναι ικανοποιητικές. Αυτή η συμπεριφορά είναι ευρέως γνωστή ως *κατάρα των μεγάλων διαστάσεων* (curse of dimensionality). Δυο πλαίσια επίλυσης που έχουν υιοθετηθεί για να ξεπεραστεί αυτή η δυσκολία είναι η εκμετάλλευση της ειδικής δομής των δεδομένων, όπως σε περιπτώσεις αραιών (sparse) δεδομένων ή στην περίπτωση που τα δεδομένα βρίσκονται σε χώρο χαμηλότερης διάστασης, και ο σχεδιασμός προσεγγιστικών αλγορίθμων. Στη διατριβή αυτή μελετάμε προβλήματα μέσα σε αυτά τα πλαίσια.

Το κύριο ερευνητικό πεδίο της παρούσας εργασίας είναι η διακριτή και υπολογιστικής γεωμετρία και οι σχέσεις της με τους κλάδους της επιστήμης των υπολογιστών και τα εφαρμοσμένα μαθηματικά, όπως είναι η θεωρία πολυτόπων, οι υλοποιήσεις αλγορίθμων, οι πιθανοθεωρητικοί γεωμετρικοί αλγόριθμοι, η υπολογιστική αλγεβρική γεωμετρία και η βελτιστοποίηση. Τα θεμελιώδη γεωμετρικά αντικείμενα της μελέτης μας είναι τα *πολύτοπα*, και οι βασικές τους ιδιότητες είναι η *κυρτότητα* και ότι ορίζονται από ένα *μαντείο* (oracle) σε ένα χώρο *υψηλής διάστασης*.

Η συμβολή αυτής της διατριβής είναι τριπλή. Πρώτον, στο σχεδιασμό και την ανάλυση των γεωμετρικών αλγορίθμων για προβλήματα σε μεγάλες διαστάσεις. Δεύτερον, σε θεωρητικά αποτελέσματα σχετικά με το συνδυαστικό χαρακτηρισμό βασικών οικογενειών πολυτόπων. Τρίτον, στην εφαρμογή και πειραματική ανάλυση των προτεινόμενων αλγορίθμων και μεθόδων. Η ανάπτυξη λογισμικού είναι ανοιχτού κώδικα. Το λογισμικό είναι διαθέσιμο στον παρακάτω σύνδεσμο



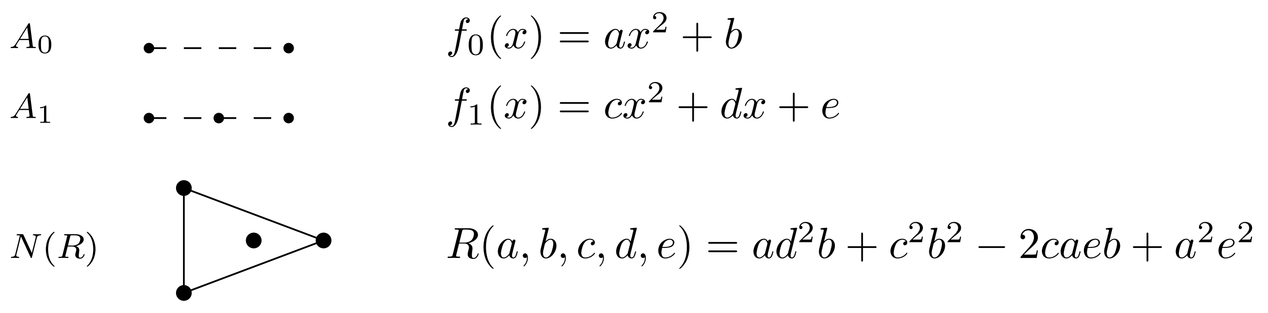
Σχήμα 1.1: Η V- και η H-αναπαράσταση ενός κυρτού πολυτόπου σε δύο διαστάσεις (πολύγωνο).

`http://sourceforge.net/users/fisikop`.

Το λογισμικό που αναπτύχθηκε βασίζεται και επεκτείνει διαδεδομένες γεωμετρικές και αλγεβρικές βιβλιοθήκες λογισμικού, όπως η `CGAL` και το `polymake`.

Στη θεωρία πολυτόπων, ένα (κυρτό) πολύτοπο P δέχεται δύο αναπαραστάσεις. Η πρώτη είναι ως το σύνολο των κορυφών του, και ονομάζεται V-αναπαράσταση ή αναπαράσταση κορυφών. Η δεύτερη είναι η τομή που οριοθετείται από ένα σύνολο γραμμικών ανισοτήτων ή υπόχωρων, η οποία ονομάζεται H-αναπαράσταση ή αναπαράσταση υπόχωρων. Δεδομένου ενός πολυτόπου σε V-αναπαράσταση, ο υπολογισμός της H-αναπαράστασης ονομάζεται πρόβλημα υπολογισμού του κυρτού περιβλήματος (convex hull problem), ενώ το αντίθετο ονομάζεται πρόβλημα απαρίθμησης κορυφών (vertex enumeration problem). Αυτά τα προβλήματα είναι ισοδύναμα λόγω *δυϊκότητας* και είναι δύο από τα πιο σημαντικά υπολογιστικά προβλήματα στη διακριτή και υπολογιστική γεωμετρία. Το Σχήμα 1.1 απεικονίζει ένα παράδειγμα στις δύο διαστάσεις. Για μια αναλυτική παρουσίαση σχετικά με διάφορα θέματα που σχετίζονται με τα κυρτά πολύτοπα αναφερόμαστε στο [140].

Ένα πολύτοπο P μπορεί επίσης να δοθεί σε μια έμμεση αναπαράσταση, που ονομάζεται *μαντείο* ή *χρησμός*. Ένας χρησμός είναι ένα μαύρο κουτί που απαντά σε ερωτήσεις σχετικά με το P. Ένας χρησμός *βελτιστοποίησης*, ή *γραμμικού προγραμματισμού (LP)*, δέχεται ένα διάνυσμα c και επιστρέφει μια κορυφή του P που έχει το μέγιστο εσωτερικό γινόμενο με το c μεταξύ όλων των κορυφών στο P. Μια άλλη σημαντική έμμεση αναπαράσταση για το P είναι να δοθεί από ένα χρησμό *διαχωρισμού*. Δηλαδή, δίνεται ένα σημείο x και το μαντείο επιστρέφει *ναι*, αν $x \in P$ ή



$$f_0(x) = ax^2 + b$$

$$f_1(x) = cx^2 + dx + e$$

$$R(a,b,c,d,e) = ad^2b + c^2b^2 - 2caeb + a^2e^2$$

Σχήμα 1.2: Η απαλείφουσα ενός συστήματος δύο πολυωνύμων σε μια μεταβλητή.

ένα υπερεπίπεδο που χωρίζει το P από το x διαφορετικά. Για παράδειγμα, αν το P δοθεί σε H-αναπαράσταση, ένας χρησμός βελτιστοποίησης για το P δεδομένου ενός διανύσματος c λύνει ένα γραμμικό πρόβλημα στο P, ενώ ένας χρησμός διαχωρισμού για το P δεδομένου σημείου x αποτιμά το σύνολο των ανισοτήτων του P στο σημείο x .

Οι σχέσεις μεταξύ των διαφόρων χρησμών έχουν μελετηθεί από τους Grötschel, Lovàsz και Schrijver στο [78] με τη χρήση ενός μοντέλου υπολογισμού σύμφωνα με το οποίο οι μηχανές Turing επιτρέπεται να καλούν χρησμούς. Για τον υπολογισμό, για παράδειγμα, ενός χρησμού βελτιστοποίησης για το P όταν το P δίνεται από ένα μαντείο διαχωρισμού, πρέπει κανείς να λύσει ένα γραμμικό πρόγραμμα στο P. Αυτό μπορεί να γίνει με τη μέθοδο του ελλειψοειδούς [96]. Λαμβάνοντας υπόψη ένα χρησμό για το P, το πολύτοπο P μπορεί να ανακατασκευαστεί σε κάποια αναπαράσταση (κορυφών ή υπόχωρων) χρησιμοποιώντας ένα αυξητικό αλγόριθμο υπολογισμού κυρτού περιβλήματος, όπως ο Beneath-and-Beyond [37].

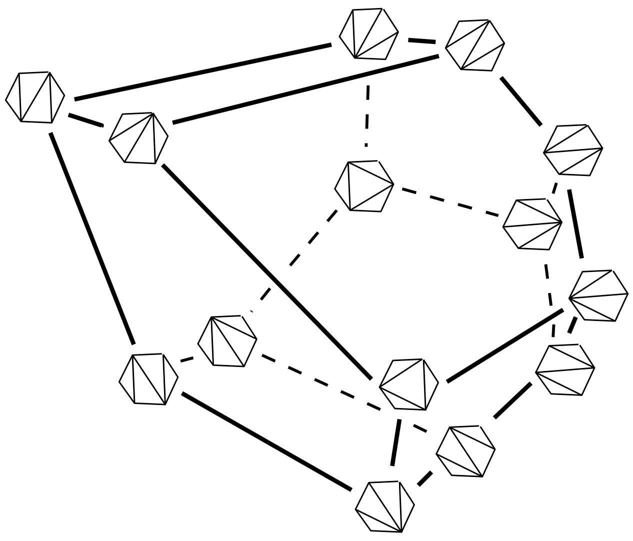
2 Αλγόριθμοι για τον υπολογισμό του πολύτοπου της απαλείφουσας

Υπό το πρίσμα της αλγεβρικής γεωμετρίας τα (κυρτά) πολύτοπα χαρακτηρίζουν με μεγαλύτερη ακρίβεια ένα πολυώνυμο από ό,τι ο συνολικός του βαθμός. Για το λόγο αυτό αποτελούν ένα θεμελιώδες αντικείμενο μελέτης στη θεωρία αραιής αλγεβρικής απαλοιφής. Μια βασική κατηγορία πολυτόπων αυτής της μορφής με εφαρμογή και στην επίλυση εξισώσεων είναι το πολύτοπο του Νεύτωνα της αραιής απαλείφουσας, ή απλά *πολύτοπο της απαλείφουσας* που το συμβολίζουμε ως $N(R)$. Τα πολύτοπα αυτά έχουν μελετηθεί από τους Gelfand, Kapranov και Zelevinsky στο [74] και από τον Sturmfels στο [131]. Ένα παράδειγμα της απαλείφουσας δύο πολυωνύμων f_0, f_1 σε μια μεταβλητή x απεικονίζεται στο Σχήμα 1.2. Είναι ένα πολυώνυμο R στους συντελεστές a, b, c, d, e των δύο πολυωνύμων που μηδενίζεται αν το σύστημα που παίρνουμε με την αντικατάσταση των a, b, c, d, e με αριθμητικές τιμές έχει λύση. Εδώ, το πολύτοπο της απαλείφουσας $N(R)$ είναι ένα τρίγωνο.

Στο [74] η μελέτη του πολυτόπου της απαλείφουσας συνδέεται με την μελέτη του *δευτερεύοντος πολυτόπου*. Το δευτερεύον πολύτοπο ενός συνόλου σημείων A είναι ένα σημαντικό αντικείμενο στην γεωμετρική συνδυαστική, δεδομένου ότι προσφέρει μια αναπαράσταση του γραφήματος των κανονικών τριγωνοποιήσεων του A. Η εικόνα 1.3 απεικονίζει το δευτερεύον πολύτοπο ενός κυρτού εξαγώνου. Αυτή είναι μια ειδική περίπτωση, όπου τα σημεία είναι σε κυρτή θέση στις δύο διαστάσεις. Σε αυτή την περίπτωση όλες οι τριγωνοποιήσεις είναι κανονικές και το δευτερεύον πολύτοπο είναι το 3-διάστατο ασοσιάεδρο (associahedron) [119].

Το κεφάλαιο 2 παρουσιάζει το σχεδιασμό και την ανάλυση του πρώτου *ευαίσθητου εξόδου* αλγόριθμου για τον υπολογισμό πολυτόπων της απαλείφουσας και προβολών αυτών. Ο αλγόριθμος είναι ευαίσθητος εξόδου, δεδομένου ότι κάνει μια κλήσης στο μαντείο ανά κορυφή και μία ανά έδρα του πολυτόπου. Τα βασικά στοιχεία του αλγορίθμου είναι η αναπαράσταση του πολυτόπου της απαλείφουσας από ένα χρησμό βελτιστοποίησης και η εκμετάλλευση της χαμηλής εγγενούς (intrinsic) του διάστασης. Το μαντείο κατασκευάζει κανονικές τριγωνοποιήσεις προκειμένου να υπολογίσει τη βέλτιστη κορυφή του πολυτόπου. Τέλος, το πολύτοπο της απαλείφουσας ανακατασκευάζεται χρησιμοποιώντας έναν αυξητικό αλγόριθμο κυρτού περιβλήματος που χρησιμοποιεί αυτό το μαντείο.

Ο αλγόριθμος υλοποιείται στο πακέτο λογισμικού `respol`, το οποίο υπολογίζει 5, 6 - και 7 διαστάσεων πολύτοπα με $35 \cdot 10^3$, $23 \cdot 10^3$ και 500 κορυφές, αντίστοιχα, μέσα σε 2 ώρες σε έναν τυπικό υπολογιστή. Επίσης υπολογίζει τα πολύτοπα πολλών σημαντικών εξισώσεων επιφανειών που απαντώνται στην γεωμετρική μοντελοποίηση σε < 1 δευτερόλεπτα, ενώ η απαρίθμηση των κορυφών των αντίστοιχων δευτερευόντων πολυτόπων είναι αδύνατη με τους υπάρχοντες υπολογιστές. Το `respol` έχει χρησιμοποιηθεί για να λύσει κάποια βασικά προβλήματα στην περιοχή της γεωμετρικής σχεδίασης με υπολογιστή [61], καθώς και για τον υπολογισμό πολυωνύμων


Σχήμα 1.3: Το δευτερεύον πολύτοπο ενός εξαγώνου.

της διακρίνουσας [62]. Επιπλέον, προτείνουμε και υλοποιούμε μια τεχνική που ονομάζεται *κατακερματισμός των οριζουσών*, η οποία αποτρέπει την αλληλοεπικάλυψη υπολογισμών με ταυτόχρονη αξιοποίηση της δομής των οριζουσών που υπολογίζονται από τον αλγόριθμο. Στην πράξη, αυτή η τεχνική επιταχύνει την εκτέλεση μέχρι και 100 φορές. Τα αποτελέσματα της εργασίας αυτής έχουν δημοσιευτεί στο διεθνές συνέδριο της υπολογιστικής γεωμετρίας [59] καθώς και στην ειδική έκδοση του περιοδικού [60]. Μια επέκταση της παραπάνω μεθόδου για τον υπολογισμό πολυτόπων της διακρίνουσας παρουσιάζεται στην ενότητα 2.6 και έχει δημοσιευτεί στο [58].

3 Υπολογισμός πολυτοπικών σκελετών από ακμές

Το γεγονός ότι ο παραπάνω αλγόριθμος δεν είναι αποδοτικός σε περισσότερες από 8 διαστάσεις μας οδηγεί στη μελέτη αλγορίθμων *συνολικά πολυωνυμικού χρόνου*. Ένας αλγόριθμος τρέχει σε συνολικά πολυωνυμικό χρόνο αν η χρονική πολυπλοκότητα του φράσσεται από ένα πολυώνυμο στο μέγεθος της εισόδου, στο μέγεθος της εξόδου και στη διάσταση. Γενικά, το πρόβλημα της εύρεσης αλγορίθμων συνολικά πολυωνυμικού χρόνου για το πρόβλημα του κυρτού περιβλήματος είναι ένα σημαντικό ανοικτό πρόβλημα στην αλγοριθμική διακριτή γεωμετρία. Ωστόσο, υπάρχουν αλγόριθμοι συνολικά πολυωνυμικού χρόνου για ειδικές περιπτώσεις πολυτόπων όπως, simplicial πολύτοπα [8] και 0/1-πολύτοπα [32].

Εδώ μελετάμε μια άλλη ειδική περίπτωση για την οποία δείχνουμε ότι υπάρχουν αλγόριθμοι συνολικά πολυωνυμικού χρόνου. Παρουσιάζουμε τον πρώτο αλγόριθμο συνολικά πολυωνυμικού χρόνου για μια ειδική περίπτωση του προβλήματος της απαρίθμησης κορυφών, όπου το πολύτοπο δίνεται από ένα χρησμό βελτιστοποίησης και μας δίνεται επίσης και ένα υπερσύνολο των κατευθύνσεων των ακμών του. Ειδικότερα ο αλγόριθμος υπολογίζει το σκελετό των ακμών του πολυτόπου, το οποίο είναι το γράφημα των κορυφών και των ακμών του πολυτόπου. Δεδομένου ότι οι κορυφές υπολογίζονται μαζί με το σκελετό, ο αλγόριθμος υπολογίζει και μια απαρίθμηση των κορυφών.

Μελετάμε δυο βασικές εφαρμογές. Εναλλασσόμενα Minkowski αθροίσματα κυρτών πολυτόπων, όπου τα πολύτοπα μπορούν να αφαιρεθούν υπό την προϋπόθεση ότι το αποτέλεσμα είναι επίσης κυρτό πολύτοπο, και δευτερεύοντα πολύτοπα, πολύτοπα της απαλείφουσας και της διακρίνουσας. Περαιτέρω εφαρμογές περιλαμβάνουν προβλήματα από την κυρτή συνδυαστική βελτιστοποίηση και κυρτό ακέραιο προγραμματισμό, όπου ο αλγόριθμος μας προσφέρει μια εναλλακτική προσέγγιση, αίροντας την εκθετική εξάρτηση από τη διάσταση στην πολυπλοκότητα.

Τα αποτελέσματα της εργασίας αυτής παρουσιάζονται στο κεφάλαιο 3. Ορισμένα αποτελέσματα έχουν δημοσιευθεί στο [56] και η πλήρη έκδοση τους στο [57].

4 Προσεγγιστικός υπολογισμός όγκου

Η προσπάθεια απαρίθμησης κορυφών σε υψηλές διαστάσεις (π.χ. εκατό) χρησιμοποιώντας τις παραπάνω μεθόδους είναι μάταιη. Η διατριβή αυτή αποσκοπεί στην εξερεύνηση των ορίων του υπολογισμού βασικών χαρακτηριστικών ενός πολυτόπου, όπως ο όγκος του. Αν και το πρόβλημα υπολογισμού του όγκου είναι #P-hard για V- και H-αναπαραστάσεις πολυτόπων [51] υπάρχουν πιθανοθεωρητικοί αλγόριθμοι πολυωνυμικού χρόνου για την προσέγγιση του όγκου ενός κυρτού σώματος με μεγάλη πιθανότητα και αυθαίρετα μικρό σχετικό σφάλμα. Ο πρώτος τέτοιος αλγόριθμος παρουσιάστηκε στο [50], και μια σειρά νέων αποτελεσμάτων μείωσε τον εκθέτη στην πολυπλοκότητα των αλγορίθμων αυτών από 27 σε 4 [104]. Ωστόσο, το πρόβλημα μιας αποδοτικής υλοποίησης παρέμεινε ανοικτό.

Αυτή η διατριβή μελετάει αυτό το πρόβλημα πειραματικά δεδομένου ενός κυρτού πολυτόπου σε H-αναπαράσταση. Εφαρμόζουμε και αξιολογούμε στην πράξη πιθανοκρατικούς αλγόριθμους για την προσέγγιση όγκου πολυτόπων σε υψηλές διαστάσεις (π.χ. εκατό). Για να πραγματοποιηθεί η υλοποίηση αποτελεσματικά συσχετίζουμε πειραματικά την επίδραση παραμέτρων, όπως το μήκος τυχαίων περιπάτων και ο αριθμός των σημείων δειγματοληψίας, με την ακρίβεια του υπολογισμού και το χρόνο. Επιπλέον, εκμεταλλευόμαστε τη γεωμετρία του προβλήματος με την εφαρμογή μιας επαναληπτικής διαδικασίας στρογγυλοποίησης, υπολογίζοντας γενιές τυχαίων σημείων και σχεδιάζοντας αποδοτικά μαντεία για τα πολύτοπα. Η υλοποίηση είναι σημαντικά ταχύτερη από αυτές που υπολογίζουν τον όγκο ντετερμινιστικά. Υπολογίζουμε επίσης προσεγγίσεις για τους όγκους των Birkhoff πολυτόπων $\mathcal{B}_{11}, \ldots, \mathcal{B}_{15}$, ενώ μόνο ο όγκος του $\mathcal{B}_{10}$ έχει υπολογιστεί με ακρίβεια.

Τα αποτελέσματα της εργασίας αυτής παρουσιάζονται στο Κεφάλαιο 4 και έχουν δημοσιευθεί στο [55].

5 Συνδυαστική των πολυτόπων της απαλείφουσας

Μελετάμε τη συνδυαστική των πολυτόπων της απαλείφουσας. Υπάρχουν γνωστά αποτελέσματα στην περίπτωση των δύο πολυωνύμων σε μία μεταβλητή [73] και στην περίπτωση όπου η διάσταση του πολυτόπου είναι μέχρι και 3 [131]. Επεκτείνουμε τα αποτελέσματα αυτά και απαντάμε σε ένα ανοιχτό ερώτημα που τίθεται στο [86] με τη μελέτη της συνδυαστικής των 4-διάστατων πολυτόπων της απαλείφουσας. Η μελέτη δείχνει μια μεγαλύτερη ποικιλία πολυτόπων σε αυτή τη διάσταση και περιλαμβάνει υπολογιστικές και συνδυαστικές προκλήσεις.

Ειδικότερα, τα πειράματα μας, με βάση το `respol`, δίνουν κάτω φράγματα στο μέγιστο πλήθος των όψεων των πολυτόπων. Με τη μελέτη των υποδιαιρέσεων των Minkowski αθροισμάτων, που ονομάζονται *μικτές υποδιαιρέσεις*, παίρνουμε σφιχτά (tight) άνω φράγματα για το μέγιστο αριθμό των όψεων που δείχνουμε ότι είναι 22.

Τα αποτελέσματα της εργασίας αυτής παρουσιάζονται στο Κεφάλαιο 5 και έχουν δημοσιευθεί στο [47].

6 Γεωμετρικά κατηγορήματα

Οι γεωμετρικοί αλγόριθμοι περιλαμβάνουν τόσο συνδυαστικούς όσο και αλγεβρικούς υπολογισμούς. Σε πολλές περιπτώσεις, όπως ο υπολογισμός του κυρτού περιβλήματος, οι τελευταίοι συνίστανται στον υπολογισμό του πρόσημου μιας ορίζουσας, που ονομάζεται γεωμετρικό κατηγόρημα. Όσο η διάσταση του χώρου υπολογισμού μεγαλώνει, ένα μεγαλύτερο ποσοστό του χρόνου υπολογισμού καταναλώνεται από αυτά τα κατηγορήματα. Στόχος μας είναι να μελετήσουμε τις ακολουθίες των κατηγορημάτων που εμφανίζονται στους γεωμετρικούς αλγόριθμους. Χρησιμοποιούμε δυναμικούς αλγορίθμους υπολογισμού της ορίζουσας για να επιταχύνουμε τον υπολογισμό του κάθε κατηγορήματος χρησιμοποιώντας πληροφορία από τα προηγουμένως υπολογισμένα κατηγορήματα.

Προτείνουμε δύο δυναμικούς αλγορίθμους υπολογισμού οριζουσών με τετραγωνική πολυπλοκότητα όταν χρησιμοποιούνται σε υπολογισμούς κυρτού περιβλήματος, και με γραμμική πολυπλοκότητα όταν χρησιμοποιούνται σε προβλήματα εύρεσης σημείου. Επιπλέον, τους υλοποιούμε και τους αναλύουμε πειραματικά. Οι υλοποιήσεις μας είναι ταχύτερες από τις πιο αποδοτικές υλοποιήσεις κυρτών περιβλημάτων, καθώς δίνουν επιτάχυνση ως και 78 φορές σε προβλήματα εύρεσης σημείου.

Τα αποτελέσματα της εργασίας αυτής παρουσιάζονται στο Κεφάλαιο 6 και έχουν δημοσιευθεί στο [65]. Το πακέτο λογισμικού που αναπτύχθηκε έχει υποβληθεί στη βιβλιοθήκη `CGAL` [35].

7 Επεκτάσεις και ανοικτά προβλήματα

Αρκετά ενδιαφέροντα ανοικτά προβλήματα αναδεικνύονται μέσα από τη μελέτη της παρούσας διατριβής. Από την οπτική της γεωμετρικής συνδυαστικής ένα ερώτημα είναι να καταλάβουμε τη συμμετρία των μέγιστων f-διανυσμάτων, που προκύπτουν από τη μελέτη των 4-διάστατων πολυτόπων της απαλείφουσας.

Επιπλέον προκύπτουν αρκετά ανοικτά προβλήματα που σχετίζονται με τη δειγματοληψία. Το πρώτο είναι να μελετηθεί ο αλγόριθμος προσέγγισης όγκου όταν δίνεται ένας χρησμός βελτιστοποίησης. Η τρέχουσα έρευνα εστιάζει σε κυρτά σώματα, ή πολύτοπα, που δίνονται από ένα χρησμό διαχωρισμού. Άλλα ανοικτά προβλήματα αφορούν τον υπολογισμό του όγκου των σπεκταέδρων (spectahedra) ή γενικά ημι-αλγεβρικών συνόλων, την εφαρμογή του τρέχοντος λογισμικού σε άλλα # P-hard προβλήματα όπως η καταμέτρηση των γραμμικών επεκτάσεων ενός μερικών διατεταγμένου συνόλου, τον υπολογισμό ολοκληρωμάτων πολυωνυμικών συναρτήσεων ορισμένα σε κυρτά πολύτοπα, τη μελέτη της ποιότητας των μεθόδων δειγματοληψίας, και μελέτη της δειγματοληψίας σε ακέραια σημεία μέσα σε πολύτοπα.

Τέλος το πρόβλημα του *πλησιέστερου γείτονα* έχει θεωρηθεί ως ένα από τα πιο θεμελιώδη προβλήματα στην επιστήμη των υπολογιστών από τη σκοπιά των εφαρμογών. Η μελέτη μας στο κεφάλαιο 4 ανοίγει το δρόμο για την εφαρμογή αλγορίθμων για τον υπολογισμό του κατά προσέγγιση πλησιέστερου γείτονα στον υπολογισμό προσεγγιστικών μαντείων πολυτόπων.

Chapter 1

Polytopes, Algorithms and Applications

1.1 Introduction

The processing and analysis of high dimensional geometric data plays a fundamental role in disciplines of science and engineering. In the last decades many successful geometric algorithms have been developed in 2 and 3 dimensions. However, in most cases their performance in higher dimensions is poor. This behaviour is commonly called *the curse of dimensionality*. A solution framework adopted for the healing of the curse of dimensionality is the exploitation of the special structure of the data, such as sparsity or low intrinsic dimension, and the design of approximation algorithms. This thesis studies problems inside this framework.

The main research area is discrete and computational geometry and its connections to branches of computer science and applied mathematics like polytope theory, algorithm engineering, randomized geometric algorithms, computational algebraic geometry and optimization. The fundamental geometric objects of the study are *polytopes*, with main properties of being *convex* and defined in a *high dimensional* space.

The contribution of this thesis is threefold. First, the design and analysis of geometric algorithms for problems concerning high-dimensional convex polytopes, such as convex hull and volume computation and their applications to computational algebraic geometry and optimization. Second, the establishment of combi-

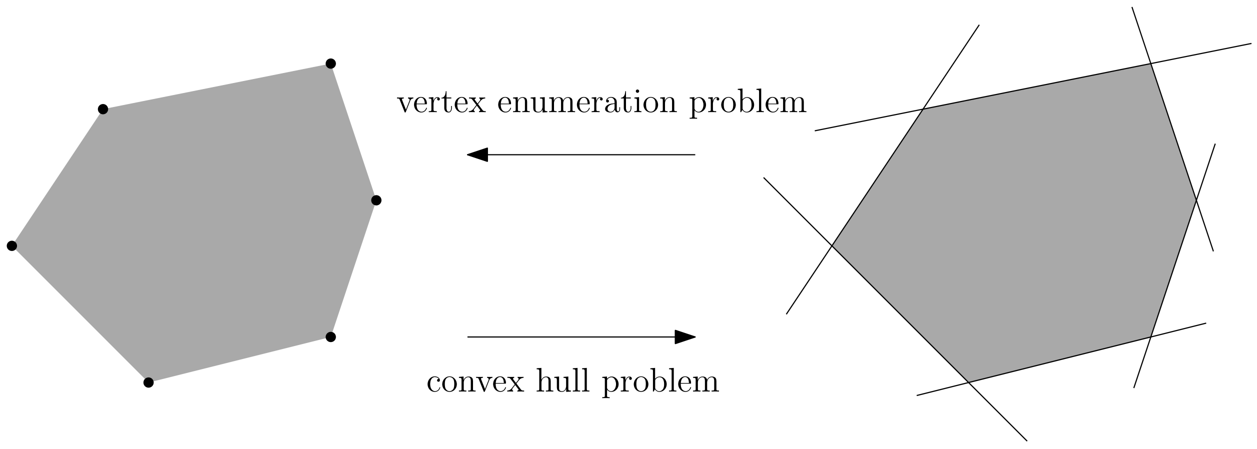


Figure 1.1: The V- and H-representation of a convex polygon.

natorial characterization results for essential polytope families. Third, the implementation and experimental analysis of the proposed algorithms and methods. The developed software is open-source, publicly available from:

`http://sourceforge.net/users/fisikop`.

It builds on, extends and is competitive with state-of-the-art geometric and algebraic software libraries such as `CGAL` [35] and `polymake` [72]. What follows is a smooth introduction to the research topics and contributions of the thesis, avoiding technical details.

In polytope theory, a (convex) polytope P admits two explicit representations. The first is the set of P vertices, which is called the V-representation or vertex representation. The second is the bounded intersection of a set of linear inequalities or half-spaces, which is called H-representation or halfspace representation. Given a polytope in V-representation, computing the H-representation constitutes the *convex hull* problem, while the opposite is the *vertex enumeration* problem. These problems are algorithmically equivalent from a computational complexity point of view by *polytope duality* and establish two of the most important computational problems in discrete geometry. See Figure 1.1 for an illustration. For a detailed presentation on several aspects related to convex polytopes we refer to [140].

A polytope P can also be given by an implicit representation, called *(polytope) oracle*. An oracle is a black box routine that answers questions regarding P. An *optimization*, or *linear programming (LP)*, or *vertex* oracle given a vector c returns

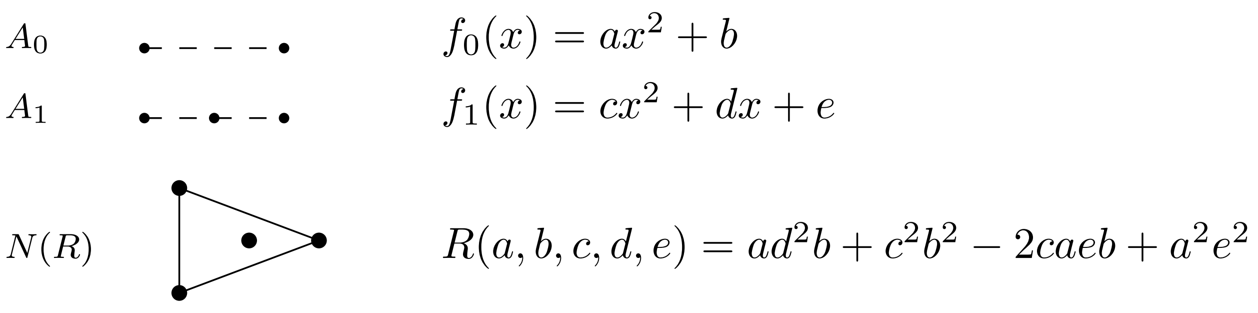


Figure 1.2: The resultant of a system of two polynomials in one variable.

a vertex of P that has the maximum inner product with c among all points in P. Another important implicit representation for P is to be given by a *separation* oracle. That is, given a point x the oracle returns yes if $x \in P$ or a hyperplane that separates P from x otherwise. To illustrate the above definitions, let P be given in H-representation. Then an optimization oracle for P given a vector c solves an LP problem on P, while a separation oracle for P given point x evaluates the set of defining inequalities of P with x.

The relations among various oracles have been studied by Grötschel, Lovàsz and Schrijver in [78] by adopting the oracle Turing machine model of computation. To acquire, for example, an optimization oracle for P when P is given by a separation oracle, one has to solve a linear program over P. This can be done by the ellipsoid method [96]. Given an oracle for P, the entire polytope P can be reconstructed and its explicit representation can be found using an incremental convex hull algorithm such as the Beneath-and-Beyond [37].

1.2 Algorithms for resultant polytopes

From the algebraic geometry perspective polytopes characterize polynomials better than total degree thus offering the fundamental representation in sparse elimination theory, called *Newton polytopes*. An important class of such polytopes is the Newton polytopes of the *sparse resultant polynomial* or simply the *resultant polytopes*. They have been studied by Gelfand, Kapranov and Zelevinsky in [74] and by Sturmfels in [131]. An example of the resultant of two polynomials f_0, f_1

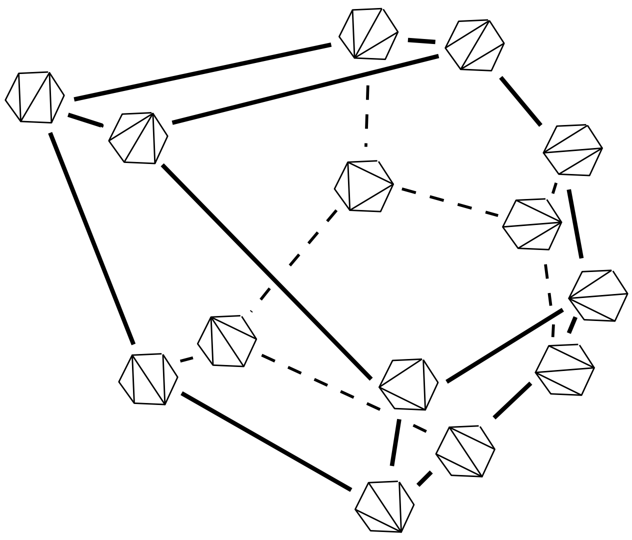


Figure 1.3: The secondary polytope of a hexagon.

in one variable x is depicted in Figure 1.2. It is a polynomial R in the coefficients a, b, c, d, e of the two polynomials which vanishes if the system we get by specializing a, b, c, d, e to numerical values has a solution. Here, the Newton polytope $N(R)$ of the resultant is a triangle.

In [74] the study of resultant polytopes is connected to the study of *secondary polytopes*. The secondary polytope of a pointset A is a fundamental object in geometric combinatorics since it offers a polytope realization of the graph of *regular triangulations* of the pointset. Figure 1.3 depicts the secondary polytope of a convex hexagon. This is a special case where the points in A are in convex position and two dimensional. In this case all triangulations are regular and the secondary polytope is the 3-dimensional associahedron [119].

Chapter 2 presents the design and the analysis of the first *output-sensitive* algorithm for computing (projections of) resultant polytopes. The algorithm is output-sensitive as it makes one oracle call per vertex and facet of the polytope. The key ingredients of that algorithm is the compact representation of resultant polytopes by an optimization oracle and the exploitation of their low intrinsic dimension. The oracle constructs regular triangulations in order to compute the optimal vertex in the polytope. Finally, the resultant polytope is reconstructed using an incremental convex hull algorithm that uses this oracle.

The algorithm is implemented in the software package `respol`, which computes 5-, 6- and 7-dimensional polytopes with $35 \cdot 10^3$, $23 \cdot 10^3$ and 500 vertices, respectively, within 2 hours on a standard computer, and the Newton polytopes of many important surface equations encountered in geometric modelling in < 1sec, whereas the enumeration of the vertices of the corresponding secondary polytopes is intractable. `respol` has been used to solve essential problems in CAD [61] as well as to compute discriminant polynomials [62]. We propose and implement a technique called *hashing of determinants*, which avoids duplication of computations by exploiting the nature of determinants computed by the algorithm. In practice, this technique accelerates execution up to 100 times.

The results of this work have been published in [59] and their full version in [60]. An extension of the above method to computing discriminant polytopes is discussed in Section 2.6 and has appeared in [58].

1.3 Edge-skeleton computation

Motivated by the fact that the above algorithm is impractical in 8 or more dimensions since it relies on an incremental convex hull algorithm, the study extends in finding more efficient, i.e. *total polynomial-time*, algorithms for convex hulls. An algorithm runs in total polynomial-time if its time complexity is bounded by a polynomial in the input *and* output size. In general dimension finding a total polynomial time algorithm for vertex enumeration is a major open problem in algorithmic geometry. However, total polynomial-time algorithms exist for vertex enumeration of special polytope cases, such as simplicial polytopes [8] and 0/1-polytopes [32].

Here we establish another case where total polynomial-time algorithms exist. We present the first total polynomial-time algorithm for a special case of the vertex enumeration problem where the polytope is given by an optimization oracle and we are also given a superset of its edge directions. In particular the algorithm computes the *edge-skeleton* (or 1-skeleton) of the polytope, which is the graph of polytope vertices and edges. Since the vertices are computed along with the skeleton, the edge-skeleton computation subsumes vertex enumeration.

We consider two main applications. We obtain total polynomial-time algorithms for computing signed Minkowski sums of convex polytopes, where polytopes can be subtracted provided the signed sum is a convex polytope, and for computing secondary, resultant, and discriminant polytopes. Further applications include convex combinatorial optimization and convex integer programming, where we offer an alternative approach, thus removing the exponential dependence on the dimension in the complexity.

The results of this work are presented in Chapter 3. Some preliminary results have been published in [56] and their full version in [57].

1.4 Approximate volume computation

Vertex enumeration in high dimensions (e.g. one hundred) using the above methods is a futile attempt. Thus, this thesis aims at exploiting the limits of learning fundamental characteristics of a polytope such as its volume. Although volume computation is #-P hard for V- and H-representations of polytopes [51] there exist randomized polynomial time algorithms to approximate the volume of a convex body with high probability and arbitrarily small relative error. Starting with the breakthrough polynomial time algorithm of [50], subsequent results brought down the exponent on the dimension from 27 to 4 [104]. However, the question of an efficient implementation had remained open.

This thesis undertakes this by experimentally studying the fundamental problem of computing the volume of a convex polytope given as an intersection of linear inequalities. We implement and evaluate practical randomized algorithms for accurately approximating the polytope's volume in high dimensions (e.g. one hundred). To carry out this efficiently we experimentally correlate the effect of parameters, such as random walk length and number of sample points, on accuracy and runtime. Moreover, we exploit the problem's geometry by implementing an iterative rounding procedure, computing partial generations of random points and designing fast polytope boundary oracles. Our publicly available code is significantly faster than exact computation. We provide volume estimations for the Birkhoff polytopes $\mathcal{B}_{11}, \ldots, \mathcal{B}_{15}$, whereas only the volume of $\mathcal{B}_{10}$ has computed exactly.

The results of this work are presented in Chapter 4 and have been published in [55].

1.5 Combinatorics of resultant polytopes

We study the combinatorics of resultant polytopes. These are known in the case of two polynomials in one variable, also known as the Sylvester case [73] and in the case where the polytope's dimension is up to 3 [131]. We extend this work and at the same time answer an open question raised in [86] by studying the combinatorial characterization of 4-dimensional resultant polytopes, which show a greater diversity and involve computational and combinatorial challenges.

In particular, our experiments, based on `respol`, establish lower bounds on the maximal number of faces. By studying subdivisions of Minkowski sums, called *mixed subdivisions*, we obtain tight upper bounds on the maximal number of facets and ridges. These yield an upper bound for the number of vertices, which is 28 whereas the previous known bound was 6608 [131]. We establish a result of independent interest, namely that the f-vector is maximized when the input is sufficiently generic, namely full dimensional and without parallel edges. Lastly, we offer a classification result of all possible 4-dimensional resultant polytopes.

The results of this work are presented in Chapter 5 and have been published in [47].

1.6 Geometric predicates

Geometric algorithms involve both combinatorial and algebraic computation. In many cases, such as convex hull computations, the later boils down to determinant sign computations, also called *geometric predicates*. As the dimension of the computation space grows, a higher percentage of the computation time is consumed by these predicates. Our goal is to study the sequences of determinants that appear in geometric algorithms. We use dynamic determinant algorithms to speed-up the computation of each predicate by using information from previously computed predicates.

We propose two dynamic determinant algorithms with quadratic complexity when employed in convex hull computations, and with linear complexity when used in point location problems. Moreover, we implement them and perform an experimental analysis. Our implementations outperform the state-of-the-art determinant and convex hull implementations in most of the tested scenarios, as well as giving a speed-up of 78 times in point location problems.

The results of this work are presented in Chapter 6 and have been published in [65]. The developed software package has been submitted in CGAL [35] and is currently under revision.

1.7 Extensions and open problems

Several intriguing open questions emerge by the study of this thesis. From the geometric combinatorics point of view one question is to understand the symmetry of the maximal f-vector, i.e. vector of polytope's face cardinalities, that appear in the study of the combinatorics of 4-dimensional resultant polytopes.

There are a few questions related to sampling. The first is to study volume approximation algorithms when an optimization oracle is available. The current research focuses on convex bodies, or polytopes, represented by a membership oracle. A special case which is also interesting is to sample random points from polytopes given in V-representation without using membership queries. Other related problems are computing the volume of spectahedra or general semi-algebraic sets, application of the current software to other #P-hard problems like counting linear extensions of partial ordered sets, integration of polynomial functions over convex polytopes, study polytopes that are easy/difficult to sample from under the assumption that they are rounded, study the quality of sampling or compare point samples, and sampling integer points from polytopes.

Nearest neighbour searching has been considered as one of the most fundamental problems in computer science. Our study in Chapter 4 paves the way for an application of approximate nearest neighbour searching to approximate polytope oracles and polytope volume approximation.

Chapter 2

Algorithms for resultant polytopes

2.1 Introduction

Given pointsets $A_0, \dots, A_n \subset \mathbb{Z}^n$, we define the pointset

$$\mathcal{A} := \bigcup_{i=0}^{n} (A_i \times \{e_i\}) \subset \mathbb{Z}^{2n}, \tag{2.1}$$

where $e_0, \dots, e_n$ form an affine basis of $\mathbb{R}^n$: e_0 is the zero vector, $e_i = (0, \dots, 0, 1, 0, \dots, 0), i = 1, \dots, n$. Clearly, $|\mathcal{A}| = |A_0| + \cdots + |A_n|$, where $|\cdot|$ denotes cardinality. By Cayley's trick (Proposition 2) the regular tight mixed subdivisions of the Minkowski sum $A_0 + \cdots + A_n$ are in bijection with the regular triangulations of $\mathcal{A}$, which are in bijection with the vertices of the *secondary polytope* $\Sigma(\mathcal{A})$ (see Section 2.2).

The *Newton polytope* of a polynomial is the convex hull of its *support*, i.e. the exponent vectors of monomials with nonzero coefficient. It subsumes the notion of degree for sparse multivariate polynomials by providing more precise information (see Figures 2.1 and 2.3). Given $n + 1$ polynomials in n variables, with fixed supports A_i and symbolic coefficients, their *sparse (or toric) resultant* $\mathcal{R}$ is a polynomial in these coefficients which vanishes exactly when the polynomials have a common root (Definition 1). The resultant is the most fundamental tool in elimination theory, it is instrumental in system solving and optimization, and is crucial in geometric modeling, most notably for changing the representation of parametric hypersurfaces to implicit.

The Newton polytope of the resultant $N(\mathcal{R})$, or *resultant polytope*, is the object of our study; it is of dimension $|\mathcal{A}| - 2n - 1$ (Proposition 4). We further consider

$f(x_1, x_2) = 8x_2 + x_1x_2 - 24x_2^2 - 16x_1^2 + 220x_1^2x_2 - 34x_1x_2^2 - 84x_1^3x_2 + 6x_1^2x_2^2 - 8x_1x_2^3 + 8x_1^3x_2^2 + 8x_1^3 + 18x_2^3$

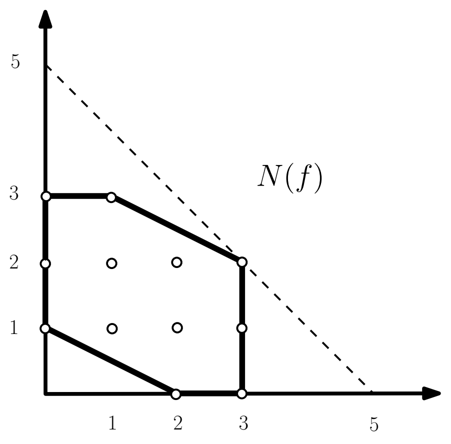


Figure 2.1: The Newton polytope of a polynomial of degree 5 in two variables. Every monomial corresponds to an integral point on the plane. The dashed triangle is the corresponding polytope of the dense polynomial of degree 5.

the case when some of the input coefficients are not symbolic, hence we seek an orthogonal projection of the resultant polytope. The lattice points in $N(\mathcal{R})$ yield a superset of the support of $\mathcal{R}$; this reduces implicitization [61, 132] and computation of $\mathcal{R}$ to sparse interpolation (Section 2.2). The number of coefficients of the $n+1$ polynomials ranges from $O(n)$ for sparse systems, to $O(n^d d^n)$, where d bounds their total degree. In system solving and implicitization, one computes $\mathcal{R}$ when all but $O(n)$ of the coefficients are specialized to constants, hence the need for resultant polytope projections.

The resultant polytope is a Minkowski summand of $\Sigma(\mathcal{A})$, which is also of dimension $|\mathcal{A}| - 2n - 1$. We consider an equivalence relation defined on the $\Sigma(\mathcal{A})$ vertices, where the classes are in bijection with the vertices of the resultant polytope. This yields an oracle producing a resultant vertex in a given direction, thus avoiding to compute $\Sigma(\mathcal{A})$, which typically has much more vertices than $N(\mathcal{R})$. This is known in the literature as an *optimization* oracle since it optimizes inner product with a given vector over the (unknown) polytope.

Example 1. [The bicubic surface] A standard benchmark in geometric modeling is the implicitization of the bicubic surface, with $n = 2$, defined by 3 polynomials in two parameters. The input polynomials have supports $A_i \subset \mathbb{Z}^2$, $i = 0, 1, 2$, with cardinalities $7, 6, 14$, respectively; the total degrees are $3, 3, 6$, respectively. The Cayley set $\mathcal{A} \subset \mathbb{Z}^4$, constructed as in Equation 2.1, has $7 + 6 + 14 = 27$ points. It is depicted in the following matrix, with coordinates as columns, where the supports from different polynomials and the Cayley coordinates are distinguished. By Proposition 4 it follows that $N(\mathcal{R})$ has dimension $|\mathcal{A}| - 4 - 1 = 22$; it lies in $\mathbb{R}^{27}$.

$$\left[\begin{array}{ccccccc|cccccc|cccccccccccccc}
0&0&1&0&2&0&3&0&0&1&2&0&3&0&0&1&0&1&2&1&2&1&2&3&2&3&3\\
0&1&0&2&0&3&0&0&1&0&0&3&0&0&1&0&2&1&0&2&1&3&2&1&3&2&3\\
\hline
0&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{array}\right]
\begin{array}{l}
\left.\begin{array}{l}\\ \\\end{array}\right\}\text{support}\\
\left.\begin{array}{l}\\ \\\end{array}\right\}\text{Cayley}
\end{array}$$

Implicitization requires eliminating the two parameters to obtain a constraint equation over the symbolic coefficients of the polynomials. Most of the coefficients are specialized except for 3 variables, hence the sought for implicit equation of the surface is trivariate and the projection of $N(\mathcal{R})$ lies in $\mathbb{R}^3$.

TOPCOM [120] needs more than a day and 9GB of RAM to compute 1, 806, 467 regular triangulations of $\mathcal{A}$, corresponding to 29 of the vertices of $N(\mathcal{R})$, and crashes before computing the entire $N(\mathcal{R})$. Our algorithm yields the projected vertices $\{(0,0,1),(0,1,0),\ (1,0,0),(0,0,9),(0,18,0),(18,0,0)\}$ of the 3-dimensional projection of $N(\mathcal{R})$, which is the Newton polytope of the implicit equation, in 30msec. Given this polytope, the implicit equation of the bicubic surface is interpolated in 42 seconds [62]. It is a polynomial of degree 18 containing 715 terms which corresponds exactly to the lattice points contained in the predicted polytope.

Our main contribution is twofold. First, we design an oracle-based algorithm for computing the Newton polytope of $\mathcal{R}$, or of specializations of $\mathcal{R}$. The algorithm utilizes the Beneath-and-Beyond method to compute both vertex (V) and halfspace (H) representations, which are required by the algorithm and may also be relevant for the targeted applications. Its incremental nature implies that we also obtain a triangulation of the polytope, which may be useful for enumerating its lattice points. The complexity is proportional to the number of output vertices and facets; in this sense, the algorithms is output sensitive. The overall cost is asymptotically dominated by computing as many regular triangulations of $\mathcal{A}$ (Theorem 11). We work in the space of the projected $N(\mathcal{R})$ and revert to the high-dimensional space of $\Sigma(\mathcal{A})$ only if needed. Our algorithm readily extends to computing $\Sigma(\mathcal{A})$, the Newton polytope of the discriminant and, more generally, any polytope that can be efficiently described by a vertex oracle or its orthogonal projection. In particular, it suffices to replace our oracle by the oracle in [122] to obtain a method for computing the discriminant polytope.

Second, we describe an efficient, publicly available implementation based on CGAL [35] and its experimental package `triangulation`. Our method computes instances of 5-, 6- or 7-dimensional polytopes with 35K, 23K or 500 vertices, respectively, in < 2hr. Our code is faster up to dimensions 5 or 6, compared to a method computing $N(\mathcal{R})$ via tropical geometry, implemented in the `Gfan` library [86]. In higher dimensions `Gfan` seems to perform better although neither implementation can compute enough instances for a fair comparison. Our code, in the critical step of computing the convex hull of the resultant polytope, uses `triangulation`. On our instances, `triangulation`, compared to state-of-the-art software `lrs`, `cdd`, and `polymake`, is the fastest together with `polymake`. We factor out repeated computation by reducing the bulk of our work to a sequence of determinants: this is often the case in high-dimensional geometric computing. Here, we exploit the nature of our problem and matrix structure to capture the similarities of the predicates, and hash the computed minors which are needed later, to speedup subsequent determinants. A variant of our algorithm computes successively tighter inner and outer approximations: when these polytopes have, respectively, 90% and 105% of the true volume, runtime is reduced up to 25 times. This may lead to an approximation algorithm.

Previous work. Sparse (or toric) elimination theory was introduced in [74]. They show that $N(\mathcal{R})$, for two univariate polynomials with k_0+1, k_1+1 monomials, has $\binom{k_0+k_1}{k_0}$ vertices and, when both $k_i \geq 2$, it has k_0k_1+3 facets. In Section 6 of [131] is proven that $N(\mathcal{R})$ is 1-dimensional if and only if $|A_i| = 2$, for all i, the only planar $N(\mathcal{R})$ is the triangle, whereas the only 3-dimensional ones are the tetrahedron, the square-based pyramid, and the resultant polytope of two univariate trinomials; we compute an affinely isomorphic instance of the latter (Figure 2.2(b)) as the resultant polytope of three bivariate polynomials. Following Theorem 6.2 of [131], the 4-dimensional polytopes include the 4-simplex, some $N(\mathcal{R})$ obtained by pairs of univariate polynomials, and those of 3 trinomials, which have been investigated with our code in [47]. The maximal (in terms of number of vertices) such polytope we have computed has f-vector $(22, 66, 66, 22)$ (Figure 2.2(c)). Furthermore, Table 2.2 presents some typical f-vectors of $4, 5, 6$ dimensional projections of resultant polytopes.

A lower bound on the volume of the Newton polytope of the discriminant polynomial that refutes a conjecture in algebraic geometry is presented in [117].

A direct approach for computing the vertices of $N(\mathcal{R})$ might consider all vertices of $\Sigma(\mathcal{A})$ since the vertices of the former are equivalence classes over the vertices of the latter. Its complexity grows with the number of vertices of $\Sigma(\mathcal{A})$, hence is impractical (Example 1).

The computation of secondary polytopes has been efficiently implemented in TOPCOM [120], which has been the reference software for computing regular or all triangulations. The software builds a search tree with flips as edges over the vertices of $\Sigma(A)$. This approach is limited by space usage. To address this, reverse search was proposed [83], but the implementation cannot compete with TOPCOM. The approach based on computing $\Sigma(\mathcal{A})$ is not efficient for computing $N(\mathcal{R})$. For instance, in implicitizing parametric surfaces with up to 100 terms, which includes all common instances in geometric modeling, we compute the Newton polytope of the equations in less than 1sec, whereas $\Sigma(\mathcal{A})$ is intractable (see e.g. Example 1).

In [109] they describe all Minkowski summands of $\Sigma(A)$. In [110] is defined an equivalence class over $\Sigma(A)$ vertices having the same mixed cells. The classes map in a many-to-one fashion to resultant vertices; our algorithm exploits a stronger equivalence relationship.

Tropical geometry is a polyhedral analogue of algebraic geometry and can be viewed as generalizing sparse elimination theory. It gives alternative ways of recovering resultant polytopes [86] and Newton polytopes of implicit equations [132]. See Section 2.5 for comparisons of the software in [86], called Gfan, with our software. In [122], tropical geometry is used to define vertex oracles for the Newton polytope of the discriminant polynomial.

In [82] there is a general implementation of a Beneath-and-Beyond based procedure which reconstructs a polytope given by a vertex oracle. This implementation, as reported in [86], is outperformed by Gfan, especially in dimensions higher than 5.

As is typical in computational geometry, the practical bottleneck is in computing determinantal predicates. For determinants, the record bit complexity is $O(n^{2.697})$ [90], while more specialized methods exist for the sign of general determinants, e.g. [27]. These results are relevant for higher dimensions and do not exploit the structure of our determinantal predicates, nor the fact that we deal with sequences of determinants whose matrices are not very different (this is formalized and addressed in Section 2.4). We compared linear algebra libraries LinBox [48] and Eigen [79], which seem most suitable in dimension greater than 100 and medium to high dimensions, respectively, whereas CGAL provides the most efficient determinant computation for the dimensions to which we focus.

The roadmap of the chapter follows: Section 2.2 describes the combinatorics of resultants, and the following section presents our algorithm. Section 2.4 overcomes the bottleneck of Orientation predicates. Section 2.5 discusses the implementation, experiments, and comparison with other software. We conclude with future work.

A preliminary version containing most of the presented results appeared in [59]. This extended version contains a more detailed presentation of the background theory of resultants, applications and examples, a more complete account of previous work, omitted proofs, an improved description of the approximation algorithm, an extended version of the hashing determinants method, and more experimental results.

2.2 Resultant polytopes and their projections

We introduce tools from combinatorial geometry [101, 140] to describe resultants [74, 44]. We shall denote by $\mathrm{vol}(\cdot) \in \mathbb{R}$ the normalized Euclidean volume, $(\mathbb{R}^m)^\times$ the linear m-dimensional functionals, $\mathrm{Aff}(\cdot)$ the affine hull, and $\mathrm{conv}(\cdot)$ the convex hull.

Let $\mathcal{A} \subset \mathbb{R}^d$ be a pointset whose convex hull is of dimension d. For any triangulation T of $\mathcal{A}$, define vector $\phi_T \in \mathbb{R}^{|\mathcal{A}|}$ with coordinate

$$\phi_T(a) = \sum_{\sigma \in T : a \in \sigma} \mathrm{vol}(\sigma), \qquad a \in \mathcal{A}, \tag{2.2}$$

summing over all simplices σ of T having a as a vertex; $\Sigma(\mathcal{A})$ is the convex hull of ϕ_T for all triangulations T. Let $\mathcal{A}^w$ denote pointset $\mathcal{A}$ lifted to $\mathbb{R}^{d+1}$ via a generic lifting function w in $(\mathbb{R}^{|\mathcal{A}|})^\times$. *Regular triangulations* of $\mathcal{A}$ are obtained by projecting the upper (or lower) hull of $\mathcal{A}^w$ back to $\mathbb{R}^d$.

Proposition 1. [[74]] *The vertices of $\Sigma(\mathcal{A})$ correspond to the regular triangulations of $\mathcal{A}$, while its face lattice corresponds to the poset of regular polyhedral subdivisions of $\mathcal{A}$, ordered by refinement. A lifting vector produces a regular triangulation T (resp. a regular polyhedral subdivision of $\mathcal{A}$) if and only if it lies in the normal cone of vertex ϕ_T (resp. of the corresponding face) of $\Sigma(\mathcal{A})$. The dimension of $\Sigma(\mathcal{A})$ is $|\mathcal{A}| - d - 1$.*

Let $A_0, \dots, A_n$ be subsets of $\mathbb{Z}^n$, $P_0, \dots, P_n \subset \mathbb{R}^n$ their convex hulls, and $P = P_0 + \cdots + P_n$ their Minkowski sum. A *Minkowski (maximal) cell* of P is any full-dimensional convex polytope $B = \sum_{i=0}^{n} B_i$, where each B_i is a convex polytope with vertices in A_i. Minkowski cells $B, B' = \sum_{i=0}^{n} B'_i$ intersect properly when $B_i \cap B'_i$ is a face of both and their Minkowski sum descriptions are compatible, i.e. coincide on the common face. A *mixed subdivision* of P is any family of Minkowski cells which partition P and intersect properly. A Minkowski cell is *i-mixed* or *v_i-mixed*, if it is the Minkowski sum of n one-dimensional segments from P_j, $j \neq i$, and some vertex $v_i \in P_i$. In the sequel we shall call a Minkowski cell, simply cell.

Mixed subdivisions contain *faces* of all dimensions between 0 and n, the maximum dimension corresponding to cells. Every face of a mixed subdivision of P has a unique description as Minkowski sum of $B_i \subset P_i$. A mixed subdivision is *regular* if it is obtained as the projection of the upper (or lower) hull of the Minkowski sum of lifted polytopes $P_i^{w_i} := \{(p_i, w_i(p_i)) \mid p_i \in P_i\}$, for lifting $w_i : P_i \to \mathbb{R}$. If the lifting function $w := (w_0 \dots, w_n)$ is sufficiently generic, then the mixed subdivision is *tight*, and $\sum_{i=0}^{n} \dim B_i = \dim \sum_{i=0}^{n} B_i$, for every cell. Given $A_0, \dots, A_n$ and the affine basis $\{e_0, \dots, e_n\}$ of $\mathbb{R}^n$, we define the Cayley pointset $\mathcal{A} \subset \mathbb{Z}^{2n}$ as in equation (2.1).

Proposition 2. [Cayley trick, [74]] *There exist bijections between: the regular tight mixed subdivisions of P and the regular triangulations of $\mathcal{A}$; the tight mixed subdivisions of P and the triangulations of $\mathcal{A}$; the mixed subdivisions of P and the polyhedral subdivisions of $\mathcal{A}$.*

The family $A_0, \dots, A_n \subset \mathbb{Z}^n$ is *essential* if they jointly affinely span $\mathbb{Z}^n$ and every subset of cardinality $j, 1 \leq j < n$, spans a space of dimension greater than or equal to j. It is straightforward to check this property algorithmically and, if it does not hold, to find an essential subset [131]. In the sequel, the input $A_0, \dots, A_n \subset \mathbb{Z}^n$ is supposed to be essential. Given a finite $A \subset \mathbb{Z}^n$, we denote by $\mathbb{C}^A$ the space of all Laurent polynomials of the form $\sum_{a \in A} c_a x^a$, $c_a \neq 0$, $x = (x_1, \dots, x_n)$. Similarly, given $A_0, \dots, A_n \subset \mathbb{Z}^n$ we denote by $\prod_{i=0}^n \mathbb{C}^{A_i}$ the space of all systems of polynomials

$$f_0 = f_1 = \cdots = f_n = 0, \tag{2.3}$$

where $\sum_{a \in A_i} c_{i,a} x^a$, $c_{i,a} \neq 0$. The vector of all coefficients $(\dots, c_{i,a}, \dots)$ of (2.3) defines a point in $\prod_{i=0}^n \mathbb{C}^{A_i}$. Let $Z \subset \prod_{i=0}^n \mathbb{C}^{A_i}$ be the set of points corresponding to systems (2.3) which have a solution in $(\mathbb{C}^*)^n$, and let $\overline{Z}$ be its closure. $\overline{Z}$ is an irreducible variety defined over $\mathbb{Q}$.

Definition 1. If $\operatorname{codim}(\overline{Z}) = 1$, then the *sparse (or toric) resultant* of the system of polynomials (2.3) is the unique (up to sign) polynomial $\mathcal{R}$ in $\mathbb{Z}[c_{i,a} : i = 0, \dots, n, a \in A_i]$, which vanishes on $\overline{Z}$. If $\operatorname{codim}(\overline{Z}) > 2$, then $\mathcal{R} = 1$.

The resultant offers a solvability condition from which x has been eliminated, hence is also known as the eliminant. For $n = 1$, it is named after Sylvester. For linear systems, it equals the determinant of the $(n+1) \times (n+1)$ coefficient matrix. The discriminant of a polynomial $F(x_1, \dots, x_n)$ is given by the resultant of $F, \partial F/\partial x_1, \dots, \partial F/\partial x_n$.

The Newton polytope $N(\mathcal{R})$ of the resultant is a lattice polytope called the *resultant polytope*. The resultant has $|\mathcal{A}| = \sum_{i=0}^n |A_i|$ variables, hence $N(\mathcal{R})$ lies in $\mathbb{R}^{|\mathcal{A}|}$, though it is of smaller dimension (Proposition 4). The monomials corresponding to vertices of $N(\mathcal{R})$ are the extreme resultant monomials.

Proposition 3. [[74, 131]] *For a sufficiently generic lifting function $w \in (\mathbb{R}^{|\mathcal{A}|})^\times$, the w-extreme monomial of $\mathcal{R}$, whose exponent vector maximizes the inner product with w, equals*

$$\pm \prod_{i=0}^{n} \prod_{\sigma} c_{i,v_i}^{\operatorname{vol}(\sigma)}, \tag{2.4}$$

where σ ranges over all v_i-mixed cells of the regular tight mixed subdivision S of P induced by w, and c_{i,v_i} is the coefficient of the monomial x^{v_i} in f_i.

Let T be the regular triangulation corresponding, via the Cayley trick, to S, and $\rho_T \in \mathbb{N}^{|\mathcal{A}|}$ the exponent of the w-extreme monomial. For simplicity we shall denote by σ, both a cell of S and its corresponding simplex in T. Then,

$$\rho_T(a) = \sum_{\substack{a-\text{mixed} \\ \sigma \in T : a \in \sigma}} \text{vol}(\sigma) \in \mathbb{N}, \qquad a \in \mathcal{A}, \tag{2.5}$$

where simplex σ is a-mixed if and only if the corresponding cell is a-mixed in S. Note that, $\rho_T(a) \in \mathbb{N}$, since it is a sum of volumes of mixed simplices $\sigma \in T$, and each of these volumes is equal to the *mixed volume*[44] of a set of *lattice* polytopes, the Minkowksi summands of the corresponding $\sigma \in S$. In particular, assuming that $\sigma \in S$ is i-mixed, it can be written as $\sigma = \sigma_0 + \cdots + \sigma_n$, $\sigma_j \subseteq A_j$, $j = 0, \ldots, n$, and $\text{vol}(\sigma) = MV(\sigma_0, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$, where MV denotes the mixed volume function which is integer valued for lattice polytopes [44]. Now, $N(\mathcal{R})$ is the convex hull of all ρ_T vectors [74, 131].

Proposition 3 establishes a many-to-one surjection from regular triangulations of $\mathcal{A}$ to regular tight mixed subdivisions of P, or, equivalently, from vertices of $\Sigma(\mathcal{A})$ to those of $N(\mathcal{R})$. One defines an *equivalence relationship* on all regular tight mixed subdivisions, where equivalent subdivisions yield the same vertex in $N(\mathcal{R})$. Thus, equivalent vertices of $\Sigma(\mathcal{A})$ correspond to the same resultant vertex. Consider $w \in (\mathbb{R}^{|\mathcal{A}|})^\times$ lying in the union of outer-normal cones of equivalent vertices of $\Sigma(\mathcal{A})$. They correspond to a resultant vertex whose outer-normal cone contains w; this defines a w-extremal resultant monomial. If w is non-generic, it specifies a sum of extremal monomials in $\mathcal{R}$, i.e. a face of $N(\mathcal{R})$. The above discussion is illustrated in Figure 2.2(a),(b).

Proposition 4. [[74]] *$N(\mathcal{R})$ is a Minkowski summand of $\Sigma(\mathcal{A})$, and both $\Sigma(\mathcal{A})$ and $N(\mathcal{R})$ have dimension $|\mathcal{A}| - 2n - 1$.*

Let us describe the $2n + 1$ hyperplanes in whose intersection lies $N(\mathcal{R})$. For this, let M be the $(2n + 1) \times |\mathcal{A}|$ matrix whose columns are the points in the A_i, where each $a \in A_i$ is followed by the i-th unit vector in $\mathbb{N}^{n+1}$. Then, the inner product of any coordinate vector of $N(\mathcal{R})$ with row i of M is: constant, for $i = 1, \ldots, n$, and known, and depends on i, for $i = n + 1, \ldots, 2n + 1$, see Prop. 7.1.11 of [74]. This implies that one obtains an isomorphic polytope when projecting $N(\mathcal{R})$ along $2n + 1$ points in $\mathcal{A}$ which affinely span $\mathbb{R}^{2n}$; this is possible because of the assumption of essential family. Having computed the projection, we obtain $N(\mathcal{R})$ by computing the missing coordinates as the solution of a linear system:

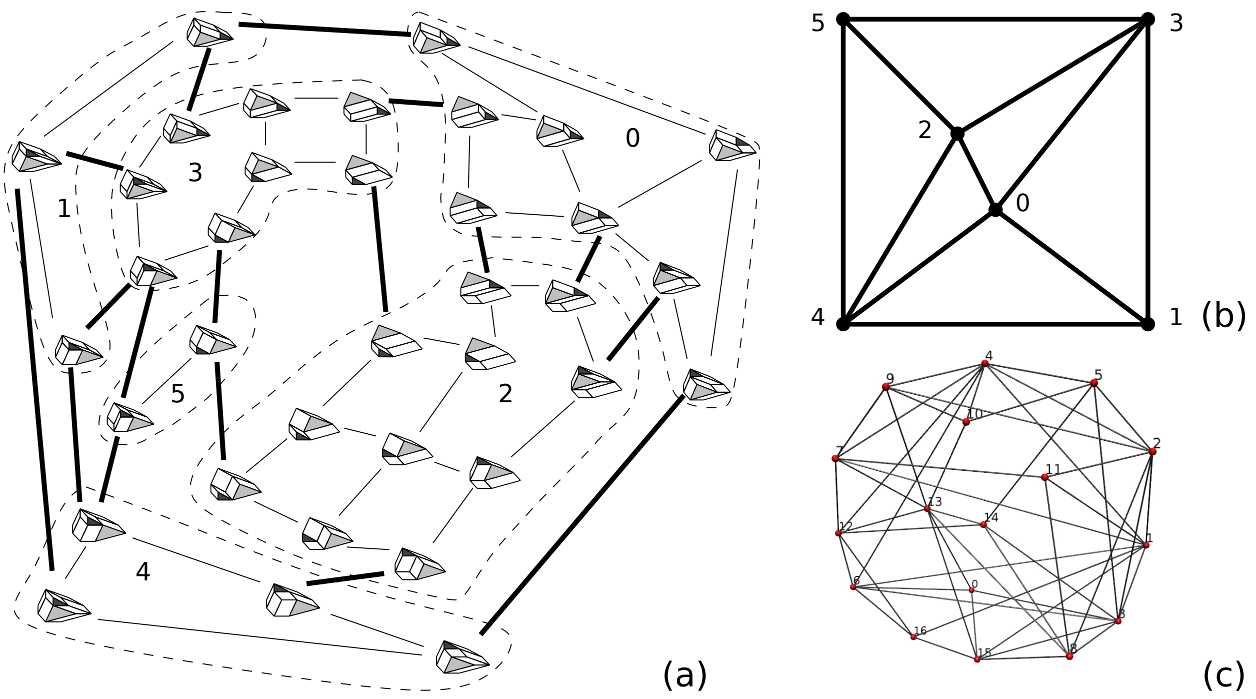


Figure 2.2: Example of secondary and resultant polytopes: (a) The secondary polytope $\Sigma(\mathcal{A})$ of two triangles (dark, light grey) and one segment $A_0 = \{(0,0),(1,2),(4,1)\}$, $A_1 = \{(0,1),(1,0)\}$, $A_2 = \{(0,0),(0,1),(2,0)\}$, where $\mathcal{A}$ is defined as in Equation 2.1; vertices correspond to mixed subdivisions of the Minkowski sum $A_0 + A_1 + A_2$ and edges to flips between them (b) $N(\mathcal{R})$, whose vertices correspond to the dashed classes of $\Sigma(\mathcal{A})$. Bold edges of $\Sigma(\mathcal{A})$, called cubical flips, map to edges of $N(\mathcal{R})$ (c) 4-dimensional $N(\mathcal{R})$ of 3 generic trinomials with f-vector $(22, 66, 66, 22)$; figure made with `polymake`.

we write the aforementioned inner products as $M[X\,V]^T = C$, where C is a known matrix and $[X\,V]^T$ is a transposed $(2n+1) \times u$ matrix, expressing the partition of the coordinates to unknown and known values, where u is the number of $N(\mathcal{R})$ vertices. If the first $2n+1$ columns of M correspond to specialized coefficients, $M = [M_1\,M_2]$, where submatrix M_1 is of dimension $2n+1$ and invertible, hence $X = M_1^{-1}(C - M_2 B)$.

We compute some orthogonal projection of $N(\mathcal{R})$, denoted Π, in $\mathbb{R}^m$:

$$\pi : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^m : N(\mathcal{R}) \to \Pi,\ m \leq |\mathcal{A}|.$$

By reindexing, this is the subspace of the first m coordinates, so $\pi(\rho) = (\rho_1, \ldots, \rho_m)$. It is possible that none of the coefficients c_{ij} is specialized, hence $m = |\mathcal{A}|$,

π is trivial, and $\Pi = N(\mathcal{R})$. Assuming the specialized coefficients take sufficiently generic values, Π is the Newton polytope of the corresponding specialization of $\mathcal{R}$. The following is used for preprocessing.

Lemma 5. [[86] Lemma 3.20] *If $a_{ij} \in A_i$ corresponds to a specialized coefficient of f_i, and lies in the convex hull of the other points in A_i corresponding to specialized coefficients, then removing a_{ij} from A_i does not change the Newton polytope of the specialized resultant.*

We focus on three applications. First, we interpolate the resultant in all coefficients, thus illustrating an alternative method for computing resultants.

Example 2. Let $f_0 = a_2x^2+a_1x+a_0$, $f_1 = b_1x^2+b_0$, with supports $A_0 = \{2, 1, 0\}$, $A_1 = \{1, 0\}$. Their (Sylvester) resultant is a polynomial in a_2, a_1, a_0, b_1, b_0. Our algorithm computes its Newton polytope with vertices $(0, 2, 0, 1, 1)$, $(0, 0, 2, 2, 0)$, $(2, 0, 0, 0, 2)$; it contains 4 lattice points, corresponding to 4 potential resultant monomials $a_1^2b_1b_0$, $a_0^2b_1^2$, $a_2a_0b_1b_0$, $a_2^2b_0^2$. Knowing these potential monomials, to interpolate the resultant, we need 4 points $(a_0, a_1, a_2, b_0, b_1)$ for which the system $f_0 = f_1 = 0$ has a solution. For computing these points we use the parameterization of resultants in [92], which yields: $a_2 = (2t_1 + t_2)t_3^2t_4$, $a_1 = (-2t_1 - 2t_2)t_3t_4$, $a_0 = t_2t_4$, $b_1 = -t_1t_3^2t_5$, $b_0 = t_1t_5$, where the t_i's are parameters. We substitute these expressions to the monomials, evaluate at 4 sufficiently random t_i's, and obtain a matrix whose kernel vector $(1, 1, -2, 1)$ yields $\mathcal{R} = a_1^2b_1b_0 + a_0^2b_1^2 - 2a_2a_0b_1b_0 + a_2^2b_0^2$.

Second, consider system solving by the rational univariate representation of roots [14]. Given $f_1, \dots, f_n \in \mathbb{C}[x_1, \dots, x_n]$, define an overconstrained system by adding $f_0 = u_0 + u_1x_1 + \cdots + u_nx_n$ with symbolic u_i's. Let coefficients $c_{ij}, i \geq 1$, take specific values, and suppose that the roots of $f_1 = \cdots = f_n = 0$ are isolated, denoted $r_i = (r_{i1}, \dots, r_{in})$. Then the u-resultant is $\mathcal{R}_u = a \prod_{r_i}(u_0 + u_1r_{i1} + \cdots + u_nr_{in})^{m_i}$, $a \in \mathbb{C}^*$, where m_i is the multiplicity of r_i. Computing $\mathcal{R}_u$ is the bottleneck; our method computes (a superset of) $N(\mathcal{R}_u)$.

Example 3. Let $f_1 = x_1^2 + x_2^2 - 4$, $f_2 = x_1 - x_2 + 2$, and $f_0 = u_0 + u_1x_1 + u_2x_2$. Our algorithm computes a polygon with vertices $\{(2, 0, 0), (0, 2, 0), (0, 0, 2)\}$, which contains $N(\mathcal{R}_u) = \mathrm{conv}(\{(2, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1)\})$. The coefficient specialization is not generic, hence $N(\mathcal{R}_u)$ is strictly contained in the computed polygon.

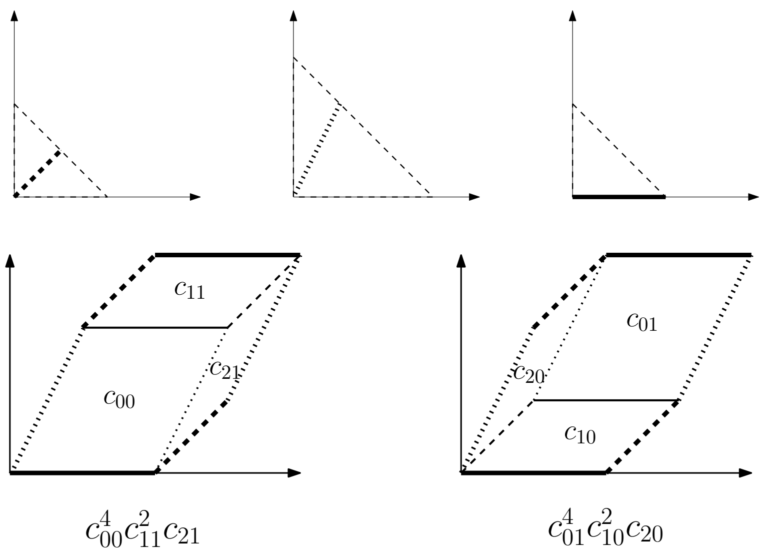


Figure 2.3: The supports A_0, A_1, A_2 of Example 4, their Newton polytopes (segments) and the two mixed subdivisions of their Minkowski sum.

Proceeding as in Example 2, $\mathcal{R}_u = 2u_0^2 + 4u_0u_1 - 4u_0u_2 - 8u_1u_2$, which factors as $2(u_0 + 2u_1)(u_0 - 2u_2)$.

The last application comes from geometric modeling, where $y_i = f_i(x)$, $i = 0, \dots, n$, $x = (x_1, \dots, x_n) \in \Omega \subset \mathbb{R}^n$, defines a parametric hypersurface. Many applications require the equivalent implicit representation $F(y_1, \dots, y_n) = 0$. This amounts to eliminating x, so it is crucial to compute the resultant when coefficients are specialized except the y_i's. Our approach computes a polytope that contains the Newton polytope of F, thus reducing implicitization to interpolation [62, 61]. In particular, we compute the polytope of surface equations within 1sec, assuming ≤ 100 terms in parametric polynomials, which includes all common instances in geometric modeling.

Example 4. Let us see how the above computation can serve in implicitization. Consider the surface given by the polynomial parameterization

$$(y_1, y_2, y_3) = (x_1x_2, x_1x_2^2, x_1^2).$$

For polynomials $f_0 := c_{00} - c_{01}x_1x_2$, $f_1 := c_{10} - c_{11}x_1x_2^2$, $f_2 := c_{20} - c_{21}x_1^2$ with supports $A_0 = \{(0,0),(1,2)\}, A_1 = \{(0,0),(1,2)\}$ and $A_2 = \{(0,0),(2,0)\}$. The resultant polytope is a segment in $\mathbb{R}^6$ with endpoints $(4,0,0,2,0,1)$, $(0,4,2,0,1,0)$ and, actually, $\mathcal{R} = -c_{00}^4c_{11}^2c_{21} + c_{01}^4c_{10}^2c_{20}$. The supports and the two mixed subdivisions corresponding to the vertices of $N(\mathcal{R})$ are illustrated in Figure 2.3. Specializing the symbolic coefficients of the polynomials as:

$$(c_{00}, c_{01}, c_{10}, c_{11}, c_{20}, c_{21}) \mapsto (y_1, -1, y_2, -1, y_3, -1)$$

yields the vertices of the implicit polytope: $(4, 0, 0), (0, 2, 1)$, which our algorithm can compute directly. The implicit equation of the surface turns out to be $-y_1^4 + y_2^2 y_3$.

2.3 Algorithms and complexity

This section analyzes our exact and approximate algorithms for computing orthogonal projections of polytopes whose vertices are defined by an *oracle*. This oracle computes a vertex of the polytope which is extremal in a given direction w. If there are more than one such vertices the oracle returns exactly one of these. Moreover, we define such an oracle for the vertices of orthogonal projections Π of $N(\mathcal{R})$ which results in algorithms for computing Π while avoiding computing $N(\mathcal{R})$. Finally, we analyze the asymptotic complexity of these algorithms.

Given a pointset V, reg_subdivision(V, ω) computes the regular subdivision of its convex hull by projecting the upper hull of V lifted by ω, and conv(V) computes the H-representation of the convex hull of V. The oracle VTX($\mathcal{A}$, w, π) computes a point in $\Pi = \pi(N(\mathcal{R}))$, extremal in the direction $w \in (\mathbb{R}^m)^\times$. First it adds to w an infinitesimal symbolic perturbation vector, thus obtaining w_p. Then calls reg_subdivision($\mathcal{A}, \widehat{w_p}$), $\widehat{w_p} = (w_p, \vec{0}) \in (\mathbb{R}^{|\mathcal{A}|})^\times$ that yields a regular triangulation T of $\mathcal{A}$, since w_p is generic, and finally returns $\pi(\rho_T)$. It is clear that the triangulation T constructed by VTX($\cdot$) is regular and corresponds to some secondary vertex ϕ_T which maximizes the inner product with $\widehat{w_p}$. Since the perturbation is arbitrarily small, both ϕ_T, ρ_T also maximize the inner product with $\widehat{w} = (w, \vec{0}) \in (\mathbb{R}^{|\mathcal{A}|})^\times$.

We use perturbation to avoid computing non-vertex points on the boundary of Π. The perturbation can be implemented in VTX($\cdot$), without affecting any other parts of the algorithm, either by case analysis or by a method of symbolic perturbation. In practice, our implementation does avoid computing non-vertex points on the boundary of Π by computing a refinement of the subdivision obtained by calling reg_subdivision($\mathcal{A}, \widehat{w}$). This refinement is implemented in `triangulation` by computing a placing triangulation with a random insertion order [22]

(Section 2.5).

Lemma 6. *All points computed by* VTX$(\cdot)$ *are vertices of Π.*

Proof. Let $v = \pi(\rho_T) = \text{VTX}(\mathcal{A}, w, \pi)$. We first prove that v lies on $\partial\Pi$. The point ρ_T of $N(\mathcal{R})$ is a Minkowski summand of the vertex ϕ_T of $\Sigma(\mathcal{A})$ extremal with respect to $\widehat{w}$, hence ρ_T is extremal with respect to $\widehat{w}$. Since $\widehat{w}$ is perpendicular to projection π, ρ_T projects to a point in $\partial\Pi$. The same argument implies that every vertex ϕ'_T, where T' is a triangulation refining the subdivision produced by $\widehat{w}$, corresponds to a resultant vertex $\rho_{T'}$ such that $\pi(\rho_{T'})$ lies on a face of Π. This is actually the same face on which $\pi(\rho_T)$ lies. Hence $\rho_{T'}$ also lies on $\partial\Pi$.

Now we prove that v is a vertex of Π by showing that it does not lie in the relative interior of a face of Π. Let w be such that the face f of $N(\mathcal{R})$ extremal with respect to $\widehat{w}$ contains a vertex ρ_T which projects to relint$(\pi(f))$, where relint$(\cdot)$ denotes relative interior. However, f will not be extremal with respect to $\widehat{w_p}$ and since VTX$(\mathcal{A}, w, \pi)$ uses the perturbed vector w_p, it will never compute a vertex of $N(\mathcal{R})$ whose projection lies inside a face of Π. □

The *initialization algorithm* computes an inner approximation of Π in both V- and H-representations (denoted Q, Q^H, respectively), and triangulated. First, it calls VTX$(\mathcal{A}, w, \pi)$ for $w \in W \subset (\mathbb{R}^m)^\times$; the set W is either random or contains, say, vectors in the $2m$ coordinate directions. Then, it updates Q by adding VTX$(\mathcal{A}, w, \pi)$ and VTX$(\mathcal{A}, -w, \pi)$, where w is normal to hyperplane $H \subset \mathbb{R}^m$ containing Q, as long as either of these points lies outside H. Since every new vertex lies outside the affine hull of the current polytope Q, all polytopes produced are simplices. We stop when these points do no longer increase dim(Q).

Lemma 7. *The initialization algorithm computes $Q \subseteq \Pi$ such that* $\dim(Q) = \dim(\Pi)$.

Proof. Suppose that the initialization algorithm computes a polytope $Q' \subset \Pi$ such that $\dim(Q') < m$. Then there exists vertex $v \in \Pi$, $v \notin \text{Aff}(Q')$ and vector $w \in (\mathbb{R}^m)^\times$ perpendicular to Aff(Q'), such that w belongs to the normal cone of v in Π and $\dim(\text{Aff}(Q' \cup v)) > \dim Q'$. This is a contradiction, since such a w would have been computed as VTX$(\mathcal{A}, w, \pi)$ or VTX$(\mathcal{A}, -w, \pi)$, where w is normal to the hyperplane H containing Q'. □

Incremental Algorithm 1 computes both V- and H-representations of Π and a triangulation of Π, given an inner approximation Q, Q^H of Π computed at the initialization. A hyperplane H is called *legal* if it is a supporting hyperplane to a facet of Π, otherwise it is called *illegal*. At every step of Algorithm 1, we compute $v = \text{VTX}(\mathcal{A}, w, \pi)$ for a supporting hyperplane H of a facet of Q with normal w. If $v \notin H$, it is a new vertex thus yielding a tighter *inner approximation* of Π by inserting it to Q, i.e. $Q \subset \text{conv}(Q \cup v) \subseteq \Pi$. This happens when the preimage $\pi^{-1}(f) \subset N(\mathcal{R})$ of the facet f of Q defined by H, is not a Minkowski summand of a face of $\Sigma(\mathcal{A})$ having normal $\widehat{w}$. Otherwise, there are two cases: either $v \in H$ and $v \in Q$, thus the algorithm simply decides hyperplane H is legal, or $v \in H$ and $v \notin Q$, in which case the algorithm again decides H is legal but also inserts v to Q.

The algorithm computes Q^H from Q, then iterates over the new hyperplanes to either compute new vertices or decide they are legal, until no increment is possible, which happens when all hyperplanes are legal. Algorithm 1 ensures that each normal w to a hyperplane supporting a facet of Q is used only *once*, by storing all used w's in a set W. When a new normal w is created, the algorithm checks if $w \notin W$, then calls $\text{VTX}(\mathcal{A}, w, \pi)$ and updates $W \leftarrow W \cup w$. If $w \in W$ then the same or a parallel hyperplane has been checked in a previous step of the algorithm. It is straightforward that w can be safely ignored; Lemma 8 formalizes the latter case.

Lemma 8. *Let H' be a hyperplane supporting a facet constructed by Algorithm 1, and $H \neq H'$ an illegal hyperplane at a previous step. If H', H are parallel then H' is legal.*

Proof. Let w, w' be the outer normal vectors of the facets supported by H, H' respectively. If H, H' are parallel then $v = \text{VTX}(\mathcal{A}, w, \pi)$ maximizes the inner product with w' in Q which implies that hyperplane H' is legal. □

The next lemma formulates the termination criterion of our algorithm.

Lemma 9. *Let $v = \text{VTX}(\mathcal{A}, w, \pi)$, where w is normal to a supporting hyperplane H of Q, then $v \notin H$ if and only if H is not a supporting hyperplane of Π.*

Proof. Let $v = \pi(\rho_T)$, where T is a triangulation refining subdivision S in $\text{VTX}(\cdot)$. It is clear that, since $v \in \partial\Pi$ is extremal with respect to w, if $v \notin H$ then H cannot

Algorithm 1: ComputeΠ $(A_0,\dots,A_n,\pi)$

```
Input  : essential $A_0,\dots,A_n \subset \mathbb{Z}^n$ processed by Lemma 5,
         projection $\pi : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^m$,
         H-, V-repres. $Q^H, Q$; triang. $T_Q$ of $Q \subseteq \Pi$.
Output : H-, V-repres. $Q^H, Q$; triang. $T_Q$ of $Q = \Pi$.

$\mathcal{A} \leftarrow \bigcup_0^n (A_i \times e_i)$   // Cayley trick
$\mathcal{H}_{illegal} \leftarrow \emptyset$
foreach $H \in Q^H$ do $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H\}$
while $\mathcal{H}_{illegal} \neq \emptyset$ do
    select $H \in \mathcal{H}_{illegal}$ and $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H\}$
    $w$ is the outer normal vector of $H$
    $v \leftarrow$ VTX$(\mathcal{A}, w, \pi)$
    if $v \notin H \cap Q$ then
        $Q^H_{temp} \leftarrow$ conv$(Q \cup \{v\})$ // convex hull computation
        foreach $(d-1)$-face $f \in T_Q$ visible from $v$ do
            $T_Q \leftarrow T_Q \cup \{$faces of $conv(f, v)\}$
        foreach $H' \in \{Q^H \setminus Q^H_{temp}\}$ do
            $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H'\}$ // $H'$ separates $Q, v$
        foreach $H' \in \{Q^H_{temp} \setminus Q^H\}$ do
            $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H'\}$ // new hyperplane
        $Q \leftarrow Q \cup \{v\}$
        $Q^H \leftarrow Q^H_{temp}$
return $Q, Q^H, T_Q$
```

be a supporting hyperplane of Π. Conversely, let $v \in H$. By the proof of Lemma 6, every other vertex $\pi(\rho'_T)$ on the face of $N(\mathcal{R})$ is extremal with respect to w, hence lies on H, thus H is a supporting hyperplane of Π. □

We now bound the *complexity* of our algorithm. Beneath-and-Beyond, given a k-dimensional polytope with l vertices, computes its H-representation and a triangulation in $O(k^5 l t^2)$, where t is the number of full-dimensional faces (cells) [87]. Let $|\Pi|, |\Pi^H|$ be the number of vertices and facets of Π.

Lemma 10. *Algorithm 1 executes VTX$(\cdot)$ at most $|\Pi| + |\Pi^H|$ times.*

Proof. The steps of Algorithm 1 increment Q. At every such step, and for each

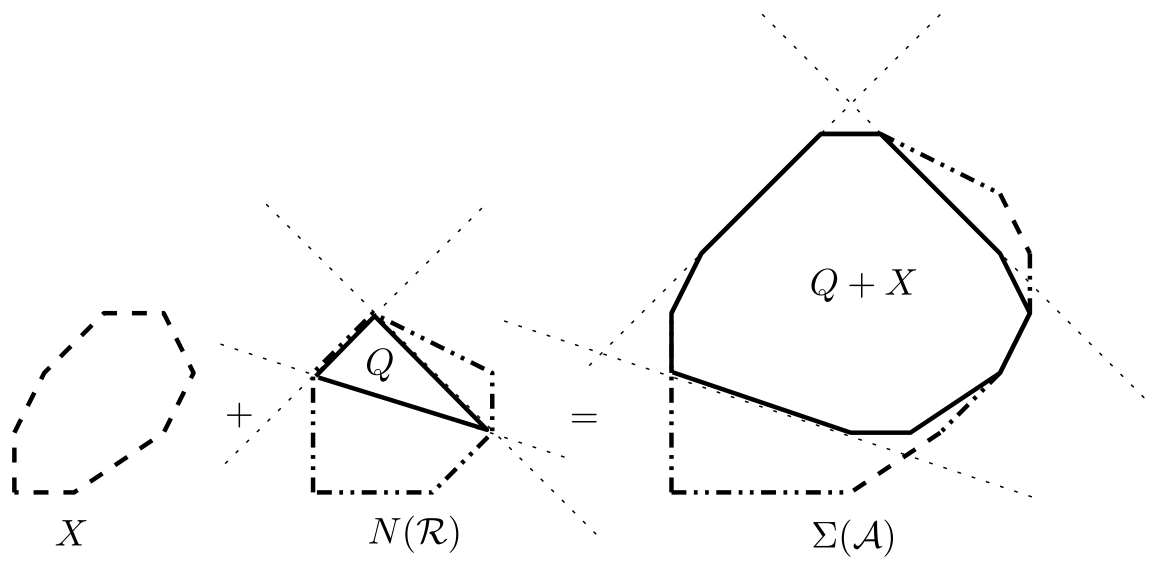


Figure 2.4: Proof sketch for Lemma 10: each illegal hyperplane of Q with normal w, separates the already computed vertices of Π (here equal to $N(\mathcal{R})$) from new ones, extremal with respect to w. X is a polytope such that $X + N(\mathcal{R}) = \Sigma(\mathcal{A})$.

supporting hyperplane H of Q with normal w, the algorithm calls VTX($\cdot$) and computes one vertex of Π, by Lemma 6. If H is illegal, this vertex is unique because H separates the set of (already computed) vertices of Q from the set of vertices of $\Pi \setminus Q$ which are extremal with respect to w, hence, an appropriate translate of H also separates the corresponding sets of vertices of $\Sigma(\mathcal{A})$ (Figure 2.4). This vertex is never computed again because it now belongs to Q. The number of VTX($\cdot$) calls yielding vertices is thus bounded by $|\Pi|$.

For a legal hyperplane of Q, we compute one vertex of Π that confirms its legality; the VTX($\cdot$) call yielding this vertex is accounted for by the legal hyperplane. The statement follows by observing that every normal to a hyperplane of Q is used only once in Algorithm 1 (by the earlier discussion concerning the set W of all used normals). $\square$

Let the size of a triangulation be the number of its cells. Let $s_{\mathcal{A}}$ denote the size of the largest triangulation of $\mathcal{A}$ computed by VTX($\cdot$), and s_{Π} that of Π computed by Algorithm 1. In VTX($\cdot$), the computation of a regular triangulation reduces to a convex hull, computed in $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$; for ρ_T we compute Volume for all cells of T in $O(s_{\mathcal{A}}n^3)$. The overall complexity of VTX($\cdot$) becomes $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$. Algorithm 1 calls, in every step, VTX($\cdot$) to find a point on $\partial\Pi$ and insert it to Q, or to conclude that a hyperplane is legal. By Lemma 10 it executes VTX($\cdot$) as many as $|\Pi|+|\Pi^H|$ times, in $O((|\Pi|+|\Pi^H|)n^5|\mathcal{A}|s_{\mathcal{A}}^2)$, and computes the H-representation of Π in $O(m^5|\Pi|s_{\Pi}^2)$. Now we have, $|\mathcal{A}| \leq (2n+1)s_{\mathcal{A}}$ and as the input $|\mathcal{A}|, m, n$ grows large we can assume that $|\Pi| \gg |\mathcal{A}|$ and thus s_{Π} dominates $s_{\mathcal{A}}$. Moreover, $s_{\Pi}(m+1) \geq |\Pi^H|$. Now, let $\tilde{O}(\cdot)$ imply that polylogarithmic factors are ignored.

Theorem 11. *The time complexity of Algorithm 1 to compute $\Pi \subset \mathbb{R}^m$ is $O(m^5|\Pi|s_{\Pi}^2 + (|\Pi|+|\Pi^H|)n^5|\mathcal{A}|s_{\mathcal{A}}^2)$, which becomes $\tilde{O}(|\Pi|s_{\Pi}^2)$ when $|\Pi| \gg |\mathcal{A}|$.*

This implies our algorithm is output sensitive. Its experimental performance confirms this property, see Section 2.5.

We have proven that oracle VTX($\cdot$) (within our algorithm) has two important properties:

1. Its output is a vertex of the target polytope (Lemma 6).
2. When the direction w is normal to an illegal facet, then the vertex computed by the oracle is computed once (Lemma 10).

The algorithm can easily be generalized to incrementally compute any polytope P if the oracle associated with the problem satisfies property (1); if it satisfies also property (2), then the computation can be done in $O(|P|+|P^H|)$ oracle calls, where $|P|$, $|P^H|$ denotes the number of vertices and number of facets of P, respectively. For example, if the described oracle returns $\pi(\phi_T)$ instead of $\pi(\rho_T)$, it can be used to compute orthogonal projections of secondary polytopes.

The algorithm readily yields an approximate variant: for each supporting hyperplane H, we use its normal w to compute v =VTX($\mathcal{A}, w, \pi$). Instead of computing a convex hull, now simply take the hyperplane parallel to H through v. The set of these hyperplanes defines a polytope $Q_o \supseteq \Pi$, i.e. an *outer approximation* of Π. In particular, at every step of the algorithm, Q and Q_o are an inner and an outer approximation of Π, respectively. Thus, we have an approximation algorithm by stopping Algorithm 1 when vol(Q)/vol(Q_o) achieves a user-defined threshold. Then, vol(Q)/vol(Π) is bounded by the same threshold. Implementing this algorithm yields a speedup of up to 25 times (Section 2.5). It is clear that vol(Q) is available by our incremental convex hull algorithm. However, vol(Q_o) is the critical step; we plan to examine algorithms that update (exactly or approximately) this volume.

When all hyperplanes of Q are checked, knowledge of legal hyperplanes accelerates subsequent computations of Q^H, although it does not affect its worst-case complexity. Specifically, it allows us to avoid checking legal facets against new vertices.

2.4 Hashing of Determinants

This section discusses methods to avoid duplication of computations by exploiting the nature of the determinants appearing in the inner loop of our algorithm. Our algorithm computes many regular triangulations, which are typically dominated by the computation of determinants. A similar technique, using dynamic determinant computations, is used to improve determinantal predicates in incremental convex hull computations [65].

Consider the $2n \times |\mathcal{A}|$ matrix with the points of $\mathcal{A}$ as columns. Define P as the extension of this matrix by adding lifting values $\widehat{w}$ as the last row. We use the Laplace (or cofactor) expansion along the last row for computing the determinant of the square submatrix formed by any $2n+1$ columns of P; without loss of generality, we assume these are the first $2n+1$ columns $a_1, \ldots, a_{2n+1}$. Let $(1, \ldots, 2n+1) \setminus i$ be the vector resulting from removing the i-th element from the vector $(1, \ldots, 2n+1)$ and let $P_{(1,\ldots,2n+1)\setminus i}$ be the $(2n) \times (2n)$ matrix obtained from the $2n$ elements of the columns whose indices are in $(1, \ldots, 2n+1) \setminus i$.

The Orientation predicate is the sign of the determinant of $P^{hom}_{(1,\ldots,2n+2)}$, constructed by columns $a_1, \ldots, a_{2n+2}$ and adding $\vec{1} \in \mathbb{R}^{2n+2}$ as the last row. Computing a regular subdivision is a long sequence of such predicates, varying a_i's on each step. We expand along the next-to-last row, which contains the lifting values, and compute the determinants $|P_{(1,\ldots,2n+2)\setminus i}|$ for $i \in \{1, \ldots, 2n+2\}$. Another predicate is Volume, used by VTX$(\cdot)$. It equals the determinant of $P^{hom}_{(1,\ldots,2n+1)}$, constructed by columns $a_1, \ldots, a_{2n+1}$ and replacing the last row of the matrix by $\vec{1} \in \mathbb{R}^{2n+1}$.

Example 5. Consider the polynomials $f_0 := c_{00} - c_{01}x_1x_2 + c_{02}x_2$, $f_1 := c_{10} - c_{11}x_1x_2^2 +$

$c_{12}x_2^2$ and $f_2 := c_{20} - c_{21}x_1^2 + c_{22}x_2$ and the lifting vector $\widehat{w}$ yielding the matrix P.

$$P = \left[\begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \begin{array}{l} \left.\begin{array}{c} \\ \\ \end{array}\right\} \text{support coordinates} \\ \left.\begin{array}{c} \\ \\ \end{array}\right\} \text{Cayley trick coordinates} \\ \}\, \widehat{w} \end{array}$$

We reduce the computations of predicates to computations of minors of the matrix obtained from deleting the last row of P. Computing an Orientation predicate using Laplace expansion consists of computing $\binom{6}{4} = 15$ minors. On the other hand, if we compute $|P^{hom}_{(1,2,3,4,5,6)}|$, the computation of $|P^{hom}_{(1,2,3,4,5,7)}|$ requires the computation of only $\binom{6}{4} - \binom{5}{4} = 10$ new minors. More interestingly, when given a new lifting $\widehat{w'}$, we compute $|P'^{\,hom}_{(1,2,3,4,5,6)}|$ without computing any new minors.

Our contribution consists in maintaining a hash table with the computed minors, which will be reused at subsequent steps of the algorithm. We store all minors of sizes between 2 and $2n$. For Orientation, they are independent of w and once computed they are stored in the hash table. The main advantage of our scheme is that, for a new w, the only change in P are m (nonzero) coordinates in the last row, hence computing the new determinants can be done by reusing hashed minors. This also saves time from matrix constructions.

Laplace expansion computation of a matrix of size n has complexity $O(n)\sum_{i=1}^{n} L_i$, where L_i is the cost of computing the i-th minor. L_i equals 1 when the i-th minor was precomputed; otherwise, it is bounded by $O\big((n-1)!\big)$. This allows us to formulate the following Lemma.

Lemma 12. *Using hashing of determinants, the complexity of the Orientation and Volume predicates is $O(n)$ and $O(1)$, respectively, if all minors have already been computed.*

Many determinant algorithms modify the input matrix; this makes necessary to create a new matrix and introduces a constant overhead on each minor computation. Computing with Laplace expansion, while hashing the minors of smaller size, performs better than state-of-the-art algorithms, in practice. Experiments in Section 2.5 show that our algorithm with hashed determinants outperforms the version without hash. For $m = 3$ and $m = 4$, we experimentally observed that the speedup factor is between 18 and 100; Figure 2.6(b) illustrates the second case.

The drawback of hashing determinants is the amount of storage, which is in $O(n!)$. The hash table can be cleared at any moment to limit memory consumption, at the cost of dropping all previously computed minors. Finding a policy to clear the hash table according to the number of times each minor was used would decrease the memory consumption, while keeping running times low. Exploring different heuristics, such as using a LRU (least recently used) cache, to choose which minors to drop when freeing memory will be an interesting research subject.

It is possible to exploit the structure of the above $(2n) \times (2n)$ minor matrices. Let M be such a matrix, with columns corresponding to points of $A_0, \ldots, A_n$. After column permutations, we split M into four $n \times n$ submatrices A, B, D, I, where I is the identity matrix and D has at most one 1 in each column. This follows from the fact that the bottom half of every column in M has at most one 1 and the last n rows of M contain at least one 1 each, unless $\det M = 0$, which is easily checked. Now, $\det M = \pm \det(B - AD)$, with AD constructed in $O(n)$. Hence, the computation of $(2n) \times (2n)$ minors is asymptotically equal to computing an $n \times n$ determinant. This only decreases the constant within the asymptotic bound. A simple implementation of this idea is not faster than Laplace expansion in the dimensions that we currently focus. However, this idea should be valuable in higher dimensions.

2.5 Implementation and Experiments

We implemented Algorithm 1 in C++ to compute Π; our code can be obtained from

`http://respol.sourceforge.net`.

All timings shown in this section were obtained on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux.

Our implementation, `respol`, relies on CGAL, using mainly a preliminary version of package `triangulation` [22], for both regular triangulations, as well as for the V- and H-representation of Π. As for hashing determinants, we looked for a hashing function, that takes as input a vector of integers and returns an integer, which minimizes collisions. We considered many different hash functions, including some variations of the well-known FNV hash [66]. We obtained the best results with the implementation of Boost Hash [85], which shows fewer collisions than the other tested functions. We clear the hash table when it contains 10^6 minors. This gives a good tradeoff between efficiency and memory consumption. Last column of Table 2.1 shows that the memory consumption of our algorithm is related to $|A|$ and $\dim(\Pi)$.

We start our experiments by comparing four state-of-the-art exact convex hull packages: `triangulation` implementing [39] and `beneath-and-beyond (bb)` in `polymake` [72]; double description implemented in `cdd` [68]; and `lrs` implementing reverse search [6]. We compute Π, actually extending the work in [7] for the new class of polytopes Π. The `triangulation` package was shown to be faster in computing Delaunay triangulations in ≤ 6 dimensions [22]. The other three packages are run through `polymake`, where we have ignored the time to load the data. We test all packages in an offline version. We first compute the V-representation of Π using our implementation and then we give this as an input to the convex hull packages that compute the H-representation of Π. Moreover, we test `triangulation` by inserting points in the order that Algorithm 1 computes them, while improving the point location of these points since we know by the execution of Algorithm 1 one facet to be removed (online version). The experiments show that `triangulation` and `bb` are faster than `lrs`, which outperforms `cdd`. Furthermore, the online version of `triangulation` is 2.5 times faster than its offline counterpart due to faster point location (Table 2.1, Figure 2.5).

A *placing triangulation* of a set of points is a triangulation produced by the Beneath-and-Beyond convex hull algorithm for some ordering of the points. That is, the algorithm places the points in the triangulation with respect to the ordering. Each point which is going to be placed, is connected to all visible faces of the current triangulation resulting to the construction of new cells. An advantage of `triangulation` is that it maintains a placing triangulation of a polytope in $\mathbb{R}^d$ by storing the $0, 1, d-1, d$ dimensional cells of the triangulation. This is useful when the oracle VTX($\mathcal{A}, w, \pi$) needs to refine the regular subdivision of $\mathcal{A}$ which is obtained by projecting the upper hull of the lifted pointset $\mathcal{A}^{\widehat{w}}$ (Section 2.3). In fact

m	$\|\mathcal{A}\|$	# of Π vertices	time (seconds)						respol Mb
			respol	tr/on	tr/off	bb	cdd	lrs	
3	2490	318	85.03	0.07	0.10	0.07	1.20	0.10	37
4	27	830	15.92	0.71	1.08	0.50	26.85	3.12	46
4	37	2852	97.82	2.85	3.91	2.29	335.23	39.41	64
5	15	510	11.25	2.31	5.57	1.22	47.87	6.65	44
5	18	2584	102.46	13.31	34.25	9.58	2332.63	215.22	88
5	24	35768	4610.31	238.76	577.47	339.05	$>$ 1hr	$>$ 1hr	360
6	15	985	102.62	20.51	61.56	28.22	610.39	146.83	2868
6	19	23066	6556.42	1191.80	2754.30	$>$ 1hr	$>$ 1hr	$>$ 1hr	6693
7	12	249	18.12	7.55	23.95	4.99	6.09	11.95	114
7	17	500	302.61	267.01	614.34	603.12	10495.14	358.79	5258

Table 2.1: Total time and memory consumption of our code (`respol`) and time comparison of online version of `triangulation (tr/on)` and offline versions of all convex hull packages for computing the H-representation of Π.

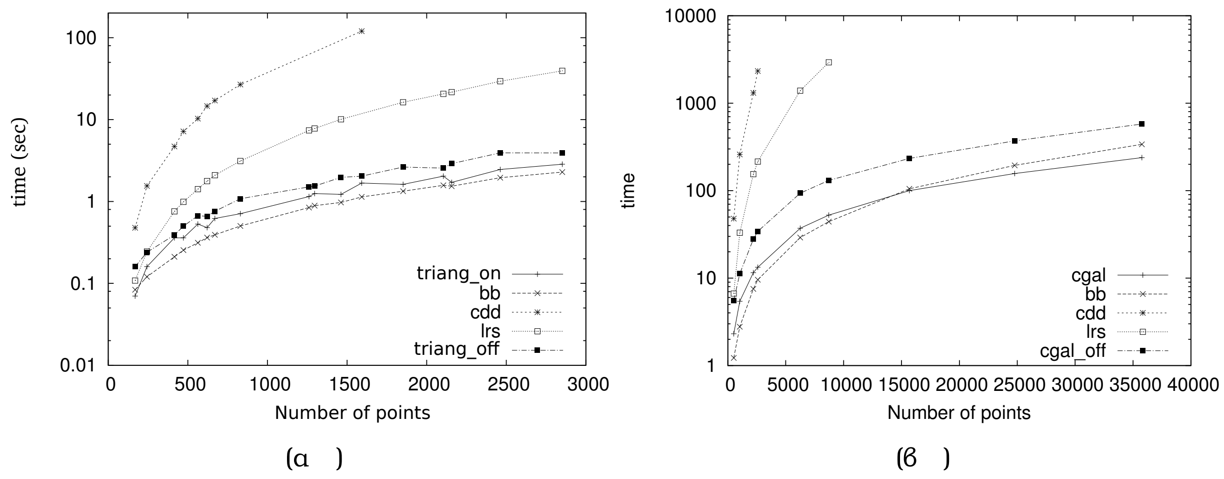


Figure 2.5: Comparison of convex hull packages for 4-dimensional (a) and 5-dimensional (b) Π. `triang_on/triang_off` are the online/offline versions of `triangulation` package (y-axis is in logarithmic scale).

this refinement is attained by a placing triangulation, i.e., by computing the projection of the upper hull of the placing triangulation of $\mathcal{A}^{\widehat{w}}$. This is implemented in two steps:

Step 1. compute the placing triangulation T_0 of the last $|\mathcal{A}| - m$ points with a random insertion order as described in [22] (they all have height zero),

Step 2. place the first m points of $\mathcal{A}^{\widehat{w}}$ in T_0 with a random insertion order [22].

Step 1 is performed only once at the beginning of the algorithm, whereas Step 2 is performed every time we check a new w. The order of placing the points in Step 2 only matters if w is not generic; otherwise, w already produces a triangulation of the m points, so any placing order results in this triangulation.

This is the implemented method; although different from the perturbation in the proof of Lemma 6, it is more efficient because of the reuse of triangulation T_0 in Step 1 above. Moreover, our experiments show that it always validates the two conditions in Section 2.3.

We can formulate this 2-step construction using a single lifting. Let $c > 0$ be a sufficiently large constant, $a_i \in \mathcal{A}$, $q_i \in \mathbb{R}$, $q_i > c q_{i+1}$, for $i = 1, \dots, |\mathcal{A}|$. Define lifting $h : \mathcal{A} \to \mathbb{R}^2$, where $h(a_i) = (w_i, q_i)$, for $i = 1, \dots, m$, and $h(a_i) = (0, q_i)$, for $i = m+1, \dots, |\mathcal{A}|$. Then, projecting the upper hull of $\mathcal{A}^h$ to $\mathbb{R}^{2n}$ yields the triangulation of $\mathcal{A}$ obtained by the 2-step construction.

Fixing the dimension of the triangulation at compile time results in $< 1\%$ speedup. We also tested a kernel that uses the filtering technique based on interval arithmetic from [26] with a similar time speedup. On the other hand, `triangulation` is expected to implement incremental high-dimensional regular triangulations with respect to a lifting, faster than the above method [46]. Moreover, we use a modified version of `triangulation` in order to benefit from our hashing scheme. Therefore, all cells of the triangulated facets of Π have the same normal vector and we use a structure (`STL set`) to maintain the set of unique normal vectors, thus computing only one regular triangulation per triangulated facet of Π.

We perform an *experimental analysis* of our algorithm. We design experiments parameterized on: the total number of input points $|\mathcal{A}|$, the dimension n of pointsets A_i, and the dimension of projection m. First, we examine our algorithm on random inputs for implicitization and u-resultants, where $m = n + 1$, while varying $|\mathcal{A}|, n$. We fix $\delta \in \mathbb{N}$ and select random points on the δ-simplex to generate dense inputs, and points on the $(\delta/2)$-cube to generate sparse inputs. For *implicitization* the projection coordinates correspond to point $a_{i1} = (0, \dots, 0) \in A_i$. For $n = 2$ the problem corresponds to implicitizing surfaces: when $|\mathcal{A}| < 60$, we compute the polytopes in < 1sec (Figure 2.6(a)). When computing the *u-resultant* polytope, the projection coordinates correspond to $A_0 = \{(1, \dots, 0), \dots, (0, \dots, 1)\}$. For $n = 2$, when $|\mathcal{A}| < 500$, we compute the polytopes in < 1sec (Figure 2.6(a)).

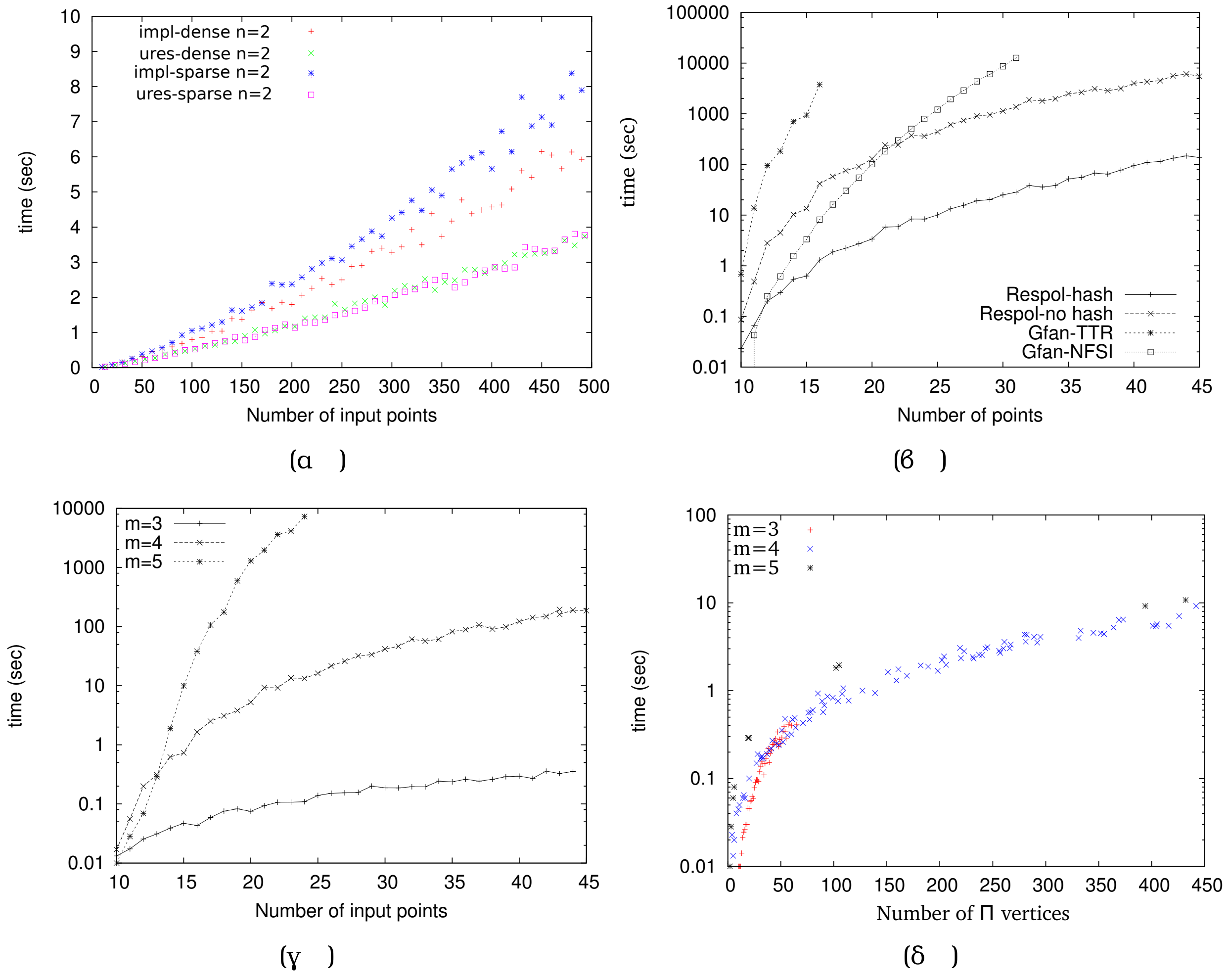


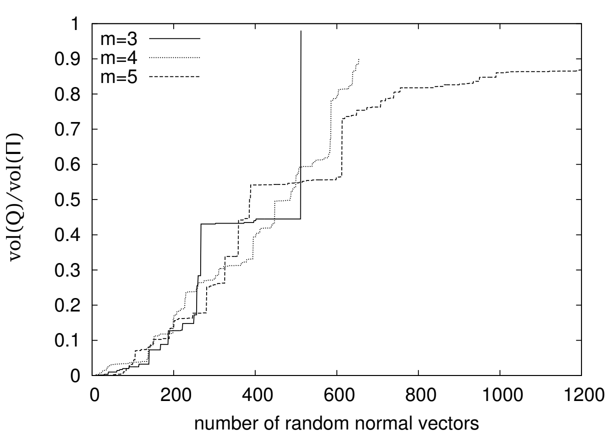
Figure 2.6: (a) Implicitization and u-resultants for $n = 2, m = 3$; (b) Comparison of `respol` (hashing and not hashing determinants) and `Gfan` (traversing tropical resultants and computing normal fan from stable intersection) for $m = 4$; (c) Performance of Alg. 1 for $m = 3, 4, 5$ as a function of input; (d) Performance of Alg. 1 as a function of its output; y-axes in (b), (c), (d) are in logarithmic scale.

By using the *hashing determinants* scheme we gain a $18\times$ speedup when $n = 2,\ m = 3$. For $m = 4$ we gain a larger speedup; we computed in $<$ 2min an instance where $|\mathcal{A}| = 37$ and would take $>$ 1hr to compute otherwise. Thus, when the dimension and $|\mathcal{A}|$ becomes larger, this method allows our algorithm to compute instances of the problem that would be intractable otherwise, as shown for $n = 3,\ m = 4$ (Figure 2.6(b)).

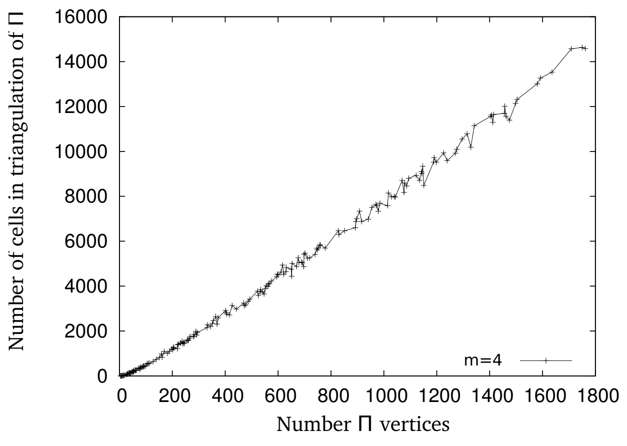
We confirm experimentally the *output-sensitivity* of our algorithm. First, our algorithm always computes vertices of Π either to extend Π or to legalize a facet.

# cells in triangulation				time (sec)				f-vector of Π					
μ	σ	min	max	μ	σ	min	max						
4781	154	4560	5087	0.35	0.01	0.34	0.38			449	1405	1438	482
16966	407	16223	17598	1.51	0.03	1.45	1.56			1412	4498	4705	1619
18229	935	16668	20058	1.92	0.10	1.77	2.11		432	1974	3121	2082	505
563838	6325	548206	578873	99	1.62	93.84	103.07		9678	43569	71004	50170	13059
289847	15788	264473	318976	69	4.88	61.67	77.31	1308	7576	16137	16324	7959	1504
400552	14424	374149	426476	96.5	4.91	88.86	107.12	1680	9740	21022	21719	10890	2133

Table 2.2: Typical f-vectors of projections of resultant polytopes and the size of their triangulations. We perform 20 runs with random insertion order of vertices for each polytope and report the minimum, maximum, average value μ and the standard deviation σ for the number of cells and the runtime.



(a)



(б)

Figure 2.7: (a) vol(Q)/vol(Π) as a function of the number of random normal vectors used to compute Q; (b) The size of the triangulation of Π as a function of the output of Alg. 1.

We experimentally show that our algorithm has, for fixed m, a subexponential behaviour with respect to both input and output (Figure 2.6(c), 2.6(d)) and its output is subexponential with respect to the input.

As the complexity analysis (Theorem 11) indicates, the runtime of the algorithm depends on the size of the constructed placing triangulation of Π. The size of the placing triangulation depends on the ordering of the inserted points. We perform experiments on the effect of the inserting order to the size of the triangulation as well as the running time of the computation of the triangulation (Table 2.2). These sizes as well as the runtimes vary in a very narrow range. Thus, the insertion order is not crucial in both the runtime and the space of our algorithm. Further experiments in 4-dimensional $N(R)$ show that the size of the input bounds polynomially the size of the triangulation of the output (Figure 2.7(b)) which explains the efficiency of our algorithm in this dimension.

We explore the *limits* of our implementation. By bounding runtime to < 2hr, we compute instances of 5-, 6-, 7-dimensional Π with 35K, 23K, 500 vertices, respectively (Table 2.1).

We also compare with the implementation of [86], which is based on Gfan library. They develop two algorithms to compute projections of $N(\mathcal{R})$. Assuming $\mathcal{R}$ defines a hypersurface, their methods compute a union of (possibly overlapping) cones, along with their multiplicities, see Theorem 2.9 of [86]. From this intermediate result they construct the normal cones to the resultant vertices.

examples in [86]	a	b	c	d	e	f	g	h	i
$\lvert\mathcal{A}\rvert$	12	12	15	12	12	16	27	16	20
m	12	12	15	6	7	9	3	4	5
n	3	2	4	2	2	3	2	3	4
Gfan(secs*)	1.40	6	55	0.70	1.30	798	0.40	2.60	184
respol(secs)	1.40	18.41	99.90	0.26	1.24	934	0.02	0.96	292.01

Table 2.3: Comparison of our implementation with Gfan. * Timings for Gfan as reported in [86].

We compare with the best timings of Gfan methods using the examples and timings of [86] (Table 2.3). Our method is faster in examples (d), (e), (g), (h) where $m < 7$, is competitive (up to 2 times slower) in (a) where $m = |\mathcal{A}| = 12$ and (i) where $m = 5, |\mathcal{A}| = 20$ and slower in (b), (c), (f) where $m \geq 12$. The bottleneck of our implementation, that makes it slower when the dimension of the projection m is high, is the incremental convex hull construction in $\mathbb{R}^m$. Moreover, since our implementation considers that $N(\mathcal{R})$ lies in $\mathbb{R}^{|\mathcal{A}|}$ instead of $\mathbb{R}^{|\mathcal{A}|-2n-1}$, (see also the discussion on the homogeneities of $\mathcal{R}$ in Section 2.2), it cannot take advantage of the fact that $\dim(N(\mathcal{R}))$ could be less than m when $|\mathcal{A}| - 2n - 1 < m < |\mathcal{A}|$. This is the case in examples (b), (c) and (f). On the other hand, we run extensive experiments for $n = 3$, considering implicitization, where $m = 4$ and our method, with and without using hashing, is much faster than any of the two algorithms based on Gfan (Figure 2.6(b)). However, for $n = 4$, $m = 5$ the beta version of Gfan used in our experiments was not stable and always crashed when $|\mathcal{A}| > 13$.

We analyze the computation of inner and outer *approximations* Q and Q_o^H. We test the variant of Section 2.3 by stopping it when $\mathrm{vol}(Q)/\mathrm{vol}(Q_o^H) > 0.9$. In the

input	m	3	3	4	4	5	5
	$\|\mathcal{A}\|$	200	490	20	30	17	20
approximation	# of Q vertices	15	11	63	121	$>$ 10hr	$>$ 10hr
	vol(Q)/vol(Π)	0.96	0.95	0.93	0.94	$>$ 10hr	$>$ 10hr
algorithm	vol(Q_o)/vol(Π)	1.02	1.03	1.04	1.03	$>$ 10hr	$>$ 10hr
	time (sec)	0.15	0.22	0.37	1.42	$>$ 10hr	$>$ 10hr
uniformly	$\|Q\|$	34	45	123	207	228	257
	random vectors	606	576	613	646	977	924
random	vol(Q)/vol(Π)	0.93	0.99	0.94	0.90	0.90	0.90
	time (sec)	5.61	12.78	1.10	4.73	8.41	16.90
exact	# of Π vertices	98	133	416	1296	1674	5093
algorithm	time (sec)	2.03	5.87	3.72	25.97	51.54	239.96

Table 2.4: Results on experiments computing Q, Q_o^H using the approximation algorithm and the random vectors procedure; we stop the approximation algorithm when vol(Q)/vol(Q_o) > 0.9; the results with random vectors are the average values over 10 independent experiments; "$>$ 10hr" indicates computation of vol(Q_o) was interrupted after 10hr.

experiments, the number of Q vertices is $< 15\%$ of the Π vertices, thus there is a speedup of up to 25 times over the exact algorithm at the largest instances. The approximation of the volume is very satisfactory: vol(Q_o^H)/vol(Π) < 1.04 and vol(Q)/vol(Π) > 0.93 for the tested instances (Table 2.4). The bottleneck here is the computation of vol(Q_o^H), where Q_o^H is given in H-representation: the runtime explodes for $m \geq 5$. We use `polymake` in every step to compute vol(Q_o^H) because we are lacking of an implementation that, given a polytope P in H-representation, its volume and a halfspace H, computes the volume of the intersection of P and H. Note that we do not include this computation time in the reported time. Our current work considers ways to extend these observations to a polynomial time approximation algorithm for the volume and the polytope itself when the latter is given by an optimization oracle, as is the case here.

Next, we study procedures that compute only the V-representation of Q. For this, we count how many *random vectors* uniformly distributed on the m-dimensional sphere are needed to obtain vol(Q)/vol(Π) > 0.9. This procedure runs up to 10 times faster than the exact algorithm (Table 2.4). Figure 2.7(a) illustrates the convergence of vol(Q)/vol(Π) to the threshold value 0.9 in typical $3, 4, 5$-dimensional examples. The basic drawback of this method is that it does not provide guarantees for vol(Q)/vol(Π) because we do not have sufficient *a priori* information on Π. These experiments also illustrate the extent in which the normal vectors required to deterministically construct Π are uniformly distributed over the sphere.

2.6 Computing discriminant polytopes

We extend `ResPol` to compute (reduced) discriminant polytopes following two approaches. The first focuses on reduced discriminants. By employing the Horn-Kapranov parameterization, the problem is reduced to implicitization. The Newton polytope of the implicit equation of the parameterization, or implicit polytope, is computed as the projection of a resultant polytope [61] and it contains (a translate of) the reduced discriminant polytope. This approach is discussed below.

The second approach defines vertex oracles for the discriminant polytope and uses Beneath-Beyond. There are several procedures to get a vertex oracle. In [122] is given a procedure and an implementation (`tropli`) for such an oracle using tropical geometry: `tropli`, given direction $c \in \mathbb{R}^{|A|}$, computes a vertex $v \in N(\Delta_A)$ s.t. $c^T v$ is minimized. `Respol` can use this oracle to reconstruct the discriminant polytope. One can also define a vertex oracle using the η-vectors from [74, ch.11], Such an oracle involves the computation of (normalized) volumes of lower dimensional simplices, and has not yet been implemented in `ResPol`.

Regarding the first approach, given A, let $B = (b_{ij}) \in \mathbb{Z}^{n\times(m-n-1)}$ be a matrix whose column vectors are a basis of the integer kernel of A. Then B is of full rank. We assume that its maximal minors have unit gcd (i.e. the rows generate $\mathbb{Z}^{m-n-1}$). Since the first row of A equals $(1, \dots, 1)$, the columns of B add up to 0. Set $d = m - n - 1$. Let $y_1, \dots, y_d$ be homogenous parameters and set $y_1 = 1$ so as to dehomogenize the parameterization. We denote by $l_i, i = 1, \dots, m$ the inner product of the i-th row of B and the parameter vector $(1, y_2, \dots, y_d)$: $l_i := \sum_{j=1}^{d} b_{ij} y_j$. The l_i correspond bijectively to the coefficients $c_a, a \in A$ of f and are thus the discriminant variables. The, so called, Horn-Kapranov parametrization [74, 92], is defined as:

$$x_j = \prod_{i=1}^{m} l_i^{b_{ij}},\ j = 1, 2, \dots, d. \tag{2.6}$$

The implicit equation of (the closure of) its image is a polynomial Δ_B in $x := (x_1, \dots, x_d)$, called the *reduced discriminant*, which is the dehomogenized version of Δ_A; it is obtained from Δ_A by specializing some $n + 1$ of its variables so as to remove the $n+1$ quasi-homogeneities. It follows that $N(\Delta_B)$ is the projection of $N(\Delta_A)$ in a space of dimension equal to its intrinsic dimension and retains the combinatorial structure of $N(\Delta_A)$.

Example 6. Let $A=\{0,1,2,3,4\}$ and $f=c_0+c_1t^1+c_2t^2+c_3t^3+c_4t^4$ be a generic quartic.

$$A=\begin{pmatrix}1&1&1&1&1\\0&1&2&3&4\end{pmatrix},\qquad B=\begin{pmatrix}3&2&1\\-4&-3&-2\\0&0&1\\0&1&0\\1&0&0\end{pmatrix}.$$

Here $m=5, n=1, d=3$ and $l_1=3+2y_2+y_3, l_2=-4-3y_2-2y_3, l_3=y_3, l_4=y_2, l_5=1$, and the Horn-Kapranov parameterization is:

$$x_1=\frac{(3+2y_2+y_3)^3}{(-4-3y_2-2y_3)^4},\quad x_2=\frac{(3+2y_2+y_3)^2y_2}{(-4-3y_2-2y_3)^3},\quad x_3=\frac{(3+2y_2+y_3)y_3}{(-4-3y_2-2y_3)^2}. \tag{2.7}$$

We prefer to have rational parameterizations with a single monomial in the denominator because this facilitates the computation of the implicit polytope. We introduce a new parameter y_4 expressing the common denominator in (2.7) and obtain the parameterization

$$x_1=\frac{(3+2y_2+y_3)^3}{y_4^4},\ x_2=\frac{(3+2y_2+y_3)^2y_2}{y_4^3},\ x_3=\frac{(3+2y_2+y_3)y_3}{y_4^2},\ y_4=-4-3y_2-2y_3,$$

from which we define the polynomials

$$F_0:=x_1y_4^4-(3+2y_2+y_3)^3,\quad F_1:=x_2y_4^3-(3+2y_2+y_3)^2y_2,$$
$$F_2:=x_3y_4^2-(3+2y_2+y_3)y_3,\quad F_3:=y_4+4+3y_2+2y_3,$$

whose supports are given as input to `ResPol`. The above procedure is demonstrated in the `Maple` file `horn_example2.mw` available with our distribution. Then, we prepare the input `file.txt`:

```
3
11 7 4 4  | 0 11 18
[[0, 0, 4], [0, 0, 0], [1, 0, 0], [0, 1, 0], [2, 0, 0], [1, 1, 0], [0, 2, 0],
 [3, 0, 0], [2, 1, 0], [1, 2, 0], [0, 3, 0], [0, 0, 3], [1, 0, 0], [2, 0, 0],
 [1, 1, 0], [3, 0, 0], [2, 1, 0], [1, 2, 0], [0, 0, 2], [0, 1, 0], [1, 1, 0],
 [0, 2, 0], [0, 0, 1], [0, 0, 0], [1, 0, 0], [0, 1, 0]]
```

The second line after ‘l’ instructs `ResPol` to project to the space defined by x_1, x_2, x_3. Executing `./res_enum_d < file.txt`, we obtain the vertices $(0,0,12)$, $(0,8,0)$, $(6,0,0)$, $(0,0,0)$ in the standard output. They define a polytope containing a translate of $N(\Delta_B)$. To compute the discriminant polytope using `tropli` we prepare a textfile `file.txt`:

```
1
5 0  l
[[0], [1], [2], [3], [4]]
```

where the zero after the cardinality 5 of the support in the second line is needed because `ResPol` expects the number of supports to be one more than the dimension. Executing the command `./res_enum_d -d < file.txt`, we obtain the vertices of $N(\Delta_A)$: $(1,0,4,0,1)$, $(0,3,0,3,0)$, $(0,4,0,0,2)$, $(0,2,3,0,1)$, $(0,2,2,2,0)$, $(2,0,0,4,0)$, $(3,0,0,0,3)$, $(1,0,3,2,0)$ in the standard output. The corresponding vertices of $N(\Delta_B)$ may be computed as follows: By renaming the l_i's as c_i's we have from (2.7) that $x_1 = c_0^3 c_1^{-4} c_4$, $x_2 = c_0^2 c_1^{-3} c_3$, $x_3 = c_0 c_1^{-2} c_2$, which gives the correspondence: $(\kappa, \lambda, \mu) \longmapsto (3\kappa + 2\lambda + \mu, -4\kappa - 3\lambda - 2\mu, \mu, \lambda, \kappa)$, between the vertices of Δ_B and Δ_A. Moreover, this yields the correspondence: $(a_1, a_2, a_3, a_4, a_5) \longmapsto (a_5, a_4, a_3)$ between the vertices of Δ_A and Δ_B. Hence, from the set of vertices of $N(\Delta_A)$ above, we obtain the vertices of $N(\Delta_B)$: $(0,2,3)$, $(0,2,2)$, $(1,0,3)$, $(1,0,4)$, $(0,3,0)$, $(0,4,0)$, $(3,0,0)$, $(2,0,0)$, which are all contained in the polytope defined by the set of vertices predicted by `ResPol`.

2.7 Future work

One algorithm that should be experimentally evaluated is the following. We perform a search over the vertices of $\Sigma(A)$, that is, we build a search tree with flips as edges. We keep a set with the extreme vertices with respect to a given projection. Each computed vertex that is not extreme in the above set is discarded and no flips are executed on it, i.e. the search tree is pruned in this vertex. The search procedure could be the algorithm of TOPCOM or the one presented in [110] which builds a search tree in some equivalence classes of $\Sigma(A)$. The main advantage of this algorithm is that it does not involve a convex hull computation. On the other hand, it is not output-sensitive with respect to the number of vertices of the resultant polytope; its complexity depends on the number of vertices on the *silhouette* of $\Sigma(A)$, with respect to a given projection and those that are connected by an edge with them.

As shown, `polymake`'s convex hull algorithm is competitive, thus one may use it for implementing our algorithm. On the other hand, `triangulation` is expected to include fast enumeration of all regular triangulations for a given (non generic) lifting, in which case Π may be extended by more than one (coplanar) vertices.

Our proposed algorithm uses an incremental convex hull algorithm and it is known that any such algorithm has a worst-case super-polynomial *total time complexity* [25] in the number of input points and output facets. The basic open question that this chapter raises is whether there is a polynomial total time algorithm for Π or even for the set of its vertices.

Chapter 3

Algorithms for the edge skeleton

3.1 Introduction

Convex polytopes in general dimension admit a number of alternative representations. The best known, explicit representations for a polytope P are either the set of its vertices (V-representation) or a bounded intersection of halfspaces (H-representation). Switching between the two representations constitutes the convex hull and vertex enumeration problems. A linear programming problem (LP) on P consists in finding a vertex of P that maximizes the inner product with a given objective vector c. This is very easy if P is in V-representation, but also if P is in H-representation, LP can be solved in polynomial time.

In general dimension, there is no polynomial-time algorithm for either convex hull or vertex enumeration, since the output size can be exponential in the worst case by the upper bound theorem [108]. In addition to, generating all vertices of a polyhedron is also hard [95]. We therefore want to take the output size into account and say that an algorithm runs in *total polynomial* time if its time complexity is bounded by a polynomial in the input *and* output size. There is no known total polynomial-time algorithm for either convex hull or vertex enumeration. In [7] they provide for each known types of convex hull algorithms, explicit families of polytopes with which as input the algorithms run in superpolynomial time.

However, finding the vertices of the convex hull of a given point set reduces to LP and has thus polynomial complexity in the input (cf. [38]). The algorithm in [8] solves, in total polynomial-time, vertex enumeration for simple polytopes and convex hull for simplicial polytopes. For 0/1-polytopes a total polynomial-time algorithm for vertex enumeration is presented in [32], where a 0/1-polytope is one, all of whose vertices have coordinates 0 or 1. On the other hand, there is no such algorithm for the more general case of 0/1-polyhedra unless P=NP [24]. In this thesis we establish another case where total polynomial-time algorithms exist.

An important explicit representation of a polytope is the *edge-skeleton* (or 1-skeleton), which is the graph of polytope vertices and edges, but none of the faces of dimension larger than one. For simple polytopes, the edge-skeleton determines the complete face lattice (see [88] and the references therein), but in general, this is false. The edge-skeleton is a useful and compact representation employed in different problems, e.g. in computing general-dimensional Delaunay triangulations of a given pointset: In [22] the authors show how the edge-skeleton suffices for point location by allowing them to recover only the needed full-dimensional simplices of the triangulation.

In this chapter we study the case where a polytope P is given by an implicit representation, where the only access to P is a black box subroutine (oracle) that solves the LP problem on P for a given vector c. Then, we say that P is given by an *optimization*, or *LP* oracle. Given such an oracle, the entire polytope can be reconstructed, and both V- and H-representations can be found using the Beneath-Beyond method; see e.g. [59, 82], although not in total polynomial-time.

Another important implicit representation of P is obtained through a *separation* oracle (Section 3.2). Celebrated results of Khachiyan [96] as well as Grötschel, Lovász and Schrijver [78] show that separation and optimization oracles are polynomial time equivalent (Proposition 14). Many important results in combinatorial optimization use the fact that separation implies optimization. In our study, we also need the other direction: Given an optimization oracle, compute a separation oracle for P.

The problem that we study is a special case of vertex enumeration. We are given an optimization oracle for a polytope P and a set of vectors that is guaranteed to contain the directions of all edges of P; edge directions are given by unit vectors. We are asked to compute the edge-skeleton of P. Since the vertices are computed along with the skeleton, our problem subsumes vertex enumeration for polytopes for which we know the edge directions. This resembles the fundamental

Minkowski reconstruction problem, e.g. [75], except that, instead of information on the facets, we have information about the 1-dimensional faces (and an oracle). The problem of the reconstruction of a simple polytope by its edge-skeleton graph is studied in [88].

The most relevant previous work is an algorithm for vertex enumeration of P, given by an optimization oracle and a superset D of all edge directions, proposed in [115] (Proposition 15). It runs in total polynomial-time in fixed dimension. The algorithm computes the zonotope Z of D, then computes an arbitrary vector in the normal cone of each vertex of Z and calls the oracle with this vector. It outputs all vertices without further information. Computing the edges from n vertices can be done by $O(n^2)$ calls to LP.

3.1.1 Applications

The problem of edge-skeleton computation given an oracle and a superset of the polytope's edge directions naturally appears in many applications. In Section 3.4 we offer new efficient algorithms for the first two applications below.

One application is the *signed Minkowski sum* problem where, besides addition, we also allow a restricted case of *Minkowski difference*. Let $A - B$ be polytope C such that A can be written as a sum $A = B+C$. In other words, a signed Minkowski sum equality such as $P-Q+R-S = T$ should be interpreted as $P+R = Q+S+T$. Such sums are motivated by the fact that resultant and discriminant polytopes (to be defined later) are written as signed sums of secondary polytopes [109], [74, Thm 11.1.3]. Also, matroid polytopes and generalized permutahedra can be written as signed Minkowski sums [4].

Minkowski sums have been studied extensively. Given r V-polytopes in $\mathbb{R}^d$, Gritzmann et al. [76] deal with the various Minkowski sum problems that occur if they regard none, one, or both of r and d as constants. They give polynomial algorithms for fixed d regardless of the input representation. For varying d they show that no polynomial-time algorithm exists except for the case of fixed r in the binary model of computation. Fukuda [67] (extended in [69]) gives an LP-based algorithm for the Minkowski sum of polytopes in V-representation whose complexity, in the binary model of computation, is total polynomial, and depends polynomially on δ, which is the sum of the maximum vertex degree in each summand. However, we are not aware of any algorithm for signed Minkowski sums and it is not obvious how the above algorithms for Minkowski sums can be extended to the signed case.

The second application is resultant, secondary as well as discriminant polytopes. For resultant polytopes at least, the only plausible representation today seems to be via optimization oracles [59]. *Resultants* are fundamental in computational algebraic geometry since they generalize determinants to nonlinear systems [131, 74]. The Newton polytope R of the resultant, or *resultant polytope*, is the convex hull of the exponent vectors corresponding to nonzero terms. A resultant is defined for $k+1$ pointsets in $\mathbb{Z}^k$. If R lies in $\mathbb{R}^d$, the total number of input points is $d+2k+1$. If n is the number of vertices in R, typically $n \gg d \gg k$, so k is assumed fixed. A polynomial-time optimization oracle yields an output-sensitive algorithm for the computation of R [59] (Lemma 25).

This approach can also be used for computing the secondary and discriminant polytopes, defined in [74]; cf [101] on secondary polytopes. The secondary polytope of a pointset is a fundamental object since it offers a polytope realization of the graph of regular triangulations of the pointset. A total polynomial-time algorithm for the secondary polytope when k is fixed is given in [106]. A specific application of discriminant polytopes is discussed in [117], where the author establishes a lower bound on the volume of the discriminant polytope of a multivariate polynomial, thus refuting a conjecture by E.I. Shustin on an asymptotic upper bound for the number of real hypersurfaces.

The size of all these polytopes is typically exponential in d: the number of vertices of R is $O(d^{2d^2})$ [131], and the number of j-faces (for any j) of the secondary polytope is $O(d^{(d-1)^2})$, which is tight if d is fixed [19].

More applications of our methods exist. One is in *convex combinatorial optimization*: given $\mathcal{F} \subset 2^N$ with $N = \{1,\dots,n\}$, a vectorial weighting $w : N \to \mathbb{Q}^d$, and a convex functional $c : \mathbb{Q}^d \to \mathbb{Q}$, find $F \in \mathcal{F}$ of maximum value $c(w(F))$. This captures a variety of (hard) problems studied in operations research and mathematical programming, including quadratic assignment, scheduling, reliability, bargaining games, and inventory management, see [114] and references therein. The standard linear combinatorial optimization problem is the special case with $d = 1, w : N \to \mathbb{Q}$, and $c : \mathbb{Q} \to \mathbb{Q} : x \mapsto x$ being the identity. As shown in [114], a convex combinatorial optimization problem can be solved in polynomial-time for fixed d, if we know the edge directions of the polytope given by the convex hull of the incidence vectors of the sets in $\mathcal{F}$.

Another application is *convex integer maximization*, where we maximize a convex function over the integer hull of a polyhedron. In [100], the vertex enumeration algorithm of [115]—based on the knowledge of edge directions—is used to come up with polynomial algorithms for many interesting cases of convex integer maximization, such as multiway transportation, packing, vector partitioning and clustering. A set that contains the directions of all edges is computed via Graver bases, and the enumeration of all vertices of a projection of the integer hull suffices to find the optimal solution.

3.1.2 Our contribution

We present the first total polynomial-time algorithm for computing the edge-skeleton of a polytope, given an optimization oracle, and a set of directions that contains the polytope's edge directions. The polytope is assumed to have some (unknown) H-representation with an arbitrary number of inequalities, but each of known bitsize, as shall be specified below. Our algorithm also works if the polytope is given by a separation oracle. All complexity bounds refer to the (oracle) Turing machine model, thus leading to (weakly) polynomial-time algorithms when the oracle is of polynomial-time complexity. By employing the reverse search method of [8] we offer a space-efficient variant of our algorithm. It remains open whether there is also a strongly polynomial-time algorithm in the real RAM model of computation.

Our algorithm yields the first (weakly) total polynomial-time algorithms for the edge-skeleton (and vertex enumeration) of signed Minkowski sum, and resultant polytopes. For both polytope classes, optimization oracles are naturally and efficiently constructed, whereas it is not straightforward to obtain the more commonly employed membership or separation oracles. For resultant polytopes, optimization oracles offer the most efficient known representation. Our results on resultant polytopes extend to secondary polytopes, for which a different approach in the same complexity class is known, as well as discriminant polytopes.

Regarding the problems of convex combinatorial optimization and convex integer programming the current approaches use the algorithm of [115] whose complexity has an exponential dependence on the dimension (Proposition 15). The utilization of our algorithm instead offers an alternative approach while removing the exponential dependence on the dimension.

Outline. The next section specifies our theoretical framework. Section 3.3 introduces polynomial-time algorithms for the edge-skeleton. Section 3.4 applies our results to signed Minkowski sums, as well as resultant and secondary polytopes. We conclude with open questions.

3.2 Well-described polytopes and oracles

This section describes our theoretical framework and relates the most relevant oracles. We start with the notation used in this chapter following by some basics from polytope theory; for a detailed presentation we refer to [140].

We denote by d the ambient space dimension and n the number of vertices of the output (bounded) polytope; k denotes dimension when it is fixed (e.g. input space for resultant polytopes); conv(A) is the convex hull of A. Moreover, φ is an upper bound for the encoding length of every inequality defining a well-described polytope (see the next section); $\langle X\rangle$ denotes the binary encoding size of an explicitly given object X (e.g., a set of vectors). For a well-described and implicitly given polytope $P \subseteq \mathbb{R}^d$, we will define $\langle P\rangle := d+\varphi$. Let $\mathbb{O} : \mathbb{R} \to \mathbb{R}$ denote a polynomial such that the oracle conversion algorithms of Proposition 14 all run in oracle polynomial-time $\mathbb{O}(\langle P\rangle)$ for a given well-described polytope P. The polynomial $\mathbb{LP} : \mathbb{R} \to \mathbb{R}$ is such that $\mathbb{LP}(\langle A\rangle + \langle b\rangle + \langle c\rangle)$ bounds the runtime of maximizing $c^T x$ over the polyhedron $\{x \mid Ax \leq b\}$.

A convex polytope $P \subseteq \mathbb{R}^d$ can be represented as the convex hull of a finite set of points, called the *V-representation* of P. In other words, $P = \text{conv}(A)$, where $A = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$. Another, equivalent representation of P is as the bounded intersection of a finite set of halfspaces or linear inequalities, called the *H-representation* of P. That is, $P = \{x \mid Ax \leq b\}, A \subseteq \mathbb{R}^{m\times d}, x \in \mathbb{R}^d, b \in \mathbb{R}^m$. Given P, an inequality or a halfspace $\{a^T x \leq \beta\}$ (where $a \in \mathbb{R}^d, \beta \in \mathbb{R}$) is called *supporting* if $a^T x \leq \beta$ for all $x \in P$ and $a^T x = \beta$ for some $x \in P$. The set $\{x \in P \mid a^T x = b\}$ is a *face* of P.

Definition 2. The *polar dual polytope* of P is defined as:

$$P^* := \{c \in \mathbb{R}^d : c^T x \leq 1 \text{ for all } x \in P\} \subseteq \mathbb{R}^d,$$

where we assume that the origin $0 \in int(P)$, the relative interior of P, i.e. 0 is not contained in any face of P of dimension $< d$.

For our results, we need to assume that the output polytope is *well-described* [78, Definition 6.2.2]. This will be the case in all our applications.

Definition 3. A rational polytope $P \subseteq \mathbb{R}^d$ is *well-described* (with a parameter φ that we need to know explicitly) if there exists an H-representation for P in which every inequality has encoding length at most φ. The *encoding length* of a well-described polytope is $\langle P \rangle = d + \varphi$. Similarly, the *encoding length* of a set of vectors $D \subseteq \mathbb{R}^d$ is $\langle D \rangle = d + \nu$ if every vector in D has encoding length at most ν.

In defining P, the inequalities are not known themselves, and we make no assumptions about their number. The following lemma connects the encoding length of inequalities with the encoding length of vertices.

Lemma 13. [78, Lemma 6.2.4] *Let $P \subseteq \mathbb{R}^d$. If every inequality in an H-representation for P has encoding length at most φ, then every vertex of P has encoding length at most $4d^2\varphi$. If every vertex of P has encoding length at most ν, then every inequality of its H-representation has encoding length at most $3d^2\nu$.*

The natural model of computation when P is given by an oracle is that of an *oracle Turing machine* [78, Section 1.2]. This is a Turing machine that can (in one step) replace any input to the oracle (to be prepared on a special oracle tape) by the output resulting from calling the oracle, where we assume that the output size is polynomially bounded in the input size. An algorithm is *oracle polynomial-time* if it can be realized by a polynomial-time oracle Turing machine. Moreover it is *total polynomial-time* if its time complexity is bounded by a polynomial in the input *and* output size.

In this chapter, we consider three oracles for polytopes; they can more generally be defined for (not necessarily bounded) polyhedra, but we do not need this:

- *Optimization* ($\mathrm{OPT}_P(c)$): Given vector $c \in \mathbb{R}^d$, either find a point $y \in P$ maximizing $c^T x$ over all $x \in P$, or assert $P = \emptyset$.
- *Violation* ($\mathrm{VIOL}_P(c, \gamma)$): Given vector $c \in \mathbb{R}^d$ and $\gamma \in \mathbb{R}$, either find point $y \in P$ such that $c^T y > \gamma$, or assert that $c^T x \leq \gamma$ for all $x \in P$.
- *Separation* ($\mathrm{SEP}_P(y)$): Given point $y \in \mathbb{R}^d$, either certify that $y \in P$, or find a hyperplane that separates y from P; i.e. find vector $c \in \mathbb{R}^d$ such that $c^T y > c^T x$ for all $x \in P$.

The following is a main result of [78] and the cornerstone of our method.

Proposition 14. [78, Theorem 6.4.9] *For a well-described polytope, any one of the three aforementioned oracles is sufficient to compute the other two in oracle polynomial-time. The runtime (polynomially) depends on the ambient dimension d and the bound φ for the maximum encoding length of an inequality in some H-representation of P.*

For applications in combinatorial optimization, an extremely important feature is that the runtime does not depend on the number of inequalities that are needed to describe P. Even if this number is exponential in d, the three oracles are polynomial-time equivalent.

Another important corollary is that linear programs can be solved in polynomial-time. Indeed, an explicitly given (bounded coefficient) system $Ax \leq b, x \in \mathbb{R}^d$ of inequalities defines a well-described polytope P, for which the separation oracle is very easy to implement in time polynomial in $\langle P \rangle$; hence, the oracle polynomial-time algorithm for $\mathrm{OPT}_P(c)$ becomes a (proper) polynomial-time algorithm.

3.3 Computing the edge-skeleton

This section studies total polynomial-time algorithms for the edge-skeleton. We are given a well-described polytope $P \subseteq \mathbb{R}^d$ via an optimization oracle $\mathrm{OPT}_P(c)$ of P. Moreover, we are given a superset D of all edge directions of P; to be precise, we define

$$D(P) := \left\{ \frac{v - w}{||v - w||} : v \text{ and } w \text{ are adjacent vertices of } P \right\}$$

to be the set of (unit) edge directions, and we assume that for every $e \in D(P)$, the set D contains some positive multiple $te, t \in \mathbb{R}, t > 0$. Slightly abusing notation, we write $D \supseteq D(P)$.

The goal is to efficiently compute the edge-skeleton of P, i.e. its vertices and the edges connecting the vertices. Even if $D = D(P)$, this set does not, in general, provide enough information for the task, so we need additional information about P; here we assume an optimization oracle.

Vertex enumeration with this input has been studied in the real RAM model of computation where we count the number of arithmetic operations:

Proposition 15. [115] *Let $P \subseteq \mathbb{R}^d$ be a polytope given by OPT$_P(c)$, and let $D \supseteq D(P)$ be a superset of the edge directions of P. The vertices of P are computed using $O(|D|^{d-1})$ arithmetic operations and $O(|D|^{d-1})$ calls to OPT$_P(c)$.*

If P has n vertices, then $|D(P)| \leq \binom{n}{2}$, and this is tight for neighborly polytopes in general position [140]. This means that the bound of Proposition 15 is $O\left(n^{2d-2}\right)$, assuming that $|D| = \Theta(|D(P)|)$.

We show below that the edge-skeleton can be computed in oracle total polynomial-time for a well-described polytope, which possesses an (unknown) H-representation with encoding size φ. Thus, we show that the exponential dependence on d in Proposition 15 can be removed in the Turing machine model of computation, leading to a (weakly) total polynomial-time algorithm. It remains open whether there is also a strongly total polynomial-time algorithm with a total polynomial runtime bound in the real RAM model of computation.

The algorithm (Algorithm 2) is as follows. Using OPT$_P(c)$, we find some vertex v_0 of P (this can be done even if OPT$_P(c)$ does not directly return a vertex [78, Lemma 6.51], [53, pp. 255–256]).

We maintain sets V_P, E_P of vertices and their incident edges, along with a queue $W \subseteq V_P$ of the vertices for which we have not found all incident edges yet. Initially, $W = \{v_0\}, V_P = E_P = \emptyset$. When we process the next vertex v from the queue, it remains to find its incident edges: equivalently, the neighbors of v. To find the neighbors, we first build a set V_{cone} of candidate vertices. We know that for every neighbor w of v, there must be an edge direction e such that $w = v + te$ for suitable $t > 0$. More precisely, w is the point corresponding to maximum t in the 1-dimensional polytope $Q(e) := P \cap \{x \mid x = v + te, t \geq 0\}$, where the latter

Algorithm 2: Edge_Skeleton (OPT_P, D)

```
Input  : Optim. oracle $\mathrm{OPT}_P(c)$, superset $D$ of edge directions $D(P)$
Output: The edge-skeleton (and vertices) of $P$

Compute some vertex $v_0 \in P$;
$V_P \leftarrow \emptyset$; $W \leftarrow \{v_0\}$; $E_P \leftarrow \emptyset$;
while $W \neq \emptyset$ do
    Choose the next element $v \in W$ and remove it from $W$;
    $V_P \leftarrow V_P \bigcup \{v\}$;
    $V_{cone} \leftarrow \emptyset$;
    foreach $e \in E$ do
        $w \leftarrow \mathrm{argmax}\{v + te \in P, t \geq 0\}$;
        if $w \neq v$ then
            $V_{cone} \leftarrow V_{cone} \bigcup \{w\}$;
    Remove non-vertices of $P$ from $V_{cone}$;
    foreach $w \in V_{cone}$ do
        if $w \notin V_P$ then $W \leftarrow W \bigcup \{w\}$;
        if $\{v, w\} \notin E_P$ then $E_P \leftarrow E_P \bigcup \{v, w\}$;
return $V_P, E_P$;
```

equals the intersection of P with the ray in direction e and apex at v. Hence, by solving $|D|$ linear programs, one for every $e \in D$, we can build a set V_{cone} that is guaranteed to contain all neighbors of v. To solve these linear programs, we need to construct optimization oracles for $Q(e)$. To do this, we first construct $\mathrm{SEP}_P(y)$ from $\mathrm{OPT}_P(c)$ in oracle polynomial-time according to Proposition 14. Thus, the construction of $\mathrm{SEP}_{Q(e)}(y)$ is elementary, and since also $Q(e)$ is well-described, we can obtain $\mathrm{OPT}_{Q(e)}(c)$ in oracle polynomial-time.

In a final step, we remove the candidates that do not yield neighboring vertices. For this, we first solve a linear program to compute a hyperplane h separating v from the candidates in V_{cone}; since V_{cone} is a finite subset of $P \setminus \{v\}$, such a hyperplane exists, and w.l.o.g. $v = \mathbf{0}$ and $h = \{x \mid x_d = 1\}$. Let C be the cone generated by the set V_{cone}. We compute the extreme points of $C \cap \{x_d = 1\}$, giving us the extremal rays of C. Finally, we remove every point from V_{cone} that is not on an extremal ray.

The correctness of the algorithm relies on the following Lemma.

Lemma 16. *Let v be a vertex of P processed during Algorithm 2, where we assume w.l.o.g. that $v = 0$ and the set V_{cone} of candidates is separated from v by the hyperplane $\{x \mid x_d = 1\}$.*

A point $w \in \mathbb{R}^d$ is a neighbor of v if and only if w is on some extremal ray of the cone C generated by V_{cone}. Here, an extremal ray is a ray whose intersection with the hyperplane $\{x_d = 1\}$ is an extreme point of the polytope $C \cap \{x \mid x_d = 1\}$.

Proof. Suppose that w is a neighbor of v. By construction, $w \in V_{cone}$. Moreover, since $\{v, w\}$ is an edge, there is a supporting hyperplane $h = \{a^T x = 0\}$ (recall that $v = 0$) such that $a^T x = 0$ for all $x \in \text{conv}(\{v, w\})$ and $a^T x > 0$ for all $p \in P \setminus \text{conv}(\{v, w\})$. For each $q \in V_{cone}$, let $c(q) = \frac{1}{q_d} q \in C \cap \{x \mid x_d = 1\}$. We have $q_d > 0$ by construction. Furthermore, $a^T c(w) = 0$ while $a^T c(q) > 0$ for $q \in V_{cone}$, unless $q \in \text{conv}(\{v, w\})$. In the latter case, $c(q) = c(w)$. Hence, $c(w)$ is the only point y in $C \cap \{x_d = 1\}$ such that $a^T y = 0$, and this implies that $c(w)$ is an extreme point of $C \cap \{x_d = 1\}$. So w is on some extremal ray of C.

For the other direction, suppose that $w \in V_{cone}$ is on the extremal ray $\{te \mid t \in \mathbb{R}\}$. So $c(w)$ is an extreme point of $C \cap \{x \mid x_d = 1\}$. This means, there exists a vertical hyperplane $h = \{a^T x = \beta\}$ with $a_d = 0$ such that $a^T c(w) = \beta$, and $a^T c(q) > \beta$, for all $q \in V_{cone}$ satisfying $c(q) \neq c(w)$. Now define the hyperplane $\bar{h} = \{\bar{a}^T x = 0\}$ with $\bar{a} = (a_1, \ldots, a_{d-1}, -\beta)$. It follows that $\bar{a}^T q \geq 0$ for all $q \in V_{cone}$, so the positive halfspace of $\bar{h}$ contains C and thus also P since $P \subseteq C$. We claim that $\bar{h} \cap P = \text{conv}(\{v, w\})$, which proves that $\text{conv}(\{v, w\})$ is an edge of P and hence w is a neighbor of v.

To see this, we first observe that $\bar{a}^T w = 0$ and $\bar{a}^T q > 0$ for all $q \in V_{cone}$ that are not multiples of p, so $\bar{h} \cap P \subseteq \bar{h} \cap C = \{te \mid t \in \mathbb{R}\}$. On the other hand, we know from the construction of V_{cone} that w is the highest point of P (the one with maximum t) on the ray $\{te \mid t \in \mathbb{R}\}$, so we indeed get $\bar{h} \cap P = \text{conv}(\{v, w\})$. □

We now bound the *time complexity* of Algorithm 2.

Theorem 17. *Given OPT_P and a superset of edge directions D of a well-described polytope $P \subseteq \mathbb{R}^d$ with n vertices, and m edges Algorithm 2 computes the edge-skeleton of P in oracle total polynomial-time*

$$O\left(n|D|\left(\mathbb{O}(\langle P \rangle + \langle D \rangle) + \mathbb{LP}(d^3|D|\left(\langle P \rangle + \langle D \rangle\right)) + d \log n\right)\right),$$

where $\langle D\rangle$ is the binary encoding length of the vector set D.

Proof. We call $\mathrm{OPT}_P(x)$ to find the first vertex of P. Then, there are $O(n)$ iterations. In each one, we construct $O(|D|)$ oracles for polytopes $Q(e)$ of encoding length at most $\langle P\rangle + \langle D\rangle$. We also compute the (at most n) extreme points from a set of at most $|D|$ candidate points. This can be done by solving $|D|$ linear programs whose inequalities have coefficients that are in turn coordinates of vertices of the $Q(e)$'s. By Lemma 13, these coordinates have encoding lengths bounded by $4d^2(\langle P\rangle+\langle D\rangle)$, and the number of coefficients in each linear program is $O(|D|d)$. At each vertex we have to test whether the computed vertices and edges are new. In the course of the algorithm these tests are at most $O(m) = O(n|D|)$, where m the number of P edges. We can test whether a vertex (or an edge) is new in $O(d \log n)$ by using a binary search tree. □

3.3.1 Reverse search for edge-skeleton.

We define a reverse search procedure based on [8] to optimize the space used by Algorithm 2. Given a vertex of P, the set of adjacent edges can be constructed as described above. Then we need to define a total order over the vertices of the polytope. Any generic vector $c \in \mathbb{R}^d$ induces such an order on the vertices, i.e. the order of a vertex u is that of $c^T u$. In other words, we can define a reverse search tree on P with root the vertex v that maximizes $c^T v$ over all the vertices of P, where c is the vector given to OPT_P for initializing P. Technically, the genericity assumption on c can be avoided by sorting the vertices w.r.t. the lexicographical ordering of their coordinates.

Reverse search also needs an *adjacency procedure* which, given a vertex v and an integer j, returns the j-th adjacent vertex of v, as well as a *local search procedure* allowing us to move from any vertex to its optimal neighbor w.r.t. the objective function. Both procedures can be implemented by computing all the adjacent vertices of a given vertex of P as described above, and then returning the best (or the j-th) w.r.t. the ordering induced by c.

The above procedures can be used by a reverse search variant of Algorithm 2 that traverses (forward and backward) the reverse search tree while keeping in memory only a constant number of P vertices and edges. On the contrary, both the original Algorithm 2 and the algorithm of Proposition 15 need to store all vertices of P whose number is exponential in d in the worst case. Note that any algorithm should use space at least $O(d|D|)$ to store the input set of edge directions. The above discussion yields the following result (encoding length of P vertices comes from Lemma 13).

Corollary 18. *Given OPT_P and a superset of edge directions D, a variant of Algorithm 2 that uses reverse search runs in additional to the input space $O(4d^2\langle P\rangle + \langle D\rangle)$ while keeping the same asymptotic time complexity.*

3.4 Applications

We examine two important classes of polytopes and provide new, total polynomial-time algorithms for their representation by an edge-skeleton: signed Minkowski sums, and resultant and secondary polytopes. These polytopes are well-described and naturally defined by optimization oracles, which provide a compact representation.

3.4.1 Signed Minkowski sums

Recall that the *Minkowski sum* of (convex) polytopes $A, B \subseteq \mathbb{R}^d$ is defined as

$$A + B := \{a + b \mid a \in A, b \in B\}.$$

Following [125] the *Minkowski difference* is defined as

$$A - B := \{x \in \mathbb{R}^d \mid B + x \subseteq A\}.$$

Here we consider a special case of Minkowski difference where B is a summand of A. Equivalently, if $A-B = C$ then $A = B+C$. A *signed Minkowski sum* combines Minkowski sums and differences, namely

$$P = s_1P_1 + s_2P_2 + \cdots + s_rP_r,\ s_i \in \{-1, 1\},$$

where all P_i are convex polytopes and so is P.

We also define the sum (or difference) of two optimization oracles as the Minkowski sum (or difference) of the resulting vertices. In particular, if $\mathrm{OPT}_P(c) = v$ and

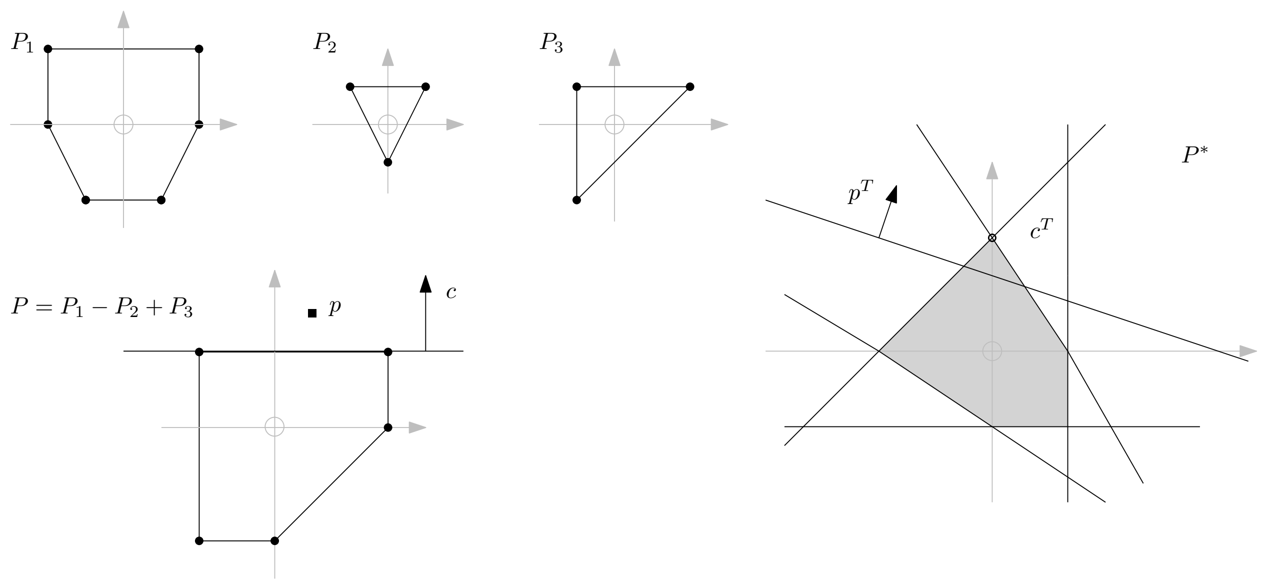


Figure 3.1: Signed Minkowski sum oracles.

$\text{OPT}_{P'}(c') = v'$ for v, v' vertices of P, P' respectively, then $\text{OPT}_P(c)+\text{OPT}_{P'}(c) = v+v'$ and $\text{OPT}_P(c) - \text{OPT}_{P'}(c) = v - v'$. An optimization oracle for the signed Minkowski sum is given by the signed sum of the optimization oracles of the summands.

Lemma 19. *If $P_1, \ldots, P_r \subset \mathbb{R}^d$ are given by optimization oracles, then we compute an optimization oracle for signed Minkowski sum $P = \sum_{i=1}^r s_i P_i$ in $O(r)$.*

Proof. Assume w.l.o.g. that $s_1 = \cdots = s_k = 1 \neq s_{k+1} = \cdots = s_r = -1$. Then, given $P = \sum_{i=1}^r s_i P_i$ we have $P + \sum_{i=k+1}^r P_i = \sum_{i=1}^k P_i = P'$. Let $\text{OPT}_{P'}(c) = v$ for some vertex v of P' and vector $c \in \mathbb{R}^d$. It suffices to show that

$$\text{OPT}_{P'}(c) = v = v_1 + \cdots + v_k = \sum_{i=1}^k \text{OPT}_{P_i},$$

which follows from Minkowski sum properties: $v = v_1 + \cdots + v_k$ for vertices v_i of P_i and $norm_P(v) \subseteq norm_{P_i}(v_i)$, for $i = 1 \ldots k$. Here $norm_P(v)$ denotes the *normal cone* of vertex v of P, i.e. the set of all vectors c such that $c^T x \leq c^T v$ for all $x \in P$. Therefore, we can compute OPT_P with r oracle calls to OPT_{P_i} for $i = 1, \ldots, r$. This yields a complexity of $O(r)$ for OPT_P since, by definition of oracle polynomial-time, the oracle calls in every summand are of unit cost. □

Example 7. Here we illustrate the above definitions and constructions as well as the standard reductions from [78]. Consider the polytopes P_1, P_2, P_3, their signed

Minkowski sum $P = P_1 - P_2 + P_3$, and its polar P^* as shown in Figure 3.1. Observe that $P_1 = P_2 + S$, where S is a square. Assume that P_1, P_2, P_3 are given by $\text{OPT}_{P_1}, \text{OPT}_{P_2}$, OPT_{P_3} oracles.

Then, $\text{OPT}_P(c) = \text{OPT}_{P_1}(c) - \text{OPT}_{P_2}(c) + \text{OPT}_{P_3}(c)$ for some vector $c \in \mathbb{R}^d$. If P satisfies the requirements of Proposition 14 then, having access to $\text{OPT}_P(c)$, we compute $\text{SEP}_P(p)$ in oracle polynomial-time for point $p \in \mathbb{R}^d$. In particular, asking if $p \in P$ is equivalent to asking if $H := \{x \mid p^T x \leq 1\}$ is a valid inequality for P^*. The latter can be solved by computing the point c^T in P^* that maximizes the inner product with the outer normal vector of H and test if it validates H. If this happens then $\text{SEP}_P(p)$ returns that $p \in P$, otherwise it returns $p \notin P$ with separating hyperplane $\{x \mid cx = 1\}$.

Let n denote the number of vertices of P. An oracle for P is provided by Lemma 19. Then, the entire polytope can be reconstructed, and both V- and H-representations can be found by Proposition 20.

Proposition 20. [59] *Given OPT_P for $P \subseteq \mathbb{R}^d$, its V- and H-representations as well as a triangulation T of P can be computed in*

$$O(d^5 n s^2) \text{ arithmetic operations, and } O(n + f) \text{ calls to } \text{OPT}_P,$$

where n and f are the number of vertices and facets of P, respectively, and s the number of d-dimensional simplices of T.

Corollary 21. *Given optimization oracles for $P_1, \ldots, P_r \subseteq \mathbb{R}^d$, we construct the V- and H-representations, and a triangulation T of signed Minkowski sum $P = P_1 + s_2 P_2 + \cdots + s_r P_r$, $s_i \in \{-1, 1\}$ in output sensitive complexity, namely $O(d^5 n s^2 + (n + f) r)$, where n, f are the number of vertices and facets in P and s the number of full-dimensional simplices of T.*

The output representation of the above algorithm can be exponential in n, thus we focus on total polynomial-time algorithms for the edge-skeleton of P. Note that it is not assumed that the polytopes are well-described. We assume the input contains a superset of all edges for each P_i. If, instead, we are given the vertices of all summands P_i, then we can compute all edges in each P_i by solving a linear program for each pair of vertices. Each such pair defines a candidate edge. Hence, the overall computation of P_i edges is polynomial.

Corollary 22. *Given optimization oracles for well-described $P_1, \dots, P_r \subseteq \mathbb{R}^d$, and supersets of their edge directions $D_1, \dots, D_r$, the edge-skeleton of the signed Minkowski sum P can be computed in oracle total polynomial-time by Algorithm 2.*

Proof. To be able to apply Algorithm 2, first we should show that P is well-described. Let $\langle P_{max} \rangle$ be the maximum encoding length of summands $P_1, \dots, P_r$. Then by Lemma 13, the encoding length of the coordinates of summand vertices is $4d^2\langle P_{max} \rangle$. Thus, $4d^2\langle P_{max} \rangle + \langle r \rangle$ is the encoding length of the coordinates of P vertices. Finally, $\langle P \rangle = d + 12d^4\langle P_{max} \rangle + 3d^2\langle r \rangle$ by Lemma 13. Now OPT_P is computed by Lemma 19 in $O(r)$. The superset of the edge directions of P is $D = \bigcup_{s_i > 0} D_i$, because $D(P_1 - P_2) \subset D(P_1)$ since $P_1 - P_2 = P_3 \Leftrightarrow P_1 = P_2 + P_3$. $\square$

Our algorithm assumes that, in the Minkowski difference $A - B$, B is a summand of A and does not verify this assumption.

3.4.2 Secondary and resultant polytopes

The *secondary polytope* Σ of a set of d points $A = \{p_1, \dots, p_d\} \subset \mathbb{Z}^k$ is a fundamental object since it expresses the triangulations of conv(A) via a polytope representation. For any triangulation T of conv(A), define vector $\phi_T \in \mathbb{R}^d$ with i-coordinate

$$\phi_T(i) = \sum_{\sigma \in T \ \mid \ p_i \in \text{vtx}(\sigma)} \text{vol}(\sigma), \tag{3.1}$$

summing over all simplices σ of T having p_i as a vertex, where vtx(σ) is the vertex set of simplex σ, and $i \in \{1, \dots, d\}$. Now the secondary polytope $\Sigma(\mathcal{A})$, or just Σ, is defined as the convex hull of ϕ_T for all triangulations T. A famous theorem of [74], which is also the central result in [101], states that there is a bijection between the vertices of Σ and the regular triangulations of conv(A). This extends to a bijection between the face poset of Σ and the poset of regular subdivisions of conv(A). Moreover, Σ, although in ambient space $\mathbb{R}^d$, has actual dimension $\dim(\Sigma) = d - k - 1$.

Let us now consider the Newton polytope of resultants, or *resultant polytopes*, for which optimization oracles provide today the only plausible approach for their computation [59].

Let us consider sets $A_0, \dots, A_k \subset \mathbb{Z}^k$. In the algebraic setting, these are the supports of $k+1$ polynomials in k variables. Let the Cayley set be defined by

$$A := \bigcup_{j=0}^{k} (A_j \times \{e_j\}) \subset \mathbb{Z}^{2k},$$

where $e_0, \dots, e_k$ form an affine basis of $\mathbb{Z}^k$. Clearly, each point in A corresponds to a unique point in some A_i. The (regular) triangulations of A are in bijective correspondance with the (regular) fine mixed subdivisions of the Minkowski sum $A_0 + \cdots + A_k$ [74]. Mixed subdivisions are those where all cells are Minkowski sums of convex hulls of subsets of the A_i. A mixed subdivision is fine if, for every cell, the sum of its summands' dimensions equals the dimension of the cell.

Let $d := \sum_{j=0}^{k} |A_j|$, then given triangulation T of conv(A), define vector $\rho_T \in \mathbb{R}^d$ with i-coordinate

$$\rho_T(i) := \sum_{i\text{-mixed } \sigma \in T} \mathrm{vol}(\sigma), \tag{3.2}$$

where $i \in \{1, \dots, d\}$. A simplex σ is called *i-mixed* if it contains $p_i \in A_\ell$ for some $\ell \in \{1, \dots, k\}$ and exactly 2 points from each A_j, where j ranges over $\{0, 1, \dots, k\} - \{\ell\}$. The *resultant polytope R* is defined as the convex hull of ρ_T for all triangulations T. Similarly with the secondary polytope, it is in ambient space $\mathbb{R}^d$ but has dimension $\dim(R) = d - 2k - 1$ [74]. There is a surjection, i.e. many to one relation, from the regular triangulations of conv(A) to the vertices of R.

Example 8. Let $A_0 = \{\{0\}, \{2\}\}$, $A_1 = \{\{0\}, \{1\}, \{2\}\}$, then the Cayley set will be $A = \{\{0,0\}, \{2,0\}, \{0,1\}, \{1,1\}, \{2,1\}\}$. The 5 vertices of the secondary polytope $\Sigma(A)$ are computed using equation (3.1):

$$\phi(T_1) = (2,4,2,0,4),$$
$$\phi(T_2) = (4,2,4,0,2),$$
$$\phi(T_3) = (4,2,3,2,1),$$
$$\phi(T_4) = (3,3,1,4,1),$$
$$\phi(T_5) = (2,4,1,2,3),$$

and the 3 vertices of the resultant polytope $N(R)$ are computed using equa-

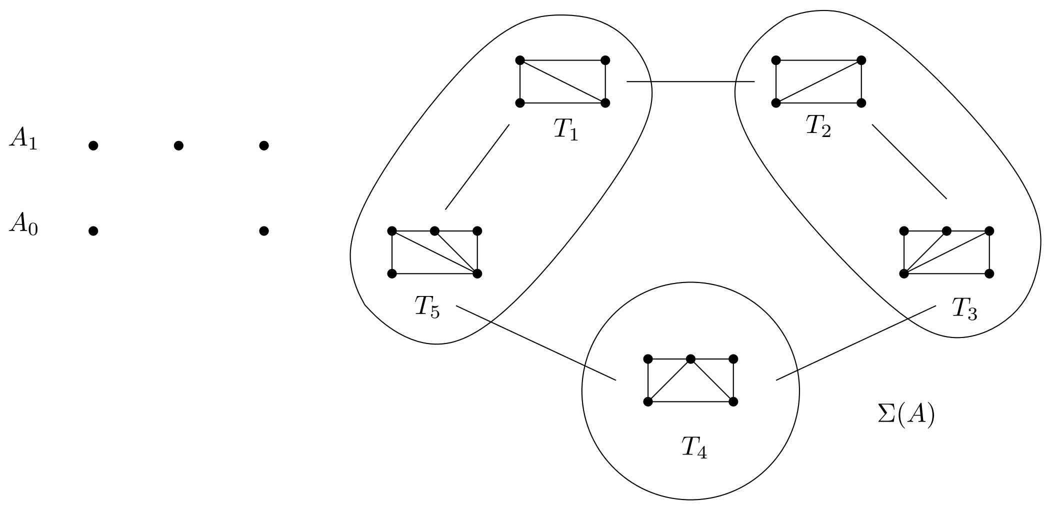


Figure 3.2: Secondary and resultant polytopes.

tion (3.2):

$$\rho(T_1) = (0, 2, 0, 0, 2),$$
$$\rho(T_2) = (2, 0, 2, 0, 0),$$
$$\rho(T_3) = (2, 0, 2, 0, 0),$$
$$\rho(T_4) = (1, 1, 0, 2, 0),$$
$$\rho(T_5) = (0, 2, 0, 0, 2).$$

Note that there are two pairs of triangulations that yield one resultant vertex each. Figure 3.2 illustrates this example.

We consider k fixed because in practice it holds $k \ll d \ll n$, where n stands for the number of polytope vertices. Note that R is computed as a full-dimensional polytope in a space of its intrinsic dimension [59] and this approach extends to Σ.

Computing the V-representation of Σ and R by the algorithm in [59] is not total polynomial. In fact, the complexity depends on the number of polytope vertices and facets, but also the number of simplices in a triangulation of the polytope (see Proposition 20). However, we show that Algorithm 2 computes Σ and R in oracle total polynomial-time. Our results readily extend to the Newton polytope of discriminants, or discriminant polytopes, discussed in [74].

Lemma 23. *Both Σ and R are well-described polytopes.*

Proof. For the case of Σ, given $A \in \mathbb{Z}^k$, let $\langle A\rangle$ be its encoding length and $\alpha :=$ vol(conv(A)). It is clear that $\alpha = O(\langle A\rangle^k)$ and thus $\langle\alpha\rangle = O(k\langle A\rangle)$. For each triangulation T each coordinate of ϕ_T is upper bounded by α, since the sum of the volumes of its adjacent simplices cannot exceed vol(conv(A)). This bound is tight for the points $a \in A$ of a regular triangulation T where the simplices containing a partition conv(A). It follows that the encoding length of Σ vertices is $\langle\alpha\rangle$ and thus $\langle\Sigma\rangle = 4n^2\langle\alpha\rangle + d = O(dn^2\langle A\rangle)$ by Lemma 13. Similarly, we bound the encoding length of ρ_T which yields that R is also a well-described polytope. □

In the sequel, we characterize the set of edge directions of Σ and R. The edge directions of both Σ, R can be computed by enumerating circuits of A. More specifically, circuit enumeration suffices to compute the *edge vectors*, i.e. both directions and lengths of the edges.

We first give some fundamental definitions from combinatorial geometry. For a detailed presentation we recommend [101]. A *circuit* $C \subseteq A$ is a minimum affinely dependent subset of A. It holds that conv(C) has exactly two triangulations C_+, C_-. The operation of switching from one triangulation to another is called *flip*. Triangulation T of A, which equals C_+ when restricted on circuit C, is supported on C if, by flipping C_+ to C_-, we obtain another triangulation T' of A. The dimension of a circuit is the dimension of its convex hull. If A is in *generic position*, then all circuits C are full dimensional. Then all the edges of Σ correspond to full dimensional circuits. If A is *not* in generic position, some edges may correspond to lower-dimensional circuits.

In the case of R, where $A = \bigcup_{j=0}^{k} A_j$, a circuit C is called *cubical* if and only if $|C \cap A_j| \in \{0, 2\}$, $j = 0, \dots, k$. If A is in *generic position*, all the edges of R correspond to full dimensional cubical circuits [131].

Lemma 24. *Given $A \in \mathbb{Z}^k$ in generic position, we compute the set of edge directions of Σ in $O(d^{k+2})$. Given $A \in \mathbb{Z}^{2k}$ in generic position the set of edge directions of R is computed in $O(d^{2k+2})$. In both cases, genericity of A is checked within the respective time complexity.*

Proof. For Σ, we enumerate all $\binom{|A|}{k+2}$ circuits in $O(d^{k+2})$, obtaining the set of all edge vectors. Genericity of A is established by checking whether all $\binom{|A|}{k}$ subsets, $k \in \{1, \dots, k+1\}$, are independent. This is in $O(d^{k+1})$ for $k = O(1)$.

In the case of R, where $A = \bigcup_{j=0}^{k} A_j$, a flip on T is cubical iff it is supported on a cubical circuit C.In generic position, $|C| = 2k + 2$. For those supporting cubical flips, $|C \cap A_j| = 2$, $j = 0, \ldots, k$. Every edge d_C of R is supported on cubical flip C, where $d_C(a)$ equals $\rho_{C_+}(a) - \rho_{C_-}(a)$, if $a \in C$, and 0 otherwise [131]. Given A, all such circuits are enumerated in $\binom{|A|}{2k+2} = O(d^{2k+2})$; a better bound is $O(t^{2k+2})$ if t bounds $|A_j|$, $j = 0, \ldots, k$. □

Lemma 25. [59] *For $k+1$ pointsets in $\mathbb{Z}^k$ of total cardinality d, optimization over R takes polynomial-time, when k is fixed.*

Corollary 26. *In total polynomial-time, we compute the edge-skeleton of $\Sigma \subset \mathbb{R}^d$, given $A \in \mathbb{Z}^k$ in generic position, and the edge-skeleton of R, given $A \in \mathbb{Z}^{2k}$ in generic position.*

Proof. Since by Lemma 23 Σ, R are well-bounded, optimization oracles are available by Lemma 25 and the set of edge directions by Lemma 24, the edge-skeletons of Σ, R can be computed by Algorithm 2 in oracle total polynomial-time. Moreover, since the optimization oracle is polynomial-time this yields a (proper) total polynomial-time algorithm for Σ, R. □

Follwoing Lemma 24, for Σ, R we also obtain their edge lengths. This can lead to a more efficient edge-skeleton algorithm on the real RAM.

3.5 Concluding remarks

We have presented the first total polynomial-time algorithm for computing the edge-skeleton of a polytope, given an optimization oracle, and a set of directions that contains the polytope's edge directions. Our algorithm yields the first (weakly) total polynomial-time algorithms for the edge-skeleton (and vertex enumeration) of signed Minkowski sum, and resultant polytopes.

An open question is a *strongly* total polynomial-time algorithm for the edge-skeleton problem. Another is to solve the edge-skeleton problem without edge directions; characterizations of edge directions for polytopes in H-representation are studied in [116]. It is also interesting to investigate new classes of convex combinatorial optimization problems where our algorithm offers a polynomial-time algorithm.

Chapter 4

Algorithms for polytope volume approximation

4.1 Introduction

A fundamental problem in discrete and computational geometry is to compute the volume of a convex body in general dimension or, more particularly, of a polytope. In the past 15 years, randomized algorithms for this problem have witnessed a remarkable progress. Starting with the breakthrough poly-time algorithm of [50], subsequent results brought down the exponent on the dimension from 27 to 4 [104]. However, the question of an efficient implementation had remained open.

Notation. Convex bodies are typically given by a membership oracle. A polytope $P \subseteq \mathbb{R}^d$ can also be represented as the convex hull of vertices (V-polytope) or, as is the case here, as the (bounded) intersection

$$P := \{x \in \mathbb{R}^d \mid Ax \leq b\}$$

of m halfspaces given by $A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$ (H-polytope); ∂P is its boundary, and $O^*(\cdot)$ hides polylog factors in the argument. The input includes approximation factor $\epsilon > 0$; W denotes the most important runtime parameter, namely random walk length.

Previous work. Volume computation is #-P hard for V- and for H-polytopes [51]. Several exact algorithms are surveyed in [30] and implemented in `VINCI` [29], which however cannot handle general polytopes for dimension $d > 15$. An interesting challenge is the volume of the n-Birkhoff polytope, computed only for $n \leq 10$ using highly specialized software (Sect. 4.4). Regarding deterministic approximation, no poly-time algorithm can compute the volume with less than exponential relative error [54]. The algorithm of [18] has error $\leq d!$.

The landmark randomized poly-time algorithm in [50] approximates the volume of a convex body with high probability and arbitrarily small relative error. The best complexity, as a function of d, given a membership oracle, is $O^*(d^4)$ oracle calls [104]. All approaches except [104] define a sequence of co-centric balls, and produce uniform point samples in their intersections with P to approximate the volume of P.

Concerning existing software (cf Sect. 4.5), [43] presented recently `Matlab` code based on [104] and [42]. The latter offers a randomized algorithm for Gaussian volume (which has no direct reduction to or from volume) in $O^*(d^3)$, as a function of d. In [102] they implement [104], focusing on variance-decreasing techniques, and an empirical estimation of mixing time. In [99], they use a straightforward acceptance-rejection method, which is not expected to work in high dimension; it was tested only for $d \leq 4$. An approach using thermodynamic integration [84] offers only experimental guarantees on runtime and accuracy.

The key ingredient of all approaches is random walks that produce an almost uniform point sample. Such samples is a fundamental problem of independent interest with important applications in, e.g., global optimization, statistics, machine learning, Monte Carlo (MC) integration, and non-redundant constraint identification. Several questions of sampling combinatorial structures such as contingency tables and more generally lattice points in polytopes may be reduced to sampling a polytope.

No simple sampling method exists unless the body has standard shape, e.g., simplex, cube or ellipsoid. Acceptance-rejection techniques are inefficient in high dimensions. E.g., the number of uniform points one needs to generate in a bounding box before finding one in P is exponential in d. A Markov chain is the only known method, and it may use geometric random walks such as the grid walk, the ball walk (or variants such as the Dikin walk), and Hit-and-run [129]. The Markov chain has to make a (large) number of steps, before the generated point becomes distributed approximately uniformly (which is the stationary limit distribution of the chain). We focus on Hit-and-run which yields the fastest algorithms today.

In contrast to other walks, Hit-and-run is implemented by computing the intersection of a line with ∂P. In general, this reduces to binary search on the line, calling membership at every step. For H-polytopes, the intersection is obtained by a *boundary oracle*; for this, we employ ray-shooting with respect to the m facet hyperplanes (Sect. 4.2). In exact form, it is possible to avoid linear-time queries by using space in $o(m^{\lfloor d/2 \rfloor})$, achieving queries in $O(\log m)$ [121]. Duality reduces oracles to (approximate) ε-nearest neighbour queries, which take $O(dm^{(1+\varepsilon)^{-2}+o(1)})$ using $O(dm + m^{1+(1+\varepsilon)^{-2}+o(1)})$ space by locality sensitive hashing [3]. Moreover, space-time tradeoffs from $O(1/\varepsilon^{(d-O(1))/8})$ time and $O(1/\varepsilon^{(d-O(1))/2})$ space to $O(1)$ time and $O(1/\varepsilon^{(d-O(1))})$ space are available by [5]. Approximate oracles are also connected to polytope approximation. Classic results, such as Dudley's, show that $O((1/\varepsilon)^{(d-1)/2})$ facets suffice to approximate a convex body of unit diameter within a Hausdorff distance of ε. This is optimized to $O(\sqrt{\text{vol}(\partial P)}/\varepsilon^{(d-1)/2})$ [5]. The boundary oracle is dual to finding the extreme point in a given direction among a known pointset. This is ε-approximated through ε-coresets for measuring extent, in particular (directional) width, but requires a subset of $O((1/\varepsilon)^{(d-1)/2})$ points [2]. The exponential dependence on d or the linear dependence on m make all aforementioned methods of little practical use. Ray shooting has been studied in practice only in low dimensions, e.g., in 6-dimensional V-polytopes [139].

Contribution. We implement and experimentally study efficient algorithms for approximating the volume of polytopes. Point sampling, which is the bottleneck of these algorithms, is key in achieving poly-time complexity and high accuracy. To this end, we study variants of Hit-and-run. It is widely believed that the theoretical bound on W is quite loose, and this is confirmed by our experiments, where we set $W = O(d)$ and obtain a $< 2\%$ error in up to 100 dimensions (Sect. 4.4).

Our emphasis is to exploit the underlying geometry. Our algorithm uses the recursive technique of co-centric balls (cf. Sect. 4.3) introduced in [50] and used in a series of papers, with the most recent to be [91]. This technique forms a sequence of *diminishing* radii which, unlike previous papers, allowing us to only sample *partial generations* of points in each intersection with P, instead of sampling N points for each. In fact, the algorithm starts with computing the largest interior ball by an LP. Unlike most theoretical approaches, that use an involved rounding procedure, we sample a set of points in P and compute the minimum enclosing ellipsoid of this set, which is then linearly transformed to a ball. This procedure is repeated until the ratio of the minimum over the maximum ellipsoid axes reaches some user-defined threshold. This *iterative rounding* allows us to handle skinny polytopes efficiently.

We study various oracles (Sect. 4.2). Line search using membership requires $O(md + \log \frac{r}{\epsilon_s})$ arithmetic operations. This is improved to a boundary oracle in $O(md)$ by avoiding membership. Using Coordinate Direction Hit-and-run, we further improve the oracle to $O(m)$ amortized complexity. We also exploit duality to reduce the oracle to ε-nearest neighbour search: although the asymptotic complexity is not improved, for certain instances such as cross-polytopes in $d = 16$, kd-trees achieve a 40x speed-up.

Our C++ code is open-source (sourceforge) and uses the CGAL library. A series of experiments establishes that it handles dimensions substantially larger than existing exact approaches, e.g., cubes and products of simplices within an error of 2% for $d \leq 100$, in about 20 min. Compared to approximate approaches, it computes significantly more accurate results. It computes in few hours volume estimations within an error of 2% for Birkhoff polytopes $\mathcal{B}_2, \dots, \mathcal{B}_{10}$; vol($\mathcal{B}_{10}$) has been exactly computed by specialized parallel software in a sequential time of years. More interestingly, it provides volume estimations for vol($\mathcal{B}_{11}$),...,vol($\mathcal{B}_{15}$), whose exact values are unknown, within 9 hours. In conclusion, we claim that the volume of general H-polytopes in high dimensions (e.g. one hundred) can be efficiently and accurately approximated on standard computers.

Paper organization. The next section discusses walks and oracles. Sect. 4.3 presents the overall volume algorithm. Sect. 4.4 discusses our experiments, and we conclude with open questions in Sect. 4.5.

4.2 Random walks and Oracles

This section introduces the paradigm of Hit-and-run walks and focuses on their implementation, with particular emphasis on exploiting the geometry of H-polytopes. The methods presented here are analysed experimentally in Sect. 4.4.

Hit-and-run random walks. The main method to randomly sample a polytope is by (geometric) random walks. We shall focus on variants of Hit-and-run, which generate a uniform distribution of points [130]. Assume we possess procedure Line(p), which returns line ℓ through point $p \in P \subseteq \mathbb{R}^d$; ℓ will be specified below. The main procedure of Hit-and-run is Walk(p, P, W), which reads in point $p \in P$ and repeats W times: (i) run Line(p), (ii) move p to a random point uniformly distributed on $P \cap \ell$. We shall consider two variants of Hit-and-run.

In *Random Directions Hit-and-run* (RDHR), Line(p) returns ℓ defined by a random vector uniformly distributed on the unit sphere centered at p. The vector coordinates are drawn from the standard normal distribution. RDHR generates a uniformly distributed point in

$$\begin{gathered} O^*(d^2r^2), \text{ or } O^*(d^3r^2) \text{ oracle calls,} \\ \text{with hidden constants } 10^{30}, \text{ or } 10^{11} \text{ respectively,} \end{gathered} \tag{4.1}$$

starting at an arbitrary, or at a uniformly distributed point (also known as warm start), respectively, where r is the ratio of the radius of the smallest enclosing ball over that of the largest enclosed ball in P [103].

In *Coordinate Directions Hit-and-run* (CDHR), Line(p) returns ℓ defined by a random vector uniformly distributed on the set $\{e_1, \ldots, e_d\}$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, $i = 1, \ldots, d$. This is a continuous variant of the Grid walk. As far as the authors know, the mixing time has not been analyzed. We offer experimental evidence that CDHR is faster than RDHR and sufficiently accurate. An intermediate variant is Artificially Centering Hit-and-run [93], where first a set S of sample points is generated as with RDHR, then Line(p) returns ℓ through p and a randomly selected point from S. This however is not a Markov chain, unlike CDHR and RDHR.

Procedure Walk(p, P, W) requires at every step an access to a *boundary oracle* which computes the intersection of line ℓ with ∂P. In the sequel we discuss various implementations of this oracle.

Boundary oracle by membership. For general convex bodies, a boundary oracle can be implemented using a *membership oracle* which, given vector $y \in \mathbb{R}^d$, decides whether $y \in P$. The intersection of ℓ with ∂P is computed by binary search on the segment defined by any point on ℓ lying in the body and the intersection of ℓ with a bounding ball. Each step calls membership to test whether the current point is internal, and stops when some accuracy ϵ_s is certified. Checking the point against a hyperplane takes $O(d)$ operations, thus obtaining the intersection of ℓ with the hyperplane. We store this intersection so that subsequent tests against this hyperplane take $O(1)$. The total complexity is $O(md + \log \frac{r}{\epsilon_s})$ arithmetic operations, where r is the ball radius.

Boundary oracle by facet intersection. Given an H-polytope P the direct method to compute the intersection of line ℓ with ∂P is to examine all m hyperplanes. Let us consider $\texttt{Walk}(p_0, P, W)$ and line $\ell = \{x \in \mathbb{R}^d : x = \lambda v + p_0\}$, where $p_0 \in \mathbb{R}^d$ lies on ℓ, and v is the direction of ℓ. We compute the intersection of ℓ with the i-th hyperplane $a_i x = b_i$, $a_i \in \mathbb{R}^d, b_i \in \mathbb{R}$, namely $p_i := p_0 + \frac{b_i - a_i p_0}{a_i v} v$, $i \in \{1, \dots, m\}$. We seek points p^+, p^- at which ℓ intersects ∂P, namely $p^+ v = \min_{1 \le i \le m}\{p_i v \mid p_i v \ge 0\}$ and $p^- v = \max_{1 \le i \le m}\{p_i v \mid p_i v \le 0\}$. This is computed in $O(md)$ arithmetic operations. In practice, only the $\lambda^{\pm}$ are computed, where $p^{\pm} = p_0 + \lambda^{\pm} v$.

In the context of the volume algorithm (Sect.4.3), the intersection points of ℓ with ∂P are compared to the intersections of ℓ with the current sphere. Assuming the sphere is centered at the origin with radius R, its intersections with ℓ are $p = p_0 + \lambda v$ such that $\lambda^2 + 2\lambda p_0 v + |p_0|^2 - R^2 = 0$. If λ^+, λ^- give a negative sign when substituted to the aforementioned equation then p^+, p^- are the endpoints of the segment of ℓ lying in the intersection of P and the current ball. Otherwise, we have to compute one or two roots of the aforementioned equation since the segment has one or two endpoints on the sphere.

However, in CDHR, where ℓ and v are vertical, after the computation of the first pair p^+, p^-, all other pairs can be computed in $O(m)$ arithmetic operations. This is because two sequential points produced by the walk differ only in one coordinate. Let j, k be the walk coordinate of the previous and the current step respectively. Then, assuming $P = \{x \in \mathbb{R}^d : Ax \le b\}$, where $A \in \mathbb{R}^{m \times d}$, $\lambda^{\pm} = \max\{\lambda \mid A(p_0 \pm \lambda v) \le b\}$. This becomes $\pm\lambda A v = \pm\lambda A_j \le -Ap_0 + b$, where A_j is the j-th column of A. The two maximizations are solved in $O(md)$ ops. Let vector $t = -Ap_0 + b \in \mathbb{R}^m$. At the next step, given point $p_0' = p_0 + c e_j$, where e_j is the j-th standard basis vector, we perform two maximizations $\lambda : \pm\lambda A_k \le t - cA_j$ in $O(m)$.

Boundary oracle by duality. Duality reduces the problem to nearest neighbour (NN) search and its variants. Given a pointset $B \subseteq \mathbb{R}^d$ and query point q, NN search returns a point $p \in B$ s.t. $dist(q, p) \le dist(q, p')$ for all $p' \in B$, where $dist(q, p)$ is the Euclidean distance between points q, p. Let us consider, w.l.o.g., boundary intersection for line ℓ parallel to the x_d-axis: $\ell = \{x : x = \lambda v + p,\ \lambda \geq 0\}$, $v = (0, \ldots, 0, -1)$. It reduces to two ray-shooting questions; it suffices to describe one, namely with the upward vertical ray, defined by $\lambda \leq 0$. We seek the first facet hyperplane hit which, equivalently, has the maximum negative signed vertical distance from p to any hyperplane H of the upper hull, for fixed v. This distance is denoted by $\mathrm{sv}(p, H)$. Let us consider the standard (aka functional) duality transform between points p and non-vertical hyperplanes H:

$$p = (p_1, \ldots, p_d) \mapsto p^* : x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} - p_d,$$
$$H : x_d = c_1 x_1 + \cdots + c_{d-1} x_{d-1} + c_0 \mapsto H^* = (c_1, \ldots, c_{d-1}, -c_0).$$

This transformation is self-dual, preserves point-hyperplane incidences, and negates vertical distance, hence $\mathrm{sv}^*(p^*, H^*) = -\mathrm{sv}(p, H)$, where $\mathrm{sv}^*(\cdot,\cdot)$ is the signed vertical distance from hyperplane p^* to point H^* in dual space. Hence, our problem is equivalent to minimizing $\mathrm{sv}^*(p^*, H^*) \geq 0$. Equivalently, we seek point H^* minimizing absolute vertical distance to hyperplane p^* on its side of positive distances. In dual space, consider

$$\begin{aligned} &\text{point } t = (t_1, \ldots, t_d), \text{ and hyperplane} \\ &p^* = q : x_d = q_1 x_1 + \cdots + q_{d-1} x_{d-1} + q_0 : \end{aligned} \tag{4.2}$$

$$\begin{aligned} \mathrm{sv}^*(q, t) &= t_d - (q_1 t_1 + \cdots + q_{d-1} t_{d-1} + q_0) \\ &= -(q_0, q_1, \ldots, q_{d-1}, -1) \cdot (1, t_1, \ldots, t_{d-1}, t_d), \end{aligned}$$

where the latter operation is inner product in Euclidean space $\mathbb{R}^{d+1}$ of "lifted" datapoint $t' = (1, t_1, \ldots, t_{d-1}, t_d)$ with "lifted" query point $q' = (q_0, q_1, \ldots, q_{d-1}, -1)$. Let

$$q'' = (q', 0),\ t'' = (t', \sqrt{M - \|t'\|_2^2}),\ \text{ for } M \geq \max_t \{1 + \|t\|_2^2\},$$

following an idea of [13]. By the cosine rule,

$$\mathrm{dist}_{d+2}^2(q'', t'') = \|q'\|_2^2 + M + 2\mathrm{sv}^*(q, t),$$

where $\mathrm{dist}_{d+2}(\cdot,\cdot)$ stands for Euclidean distance in $\mathbb{R}^{d+2}$. Since the t'' lie on hyperplane $x_1 = 1$, optimizing $\mathrm{dist}_{d+2}(q'', t'')$ over a set of points t'' is equivalent to optimizing $\mathrm{dist}_{d+1}(\hat{q},\hat{t})$, $\hat{q}=(q_1,\dots,q_{d-1},-1,0)$, over points $\hat{t}=(t,\sqrt{M-1-\|t\|_2^2})$. Hence, point t minimizing $\mathrm{sv}^*(q,t)\geq 0$ corresponds to $\hat{t}$ minimizing $\mathrm{dist}^2_{d+1}(\hat{q},\hat{t})$. Thus the problem is reduced to (exact) nearest neighbor in $\mathbb{R}^{d+1}$. Ray shooting to the lower hull with same v reduces to farthest neighbor. Unfortunately, an approximate solution to these problems incurs an additive error to the corresponding original problem.

Alternatively, we shall consider hyperplane queries. Let us concentrate on hyperplanes supporting facets on the lower hull of P. Their dual points lie in convex position. Given that point p is interior in P, the dual points of the lower hull facets lie on the upper halfspace of p^*. In dual space, consider point t and hyperplane q as in expression (4.2). Let $\mathrm{sd}^*(q,t)$ be the signed Euclidean distance from q to t, i.e. the minimum Euclidean distance of any point on q to t. Then $\mathrm{sv}^*(q,t)=\mathrm{sd}^*(q,t)/\|(q_1,\dots,q_{d-1},1)\|_2$, where the normal is $(q_1,\dots,q_{d-1},1)$. Our question, therefore, becomes equivalent to minimizing $\mathrm{sd}^*(q,t)$ over all datapoints $t\in\mathbb{R}^d$ for which $\mathrm{sd}^*(q,t)\geq 0$; i.e., we seek the NN above q. Starting with facets on the upper hull, the problem becomes that of maximizing $\mathrm{sd}^*(q,t)\leq 0$, i.e. finding the NN below q.

The above approaches motivate us to use NN software for exact point and hyperplane queries (Sect. 4.4).

4.3 The volume algorithm

This section details our poly-time methods for approximating the volume of P. Algorithms in this family are the current state-of-the-art with respect to asymptotic complexity bounds. Moreover, they can achieve any approximation ratio given by the user, i.e., they form a fully polynomial randomized approximation scheme (FPRAS). Given polytope $P\subseteq\mathbb{R}^d$, they execute *sandwiching* and *Multiphase Monte Carlo* (MMC) [129].

We consider that P is a full-dimensional H-polytope. However, we can also consider P to be lower dimensional and be given in form $\{x\in\mathbb{R}^d\,|\,Ax=b,\ x\geq 0\}$, where $A\in\mathbb{R}^{m\times d}$, $x\in\mathbb{R}^d$, $b\in\mathbb{R}^m$, $A'\in\mathbb{R}^{m\times m-d+1}$, $x'\in\mathbb{R}^{m-d+1}$. Using *Gauss-Jordan elimination* the linear system $Ax=b$ can be transformed to its unique *reduced row echelon form* $[I|A']x=b'$, where I is the identity matrix. Then P can be written as $\{x' \in \mathbb{R}^{m-d+1} \mid A'x' \geq b', x' \geq 0\}$, i.e. a full-dimensional H-polytope in $\mathbb{R}^{m-d+1}$.

Rounding and sandwiching. This stage involves first rounding P to reach a near isotropic position, second sandwiching, i.e. to compute ball B and scalar ρ such that $B \subseteq P \subseteq \rho B$. There is an abundance of methods in literature for rounding and sandwiching (cf. [129] and references therein). However, here we develop a simple, efficient method that succeed significantly accurate results in practice (cf. Sect. 4.4 and Table 4.5). The method doesn't compute a ball that covers P but a ball B' such that $B' \cap P$ contains almost all the volume of P.

For rounding, we sample a set S of $O(n)$ random points in P. Then we approximate the minimum volume ellipsoid $\mathcal{E}$ that covers S, and satisfies the inclusions $\frac{1}{(1+\varepsilon)d}\mathcal{E} \subseteq \text{conv}(S) \subseteq \mathcal{E}$, in time $O(nd^2(\varepsilon^{-1} + \ln d + \ln\ln n))$ [97]. Let us write

$$\begin{aligned}\mathcal{E} &= \{x \in \mathbb{R}^d \mid (x - c_{\mathcal{E}})^T E\, (x - c_{\mathcal{E}}) \leq 1\} \\ &= \{x \in \mathbb{R}^d \mid L^T(x - c_{\mathcal{E}}) \leq 1\},\end{aligned} \tag{4.3}$$

where $E \subseteq \mathbb{R}^{d\times d}$ is a positive semi-definite (p.s.d.) matrix and $L^T L$ its Cholesky decomposition. By substituting $x = (L^T)^{-1}y + c_{\mathcal{E}}$ we map the ellipsoid to the ball $\{y \in \mathbb{R}^d \mid y^T y \leq 1\}$. Applying this transformation to P we have $P' = \{y \in \mathbb{R}^d \mid A(L^T)^{-1} \leq b - Ac_{\mathcal{E}}\}$ which is the rounded polytope, where $\text{vol}(P) = \det(L^T)^{-1}\text{vol}(P')$. We iterate this procedure until the ratio of the minimum over the maximum ellipsoid axes reaches some user defined threshold.

For sandwiching P we first compute the *Chebychev ball* $B(c, r)$ of P, i.e. the largest inscribed ball in P. It suffices to solve the LP: $\{\text{maximize } R, \text{ subject to: } A_i x + R\|A_i\|_2 \leq b_i,\ i = 1, \dots, m,\ R \geq 0\}$, where A_i is the i-th row of A, and the optimal values of R and $x \in \mathbb{R}^d$ yield, respectively, the radius r and the center c of the Chebychev ball.

Then we may compute a uniform random point in $B(c, r)$ and use it as a start to perform a random walk in P, eventually generating N random points. Now, compute the largest distance between each of the N points and c; this defines a (approximate) bounding ball. Finally, define the sequence of balls $B(c, 2^{i/d})$, $i = \alpha, \alpha + 1, \dots, \beta$, where $\alpha = \lfloor d \log r \rfloor$ and $\beta = \lceil d \log \rho \rceil$.

Multiphase Monte Carlo (MMC). MMC constructs a sequence of bodies $P_i :=$

$P \cap B(c, 2^{i/d})$, $i = \alpha, \alpha+1, \dots, \beta$, where $P_\alpha = B(c, 2^{\alpha/d}) \subseteq B(c, r)$ and P_β (almost) contains P. Then it approximates vol(P) by the telescopic product

$$\text{vol}(P_\alpha) \prod_{i=\alpha+1}^{\beta} \frac{\text{vol}(P_i)}{\text{vol}(P_{i-1})}, \text{ where } \text{vol}(P_\alpha) = \frac{2\pi^{d/2}(2^{\lfloor \log r \rfloor})^d}{d\,\Gamma(d/2)}.$$

This reduces to estimating the ratios vol(P_i)/vol(P_{i-1}), which is achieved by generating N uniformly distributed points in P_i and by counting how many of them fall in P_{i-1}.

For point generation we use random walks as in Sect. 4.2. We set the walk length $W = \lfloor 10 + d/10 \rfloor = O(d)$, which is of the same order as in [102] but significantly lower than theoretical bounds. This choice is corroborated experimentally (Sect. 4.4).

Unlike typical approaches, which generate points in P_i for $i = \alpha, \alpha+1, \dots, \beta$, here we proceed inversely. First, let us describe initialization. We generate an (almost) uniformly distributed random point $p \in P_\alpha$, which is easy since $P_\alpha = B(c, 2^{\alpha/d}) \subseteq B(c, r)$. Then we use p to start a random walk in $P_\alpha, P_{\alpha+1}, P_{\alpha+2}$ and so on, until we obtain a uniformly distributed point in P_β. We perform N random walks starting from this point to generate N (almost) uniformly distributed points in P_β and then count how many of them fall into $P_{\beta-1}$. This yields an estimate of vol(P_β)/vol$(P_{\beta-1})$. Next we keep the points that lie in $P_{\beta-1}$, and use them to start walks so as to gather a total of N (almost) uniformly distributed points in $P_{\beta-1}$. We repeat until we compute the last ratio vol$(P_{\alpha+1})$/vol(P_α).

The implementation is based on a data structure S that stores the random points. In step $i > \alpha$, we wish to compute vol$(P_{\beta-i})$/vol$(P_{\beta-i-1})$ and S contains N random points in $P_{\beta-i+1}$ from the previous step. The computation in this step consists in removing from S the points not in $P_{\beta-i}$, then sampling $N - size(S)$ new points in $P_{\beta-i}$ and, finally, counting how many lie in $P_{\beta-i-1}$. Testing whether such a point lies in some P_i reduces to testing whether $p \in B(2^{i/d})$ because $p \in P$.

One main advantage of our method is that it creates partial generations of random points for every new body P_i, as opposed to having always to generate N points. This has a significant effect on runtime since it reduces it by a constant raised to β. Partial generations of points have been used in convex optimization [17].

Algorithm 3: VolEsti (P, ϵ, t_r)

```
Input  : H-polytope $P$, objective approximation $\epsilon$, rounding threshold $t_r$
Output : approximation of vol($P$)
$N \leftarrow 400\epsilon^{-2} d \log d$;   $W \leftarrow \lfloor 10 + d/10 \rfloor$;
// rounding and sandwiching
compute the Chebychev ball $B(c, r)$;
generate a random point $p$ in $B(c, r)$;
repeat
    $S \leftarrow \emptyset$;
    for $i = 1$ to $N$ do
        $p \leftarrow \texttt{Walk}(p, P, W)$;
        add $p$ in $S$;
    compute min encl. ellipsoid $\mathcal{E}$ of $S$, with p.s.d. $E$;
    set as $\mathcal{E}min, \mathcal{E}max$ the min and max $\mathcal{E}$ axes;
    compute the Cholesky decomposition $L^T L$ of $E$;
    transform $P$ and $p$ w.r.t. $L$;
until $\mathcal{E}max/\mathcal{E}min < t_r$;
set $\rho$ the largest distance from $c$ to any point in $S$;
// MMC
set $\alpha \leftarrow \lfloor \log r \rfloor$;   $\beta \leftarrow \lceil \log \rho \rceil$;
$P_i \leftarrow P \cap B(c, 2^{i/d})$ for $i = \alpha, \alpha + 1, \ldots, \beta$;
vol($P_\alpha$) $\leftarrow 2\pi^{d/2}(2^{\lfloor \log r \rfloor})^d / d\Gamma(d/2)$;
$i \leftarrow \beta$;
while $i > \alpha$ do
    $P_{\text{large}} \leftarrow P_i$;   $i \leftarrow i - 1$;   $P_{\text{small}} \leftarrow P_i$;
    $count\_prev \leftarrow size(S)$;   remove from $S$ the points not in $P_{\text{small}}$;
    $count \leftarrow size(S)$;
    Set $p$ to be an arbitrary point from $S$;
    for $j = 1$ to $N - count\_prev$ do
        $p \leftarrow \texttt{Walk}(p, P_{\text{large}}, W)$;
        if $p \in B(c, 2^{i/d})$ then
            $count \leftarrow count + 1$;
            add $p$ in $S$;
    $vol \leftarrow vol \cdot (N/count)$;
return vol/ $\det(L^T)$ ;
```

We use *threads*, also in [102], to ensure independence of the points. A thread is a sequence of points each generated from the previous point in the sequence by a random walk. The first point in the sequence is uniformly distributed in the ball inscribed in P. Alg. 3 describes our algorithm using a single thread.

Complexity. The first $O^*(d^5)$ algorithm was in [91], using a sequence of subsets defined as the intersection of the given body with a ball. It uses isotropic sandwiching to bound the number of balls by $O^*(d)$, it samples $N = 400\epsilon^{-2} d \log d = O^*(d)$ points per ball, and follows a ball walk to generate each point in $O^*(d^3)$ oracle calls. Interestingly, both sandwiching and MMC each require $O^*(d^5)$ oracle calls. Later the same complexity was obtained by Hit-and-run under the assumption the convex body is well sandwiched.

Proposition 27. [91] *Assuming $B(0,1) \subseteq P \subseteq B(0,\rho)$, the volume algorithm of [91] returns an estimation of vol(P), which lies between $(1-\epsilon)$vol(P) and $(1+\epsilon)$vol(P), with probability $\geq 3/4$, by*

$$O\left(\frac{d^4\rho^2}{\epsilon^2} \ln d \ln \rho \ \ln^2 \frac{d}{\epsilon}\right) = O^*(d^4\rho^2)$$

oracle calls with probability $\geq 9/10$, where we have assumed ϵ is fixed. Sandwiching yields $\rho = \sqrt{d/\log(1/\epsilon)}$, implying a total of $O^*(d^5)$ calls.

In [104], they construct a sequence of log-concave functions and estimate ratios of integrals, instead of ratios of balls, using simulated annealing. The complexity reduces to $O^*(d^4)$ by decreasing both number of phases and number of samples per phase to $O^*(\sqrt{d})$. Using Hit-and-run, $O^*(d^3)$ still bounds the time to sample each point. Moreover, they improve isoperimetric sandwiching to $O^*(d^4)$.

The following Lemma states the runtime of Alg. 3, which is in fact a variant of the algorithm analysed in [91] (see also Prop. 27). Although there is no theoretical bound on the approximation error of Alg. 3, our experimental analysis in Sect. 4.4 shows that in practice the achieved error is always better than the one proved in Prop. 27.

Lemma 28. *Given H-polytope P, Alg. 3 performs k phases of rounding in $O^*(d^3mk)$, and approximates vol(P) in $O(md^3 \log d \log(\rho/r))$ arithmetic operations, assuming $\epsilon > 0$ is fixed, where r and ρ denote the radii of the largest inscribed ball and of the co-centric ball covering P.*

Proof. Our approach generates $d\log(\rho/r)$ balls and uses Hit-and-run. Assuming P contains the unit ball, an upper bound on ρ/r is diameter δ. In each ball intersected with P, we generate $\leq N = 400\epsilon^{-2} d\log d$ random points. Each point is computed after $W = O(d)$ steps of CDHR.

The boundary oracle of CDHR is implemented in Sect. 4.2. In particular, k CDHR steps require $O(dm+(k-1)m+kd)$ arithmetic operations. It holds $d = O(m)$ and $k = \Omega(d)$. Thus, the amortized complexity of a CDHR step is $O(m)$. Overall, the algorithm needs $O(\epsilon^{-2}md^3 \log d \log(\rho/r))$ operations.

Each rounding iteration decreases δ and runs in $O(nd^2(\varepsilon^{-1} + \ln d + \ln\ln(n)))$, where n stands for the number of sampled points, and ε is the approximation of the minimum volume ellipsoid of Eq. (4.3). We generate $n = O(d)$ points, each in $O(m)$ arithmetic operations. Hence, rounding runs in $O^*(d^3mk)$, where ε is fixed. Moreover, k is typically constant since $k = 1$ is enough to handle, e.g., polytopes with $\rho/r = 100$ in dimension up to 20. □

Let us check this bound with the experimental data for cubes, products of simplices, and Birkhoff polytopes, with $d \leq 100$ and $\epsilon = 1$, where $m = 2d$, $d+2$ and $d+1+2\sqrt{d}$, respectively, for the 3 classes, and for cubes $\log(\rho/r) \leq \log(\sqrt{d}) = O(\log d)$. Fig. 4.1 shows that the 3 classes behave similarly. Performing a fit of $ad^b \log^2 d$, runtime follows $10^{-5}d^{3.08}\log^2 d$ which shows a smaller dependence on d than our bounds, at this range of experiments.

4.4 Experiments

We implement and experimentally test the above algorithms and methods in the software package `VolEsti`. The code currently consists of around 2.5K lines in C++ and is open-source[1]. It relies on the `CGAL` library [35] for its d-dimensional kernel to represent objects such as points and vectors, for its LP solver [64], for the approximate minimum ellipsoid [63], and for generating random points in balls. We use `Eigen` [79] for linear algebra. The memory consumption is dominated by the list of random points which needs $O(dN)$ space during the entire execution of the algorithm (Sect. 4.3). Arithmetic uses the `double` data type of C++, except from the LP solver, which uses the *GNU Multiple Precision arithmetic library* to

[1] `http://sourceforge.net/projects/randgeom`

P	d	m	vol(P)	N	μ	[min, max]	std-dev	$\frac{\text{vol}(P)-\mu}{\text{vol}(P)}$	`VolEsti` (sec)	`Exact` (sec)
cube-10	10	20	1.024E+003	9210	1.027E+003	[0.950E+003,1.107E+003]	3.16E+001	0.0030	0.42	0.01
cube-15	15	30	3.277E+004	16248	3.24E+004	[3.037E+004,3.436E+004]	9.41E+002	0.0088	1.44	0.40
cube-20	20	40	1.048E+006	23965	1.046E+006	[0.974E+006,1.116E+006]	3.15E+004	0.0028	4.62	swap
cube-50	50	100	1.126E+015	78240	1.125E+015	[1.003E+015,1.253E+015]	4.39E+013	0.0007	117.51	swap
cube-100	100	200	1.268E+030	184206	1.278E+030	[1.165E+030,1.402E+030]	4.82E+028	0.0081	1285.08	swap
Δ-10	10	11	2.756E-007	9210	2.76E-007	[2.50E-007,3.08E-007]	1.08E-008	0.0021	0.56	0.01
Δ-50	60	61	1.202E-082	98264	1.21E-082	[1.07E-082,1.38E-082]	6.44E-084	0.0068	183.12	0.01
Δ-100	100	101	1.072E-158	184206	1.07E-158	[9.95E-159,1.21E-158]	4.24E-160	0.0032	907.52	0.02
Δ-20-20	40	42	1.689E-037	59022	1.70E-037	[1.54E-037,1.87E-037]	7.33E-039	0.0088	53.13	0.01
Δ-40-40	80	82	1.502E-096	140224	1.50E-096	[1.32E-096,1.70E-096]	7.70E-098	0.0015	452.05	0.01
Δ-50-50	100	102	1.081E-129	184206	1.10E-129	[1.01E-129,1.19E-129]	4.65E-131	0.0154	919.01	0.02
cross-10	10	1024	2.822E-004	9210	2.821E-004	[2.693E+004,2.944E+004]	5.15E-006	0.0003	1.58	388.50
cross-11	11	2048	5.131E-005	10550	5.126E-005	[4.888E-005,5.437E-005]	1.15E-006	0.0010	5.19	6141.40
cross-12	12	4096	8.551E-006	11927	8.557E-006	[8.130E-006,9.020E-006]	1.69E-007	0.0007	12.21	—
cross-15	15	32768	2.506E-008	16248	2.505E-008	[2.332E-008,2.622E-008]	5.15E-010	0.0004	541.22	—
cross-18	18	262144	4.09E-011	20810	4.027E-011	[3.97E-011,4.08E-011]	5.58E-013	0.0165	5791.06	—
rh-8-25	8	25	7.859E+002	6654	7.826E+002	[7.47E+002,8.15E+002]	1.93E+001	0.0042	0.30	1.14
rh-8-30	8	30	2.473E+002	6654	2.449E+002	[2.28E+002,2.68E+002]	1.06E+001	0.0099	0.27	5.56
rh-10-25	10	25	5.729E+003	9210	5.806E+003	[5.55E+003,6.06E+003]	1.85E+002	0.0134	0.66	6.88
rh-10-30	10	30	2.015E+003	9210	2.042E+003	[1.96E+003,2.21E+003]	7.06E+001	0.0132	0.67	swap
rv-8-10	8	24	1.409E+019	6654	1.418E+019	[1.339E+019,1.497E+019]	5.24E+017	0.0107	0.37	0.01
rv-8-11	8	54	3.047E+018	6654	3.056E+018	[2.562E+018,3.741E+018]	3.98E+017	0.0028	0.76	0.54
rv-8-12	8	94	4.385E+019	6654	4.426E+019	[4.105E+019,4.632E+019]	2.07E+018	0.0093	0.59	261.37
rv-8-20	8	1191	2.691E+021	6654	2.724E+021	[2.517E+021,2.871E+021]	1.05E+020	0.0123	3.69	swap
rv-8-30	8	4482	7.350E+021	6654	7.402E+021	[7.126E+021,7.997E+021]	2.19E+020	0.0072	12.73	swap
rv-10-12	10	35	2.136E+022	9210	2.155E+022	[1.952E+022,2.430E+022]	1.53E+021	0.0093	1.00	0.01
rv-10-13	10	89	1.632E+023	9210	1.618E+023	[1.514E+023,1.714E+023]	6.23E+021	0.0088	1.24	59.50
rv-10-14	10	177	2.931E+023	9210	2.962E+023	[2.729E+023,3.195E+023]	1.71E+022	0.0135	2.08	swap
cc-8-10	8	70	1.568E+005	26616	1.589E+005	[1.52E+005,1.64E+005]	3.50E+003	0.0138	1.95	0.05
cc-8-11	8	88	1.391E+006	26616	1.387E+006	[1.35E+006,1.43E+006]	2.65E+004	0.0034	2.10	0.08
Fm-4	6	7	8.640E+001	4300	8.593E+001	[7.13E+001,1.12E+002]	8.38E+000	0.0055	0.19	0.01
Fm-5	10	25	7.110E+003	9210	7.116E+003	[6.35E+003,8.10E+003]	3.01E+002	0.0009	0.69	0.02
Fm-6	15	59	2.861E+005	16248	2.850E+005	[2.42E+005,3.22E+005]	1.55E+004	0.0038	3.24	swap
ccp-5	10	56	2.312E+000	9210	2.326E+000	[2.16E+000,2.52E+000]	7.43E-002	0.0064	0.49	38.00
ccp-6	15	368	1.346E+000	16248	1.346E+000	[1.26E+000,1.45E+000]	3.81E-002	0.0002	6.14	swap
$\mathcal{B}_8$	49	64	4.42E-023	76279	4.46E-023	[4.05E-023, 7.32E-024]	1.93E+004	0.0092	192.97	1920.00
$\mathcal{B}_9$	64	81	2.60E-033	106467	2.58E-033	[2.23E-033, 3.07E-033]	2.13E-034	0.0069	499.56	8 days
$\mathcal{B}_{10}$	81	100	8.78E-046	142380	8.92E-046	[7.97E-046, 9.96E-046]	4.99E-047	0.0152	1034.74	6160 days
$\mathcal{B}_{11}$	100	121	???	184206	1.40E-060	[1.06E-060, 1.67E-060]	1.10E-061	???	2398.17	—
$\mathcal{B}_{12}$	121	144	???	232116	7.85E-078	[6.50E-078, 9.31E-078]	5.69E-079	???	4946.42	—
$\mathcal{B}_{13}$	144	169	???	286261	1.33E-097	[1.13E-097, 1.62E-097]	1.09E-098	???	9802.73	—
$\mathcal{B}_{14}$	169	196	???	346781	5.96E-120	[5.30E-120, 6.96E-120]	3.82E-121	???	17257.61	—
$\mathcal{B}_{15}$	196	225	???	413804	5.70E-145	[5.07E-145, 6.52E-145]	1.55E-145	???	31812.67	—

Table 4.1: Overall results; $\epsilon = 1$, "swap" indicates it ran out of memory and started swapping. "???" indicates that the exact volume is unknown; "—" indicates it didn't terminate after at least 10h. `VINCI` is used for exact volume computation except Birkhoff polytopes where `birkhoff` is used instead.

avoid double exponent overflow. We experimented with several pseudo-random number generators in Boost [107] and chose the fastest, namely mersenne twister generator `mt19937`. All timings are on an Intel Core i5-2400 3.1GHz, 6MB L2 cache, 8GB RAM, 64-bit Debian GNU/Linux.

Data. The following polytopes are tested (the first 7 are from the `VINCI` webpage):

- cube-d: $\{x = (x_1, \ldots, x_d) \mid x_i \leq 1, x_i \geq -1, x_i \in \mathbb{R} \text{ for all } i = 1, \ldots, d\}$,
- cross-d: cross polytope, the dual of cube, i.e. conv($\{-e_i, e_i,\ i = 1, \ldots, d\}$),
- rh-d-m: polytopes constructed by randomly choosing m hyperplanes tangent to the sphere,
- rv-d-n: dual to rh-d-m, i.e. polytopes with n vertices randomly distributed on the sphere,

			RDHR				CDHR			
P	d	ϵ	μ	[min, max]	$(\mathrm{vol}(P)-\mu)$ $/\mathrm{vol}(P)$	VolEsti (sec)	μ	[min, max]	$(\mathrm{vol}(P)-\mu)$ $/\mathrm{vol}(P)$	VolEsti (sec)
$\mathcal{B}_5$	16	1	2.27E-07	[1.66E-07,2.85E-07]	0.0072	22.90	2.25E-07	[1.87E-07,2.80E-07]	0.0003	4.06
$\mathcal{B}_6$	25	1	8.53E-13	[3.72E-13,1.22E-12]	0.0982	105.96	9.53E-13	[7.30E-13,1.15E-12]	0.0083	17.26
$\mathcal{B}_7$	36	1	2.75E-20	[1.78E-21,6.71E-20]	0.4259	479.40	4.82E-20	[3.86E-20,6.18E-20]	0.0056	56.64
cube-10	10	1	1022.8	[944.3951,1103.968]	0.0012	2.03	1026.83	[970.3117,1096.469]	0.0027	0.34
cube-10	10	0.4	–	–	–	–	1022.88	[993.0782,1060.409]	0.0011	2.02
cube-20	20	1	1.04E+6	[9.38E+5,1.14E+6]	0.0033	25.44	1.04E+6	[9.74E+5,1.12E+6]	0.0028	4.62

Table 4.2: Experiments with CDHR vs RDHR; $W = 10$.

d	m	$\mathrm{vol}(P)$	N	μ	[min, max]	std-dev	$(\mathrm{vol}(P)-\mu)$ $/\mathrm{vol}(P)$	VolEsti (sec)	mem. MB	VolEsti* (sec)	mem. MB
10	1024	2.82E-04	9210	2.82E-04	[2.67E-04,3.00E-04]	5.74E-06	0.0001	1.58	35	0.51	42
12	4096	8.55E-06	11927	8.54E-06	[8.04E-06,8.89E-06]	1.72E-07	0.0010	12.21	35	1.62	72
14	16384	1.88E-07	14778	1.88E-07	[1.80E-07,1.99E-07]	4.09E-09	0.0006	237.22	36	6.49	230
16	65536	3.13E-09	17744	3.13E-09	[2.97E-09,3.33E-09]	6.44E-11	0.0004	1430.93	37	32.87	992
18	262144	4.09E-11	20810	4.09E-11	[3.99E-11,4.29E-11]	7.19E-13	0.0013	5791.06	38	188.43	4781

Table 4.3: Experiments with NN for boundary oracle on cross-polytopes; VolEsti* uses `flann`; $\epsilon = 1$.

- cc-8-n: the 8-dimensional product of two 4-dimensional cyclic polyhedra with n vertices,
- ccp-n: complete cut polytopes on n vertices,
- Fm-d: one facet of the metric polytope in dimension d,
- Δ-d: the d-dimensional simplex $\mathrm{conv}(\{e_i, \text{ for } i = 0, 1, \ldots, d\})$,
- Δ-d-d: product of two simplices, i.e $\{(p, p') \in \mathbb{R}^{2d} \,|\, p \in \Delta\text{-}d, p' \in \Delta\text{-}d\}$,
- skinny-cube-d: $\{x = (x_1, \ldots, x_d) \,|\, x_1 \leq 100, x_1 \geq -100, x_i \leq 1, x_i \geq -1, x_i \in \mathbb{R}\, i = 2, \ldots, d\}$, rotated by 30^o in the plane defined by the first two coordinate axes,
- $\mathcal{B}_n$: the n-Birkhoff polytope (defined below).

Each experiment is repeated 100 times with $\epsilon = 1$ unless otherwise stated. The reported timing for each experiment is the mean of 100 timings. We keep track of and report the *min* and the *max* computed values, the mean μ, and the standard deviation. We measure the accuracy of our method by $(\mathrm{vol}(P) - \mu)/\mathrm{vol}(P)$ and $(\mathit{max}{-}\mathit{min})/\mu$; unless otherwise stated mean error of approximation refers to the first quantity. The reader should not confuse these quantities which refer to the approximation error that computed *in practice* with ϵ which refers to the *objective* approximation error. Comparing the practical and objective approximation error, our method is in practice more accurate than indicated by the theoretical bounds. In particular, in all experiments all computed values are contained in the interval $((1 - \epsilon)\mathrm{vol}(P), (1 + \epsilon)\mathrm{vol}(P))$, while theoretical results in [91] guarantee only 75% of them. Actually, the above interval is larger than [*min*, *max*]. In general our experimental results show that our software can approximate the volume of general polytopes up to dimension 100 in less than 2 hours with mean approximation error at most 2% (cf. Table 4.1).

Random walks and oracles. First, we compare the implementations of boundary oracles using membership oracles versus using facet intersection. By performing experiments with RDHR our algorithm approximate the volume of a 10-cube in 42.58 sec using the former, whereas it runs in 2.03 sec using the latter.

We compare RDHR to CDHR. The latter take advantage of more efficient boundary oracle implementations as described in Sect. 4.2. Table 4.2 shows that our algorithm using CDHR becomes faster and more accurate than using RDHR by means of smaller $[min, max]$ interval. Additionally, since CDHR is faster we can increase the accuracy (decrease ϵ) and obtain even more accurate results than RDHR, including smaller error $(\text{vol}(P) - \mu)/\text{vol}(P)$.

Finally, we evaluate our implementations of boundary oracles using *duality* and *NN search* (Sect. 4.2). The motivation comes from the fact that the boundary oracle becomes slow when the number of facets is large, e.g., for cross-d, $m = 2^d$. We consider state-of-the-art NN software: CGAL's dD Spatial Searching implements kd-trees [133], ANN [111] implements kd- and BBD-trees, LSH implements Locality Sensitive Hashing [3], and FLANN [112] implements randomized kd-trees. We compare them against our oracle running in $O(m)$, on cross-17, $\mathcal{B}_{10}$ and cpp-7. We build two kd-trees per coordinate, i.e. one per direction, each tree storing the dual of the corresponding lower and upper hulls.

Consider point queries. FLANN, is very fast in high dimensions (typically > 100), but lacks theoretical guarantees. It turns out that KDTreeSingleIndexParams on cross-d returns exact results for all ε and d tested, since the tree stores vertices of a cube. Compared to the $O(m)$ oracle, for $\varepsilon = 0$ it is 10x slower, for $\varepsilon = 2$ it is competitive, and for $\varepsilon = 5$ it lets us approximate vol(cross-18) with a 40x speed-up, but with extra memory usage (Table 4.3). On other datasets, FLANN does not always compute the exact NN even for $\varepsilon = 0$. ANN, is very fast up to dimension 20 and offers theoretical guarantees. For $\varepsilon = 0$, it guarantees the exact NN, but is $> 10^3$x slower than our $O(m)$ oracle, though it becomes significantly faster for $\varepsilon > 1$. In [113], LSH is reported to be 10x slower than FLANN and competitive with ANN, thus we do test it here.

CGAL for point queries is slower than ANN, but can be parametrized to handle

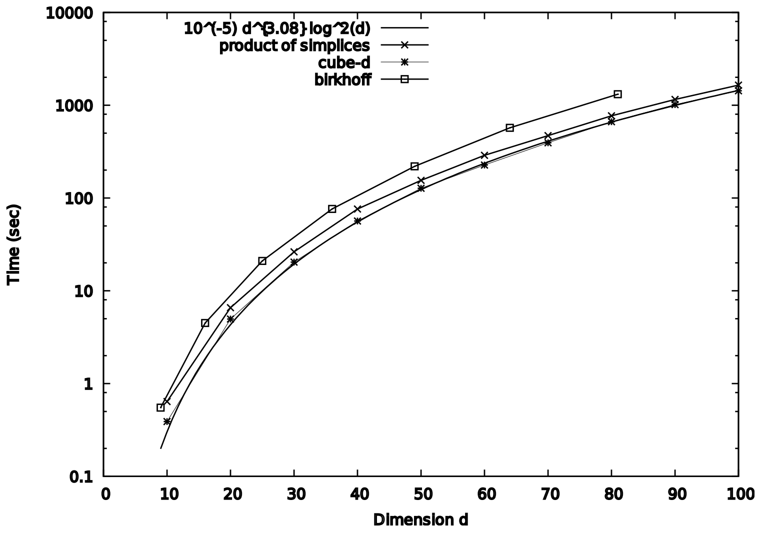


Figure 4.1: Runtime of `VolEsti` w.r.t. dimension; $\epsilon = 1$, y-axis in logscale; fitting on cube-d results.

hyperplane queries with theoretical guarantees. Given hyperplane H, we set as query point the projection of the origin on H and as distance-function the inner product between points. With the `Sliding_midpoint` rule and $\varepsilon = 0$, this is a bit (while `ANN` is 1000x) slower than our boundary oracle for cross-17. It is important to design methods for which $\varepsilon > 0$ accelerates computation so as to use them with approximate boundary oracles.

The above study provides motivation for the design of algorithms that can use approximate boundary queries and hence take advantage of NN software to handle more general polytopes with large number of facets. Of particular relevance is the development of efficient methods and data-structures for approximate hyperplane queries.

Choice of parameters and rounding. We consider two crucial parameters, the length of a random walk, denoted by W, and approximation ϵ, which determines the number N of random points. We set $W = \lfloor 10 + d/10 \rfloor$. Our experiments indicate that, with this choice, either (vol(P)-μ)/vol(P) or $(min, max)/\mu$ is $< 1\%$ up to $d = 100$ (Table 4.4). Moreover, for higher W the improvement in accuracy is not significant, which supports the claim that asymptotic bounds are unrealistically high. Fig. 4.2 correlates runtime (expressed by NW) and accuracy (expressed by $(min, max)/\mu$ which actually measures some "deviation") to W and ϵ (expressed by N). A positive observation is that accuracy tightly correlates with runtime: e.g.,

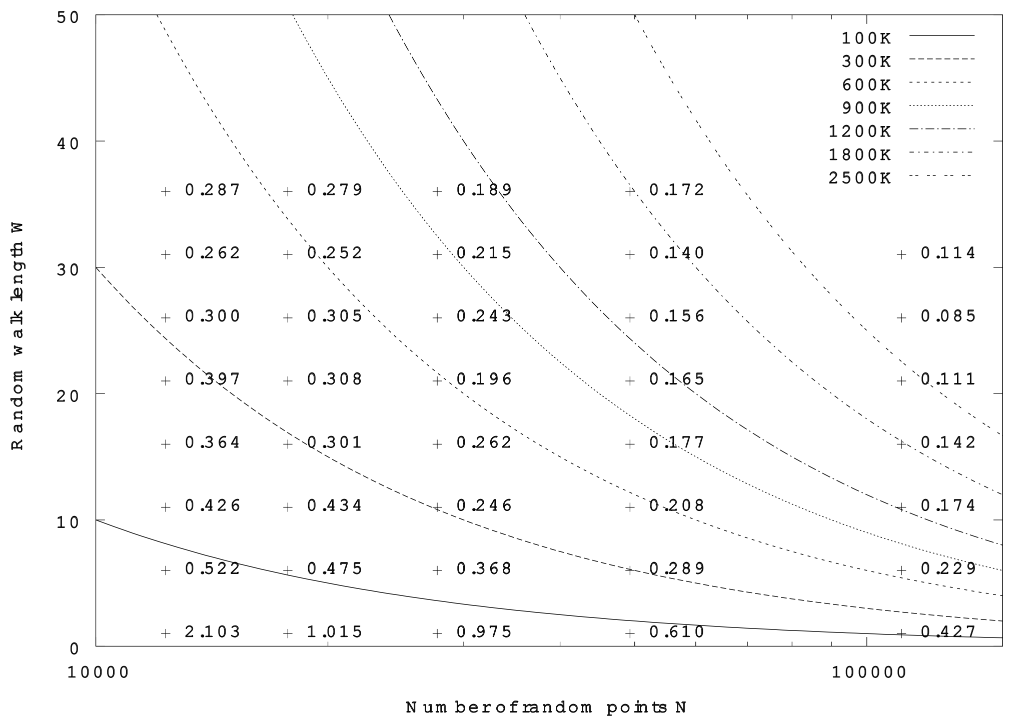


Figure 4.2: Experiments with $\mathcal{B}_5$ on the effect of W and ϵ (or N) on accuracy, measured by (min, max)/μ (crosses), and runtime, measured by levels of $N \cdot W = c$, for $c = 10^5, \ldots, 2.5 \cdot 10^6$.

accuracy values close to or beyond 1 lie under the curve $NW = 10^5$, and those rounded to ≤ 0.3 lie roughly above $NW = 3 \cdot 10^5$. It also shows that, increasing W converges faster than increasing N to a value beyond which the improvement in accuracy is not significant.

To experimentally test the effect of *rounding* we construct skinny hypercubes skinny-cube-d. We rotate them to avoid CDHR taking unfair advantage of the degenerate situation where the long edge is parallel to an axis. Table 4.5 on these and other polytopes shows that rounding reduces approximation error by 2 orders of magnitude. Without rounding, for polytope rv-8-11 one needs to multiply N (thus runtime) by 100 in order to achieve approximation error same as with rounding.

Other software. Exact volume computation concerns software computing the exact value of the volume, up to round-off errors in case it uses floating point arithmetic. We mainly test against `VINCI` 1.0.5 [29], which implements state-of-the art algorithms, cf. Table 4.1. For H-polytopes, the method based on

P	d	m	W	μ	$[min, max]$	std-dev	(vol(P)-μ) /vol(P)	(min, max) /μ
(*) cube-10	10	20	**10**	1026.953	[925.296,1147.101]	33.91331	0.0029	0.2160
cube-10	10	20	15	1024.157	[928.667,1131.928]	31.34121	0.0002	0.1985
cube-10	10	20	20	1026.910	[932.118,1144.601]	30.97023	0.0028	0.2069
cube-50	50	100	10	1.123E+15	[1.019E+15,1.257E+15]	4.135E+13	0.0022	0.2125
(*) cube-50	50	100	**15**	1.131E+15	[1.039E+15,1.237E+15]	3.882E+13	0.0044	0.1744
cube-50	50	100	20	1.127E+15	[1.033E+15,1.216E+15]	3.893E+13	0.0007	0.1629
cube-100	100	200	10	1.278E+30	[1.165E+30,1.402E+30]	4.819E+28	0.0081	0.1856
cube-100	100	200	15	1.250E+30	[1.243E+30,1.253E+30]	4.075E+27	0.0140	0.0083
(*) cube-100	100	200	**20**	1.263E+30	[1.190E+30,1.321E+30]	3.987E+28	0.0038	0.1038
Δ-20-20	40	42	10	1.699E-37	[1.527E-37,1.881E-37]	7.670E-39	0.0056	0.2083
(*) Δ-20-20	40	42	**14**	1.694E-37	[1.526E-37,1.892E-37]	7.096E-39	0.0025	0.2166
Δ-20-20	40	42	20	1.694E-37	[1.433E-37,1.836E-37]	7.006E-39	0.0024	0.2382
Δ-50-50	100	102	10	1.098E-129	[1.012E-129,1.189E-129]	4.652E-131	0.0154	0.1612
Δ-50-50	100	102	15	1.111E-129	[1.090E-129,1.139E-129]	1.610E-131	0.0281	0.0437
(*) Δ-50-50	100	102	**20**	1.079E-129	[1.011E-129,1.148E-129]	3.685E-131	0.0015	0.1266
$\mathcal{B}_{10}$	81	100	10	7.951E-55	[6.291E-55,9.077E-55]	8.533E-56	0.0946	0.3504
$\mathcal{B}_{10}$	81	100	15	8.124E-55	[7.451E-55,8.774E-55]	5.015E-56	0.0750	0.1629
(*) $\mathcal{B}_{10}$	81	100	**20**	7.489E-55	[7.398E-55,7.552E-55]	6.615E-57	0.1472	0.0106

Table 4.4: Experiments with varying W; $\epsilon = 1$. (*) indicate minimum W where either (vol(P)-μ)/vol(P) or (min, max)/μ is $< 1\%$.

P	vol(P)	N	μ	$[min, max]$	$\frac{\mathrm{vol}(P)-\mu}{\mathrm{vol}(P)}$	`VolEsti`(sec)
rv-8-11	3.047E+18	6654	1.595E+18	[6.038E+17,3.467E+18]	0.4766	1.48
rv-8-11	3.047E+18	665421	3.134E+18	[3.134E+18,3.134E+18]	0.0283	157.46
(*) rv-8-11	3.047E+18	6654	3.052E+18	[2.755E+18,3.383E+18]	0.0013	1.34
skinny-cube-10	1.024E+05	9210	5.175E+04	[2.147E+04,1.228E+05]	0.4946	0.69
(*) skinny-cube-10	1.024E+05	9210	1.029E+05	[8.445E+04,1.149E+05]	0.0050	0.71
skinny-cube-20	1.049E+08	23965	4.193E+07	[2.497E+07,7.259E+07]	0.6001	5.59
(*) skinny-cube-20	1.049E+08	23965	1.040E+08	[8.458E+07,1.163E+08]	0.0084	6.70

Table 4.5: Experiments with rounding; (*): means that we use rounding.

Lawrence's general formula is numerically unstable resulting in wrong results in many examples [30], and thus was excluded. Therefore, we focused on Lasserre's method. For all polytopes there is a threshold dimension for which `VINCI` cannot compute the volume: it takes a lot of time (e.g. $>$ 4 hrs for cube-20) and consumes all system memory, thus starts swapping.

`LRS` is not useful for H-polytopes as stated on its webpage: "If the volume option is applied to an H-representation, the results are not predictable." `Latte` implements the same decomposition methods as `VINCI`; it is less prone to round-off error but slower [45]. `Normaliz` applies triangulation: it handles cubes for $d \leq 10$, in < 1 min, but for $d = 15$, it did not terminate after 5 hours. `Qhull` handles V-polytopes but does not terminate for cube-10 nor random polytope rv-

P:	rv-15-				rv-10-				cube-			
	30	40	50	60	100	150	200	250	7	8	9	10
time (sec)	7.7	82.8	473.3	swap	37.3	107.8	282.5	449.0	0.1	2.2	119.5	>5h

Table 4.6: Experiments with `qhull`; “swap” indicates it ran out of memory and started swapping; “>5h” indicates it did not terminate after 5 hours.

	software of [42]					Volesti				
P	[min, max]	std-dev	$\frac{\text{vol}(P)-\mu}{\text{vol}(P)}$	# total steps	time(sec)	[min, max]	std-dev	$\frac{\text{vol}(P)-\mu}{\text{vol}(P)}$	# total steps	time(sec)
cube-20	[5.11E+05, 1.55E+06]	1.67E+05	0.0198	7.96E+04	21.48	[9.74E+05, 1.12E+06]	3.15E+04	0.0028	3.61E+06	4.62
cube-30	[6.75E+08, 1.45E+09]	1.72E+08	0.0440	2.22E+05	49.24	[9.91E+08, 1.16E+09]	3.89E+07	0.0039	1.21E+07	17.96
cube-40	[7.90E+11, 1.38E+12]	1.67E+11	0.0731	4.30E+05	88.09	[1.01E+12, 1.23E+12]	4.46E+10	0.0039	2.84E+07	50.72
cube-50	[8.75E+14, 1.45E+15]	1.43E+14	0.0327	7.16E+05	148.06	[1.00E+15, 1.25E+15]	4.39E+13	0.0007	5.49E+07	117.51
cube-60	[8.89E+17, 1.43E+18]	1.64E+17	0.0473	1.15E+06	229.33	[1.06E+18, 1.27E+18]	4.00E+16	0.0051	9.42E+07	222.10
cube-70	[9.01E+20, 1.36E+21]	1.49E+20	0.0707	1.66E+06	427.82	[1.02E+21, 1.32E+21]	5.42E+19	0.0013	1.49E+08	358.93
cube-80	[9.30E+23, 1.36E+24]	1.46E+23	0.1145	2.30E+06	531.46	[1.13E+24, 1.30E+24]	4.42E+22	0.0009	2.21E+08	582.19
cube-90	[1.07E+27, 1.88E+27]	2.20E+26	0.0394	3.30E+06	701.54	[1.09E+27, 1.44E+27]	5.18E+25	0.0019	3.15E+08	875.69
cube-100	[9.53E+29, 1.64E+30]	1.93E+29	0.0357	4.19E+06	884.43	[1.17E+30, 1.40E+30]	4.82E+28	0.0081	4.33E+08	1285.08
$\mathcal{B}_8$	[2.12E-23, 2.45E-22]	6.25E-23	0.3970	9.31E+05	221.30	[4.05E-23, 7.32E-24]	1.93E+04	0.0092	1.01E+08	192.97
$\mathcal{B}_9$	[1.54E-33, 2.77E-33]	3.71E-34	0.1830	2.05E+06	420.07	[2.23E-33, 3.07E-33]	2.13E-34	0.0069	2.27E+08	499.56
$\mathcal{B}_{10}$	[3.39E-46, 1.92E-45]	4.75E-46	0.1207	3.69E+06	691.97	[7.97E-46, 9.96E-46]	4.99E-47	0.0152	4.62E+08	1034.74

Table 4.7: Comparison of the software [42] vs `VolEsti`; each experiment is run 10 times, total steps refer to the mean of the total number of Hit-and-run steps in each execution.

	n	3	4	5	6	7	8	9	10
$\frac{\text{estimate}}{\text{actual}}$	[33]	1.25408	1.22556	1.19608	1.17258	1.15403	1.13910	1.12684	1.11627
	VolEsti	0.99485	1.09315	1.00029	1.00830	1.00564	0.99440	0.99313	1.01525

Table 4.8: Comparison between asymptotic and experimental approximation of the volume of $\mathcal{B}_n$.

15-60 (Table 4.6). This should be juxtaposed to the duals, namely our software approximates the volume of cross-10 in 2 sec with $< 1\%$ error and rh-15-60 in 3.44 sec. A general conclusion for exact software is that it cannot handle $d > 15$.

We compare with the most relevant approximation method, namely the `Matlab` implementation of [43] for bodies represented as the intersection of an H-polytope and an ellipsoid. They report that the code is optimized to achieve about 75% success rate for bodies of dimension ≤ 100 and $\epsilon \in [0.1, 0.2]$ (not to be confused with the ϵ of our method). Testing [43] with default options and $\epsilon = 0.1$, our implementation with $\epsilon = 1$ runs faster for $d < 80$, performs roughly 100 times more total Hit-and-run steps and returns significantly more accurate results, e.g. from 4 to 100 times smaller error on cube-d when $d > 70$, and from 5 to 80 times on Birkhoff polytopes (Table 4.7).

Birkhoff polytopes are well studied in combinatorial geometry and offer an important benchmark. The n-th Birkhoff polytope $\mathcal{B}_n = \{x \in \mathbb{R}^{n\times n} \mid x_{ij} \geq 0,\ \sum_i x_{ij} = 1,\ \sum_j x_{ij} = 1,\ 1 \leq i \leq n\}$, also described as the polytope of the perfect matchings of the complete bipartite graph $K_{n,n}$, the polytope of the $n \times n$ doubly stochastic matrices, and the Newton polytope of the determinant. In [15], they present a complex-analytic method for this volume, implemented in package `birkhoff`, which has managed to compute vol($\mathcal{B}_{10}$) in parallel execution, which corresponds to a single processor running at 1 GHz for almost 17 years.

First, $\dim \mathcal{B}_n = n^2 - 2n + 1$: we project $\mathcal{B}_n$ to a subspace of this dimension. Our software, with $\epsilon = 1$, computes the volume of polytopes up to $\mathcal{B}_{10}$ in < 1 hour with mean error of $\leq 2\%$ (Table 4.1). The computed approximation values improves upon the best known upper bounds on vol($\mathcal{B}_n$), obtained through the asymptotic formula of [33], cf. Table 4.8. By setting $\epsilon = .5$ we obtain an error of 0.7% for vol($\mathcal{B}_{10}$), in 6 hours. The computed approximation of the volume has two correct digits, i.e. its first two digits equal to the ones of the exact volume. More interestingly, using $\epsilon = 1$ we compute, in < 9 hours, an approximation as well as an interval of values for vol($\mathcal{B}_{11}$),..., vol($\mathcal{B}_{15}$), whose exact values are unknown (Table 4.1).

4.5 Further work

NN search seems promising and could accelerate our code, especially if it were performed approximately with hyperplane queries. Producing (almost) uniform point samples is of independent interest in machine learning, including sampling contingency tables and learning the p-value. We plan to exploit such applications of our software. We may also study sampling for special polytopes such as Birkhoff. It is straightforward to parallelize certain aspects of the algorithm, such as random walks assigning each thread to a processor, though other aspects, such as the algorithm's phases, require more sophisticated parallelization. Our original motivation and ultimate goal is to extend these methods to V-polytopes represented by an optimization oracle.

Chapter 5

Combinatorics of 4-dimensional resultant polytopes

5.1 Introduction

Let $\mathcal{A} = (A_0, \dots, A_n)$ be a family of subsets of $\mathbb{Z}^n$ and let $f_0, \dots, f_n \in \mathbb{C}[x_1, \dots, x_n]$ be polynomials with this family of sets as supports, and symbolic coefficients $c_{ij} \neq 0$, $i = 0, \dots, n$, $j = 1, \dots, |A_i|$, i.e. $f_i = \sum_{a \in A_i} c_{ij} x^a$. The family $\mathcal{A}$ is *essential* if these sets jointly affinely span $\mathbb{Z}^n$, and every subfamily of A_i's of cardinality j, $1 \leq j < n$, spans an affine space of dimension $\geq j$. In this chapter we assume that $\mathcal{A}$ is essential. The *sparse (or toric) resultant* $\mathcal{R} = \mathcal{R}_{\mathcal{A}}$ of $f_0, \dots, f_n$ is then a non-constant irreducible polynomial in $\mathbb{Z}[c_{ij} : i = 0, \dots, n,\ j = 1, \dots, |A_i|]$, defined up to sign, which vanishes if $f_0 = f_1 = \cdots = f_n = 0$ has a solution in $(\mathbb{C}^*)^n$, $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$. The Newton polytope $N(\mathcal{R})$ of the resultant, that is, the convex hull of the exponents occurring in $\mathcal{R}$ with non-zero coefficient, is a lattice polytope called *resultant polytope*. A famous example is the Birkhoff polytope of a linear system, cf Example 9.

The resultant has

$$m = \sum_{i=0}^{n} |A_i|$$

variables, hence $N(\mathcal{R})$ lies in $\mathbb{R}^m$. However, for essential families, $\mathcal{R}$ satisfies $n + (n+1)$ natural homogeneities [74], so its dimension is $\dim(N(\mathcal{R})) = m - 2n - 1$. If $\mathcal{A}$ is not essential, but contains a single essential subfamily, the resultant depends only on the coefficients of the polynomials in this subfamily. Otherwise, the resultant locus has codimension bigger than one, and then the sparse resultant is defined to be the constant 1.

Previous work. In [60] an algorithm is described for computing the vertex and facet representations of $N(\mathcal{R})$; the algorithm also produces a triangulation of the polytope's interior into simplices. The input to this algorithm is an oracle for computing the extreme resultant vertex given a direction. The software implementation is called `respol` (available at `http://respol.sourceforge.net`) and is the one used in our experiments. This method is readily generalized to compute the discriminant and secondary polytopes, although for the latter there exists a faster method to enumerate the vertices, when only these are needed [120]. An alternative way for computing resultant polytopes exploits tropical geometry [86] and is implemented based on the software library `Gfan`.

The combinatorics of resultant polytopes is known only in small cases, namely for linear systems (Example 9), in the Sylvester case ($n = 1$), and when $\dim N(\mathcal{R}) = 3$. The univariate case is fully described in [73]: $N(\mathcal{R})$ is combinatorially isomorphic to a polytope denoted by $\mathcal{N}_{k_0,k_1}$, of dimension $k_0 + k_1 - 1$, where the A_i may be multisets with cardinality k_i. They may lead to polytopes in any dimension if one picks the A_i accordingly. In [74] they show that $\mathcal{N}_{k_0,k_1}$ has $\binom{k_0+k_1}{k_0}$ vertices and, when both $k_i \geq 2$, it has $k_0 k_1 + 3$ facets. Sturmfels [131] classifies all resultant polytopes up to dimension 3. In his notation, the 3-dimensional polytope $\mathcal{N}_{111,111}$, denoted by $\mathcal{N}_{2,2}$ in [73], depicted in Figure 5.8 (resultant), has maximal f-vector.

Proposition 29. [131, Section 6] *Assume $\mathcal{A}$ is an essential family. Then, $N(\mathcal{R})$ is 1-dimensional if and only if $|A_i| = 2$, for all i. The only planar resultant polytope is the triangle. The only 3-dimensional $N(\mathcal{R})$ are, combinatorially: (a) the tetrahedron, (b) the square-based pyramid, and (c) the polytope $\mathcal{N}_{3,2}$, first in Figure 5.8 (resultant).*

In [86] the authors raise explicitly the open question of describing 4-dimensional resultant polytopes, which we undertake here.

Our contribution. We study the combinatorial characterization of 4-dimensional resultant polytopes. To bound the maximum number of faces, we prove that it suffices to focus on one case, which corresponds to 3 Newton polygons with support cardinalities $|A_i| = 3$, thus $m = 9$ and $n = 2$. We further show it is enough to consider sufficiently generic polygons, namely where they all have non-zero area, and no parallel edges exist among them. Our experiments, based on `respol` [60], establish lower bounds on the maximal number of faces (Table 5.1). By studying mixed subdivisions, we obtain tight upper bounds on the maximal number of facets and ridges, thus raising new conjectures, the most important of which is that the maximal f-vector is $(22, 66, 66, 22)$ for 4-dimensional $N(\mathcal{R})$. These results are summarized in Theorem 63. Our (loose) upper bound on the number of vertices, namely 28, significantly improves the known bound of 6608 [131, Corollary 6.2]. Certain general features emerge, such as the symmetry of the maximal f-vector, which are intriguing but still under investigation. However, the Newton polytopes are not *self-dual*. Our main result is Theorem 64, where we offer a characterization of all possible 4-dimensional resultant polytopes.

The rest of the chapter is organized as follows. The next section introduces 4-dimensional resultant polytopes. Section 5.3 focuses on three $2d$-triangles with non-parallel edges, which maximizes the number of faces, and upper bounds the number of facets and ridges in $N(\mathcal{R})$ by combinatorial arguments. Section 5.5 classifies all 4-dimensional resultant polytopes, and proves we can ignore parallel edges when maximizing the number of faces. We conclude with open questions and generalizations.

5.2 Resultant polytopes

This section defines resultant polytopes and recalls the concepts needed for their study, including some previous results from [131, 74].

The *polar (dual) polytope* of a polytope $P \subseteq \mathbb{R}^d$ is defined as:

$$P^* := \{c \in \mathbb{R}^d : c^T x \leq 1 \text{ for all } x \in P\} \subseteq \mathbb{R}^d,$$

where we assume that the origin $0 \in relint(P)$, the relative interior of P, i.e. 0 is not contained in any face of P of dimension $< d$. Two polytopes are *combinatorially equivalent* if and only if their face lattices are isomorphic.

The main tool for computing sparse resultants are the *regular mixed subdivisions* of the convex hull of Minkowski sum $P = \sum_i A_i$. By abuse of notation, we may also refer to this sum as $\sum_i P_i$, where P_i denotes the convex hull of A_i and it is understood that the information from the A_i's is preserved. A *subdivision* of P

is a collection of subsets of P, the cells of the subdivision, such that the union of the cells' convex hulls equals the convex hull of P and every pair of convex hulls of cells intersect at a common face. *Maximal* cells are those with dimension equal to the dimension of the subdivision. *Fine (or tight)* are those whose dimension equals the sum of its summands' dimensions.

Definition 4. *A subdivision is regular if it can be obtained as the projection of the lower hull of the Minkowski sum $\sum_i \widehat{A}_i$ of the lifted point sets $\widehat{A}_i$, for some lifting to $\mathbb{R}^{n+1}$. A subdivision is* mixed *when its cells are* expressed *(or, can be written) as Minkowski sums of convex hulls of point subsets in the A_i's; these expressions are unique. The subdivision is* fine (or tight) *if all its cells are fine, otherwise, it is* coarse.

In the sequel, we typically refer to mixed subdivisions simply as subdivisions. Usually they are regular; if not, we explicitly mention it. However, no subdivision is necessarily fine and often we work with coarse subdivisions.

Maximal cells are *mixed* if the dimension of every summand, except possibly one, equals one. In the sequel, we focus only on *regular* subdivisions, thus we occasionally omit the word "regular" in general.

Given a family $\mathcal{A}$, the associated Cayley configuration $\mathcal{C}$ is the lattice configuration in $\mathbb{Z}^{n+1} \times \mathbb{Z}^n = \mathbb{Z}^{2n+1}$ defined by

$$\{e_0\} \times A_0 \cup \cdots \cup \{e_n\} \times A_n,$$

where $e_0, \ldots, e_n$ denotes the canonical basis in $\mathbb{Z}^{n+1}$. We denote by Q its convex hull. Regular fine mixed subdivisions of P are in bijection with regular triangulations of Q. Indeed, there is a bijection of maximal cells given as follows: any maximal cell (simplex) σ_T in a given regular triangulation $T = T_w$ of Q (with vertices in $\mathcal{C}$) has $2n$ vertices; the corresponding maximal cell in the associated regular fine subdivision $S = S_w$ of P has vertices of the form $a_0 + \cdots + a_n$, with (e_i, a_i) a vertex of σ_T. Note that for σ_T to be of maximal dimension, at least one of its vertices lies in $e_i \times A_i$, for all i. For more details about the translation between regular subdivisions of Q and regular mixed subdivisions of P, see [101, Section 9.2].

Let C be the $(2n + 1) \times m$ associated Cayley matrix, i.e., the matrix whose columns are the points in the Cayley configuration $\mathcal{C}$. The inner product of any point in $N(\mathcal{R})$ with any vector in the rowspan of the Cayley matrix C is constant, and so $N(\mathcal{R})$ lies in a parallel translate to the null-space of C. This explains why $\dim(N(\mathcal{R})) = m - 2n - 1$.

The faces (resp. vertices) of $N(\mathcal{R})$ can be obtained, by a many-to-one mapping, from the set of all regular (resp. fine) mixed subdivisions of P [131]. Given a mixed subdivision of P, every cell σ defines a subsystem of the $f_i|_\sigma$, where each polynomial is a restriction of f_i on the face of A_i appearing as a summand in σ. If the subdivision is the projection of the lower hull under a lifting w, then the face of $N(\mathcal{R})$ whose outer normal is w is

$$\prod_{\sigma} \mathcal{R}(f_0|_\sigma, \ldots, f_n|_\sigma)^{d_\sigma}, \tag{5.1}$$

where $d_\sigma \in \mathbb{N}$ is specified in [131, Theorem 4.1]. We shall call σ *essential* if the corresponding $f_i|_\sigma$ define an essential subsystem. Hence, all faces of $N(\mathcal{R})$ are Minkowski sums of lower-dimensional resultant polytopes, corresponding to essential subsystems. These lower-dimensional resultant polytopes correspond to subsets of the cells of the subdivision defining the face of $N(\mathcal{R})$. In particular, resultant vertices are obtained when all resultants in (5.1) are monomials, hence all σ are mixed. In general, d_σ is the normalized volume of σ.

We call *flip* the transformation of a fine mixed subdivision of P to another fine mixed subdivision of P. Following [131], if these subdivisions correspond to different vertices of $N(\mathcal{R})$ we call this flip *cubical*. In other words, a cubical flip corresponds to a resultant edge.

In short, a mixed subdivision S is specified by a lifting function w. A face of $N(\mathcal{R})$ corresponds to the initial form of $\mathcal{R}$ specified by w. In [131, Theorem 4.1] under the assumption that $\{A_0, A_1, A_2\}$ is essential, which we have also assumed, it is shown that the initial form of $\mathcal{R}$ w.r.t. w is the product of resultant polytopes corresponding to the cells $\sigma \subset S$, each raised to the power d_σ. The above discussion yields the following, which will be our basic tool for counting the faces of $N(\mathcal{R})$, and is direct consequence of [131, Theorem 4.1].

Proposition 30. *A mixed subdivision S of P corresponds to a face of $N(\mathcal{R})$, which is the Minkowski sum of the resultant polytopes of the cells σ of S, each scaled by d_σ.*

5.2.1 4-dimensional resultant polytopes.

In this case, $m = 2n + 5$, where $m = \sum_{i=0}^{n} |A_i|$, and $|A_i| \geq 2$. So, there are only 3 cases, up to reordering:

(i) All $|A_i| = 2$, except for one with cardinality 5.

(ii) All $|A_i| = 2$, except for two with cardinalities 3 and 4.

(iii) All $|A_i| = 2$, except for three with cardinality 3.

Cases (i) and (ii) are similar to the study of 3d-resultant polytopes in [131], cf Theorem 64. So, we concentrate on the new case (iii) and, more precisely, on the main case $n = 2$ and each $|A_i| = 3$, which we term the case $(3, 3, 3)$. This is done without loss of generality, by the following:

Theorem 31. [131, Theorem 6.2] *Every resultant polytope of an essential family is affinely isomorphic to a resultant polytope of an essential family* $(A_0, \dots, A_n)$ *with* $|A_i| \geq 3$, *for all* $i = 0, \dots, n$.

Let us focus on case (iii). The proof is an algorithm to produce this reduction: up to an affine change of variables and reordering, we can assume that $A_i = \{0, \nu_i e_{i+1}\}, i = 0, \dots, n-3$, so we can solve (with rational powers) the first $n-2$ variables and replace them in the last 3 polynomials. Then, $N(\mathcal{R})$ has the same combinatorial type as an essential $(3, 3, 3)$ configuration, where we could have repeated points or parallel edges, even if they were not present in A_{n-2}, A_{n-1} and A_n (and some coefficients could be equal to the sum of Laurent monomials in the original coefficients).

Our study shows the richness of possible polytopes, in contrast to the case of resultant polytopes with dimension ≤ 3.

5.3 The case $(3, 3, 3)$ in general

In this section, we start with some examples and computational experiments for a family with $n = 3, m = 9$, where each A_i has cardinality 3. This is a $(3, 3, 3)$ configuration $\mathcal{A} = (A_0, A_1, A_2)$. Then, we focus on the case of non-parallel edges and study the combinatorics of the corresponding resultant polytope, denoted by $N(\mathcal{R})$.

(6, 15, 18, 9)	(13, 37, 37, 13)	(16, 43, 40, 13)	(18, 52, 50, 16)
(8, 20, 21, 9)	(14, 35, 32, 11)	(16, 43, 41, 14)	(18, 52, 51, 17)
(9, 22, 21, 8)	(14, 36, 33, 11)	(16, 44, 41, 13)	(18, 53, 51, 16)
(9, 24, 25, 10)	(14, 36, 34, 12)	(16, 44, 42, 14)	(18, 53, 53, 18)
(10, 24, 23, 9)	(14, 37, 34, 11)	(16, 45, 43, 14	(18 54 54 18)
(10, 25, 24, 9)	(14, 37, 35, 12)	(16, 45, 44, 15)	(19, 54, 52, 17)
(10, 25, 25, 10)	(14, 37, 36, 13)	(16, 46, 45, 15)	(19, 55, 51, 15)
(10, 26, 25, 9)	(14, 38, 36, 12)	(16, 46, 46, 16	(19 55 52 16)
(11, 28, 27, 10)	(14, 38, 37, 13)	(17, 46, 43, 14)	(19, 55, 54, 18)
(11, 29, 28, 10)	(14, 38, 38, 14)	(17, 47, 43, 13	(19 56 54 17)
(11, 29, 29, 11)	(14, 40, 40, 14)	(17, 47, 44, 14)	(19, 56, 56, 19)
(12, 29, 26, 9)	(15, 39, 36, 12)	(17, 47, 45, 15	(19 57 57 19)
(12, 30, 27, 9)	(15, 40, 36, 11)	(17, 48, 45, 14)	(20, 58, 54, 16)
(12, 30, 28, 10)	(15, 40, 37, 12)	(17, 48, 46, 15	(20 59 57 18)
(12, 32, 31, 11)	(15, 40, 38, 13)	(17, 48, 47, 16	(20 60 60 20)
(12, 33, 33, 12)	(15, 41, 39, 13)	(17, 49, 47, 15	(21 62 60 19)
(13, 32, 29, 10)	(15, 41, 40, 14)	(17, 49, 48, 16	(21 63 63 21)
(13, 33, 30, 10)	(15, 42, 41, 14)	(17, 49, 49, 17	(22 66 66 22)
(13, 33, 31, 11)	(15, 42, 42, 15)	(17, 50, 50, 17)	
(13, 34, 32, 11)	(16, 42, 39, 13)	(18, 51, 48, 15)	
(13, 34, 33, 12)	(16, 43, 39, 12)	(18, 51, 49, 16)	

Table 5.1: The largest f-vectors of 4d $N(\mathcal{R})$ computed: 9 highlighted f-vectors correspond case $(3, 3, 3)$ without parallel edges.

We write the f-vectors as (f_0, f_1, f_2, f_3), omitting the $f_4 = 1$ corresponding to the unique 4-face, where f_i stands for the cardinality of i-dimensional faces. We define the *minimum* and *maximum* f-vector to be the one with minimum and maximum *number of facets*, i.e., with minimum or maximum value of f_3.

A complete list of f-vectors when $A_0 = \{(0,0),(0,1),(1,0)\}$ and $A_1 = \{(5,5), a_{11}, a_{12}\}, A_2 = \{(5,5), a_{21}, a_{22}\}$ where a_{ij} take all the possible values from the set $\{(\alpha, \beta) \mid \alpha, \beta \in \mathbb{N} \wedge \alpha, \beta \leq 10\}$ is presented in Table 5.1. There is a unique f-vector, $(22, 66, 66, 22)$, which is maximal, and corresponds to more than one input family of supports. Highlighted f-vectors correspond to triangles that share no parallel edges between any of them. Table 5.2 shows one example for each of these cases and the types of facets for each resultant polytope. The computations have been performed using `respol` and last several days. The minimum f-vector is $(6, 15, 18, 9)$ and is attained by Example 9.

Example 9 (Birkhoff polytope). Let $A_0 = A_1 = A_2 = \{(0,0),(1,0),(0,1)\}$. Then $N(\mathcal{R})$

a_{11}	a_{12}	a_{21}	a_{22}	f-vector	facets of $N(\mathcal{R})$
(0, 0)	(1, 7)	(1, 0)	(2, 6)	$(19, 55, 52, 16)$	$9\,R,\ 5\,P,\ 2\,P_8$
(0, 0)	(1, 2)	(0, 1)	(2, 6)	$(19, 56, 54, 17)$	$9\,R,\ 7\,P_5,\ 1\,P_{10}$
(0, 0)	(1, 2)	(0, 1)	(2, 0)	$(18, 54, 54, 18)$	$9\,R,\ 9\,P_5$
(0, 0)	(1, 2)	(2, 1)	(4, 7)	$(20, 59, 57, 18)$	$9\,R,\ 7\,P_5,\ 1\,P_{10},\ 1\,Z_3$
(0, 0)	(1, 2)	(0, 1)	(1, 4)	$(19, 57, 57, 19)$	$9\,R,\ 9\,P_5,\ 1\,Z_3$
(0, 1)	(9, 2)	(1, 6)	(2, 7)	$(21, 62, 60, 19)$	$9\,R,\ 9\,P_5,\ 1\,Z_4$
(0, 0)	(1, 2)	(0, 2)	(2, 3)	$(20, 60, 60, 20)$	$9\,R,\ 9\,P_5,\ 2\,Z_3$
(0, 0)	(1, 2)	(1, 0)	(3, 8)	$(21, 63, 63, 21)$	$9\,R,\ 9\,P_5,\ 3\,Z_3$
(0, 1)	(8, 3)	(0, 6)	(1, 7)	$(22, 66, 66, 22)$	$9\,R,\ 9\,P_5,\ 4\,Z_3$

Table 5.2: f-vectors correspond to case $(3, 3, 3)$ without parallel edges. Generic resultant, prism, and zonotope facets are denoted R, P_5, Z_3 respectively. They depicted in Figure 5.8. The degenerate zonotope and two different prism facets are denoted Z_4, P_8, P_{10} respectively. They depicted in Figure 5.9.

is the 4-dimensional Birkhoff polytope [140] which has f-vector (6,15,18,9).

A particular extremal case follows:

Example 10. Let $A_0 = \{(0,0),(1,0),(0,1)\}$, $A_1 = \{(0,0),\ (5,4),(9,1)\}$, $A_2 = \{(5,0),(0,1),(1,2)\}$. Then, $N(\mathcal{R})$ has f-vector $(22, 66, 66, 22)$; the vertex and facet graphs are in Figure 5.1. A non-regular subdivision of $P = \sum_i P_i$ is depicted in Figure 5.7 (see also Rem. 3).

A particular non-extremal case follows:

Example 11. Let $A_0 = \{(0,0),(2,0),(1,3)\}$, $A_1 = \{(0,0),\ (3,1),(2,3)\}$, $A_2 = \{(0,1),(3,0),(1,3)\}$. Then, $N(\mathcal{R})$ has f-vector $(21, 62, 60, 19)$. A regular subdivision of $P = \sum_i P_i$ is depicted in Figure 5.7 (see also Rem. 3).

On the other hand, the following example concerns the case of three 1-dimensional configurations, excluded from the above list.

Example 12. Let $A_0 = \{(0,0),(0,1),(0,2)\}$, $A_1 = \{(0,0),\ (1,0),(2,0)\}$, $A_2 = \{(0,0),(1,1),(2,2)\}$. Then, $N(\mathcal{R})$ has f-vector $(20, 57, 51, 14)$.

If we replace the configuration (A_0, A_1, A_2) in Example 12 by the configuration $(A_0, A_1, \{(0,0),(1,1),\ (2,3)\})$, where two parallelisms are broken without introducing any new one, the f-vector becomes $(20, 58, 54, 16)$. Note that we get higher values for the different f_i.

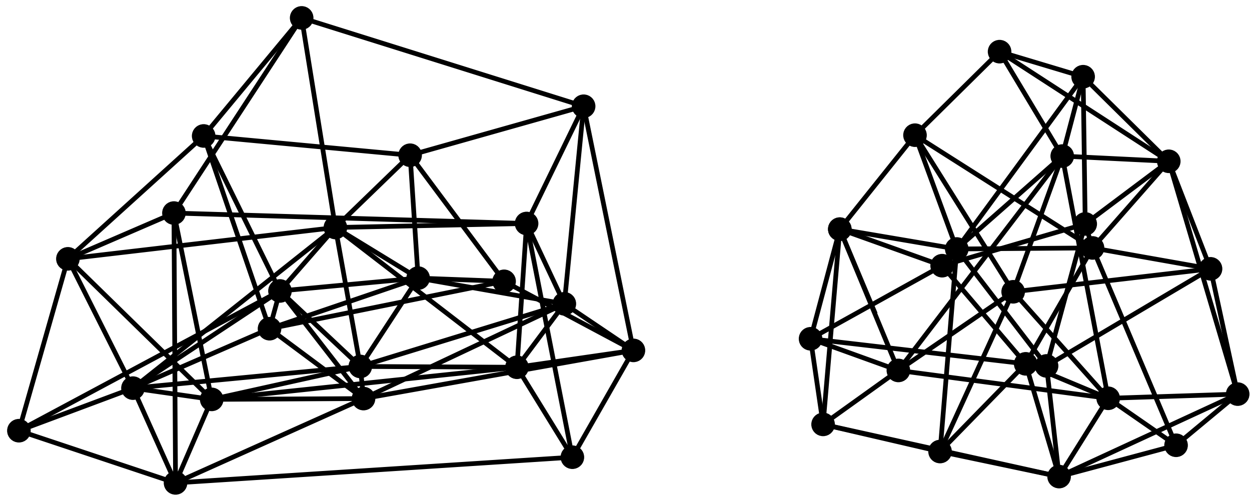


Figure 5.1: Vertex graph and facet graph (courtesy of M. Joswig) of the resultant polytope in Example 10.

An interesting observation regarding Table 5.1 is the existence of *symmetric f-vectors*. It is known that self-dual (or self-polar) polytopes enjoy this property. A polytope is self-dual if it is combinatorially equivalent to its polar-dual polytope. However, this is not the case of resultant polytopes. The polytope of Example 10 has 36 triangular and 30 parallelogram ridges while its polar-dual has 42, 19 and 5 triangular, parallelogram and pentagonal ridges respectively. Additionally, note that in the case without parallel edges symmetric f-vectors appear when the facets are generic (cf. Table 5.2). Generic facets are described in detail in the next sections. Here we observe that polytopes with symmetric f-vectors have facets that are either resultant or prism or cube depicted in Figure 5.8.

5.3.1 Input genericity maximizes complexity

In this section we prove a stronger version of the following result for the special case where the Minkowski summands are triangles.

Proposition 32. *[71, Theorem 1] Let $P = P_1 + \cdots + P_r$ be a Minkowski sum. There is a Minkowski sum $P = P_1 + \cdots + P_r$ of polytopes relatively in general position so that $f_k(P_i') = f_k(P_i)$ for all i and k, and so that $f_k(P') \geq f_k(P)$ for all k.*

Given a polytope $Q \subset \mathbb{R}^3$ and a direction $u \in \mathbb{R}^3$, its lower hull along u, denoted $\mathrm{LH}_u Q$, is the union of all facets whose outer normal has negative or zero inner product with u. In the case of zero inner product, the facet is called degenerate and its projection is not a maximal cell. We assume that the triangles $A_i = \{p_{ij}, j = 0,1,2\}$, $i = 0,1,2$, have 2d convex hulls P_i. Let S be a regular subdivision of P, and $\widehat{A}_i, \widehat{P}$ be the lifted Newton polytopes and their Minkowski sum; $\mathrm{LH}_{e_3}\widehat{P}$, where e_3 is the unit vector on the x_3-axis, is in bijection with S. Consider edges E_0, E_1 with the same outer normal v:

$$E_0 = (p_{00}, p_{01}) \subset P_0,\ E_1 = (p_{10}, p_{11}) \subset P_1.$$

For some vertex $p_{2k} \in A_2$, $E_1 + E_2 + p_{2k}$ is an edge of P with outer normal v. Thus, their lifting $\widehat{E}_1 + \widehat{E}_2 + \widehat{p}_{2k}$ has outer normal $(v, 0)$ and yields one or two facets of $\widehat{P}$, yielding one or two degenerate facets on $\mathrm{LH}_{e_3}\widehat{P}$, i.e. segments, depending on whether the lifting leads, resp., to a coarse or fine subdivision. In the latter case, the two segments are collinear but their union has been subdivided into one of two possible mixed subdivisions, each with two cells. W.l.o.g., these are:

$$\{p_{00} + E_1 + p_{2k}, E_0 + p_{11} + p_{2k}\},\ \{E_0 + p_{10} + p_{2k}, p_{01} + E_1 + p_{2k}\} \tag{5.2}$$

We consider a perturbation in the direction of v

$$p^*_{00} := p_{00} + \epsilon v, \tag{5.3}$$

with indeterminate $\epsilon \to 0^+$. Since we are considering a finite process that branches on signs of algebraic expressions, namely Cayley minors, ϵ can take sufficiently small positive rational values, as is the case in standard symbolic perturbation methods.

Lemma 33. *With the above hypotheses and notation, let*

$$\mathcal{A}^* := (\{p^*_{00}, p_{01}, p_{02}\}, A_1, A_2),$$

and $P^* := A^*_0 + A_1 + A_2$ be the family and Minkowski sum associated with a perturbation (5.3). *Let S be a (regular fine) mixed subdivision of P associated with a generic weight vector w, and S^* the regular subdivision of P^* associated to the same vector w. Then, S^* is mixed and contains at most one more cell σ than does S. There is a bijection between all cells of S^* (except σ, if it exists) and the cells of*

S, which associates combinatorially equivalent cells.

We expect this lemma to extend to any dimension.

Proof. Eq. (5.3) defines $p^*_{00} \in \mathbb{Q}^2$. As in the proof of Theorem 64, by an appropriate dilation we define a family of supports in $\mathbb{Z}^2$. By abuse of notation, we denote the latter by $\mathcal{A}^*$. To prove the lemma for any S, we consider two cases according to the subdivisions of $E_0 + E_1 + p_{2k}$ in (5.2). In the first case, $p_{00} + p_{11} + p_{2k}$ is not a vertex of P but $p^*_{00} + p_{11} + p_{2k}$ is a vertex of P^*: the perturbation has moved outward the middle point of $E_0 + E_1 + p_{2k}$. $\mathrm{LH}_{e_3}\widehat{P}$ is combinatorially equivalent to $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$, where the latter is defined by shifting our viewpoint by an infinitesimal amount: the two degenerate facets whose union is $\widehat{E}_0 + \widehat{E}_1 + \widehat{p}_{2k}$ appear in both lower hulls (the subdivision to two facets occurs because S is fine). The non-degenerate facets are clearly combinatorially equivalent in both lower hulls. Formally, non-degenerate facets on $\mathrm{LH}_{e_3}\widehat{P}$, i.e. with positive area, have outer normal $(w, -1)$ and we claim that

$$(w, -1) \cdot (e_3 + \epsilon(v, 0)) = -1 + \epsilon w \cdot v < 0,$$

for sufficiently small $\epsilon > 0$. Thus, these facets also lie on $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$. Non-degenerate facets of $\widehat{P}$ but not on $\mathrm{LH}_{e_3}\widehat{P}$ have outer normal $(w, 1)$ and we claim that

$$(w, 1) \cdot (e_3 + \epsilon(v, 0)) = 1 + \epsilon w \cdot v > 0,$$

for sufficiently small $\epsilon > 0$. So, these facets do not lie on $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$. We now show $\mathrm{LH}_{e_3}\widehat{P}^*$ is combinatorially equivalent to $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$. Any facet except the degenerate ones in $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$ clearly corresponds to a combinatorially equivalent facet in $\mathrm{LH}_{e_3}\widehat{P}^*$. The degenerate facets give rise to two edges in $\widehat{P}^*$, which proves that S^* is fine, hence a mixed subdivision; moreover, these edges are combinatorially equivalent to those on $\mathrm{LH}_{e_3+\epsilon v}\widehat{P}$. Thus the lemma is proved in the case no new cell is created.

In the second subdivision of $E_0 + E_1 + p_{2k}$, the middle point is $p_{01} + p_{10} + p_{2k}$; this point is perturbed to the relative interior of P^*. The perturbation creates an extra (mixed) cell $E^*_0 + E_1 + p_{2k}$ which intersects ∂P^*. For all other cells in S^* the discussion for the above case holds. This settles the case a new cell is created. □

Theorem 34. *For any family $\mathcal{A}$ whose triangles have one or more pairs of parallel edges, there exists a family of triangles $\mathcal{A}^*$ without any parallel edges as in* *section 5.4, whose resultant polytope $N(\mathcal{R}^*)$ has at least as many faces of any dimension as those in the polytope $N(\mathcal{R})$ of $\mathcal{A}$.*

Proof. We first assume all P_i's have non-zero area. Given $\mathcal{A}$ with strongly parallel edges $E_0 \subset P_0$, $E_1 \subset P_1$, perturbation (5.3) defines $\mathcal{A}^*$, where the corresponding edges are not parallel. In the case of other strongly parallel edges, we apply the same procedure sufficiently many times. For every mixed subdivision S of $\mathcal{A}$ the same lifting defines a mixed subdivision S^*, as in Lemma 33. This shows that the vertices of $N(\mathcal{R})$ can be mapped in a 1-1 fashion to, possibly a subset of, vertices of $N(\mathcal{R}^*)$. Hence the number of vertices in $N(\mathcal{R}^*)$ is at least as large as that of $N(\mathcal{R})$.

To prove the statement for k-faces, $k \geq 1$, we extend Lemma 33 to an arbitrary (coarse) regular subdivision S and its perturbed counterpart S^*. The only difference is that S may contain a single 1d cell $E_0 + E_1 + p_{2k}$ and cells of the form $\sigma = E_0 + E_1 + F_2$, for a face $F_2 \subset P_2$. Each σ is subdivided to 3 or 2 cells in S^*, depending on whether a new cell is created or not. The subdivision follows one of the subdivisions of $E_0 + E_1 + p_{2k}$ discussed in the proof of Lemma 33. Now σ is not essential hence contributes a point summand to the $N(\mathcal{R})$ face corresponding to S. The $N(\mathcal{R}^*)$ face corresponding to S^* is an edge if σ^* is a hexagon, thus establishing the lemma for k-faces.

If parallel edges E_0, E_1 have anti-parallel outer normals, no regular subdivision (even coarse) may contain a cell of the form $E_0 + E_1 + F_2$, though there may be adjacent cells $E_0 + p_{1j} + F_2, p_{0i} + E_1 + F_2$. Any infinitesimal perturbation, such as (5.3), yields S^* combinatorially equivalent to S.

When some P_i's have zero area, the result still holds in a similar way after a detailed study of each possible case (including repeated points), which we omit due to space restrictions. The key case is the following: $\mathcal{A}$ satisfies $|A_0| = |A_1| = |A_2| = 3$, $\dim P_0 = 1$, $\dim P_1 = \dim P_2 = 2$, then let $\mathcal{A}^* = (A_0^*, A_1, A_2)$ such that the middle point of A_0 is infinitesimally perturbed to yield $\dim P_0^* = 2$. Then there is an injection of regular subdivisions of $\mathcal{A}$ to those of $\mathcal{A}^*$, such that if S maps to S^* then S^* contains one more cell equal to $A_0^* + p_{1j} + p_{2k}$, for vertices $p_{1j} \in A_1, p_{2k} \in A_2$, and all other cells are combinatorially equivalent to the corresponding cells in S. □

5.4 The case $(3,3,3)$ with non-parallel edges

In this section, we assume that we have an essential family with $n = 3,\ m = 9$ and each A_i has cardinality 3, i.e., $\dim(P_i) = 2$, for all i. Moreover no edges coming from two different P_i are parallel, i.e. any pair of edges $e \in P_i,\ e' \in P_j$ where $i \neq j$ for all i, j are non-parallel. This is an essential $(3,3,3)$ configuration $\mathcal{A} = (A_0, A_1, A_2)$.

5.4.1 Polar mixed subdivisions

Recall that a regular mixed subdivision of $\sum_i \mathcal{A}_i \subseteq \mathbb{R}^2$ can be obtained as the projection of the lower hull $\widehat{P}_\ell$ of the Minkowski sum $\widehat{P} = \sum_i \widehat{A}_i$ of the lifted point sets $\widehat{A}_i$, for some lifting to $\mathbb{R}^3$.

Given a vector $w \in \mathbb{R}^d$ and polytope $P \subseteq \mathbb{R}^d$, denote

$$face_w(P) = \{x \in P \mid x \cdot w \geq y \cdot w, \forall y \in P\}$$

the extremal face of P w.r.t. w. Then w is an outer normal to the face. The normal cone of P at face F is

$$NC(F) = \{w \in \mathbb{R}^d \mid face_w(P) = F\}.$$

The normal fan of P is the collection of all normal cones of P i.e.

$$NF(P) = \{NC(F) \mid F \text{ a face of } P\}.$$

Given a regular mixed subdivision S of P, we define the *polar (dual) mixed subdivision* or *configuration* S^* to be the intersection of a generic hyperplane h in the dual space $(\mathbb{R}^3)^*$ with the normal fan $NF(\widehat{P}_\ell)$. Given $A_i \subset \mathbb{Z}^2$ for some $i \in \{0,1,2\}$ we call A_i^* the *dual triangle*. The dual of an edge of A_i is called a *ray* and the dual of a vertex of A_i is called a *cone*.

There is a bijection from vertices, edges and facets of $\sum_i \widehat{A}_i$ to cells, edges and vertices of S^* respectively, which induce a bijection from vertices, edges and cells of S to cells, edges and vertices of S^* respectively. Moreover, there is a bijection of boundary vertices and edges of S to cones and rays of S^*. The above construction is similar to the construction of the polar (dual) polytope defined in Section 5.5.

Lemma 35. *Given an edge e from some A_i, consider all edges in S that have e as Minkowski summand. Then, the union of their polar edges in S^* is a ray.*

Proof. Consider the set of edges of $\widehat{P}_\ell$ whose summand is the lifted e. The union of the normal cones of the edges in that set is two dimensional. To see this observe that all the edges have the same edge as summand (and the other summands are vertices) and thus they are translations of the same edge. The intersection of that union with h is a ray. □

Observe that rays in S^* are in bijection with edges of A_i's. Figure 5.2 provides an illustration. We call *mixed* the points that are defined as an intersection of three rays from different A_i's. We define a_{A_i} to be the apex of the dual triangle A_i^*, that is the common apex of its rays, and define a_r to be the apex of a ray r. Let n_r denotes the normal vector parallel to r.

Given $A_0, A_1, \dots, A_n \subset \mathbb{Z}^n$ we define $\text{int}(A_0, A_1, \dots, A_n)$ to be the maximum number of intersection points among $A_0^*, A_1^*, \dots, A_n^*$. Given a ray $r \in A_i^*$, let $\text{int}_r(C)$ be the maximum number of intersection points of a ray r with A_j^* when $a_r \in C \subset A_j^*$ and $i \neq j$.

Consider the 3 rays of A_i^* and count the intersection points with some A_j when $i \neq j$. We call *signature* of A_i the multiset of intersection cardinalities of the 3 rays of A_i^* with respect to some A_j when $i \neq j$. Note that the each element of the signature that correspond to a ray r is less than $int_r(C)$, where C is the cone such that $\alpha_{A_i} \in C$. The *signature vector* of A_i is the vector whose i-th coordinate is the intersection cardinality of the i-th ray of A_i^*.

Lemma 36. *Given ray $r \in A_i^*$ and cone $C \subseteq A_j^*$ such that $i \neq j$ then*

$$int_r(C) = \begin{cases} 0, & \textit{if } n_r \in C, \\ 2, & \textit{if } -n_r \in C, \\ 1, & \textit{otherwise.} \end{cases}$$

Proof. Define H_r to be the halfplane defined by the supporting line of r for which there exist a unique cone $C' \in A_0^*$ such that $C \cap H_r = C$. Let $a_r \in H_r$. If $n_r \in C$ then r does not intersect any cone. If $-n_r \in C$ then r does intersect C' and $int_r(C) = 2$. Otherwise, $n_r \in C'$ and $int_r(C) = 1$. □

If we sum over all cones of A_j of Lemma 36 then we have the following.

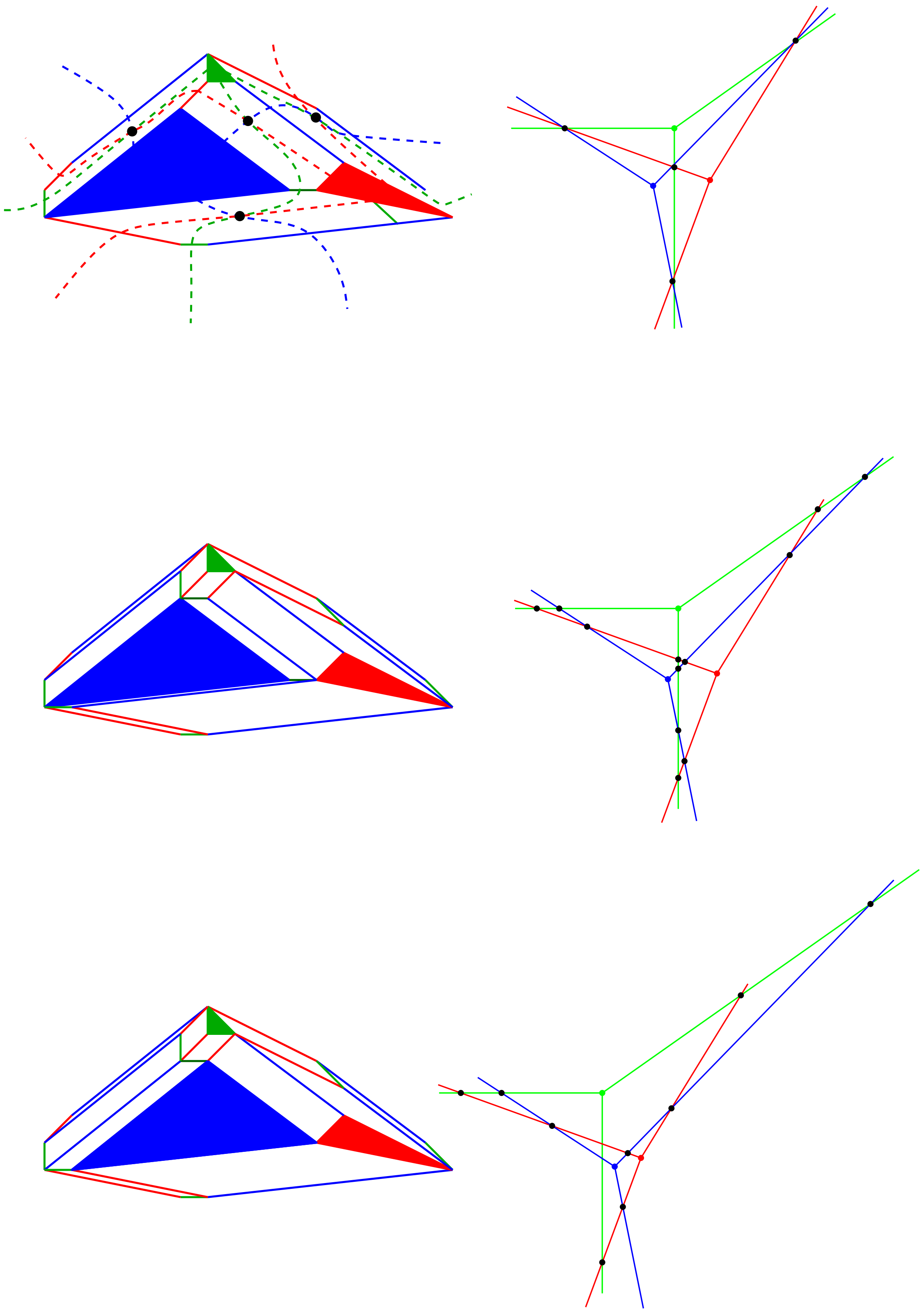


Figure 5.2: A subdivision S and its polar dual S^*.

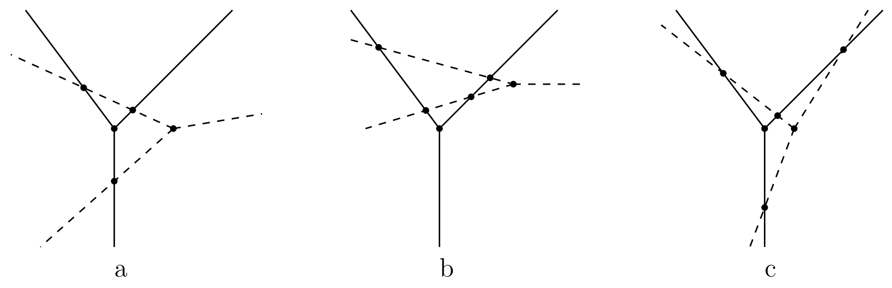


Figure 5.3: Arrangements of two polar triangles with ≥ 3 intersection points.

Corollary 37. *For any $A_j \subset \mathbb{Z}^2$ it holds $\sum_{C \in A_j^*} int_r(C) = 3$, where the sum is over all cones C of A_j^* and r is a ray of A_i^* for some $i \neq j$.*

5.4.2 Bounds on the number of cells in a subdivision

We study upper bounds on the number of cells of a subdivision. This can be derived from the upper bound on the number of facets of the Minkowski sum of the lifted A_i's for any lifting function.

Given polytopes $P_1, \dots, P_r$ in $\mathbb{R}^d$, Gritzmann and Sturmfels [77] prove that

$$f_i \leq 2\binom{k}{i} \sum_{j=0}^{d-1-i} \binom{k-1-i}{j}$$

where k denote the number of non-parallel edges of $P_1, \dots, P_r$, and here f_i denotes the cardinality of i-dimensional faces of $P_1 + \cdots + P_r$. The equality holds if $P_1, \dots, P_r$ are zonotopes. For our case where $k = 9$, $i = 2$, $d = 3$, $r = 3$ the above formula gives $2\binom{9}{2} = 72$. Thus, the best upper bound that can be implied for the cells of the lower hull and therefore for the cells of a subdivision is 36.

In the sequel we introduce a tool that allow as to derive an better upper bound, namely 15. Similar constructions have been used in tropical geometry [134].

We first prove a technical lemma.

Lemma 38. *For fixed (A_0, A_1, A_2) there is no subdivision S with cells $a + s + s'$ and $a' + s + s'$ for any points $a \in A_i$, $a' \in A_j$, segments $s \in P_k$, $s' \in P_l$, $i, j, k, l \in \{0, 1, 2\}$.*

Proof. If there exist a subdivision S with cells $a + s + s'$ and $a' + s + s'$, then there is a lifting on A_i's such that the lower hull of the Minkowski sum of the lifted P_i's contains the facets $a^* + s^* + s'^*$ and $a'^* + s^* + s'^*$. Where * denotes the lifted faces. Note that the normal vectors of these facets are equal, a contradiction. $\square$

A basic property of polar mixed subdivisions follows.

Corollary 39. *Two rays in S^* have at most one common intersection point.*

Lemma 40. *For any $A_0, A_1 \subset \mathbb{Z}^2$, $int(A_0, A_1) \leq 4$.*

Proof. Fix A_0 and assume there is an A_1 such that $int(A_0, A_1) = 5$. Observe that there should exists a signature $(2, 2, 1)$, since there are 5 intersection points and each ray can intersect a dual triangle at most twice by Corollary 37. By Lemma 36, the signature $(2, 2, 1)$ forces the inverse of 2 normals to be in C and forbids the third from being in C. Note that there is no triangle A_1 with such a property; a contradiction. □

Lemma 41. *For any configuration S^* of $A_0^*, A_1^*, \ldots, A_n^*$ where $|A_0| = |A_1| = \cdots = |A_n| = 3$ it holds $int(A_0, A_1, \ldots, A_n) \leq 4\binom{n}{2}$.*

Proof. By Lemma 40 a configuration of two dual triangles has at most 4 intersections. In any configuration of n triangles each of the $\binom{n}{2}$ pairs of triangles will contribute at most 4 intersections in the total intersections and thus at most $4\binom{n}{2}$. □

By duality the following result holds for the special case of $n = 3$ triangles.

Corollary 42. *For fixed (A_0, A_1, A_2) where $|A_0| = |A_1| = |A_2| = 3$ there is no subdivision with more than 15 cells.*

Lemma 43. *For fixed (A_0, A_1, A_2) there is no subdivision with more than 10 and 6 cells when w.l.o.g. $(|A_0| = |A_1| = 3, |A_2| = 2)$ and $(|A_0| = 3, |A_1| = |A_2| = 2)$ respectively.*

Proof. The polar of A_i is a line when $|A_i| = 2$. Hence, for the case $|A_0| = 3, |A_1| = |A_2| = 2$ we have at most 2 intersections between a line and a polar triangle (i.e. the polar of A_0) and at most 1 intersection between two lines. Therefore, there are in total at most 5 intersection points plus the point of the polar triangle that yield a bound of at most 10 cells in any subdivision.

For the case $|A_0| = |A_1| = 3, |A_2| = 2$, we have at most 4 intersection points between the two polar triangles by Lemma 40. Moreover, the line that corresponds to A_2 can have at most 2 intersection points with each of the polar triangles. In total there are at most 8 intersection points and two points of the polar triangles yielding a bound of at most 10 cells in any subdivision. □

5.4.3 Bounds on the number of types of subdivisions

First we study the case of intersection of two dual triangles, A_0^*, A_1^*. We restrict to the cases with at least 3 intersections. By Corollary 37 each coordinate of the signature vectors is at most 3. Then we have the following cases of signatures: $\{1,1,1\}, \{2,1,0\}, \{2,2,0\}, \{2,1,1\}$ that denote intersection cardinalities in the 3 rays of A_0^* with A_1^*. By Lemma 40 the sum of coordinates is at most 4. Then we have the following corollary.

Corollary 44. *The only possible cases in the intersections of two dual triangles with at least* 3 *intersection points are the* $\{1,1,1\}$, $\{2,1,0\}$, $\{2,2,0\}$, $\{2,1,1\}$ *depicted in Figure 5.3.*

The cases $\{1,1,1\}$ and $\{2,1,0\}$ are symmetric. A dual triangle has exactly one intersection point at each ray. They depicted in Figure 5.3 (a), $\{2,2,0\}$ is depicted in Figure 5.3 (b) and $\{2,1,1\}$ in Figure 5.3 (c).

Lemma 45. *Given a fixed set* A_0, A_1 *either* $\{2,2,0\}$ *or* $\{2,1,1\}$ *will occur in the set of all subdivisions.*

Proof. Assume that A_0, A_1 are fixed and there exist two subdivisions of $A_0 + A_1$ with signatures $\{2,2,0\}$ and $\{2,1,1\}$ respectively. First note that a_{A_1} is forced to belong in different cones of A_0^* in each of the two subdivisions. By Corollary 37 the per coordinate sum of the signature vectors is a vector where every coordinate is ≤ 3. Thus, the two signature vectors in our case are $(2,1,1)$ and $(0,2,2)$. Hence the signature vector for placing the apex of A_1 in the third cone has two 0 coordinates and the third coordinate should be ≤ 1. The contradiction comes from the fact that it is always possible to place any dual triangle in the cone of another in order to have at least two intersection points. □

Consider the normal fan $NF(P)$ of $P = A_0+A_1+A_2$. A cone A_i for some i is called *empty* if it contains no ray in $NF(P)$. We call a cone of A_i^* *full* if it contains one non-empty cone of A_j and one non-empty cone of A_k in $NF(P)$, where $i \neq j \neq k$.

Consider all configurations with at least 3 mixed points. We categorize them by the number of apexes of the dual triangles that are in the convex hull T of the 3 mixed points. Clearly, the only possible cases are $3, 2, 1, 0$ denoted $(a), (b), (c), (d)$

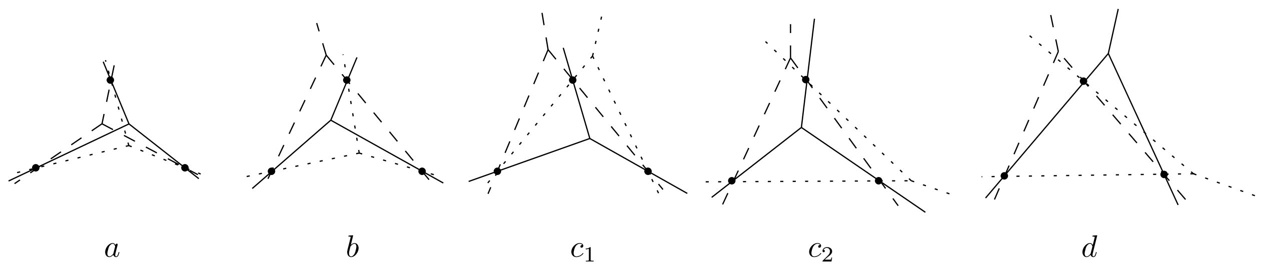


Figure 5.4: Arrangements of three polar triangles with 3 mixed points.

respectively. They are depicted in Figure 5.4. Observe that a dual triangle has either its apex in T or out with one of its rays passes through two mixed points.

Observe that configurations of type (a) have signatures $\{2, 1, 1\}$, since all the dual triangles are in T and therefore they have no intersection in each ray and there is one more intersection point between each pair of dual triangles that lays inside T. Configurations of type (b) have 1 empty cone that belongs to the dual triangle whose apex is outside T. Moreover, the dual triangles which apexes are inside T have $\{2, 1, 1\}$ signatures. Configurations of type (c) are of two kinds. The first case (c_1), are those with signature $\{2, 1, 0\}$ in the two dual triangles, which apexes are outside T (Cf. Figure 5.4 (c_1)). The second case (c_2), are those with signature $\{2, 2, 0\}$ in the two dual triangles, which apexes are outside T (Cf. Figure 5.4 (c_2)). Observe that (c_1) has 1 full cone that belongs to the dual triangle whose apex is inside T. Observe that (c_2) has 2 empty cones, one that belongs to each dual triangle that lay outside T. Configurations of type (d) have at least one empty cone and signature $\{2, 2, 0\}$ (that belongs to the blue dual triangle in Figure 5.4 (d)).

Lemma 46. *The configuration of type* (a) *can appear at most once.*

Proof. Observe that the sequence of rays in a mixed point is uniquely defined by the normal fan of P. This implies a unique configuration of type (a). □

Lemma 47. *Given a fixed set* A_0, A_1, A_2 *the sum of the number of empty and the number of full cones is at most* 3. *In particular, the possible cases are* $3, 2, 1, 0, 0$ *full and* $0, 0, 1, 2, 3$ *empty cones respectively.*

Proof. Observe that every dual triangle has at most one full and one empty cone. By the definition of full if there exist one in A_i then there is no empty in A_j, A_k, where $i \neq j \neq k$. This yields possible cases of one full and two empty and two full and zero empty. This completes the proof since in all the possible cases the sum of the number of empty and the number of full cones is at most 3. □

Lemma 48. *Given a fixed set A_0, A_1, A_2, a configuration of type (d) can occur at most once. It has two pairs of dual triangles with signature $\{2,2,0\}$ and implies the existence of at least one empty cone.*

Proof. We consider the 3 lines that support the edges of T and take cases of the apices of the dual triangles on these lines. Let a, b, c the three mixed points and $\ell_{ab}, \ell_{bc}, \ell_{ac}$ the three lines index by the points that they intersect. W.l.o.g. let $\alpha_0, \alpha_1, \alpha_2$ lay on $\ell_{ab}, \ell_{ac}, \ell_{bc}$ respectively. Moreover, w.l.o.g. α_0 lay on the side of a. First, if α_1 lay on the other side of a then this implies a $\{2,2,0\}$ signature. Then placing α_2 on the one side of bc implies a $\{2,2,0\}$ signature with the one dual triangle and placing to the other side implies a $\{2,2,0\}$ signature with the other dual triangle. Second, if α_1 lay on the same side of a as α_0 any placement of α_2 implies a $\{2,2,0\}$ signature with both dual triangles. This proves the existence of at least two pairs with $\{2,2,0\}$ signatures.

Since there are two pairs of dual triangles with $\{2,2,0\}$ signature there is one dual triangle, say A_0^*, that belongs to both pairs. Now observe that a $\{2,2,0\}$ signature implies one empty cone to each participant dual triangle. Thus, A_0^* has an empty cone. This completes the proof of the existence of at least one empty cone.

To show that there is only one occurrence of (d), observe (similar to the proof of Lemma 46) that the sequence of rays in a mixed point is uniquely defined by the normal fan of P. This implies a unique configuration of type (d). □

Proposition 49. *Given a fixed set A_0, A_1, A_2 there are at most 4 distinct triplets of mixed points that appear in the set of all subdivisions.*

Proof. Since in case (a) the three pairs of dual triangles all have a $\{2,1,1\}$ signature, we know (a) cannot occur together with (c_2) nor with (d) in the set of all configurations for a fixed input. On the other hand, (a) may occur together with (b) and (c_1). By Lemma 47, there are at most 3 triplets of mixed points of type (b) or (c_1), and by Lemma 46 there is at most one triplet of mixed points of type (a). In total, there are at most 4 triplets of mixed points.

We consider two cases where (a) does not appear and (d) appears. The appearance of (d) implies there is at most one pair with $\{2,1,1\}$ since (d) has two

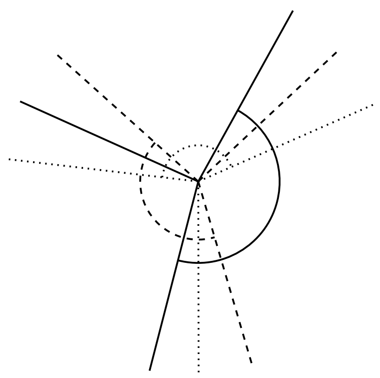


Figure 5.5: The normal fan of the Example 10.

pairs with $\{2, 2, 0\}$ by Lemma 48. This implies the appearance of at most one (b). Now the two cases depend on whether (c_1) or (c_2) may occur. The appearance of (d) implies there is at least one empty cone by Lemma 48, hence at most one full cone by Lemma 47, which implies that there is at most one occurrence of (c_1). Thus, in total, there are at most 3 triplets of mixed points. In the other case, the maximum number of the occurrences of (c_2) and (b) is 3, since there are 3 pairs of dual triangles that have either $\{2, 2, 0\}$ or $\{2, 1, 1\}$ signature and imply the occurrence of either (c_2) or (b), respectively. Since (d) occur only once, in total, there are at most 4 triplets of mixed points.

We then consider the case where none of (a) and (d) appear. Then the two possible cases of $(b), (c_1)$ and of $(b), (c_2)$ yield at most 3 mixed triplets, by Lemmas 47 and 45 respectively.

Therefore, 4 is the total maximum number of triplets of mixed points. $\square$

The above bound is tight as shown in Example 10, which corresponds to the first case, namely (a) and three (c_2) occurrences. Figure 5.5 illustrates this case. Observe the existence of the 3 full cones, the absence of an empty cone and the existence of the (a). We do not have an example with 4 triplets of mixed points in the case of $(b), (c_2), (d)$, which would have made the corresponding analysis tight.

5.4.4 Subsystems and cells of subdivisions

Let us now describe the subsets $A_i' \subset A_i$, which form *subsystems*, that define cells of a subdivision S of P, and their connection to the faces of $N(\mathcal{R})$. Clearly, the possible cells of S are a point, a segment, or an i-gon for i in $\{3, \ldots, 9\}$. *Non-*

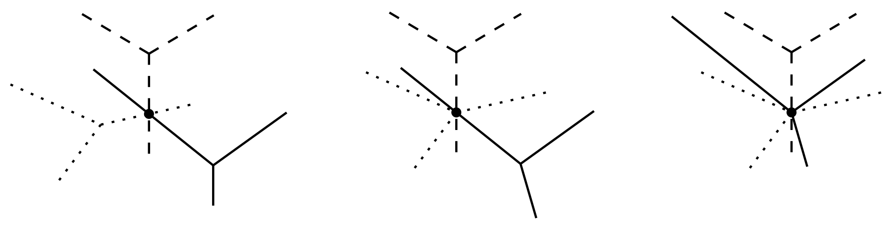


Figure 5.6: The dual points of a hexagon, a heptagon and an octagon cell (from left to right).

trivial subsystems are those that correspond to a face of $N(\mathcal{R})$ of dimension $1, 2$, or 3.

Lemma 50. *The only non-trivial subsystems in the* $(3, 3, 3)$ *case of non-parallel edges are the* 6, 7, 8*-gons or the hexagon, heptagon and octagon respectively.*

Proof. The triangular cells and parallelogram cells (Minkowski sums of two segments) correspond to zero dimensional faces of $N(\mathcal{R})$ (Proposition 4.1 in [131]). A pentagonal cell (Minkowski sum of a triangle, a segment and a point) also corresponds to a zero dimensional face of $N(\mathcal{R})$ because it corresponds to a non-cubical flip (see also Theorem 5.2 in [131]). A 9-gon (Minkowski sum of three triangles) is P and thus corresponds to the whole 4-dimensional $N(\mathcal{R})$. □

Hexagon. The simplest non-trivial subsystem includes 2-element subsets A_i' in each A_i (namely edges). Such a subsystem is essential when no two of the convex hulls of the A_i' are parallel. In this case, the cell in S is a Minkowski sum of 3 edges from the different A_i's which we call a *hexagon*. Every hexagon can be refined in two possible ways to a regular mixed decomposition and corresponds to an *edge* of $N(\mathcal{R})$ (see the cubical flips discussed above).

Remark 1. In the regular subdivision of an essential family, the existence of two hexagons implies a parallelogram resultant face for a $(3, 3, 3)$ family, or resultant facet for a $(2, 3, 3)$ family.

Fact: Each hexagon corresponds to a full-dimensional circuit hence (Sturmfels) it can be refined without affecting the rest of the subdivision. Thus, refining the two hexagons defines 4 points in $N(\mathcal{R})$.

These points are vertices iff the subdivision is regular because, if the overall subdivision is regular before refinement, the new subdivision is again regular (we have such a Lemma).

We now show that it is NOT possible for the 4 points to be collinear (Irrespective of whether the hexagons are contained in a regular subdivision or not). Assume $X_1 \cap X_2$ is an edge of P_0 and, by convexity, this is the unique common edge. Let

$$X_1 = (p_{00}, p_{01}) + (p_{10}, p_{11}) + (p_{20}, p_{21}), X_2 = (p_{00}, p_{01}) + (p_{10}, p_{12}) + (p_{20}, p_{22}).$$

Refining X_1, X_2 each in two ways corresponds to two segments (possibly resultant edges) with directions

$$(a, -a, 0, b, -b, 0, c, -c, 0),\ (a', -a', 0, b', 0, -b', c', 0, -c') \text{ for } a, b, c, a', b', c' > 0.$$

The zero pattern in each vector implies that the two segments cannot be parallel.

Two hexagons in S give rise to the Minkowski sum of two segments that form a parallelogram 2-face of $N(R)$ in the general case. Similarly, three hexagons in S give rise to the Minkowski sum of three segments that form a 2-face of $N(R)$ which is a hexagon (cf Figure 5.9a) or a facet of $N(R)$ which is a 3-cube (cf last polytope in Figure 5.8). In general, k hexagons in S give rise to the Minkowski sum of k segments, also known as a zonotope, that form a face of $N(\mathcal{R})$ of proper dimension (cf. proof of Lemma 57).

Heptagon. A *heptagon* cell is a Minkowski sum of an A_i, w.l.o.g. A_0, and 2 edges A'_1, A'_2 from A_1, A_2 respectively, which form an essential subfamily $\mathcal{A}' = (A_0, A'_1, A'_2)$ provided the A'_1, A'_2 are not parallel. The heptagon contains a hexagon cell which is the sum of one edge of A_0 and $A'_1 + A'_2$, where the former is not parallel to any of A'_i.

Remark 2. If it lies in a regular subdivision, then the heptagon has up to 3 refinements, each preserving regularity. To see this, observe that all fine subdivisions contain mixed cell $v_{0i} + A'_1 + A'_2$, for some vertex $p_{0i} \in A_0$; clearly, this cell refines the hexagon. The fine subdivisions correspond to up to 3 choices for p_{0i}.

In this general case, a heptagon corresponds to a triangular 2-face of $N(\mathcal{R})$.

Octagon. An *octagon* cell is a Minkowski sum of two A_i, w.l.o.g. A_0, A_1, and an edge A'_2 of A_2. If it lies in a regular subdivision it give rise to a facet of $N(\mathcal{R})$, in particular, the first polytope in Figure 5.9.

Lemma 51 (Technical). *Assume that we have, for $i \in \{0,1,2\}$, the subsets $A'_i \subset A_i$. Consider a regular subdivision of $A_0 + A_1 + A_2$ defined by lifting w containing cells $B_j = B_{j0} + B_{j1} + B_{j2}$, for j ranging in a finite set J, where $B_{ji} \subset A'_i$. Then, it is possible to construct a regular subdivision of $A'_0 + A'_1 + A'_2$ containing all cells B_j, by restricting w to the A'_i.*

Proof. By assumption, $\widehat{B}_{j0}+\widehat{B}_{j1}+\widehat{B}_{j2}$ is a facet, for every $j \in J$, on the lower hull of $\widehat{A}_0+\widehat{A}_1+\widehat{A}_2$, obtained by lifting according to w. Then w, restricted to the A'_i, yields a regular subdivision of $A'_0+A'_1+A'_2$. Moreover, the hyperplane of $\widehat{B}_{j0}+\widehat{B}_{j1}+\widehat{B}_{j2}$ supports the lower hull of $\widehat{A}'_0+\widehat{A}'_1+\widehat{A}'_2$, hence $\widehat{B}_{j0}+\widehat{B}_{j1}+\widehat{B}_{j2}$ is a facet of this lower hull for every $j \in J$. □

Lemma 52 (Technical). *Given a regular coarse subdivision S with a coarse cell C, it is possible to define a new regular subdivision S' that contains a fine subdivision of C and, for every other cell of S, it contains either the same cell or a refinement of it.*

In this setting, if $H \subset S$ is heptagon $A_0 + s_1 + s_2$ and $a \in A_0$ is a specific vertex, it is possible to create in S' a hexagon $X = s_0 + s_1 + s_2$, where $s_0 = conv(A_0 \setminus \{a\})$.

Proof. Let w be the lifting defining S. Let $C = C_0 + C_1 + C_2$, $C_i \subset A_i$, and consider a sufficiently generic lifting σ which is nonzero only for the C_i. Now consider lifting $w + \epsilon\sigma$, for sufficiently small $\epsilon > 0$. This is different from w therefore, by construction of σ, it yields a fine subdivision of C. The new lifting does not affect any fine cell of S but may refine coarse cells of S whose expression includes at least one point in the C_i's for which σ is nonzero.

For the second part, let $C = H$ and let σ be zero for every support point except a. Then S' is regular, X is created by refining H, and all other cells of S are either maintained or refined. Those that are refined contain a in their summand from A_0. □

5.4.5 Types of $N(\mathcal{R})$ facets

We describe resultant polytopes corresponding to maximum facet cardinality. We start with several lemmata that serve as tools later.

First we consider the case $(3, 3, 2)$.

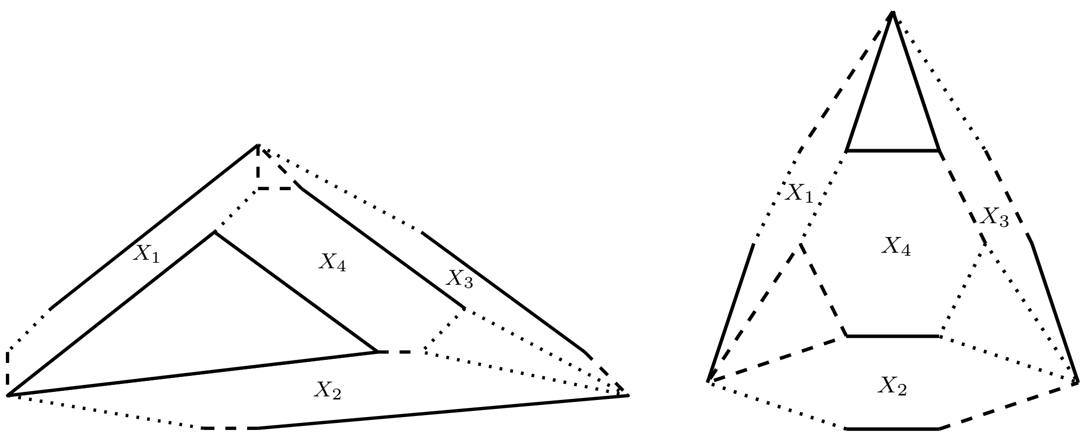


Figure 5.7: Left: A non-regular subdivision of P of Example 10. A regular subdivision of P of Example 11. Each subdivision has 4 hexagons X_1, X_2, X_3, X_4.

Lemma 53. *Given is an essential* $(3,3,3)$ *configuration* $\mathcal{A} = (A_0, A_1, A_2)$*. When there are no parallel edges in the subfamily* $\mathcal{A}' = (A_0, A_1, A_2')$*,* $A_2' \subset A_2$*,* $|A_2'| = 2$*, their corresponding resultant polytope* $N(\mathcal{R}_{\mathcal{A}'})$ *gives a facet of* $N(\mathcal{R})$*, which is combinatorially equivalent to the first polytope of Fig 5.8 (resultant).*

Proof. If all edges in $\mathcal{A}'$ are non-parallel, then every set of subsets of two points from each A_0, A_1, together with A_2', define a hexagon within a regular subdivision of $\mathcal{A}'$. For this, it suffices to lift only the third vertex of A_0 and of A_1; moreover, the hexagon can be refined independently of the rest of the subdivision because it forms a full-dimensional circuit. This gives a different resultant edge direction of $N(\mathcal{R}_{\mathcal{A}'})$. Thus, the number of edge directions of $N(\mathcal{R}_{\mathcal{A}'})$ is 9.

The dimension of $N(\mathcal{R}_{\mathcal{A}'})$ is 3 hence, by [131, Cor.6.3(b)], there are three types of 3-dimensional resultant polytopes, see also Prop. 29. Two of these, namely the tetrahedron and square-based pyramid, have 6 and 8 edges, respectively. Hence the claim follows. $\square$

Assume we have

$$\text{hexagon } X = s_0 + s_1 + s_2, \text{ and heptagon } H = A_0 + s_1' + s_2', \tag{5.4}$$

where $s_i, s_i' \subset A_i$ are all of cardinality 2, for $i \in \{1,2\}$, with the corresponding support sets for X and for H being essential. The next lemma proves that edges s_i, s_i' are distinct, $i \in \{1,2\}$.

Lemma 54. *Given any regular subdivision of $A_0 + A_1 + A_2$ which includes a heptagon H and a hexagon X, there is exactly one edge from A_i, for some $i \in \{0, 1, 2\}$, which appears in the expression of both H and X.*

Proof. Observe that X, H always share one edge since the latter has a triangle as a summand: w.l.o.g., this edge is $s_0 \subset A_0$, using the notation of expression (5.4). Thus, if they have one more common edge, this should be from A_1 or A_2. W.l.o.g. it is edge $s_1 = s_i' \subset A_1$. By lemma 51, we can construct a subdivision of $A_0+s_1+A_2$ that contains X, H. This yields a 3d (prism) or 2d (trapezoid or pentagon) resultant polytope which is the Minkowski sum of a segment and the polytope or polygon corresponding to H. Neither of these exists in the list of Proposition 29. □

Lemma 55. *If we fix an edge $s_0 \subset A_0$, there is at most one way to construct a regular subdivision which contains a hexagon and a heptagon sharing s_0 (i.e., where their intersection equals a copy of s_0).*

Proof. Assume there are two such subdivisions: one with X, H, following the notation of expression (5.4), and one with $X^* \subset s_0 + A_1 + A_2$, $H^* = A_0 + s_1^* + s_2^*$, $s_i^* \subset A_i$, for $i \in \{1, 2\}$. H can be refined just enough so as to create a hexagon without destroying overall regularity, by Lemma 52 and Remark 2, namely hexagon $s_0 + s_1' + s_2'$, which is different from X. The two hexagons define a parallelogram facet, by remark 1, in the resultant polytope of $\{s_0, A_1, A_2\}$.

Analogously, we can subdivide H^* s.t. it contains $s_0+s_1^*+s_2^*$ as a cell, which is different from X^*, defining a parallelogram facet in the same resultant polytope. Thus, we have defined two parallelogram facets which are distinct, because the subdivisions containing X, H and X^*, H^* are distinct, since they were so at the beginning and remain so after refinement. But it is impossible to have two such facets in the 3d resultant polytope of $s_0 + A_1 + A_2$, by examining the possible 3d polytopes. □

Corollary 56. *There is no regular subdivision with an octagon and a hexagon cell, nor with two heptagon cells.*

Proof. Assume that there is a subdivision with an octagon cell and a hexagon cell X. Then we can subdivide the octagon cell in (at least) two ways such that there is a heptagon cell in the octagons subdivision. This yields two subdivisions of P with a different heptagon cell each and X, which contradicts Lemma 55.

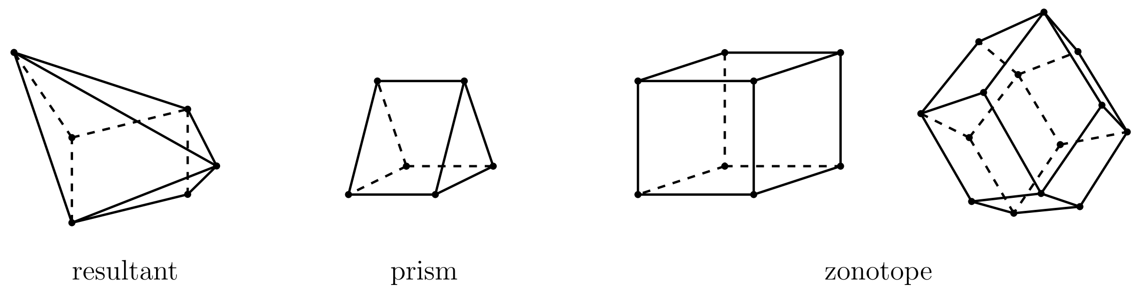


Figure 5.8: The 3 types of generic facets of 4d resultant polytopes in the $(3, 3, 3)$ case with non-parallel edges. The black, red, blue colors indicate the 3, 2, 1 dimensional resultant polytopes respectively.

Assume now there is a subdivision with two heptagon cells H, H' written, w.l.o.g., $H = A_0 + s_1 + s_2$ and $H' = s_0 + A_1 + s'_2$. We apply the second part of lemma 52, by using function σ that lifts vertex $A_0 \setminus s_0$. This creates a new regular subdivision containing hexagon $X = s_0 + s_1 + s_2 \subset H$, which contradicts Lemma 54, since X shares two edges with H'. □

Alternative proof. Let, $i \neq j \neq k$ have values $\{0, 1, 2\}$. The dual point of an octagon cell is two apexes of the dual triangles A_i^*, A_j^* intersected by a ray from A_k^*. If we have a hexagon cell with an octagon cell in a configuration this implies that a ray from A_i^* and a ray from A_j^* should intersect at some point different than the apexes of the dual triangles. This is not possible since the apexes of A_i^*, A_j^* coincide and the rays are non parallel. Given a configuration the dual point of a heptagon cell is the apex of a dual triangle A_i^* intersected by a ray from A_j^* and one from A_k^*. If there exist another heptagon cells in this configuration then the apex of either A_j^* or A_k^* should be intersected by a ray from A_i^* which is impossible since there are no parallel rays. See also Figure 5.6 for an illustration. □

We are now ready to describe all the possible types of facets. We call *resultant* facets the facets that correspond to a 3-dimensional resultant polytope; *prism* facets the facets that correspond to Minkowski sums of a 2-dimensional resultant polytope and 1-dimensional resultant polytopes; and *zonotope* facets the facets that correspond to Minkowski sums of 1-dimensional resultant polytopes.

Lemma 57. *If the 3 triangles A_i share no parallel edges, then the only possible facet types are resultant, prism and zonotope.*

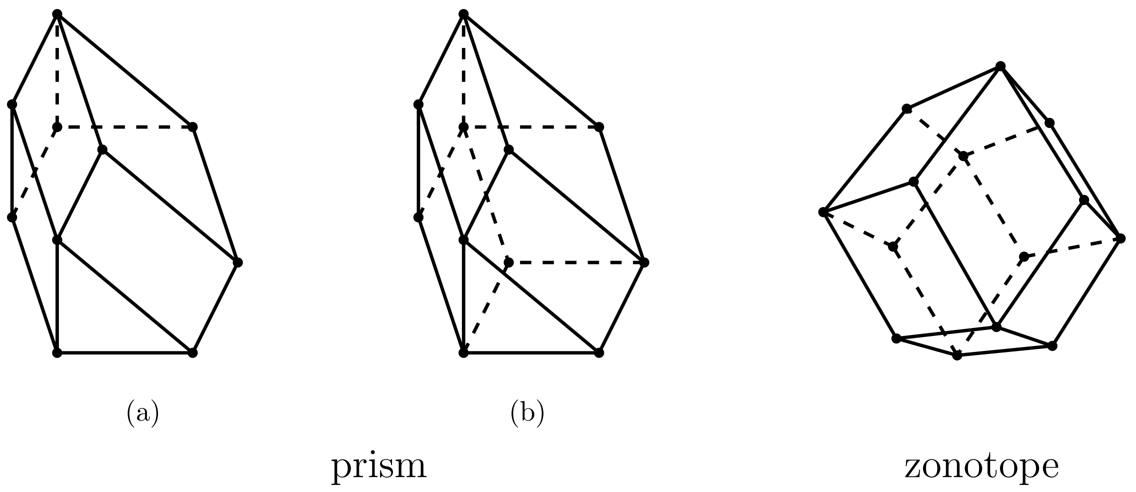


Figure 5.9: Types of degenerate facets of 4d resultant polytopes in the $(3, 3, 3)$ case with non-parallel edges. The red and blue colors indicate the 2 and 1 dimensional resultant polytopes respectively.

Proof. Considering that the facets are Minkowski sums of lower-dimensional resultant polytopes [131]. Let S a subdivision of P. By Corollary 42 the number of mixed cells of S is at most 12. The possible non-trivial cells by Lemma 50 are the hexagon, the heptagon and the octagon, that can be subdivided in at most $3, 5, 8$ mixed cells respectively by Lemma 43. Using these cardinalities we perform a case analysis over all these possible Minkowski sums and hence $N(\mathcal{R})$ facets.

If S contains an octagon, then the only other non-trivial cell is hexagon. This is impossible by Corollary 56. Then the sum has a 3-dimensional summand and the facet is the resultant facet of Figure 5.8 with f-vector $(6, 11, 7)$.

If S contains a heptagon, then the only possible cases for the other non-trivial cells are a heptagon (which is impossible by Corollary 56 and one or two hexagons. The case of a heptagon and a hexagon yields generically a $N(\mathcal{R})$ facet which is a Minkowski sum of a segment and a triangle, called prism (cf. Figure 5.8). The case of a heptagon and two hexagons yields a face which is a Minkowski sum of two segments and a triangle. If all the summands are relatively in general position the sum is a 4 dimensional polytope that contains two prism facets. Otherwise, we encounter the degenerate cases depicted in Figure 5.9.

The last possible case is to have k hexagons in S. These yield facets that are Minkowski sums of segments, i.e. zonotope facets. By the discussion above it follows $k \leq 4$. If the summands are relatively in general position and k is 3 or 4 then the sum yields a polytope of dimension 4 with 4 cube facets (cf. Figure 5.10) or a 3-dimensional cube facet depicted in Figure 5.8, respectively. A degenerate case is the 3-dimensional Minkowski sum of 4 segments depicted in Figure 5.9. □

5.4.6 The number of $N(\mathcal{R})$ facets

Now we bound the number of the facet types described above.

Resultant facets: counting octagons

We start by bounding the number of *resultant facets*, see Figure 5.8 (resultant). They contain 6 vertices, 11 edges, and 7 ridges: 6 triangular and one parallelogram.

Lemma 58. *A 4-dimensional resultant polytope can have at most 9 resultant facets in the $(3, 3, 3)$ case, and this is tight.*

Proof. The resultant facet is a 3d resultant polytope, corresponding to a subsystem with no parallel edges and support cardinalities $(3, 3, 2)$. This subsystem, comprised of two triangles and an edge, defines a Minkowski sum equal to an octagon. Consider a coarse mixed subdivision S of A_0, A_1, A_2, containing this octagon as a cell. All the other cells of S correspond to non-essential subsystems, hence their resultant is a monomial. There are 9 different subsystems with support cardinalities $3, 3, 2$, because there are 3 ways to choose the A_i contributing an edge, and 3 ways to specify this edge. This bound is tight because it is achieved in Example 10. □

Prism facets: counting heptagon-hexagon pairs

A prism facet is the Minkowski sum of a triangular ridge T and an edge E of $N(\mathcal{R})$, see Figure 5.8 (prism), where T, E are resultant polytopes of subsystems with cardinalities $(3, 2, 2)$ and $(2, 2, 2)$ resp. This type of facet has 6 vertices. The ridges are two translates of T, and 3 Minkowski sums of E with every edge of T. Each prism facet has 9 edges: 3 translates of E, and two translates of each edge of T. The subdivision of P which corresponds to a prism facet should contain a hexagon X and a heptagon H, where X corresponds to E and H to T.

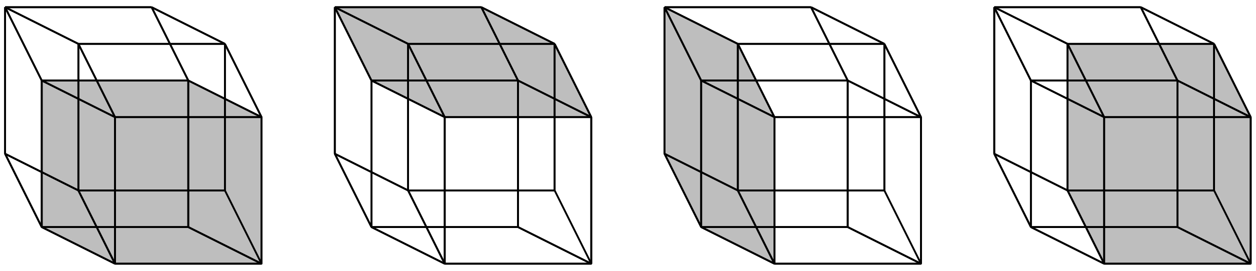


Figure 5.10: The complex of 4 zonotope (cube) facets: a Minkowski sum of 4 affinely independent segments, each associated to a hexagon in the subdivision; each subfigure highlights a cube.

Lemma 59. *There are at most* 9 *different hexagon-heptagon pairs in the set of all subdivisions of a fixed family* (A_0, A_1, A_2) *in the* $(3, 3, 3)$ *case, and this is tight.*

Proof. For fixed A_0, A_1, A_2, consider hexagon $X = s_0 + s_1 + s_2$, where $s_i \subset A_i$, and heptagon H, which is the Minkowski sum of A_0 and segments $s'_1 \neq s_1, s'_2 \neq s_2$, where $s'_i \subset A_i$, for $i \in \{1, 2\}$. By Lemma 54, X and H have common edge $s_0 \subset A_0$. By Lemma 55, if we fix $s_0 \subset A_0$, there is a unique way to construct a regular mixed subdivision with X and H. Hence, there are at most 9 possible different hexagon-heptagon pairs. This bound is tight because it is achieved in Example 10. □

Corollary 60. *A* 4*-dimensional resultant polytope can have at most* 9 *prism facets in the* $(3, 3, 3)$ *case, and this is tight.*

Proof. The maximal number of prism facets occur when the 9 different hexagon-heptagon pairs appear in different subdivisions. Degenerate cases occur when a subdivision with a heptagon and two hexagons yields a facet of $N(\mathcal{R})$. This is a degenerate prism facet (cf. Figure 5.9). These cases decrease the total number of prism facets since two different hexagon-heptagon pairs yield only one facet. This bound is tight because it is achieved in Example 10. □

Zonotope facets: counting tuples of hexagons

The zonotope facet is a Minkowski sum of $N(\mathcal{R})$ edges, each corresponding to a hexagon, see the zonotope facet types in Figures 5.8, 5.9. We study tuples of hexagons that appear in subdivisions to prove lemmas that bound the number of zonotope facets.

Since mixed points in dual subdivisions correspond to hexagon cells in subdivisions, we use counting of triplets of mixed points to count zonotope facets. Therefore, by Proposition 49 we get the following result.

Corollary 61. *A* 4*-dimensional resultant polytope can have at most 4 zonotope facets in the* $(3, 3, 3)$ *case, and this is tight.*

A note on regularity

Remark 3. *We describe here when a subdivision that contain* 4 *hexagons is regular or when is not. For a fixed family* A_0, A_1, A_2*, let* X_1, X_2, X_3, X_4 *be the hexagons and* $S_0 \geq S_1 \geq \cdots \geq S_4$ *be a chain in the refinement poset of all subdivisions where* S_0 *is the most coarse subdivision among them and every element differs from the next by a refinement in one hexagon. Since the length of this chain is* 5 *the refinement poset has a chain of length* 6*, by considering the maximal element (the coarsest subdivision). Thus one of these subdivisions should be non regular and not correspond to any face because the corresponding resultant polytope has dimension* 4 *and its face poset cannot have a chain longer than* 5*. When the hexagons correspond to* 4 *affinely independent segments then* S_0 *should be non regular because the Minkowski sum is* 4*-dimensional and the subdivision neither corresponds to the whole* 4*-dimensional polytope nor to any of its faces. In the case of* 4 *affinely dependent segments* S_0 *corresponds to a facet and* S_1 *should be non regular because the Minkowski sum is* 3*-dimensional and it neither corresponds to a facet nor to a ridge.*

We indicate now the topology of the zonotope facets correspond to refinements of this subdivision. Let $(a, b, c, d) \in \{0, 1\}^4$ *stand for the two possible flips in the 4 hexagons. There are* 16 *fine subdivisions of* S*: those which are regular correspond to resultant vertices. Let us denote, w.l.o.g., by*

$$(0bcd), (a0cd), (ab0d), (abc0) \ \subset \{0, 1\}^4,$$

the subsets of regular fine subdivisions defining zonotope facets, each with cardinality 8. The flip graph corresponds to 4 *zonotope facets, each defined by all possible flips in 3 of the hexagons. Hence, each is a neighbour of the other* 3*, as shown in Figure 5.10, with a parallelogram in common. The facet graph is a 4-clique. Over-*

all, 15 fine subdivisions are involved, hence regular, while one fine subdivision, namely (1111), is non-regular and not contained in any of the 4 facets. Of course, each zonotope (cube) has another 3 parallelograms in common with prism or resultant facets.

5.4.7 The number of $N(\mathcal{R})$ faces

We denote by $\tilde{f}_i$ the maximum number of faces of dimension i of any $(3,3,3)$ resultant polytope. It follows from Theorem 34 that it is enough to bound the maximal number of faces in the generic case with no parallel edges, considered in Section 5.4.

We will make use of a powerful result extending Barnette's Lower bound to non-simplicial polytopes:

Proposition 62. [89, thm.1.4] *For d-dimensional polytopes:*

$$f_1 + \sum_{i\geq 4}(i-3)f_2^i \geq df_0 - \binom{d+1}{2},$$

where f_2^i is the number of 2-faces which are i-gons.

The following theorem summarizes our results on the maximum numbers $\tilde{f}_i$.

Theorem 63. . *The maximal number of ridges of a $(3,3,3)$ resultant polytope is $\tilde{f}_2 = 66$ and the maximal number of facets is $\tilde{f}_3 = 22$. Moreover, $\tilde{f}_1 = \tilde{f}_0 + 44$, $22 \leq \tilde{f}_0 \leq 28$, and $66 \leq \tilde{f}_1 \leq 72$. The lower bounds are tight.*

Proof. Assume that we have a non parallel $(3,3,3)$ configuration and let us relate f_2 and f_3. Let ϕ_1, ϕ_2, ϕ_3 be the number of resultant, prism and cube facets resp.; i.e. ϕ_i is the number of facets with i summands. By Lemma 57, the total number of facets is $f_3 = \phi_1 + \phi_2 + \phi_3$. We observe that there are only triangular and parallelogram ridges, whose cardinalities are at most 36 and 30, resp.:

$$\frac{1}{2}(6\phi_1 + 2\phi_2) = 3\phi_1 + \phi_2 \leq 36,$$
$$\frac{1}{2}(\phi_1 + 3\phi_2 + 6\phi_3) = \frac{1}{2}(\phi_1 + 3\phi_2) + 3\phi_3 \leq 30.$$

The total number of ridges is then

$$f_2 = \frac{1}{2}(7\phi_1 + 5\phi_2) + 3\phi_3 \leq 66. \tag{5.5}$$

Thus, $\tilde{f}_2 \leq 66$ and our maximal instance establishes the lower bound.

With respect to the number of facets, there are at most 9 resultant, 9 prism, and 4 cubical facets by Lemmata 58, 59, and 61. Thus $\tilde{f}_3 \leq 22$ and again, our maximal instance in Example 10 establishes the lower bound.

By Euler's equality, for any resultant polytope we have $f_0+f_2 = f_1+f_3 \leq \tilde{f}_1+\tilde{f}_3$, therefore $\tilde{f}_0 + \tilde{f}_2 \leq \tilde{f}_1 + \tilde{f}_3$. By symmetry, we get $\tilde{f}_0 + \tilde{f}_2 = \tilde{f}_1 + \tilde{f}_3$. Then,

$$\tilde{f}_1 - \tilde{f}_0 = \tilde{f}_2 - \tilde{f}_3 = 44 \tag{5.6}$$

With respect to the two last inequalities in the statement, the lower bounds are given by our maximal instance and by equality (5.6), it is enough to prove $\tilde{f}_0 \leq 28$. Again, assume we are in the non parallel case. In the resultant polytope with maximal number of facets, the 2-faces are either triangles or parallelograms and there are $f_2^4 = 30$ parallelograms. Proposition 62 becomes $\tilde{f}_1 + 30 \geq 4\tilde{f}_0 - 10$. Then,

$$\tilde{f}_1 + 40 = \tilde{f}_0 + 84 \geq 4\tilde{f}_0,$$

and the desired bound follows. □

5.5 Classification

Let us summarize the characterization of 4d resultant polytopes. We need to consider 3 special instances, corresponding to 3 possible cardinalities of supports in Section 5.2.1. As mentioned before, the cases $n = 0, 1$ are similar to those in [131], so we concentrate on $(3,3,3)$. We fix $n = 2$ and $m = 9 = 3+3+3$ and consider such families. The associated *mixed Grassmannian* $G(2; 3, 3, 3; \mathbb{Q})$, defined in [34], is the linear subvariety of the Grassmanian of 5-dimensional subspaces in $\mathbb{Q}^9$ which contain the vectors $e_1 + e_2 + e_3, e_4 + e_5 + e_6$, and $e_7 + e_8 + e_9$. Given a $(3,3,3)$ family $\mathcal{A}$, its associated Cayley matrix C represents (via its rowspan) an element in $G(2; 3, 3, 3; \mathbb{Q})$. All 5×9 matrices representing an element in $G(2; 3, 3, 3; \mathbb{Q})$ are affinely equivalent to an integer Cayley matrix of an integer $(3,3,3)$ family and have some structural vanishing minors. In the case of Cayley matrices of essential configurations, not too many minors can be 0, but there could be parallel vectors and repeated points. In Sturmfels' notation [131], the Newton polytope $\mathcal{N}_{12,111}$ corresponds to two univariate configurations of multisets of 3 points, but in the first, two of the points coincide: this is a square-based pyramid. Thus, this is a degeneration of $\mathcal{N}_{111,111}$, which is the Newton polytope for two univariate configurations with 3 different points each, cf. the first polytope in Figure 5.8. Note that from the point of view of the Cayley matrix C, having a configuration with a repeated point is just an occurrence of the fact that some minors of C vanish, similarly to the existence of parallel edges in $\mathcal{A}$, which is the new feature that we have encountered in the study of 4-d resultant polytopes.

Theorem 64. *Assume we have an essential family $\mathcal{A}$ of $n+1$ (finite) lattice point configurations in $\mathbb{R}^n$ with $N(\mathcal{R})$ of dimension 4. Then, up to reordering, we are in one of the situations (i), (ii) or (iii) in Section 5.2.1. These resultant polytopes are, resp., a degeneration of the following:*

1. *$n = 0$, $|A_0| = 5$, which is a 4-simplex with f-vector $(5, 10, 10, 5)$,*
2. *$n = 1$, $|A_0| = 3, |A_1| = 4$, which is a Sylvester case, with f-vector $(10, 26, 25, 9)$,*
3. *$n = 2$, $|A_0| = |A_1| = |A_2| = 3$, which are the polytopes described in Section 5.3.*

In particular, no resultant polytope of dimension 4 can have more than 22 facets and 66 ridges.

Proof. By Theorem 31, we restrict our attention to cases 1 to 3. We discuss case 3 because cases 1 and 2 are settled, resp., in [131] and [74, ch.12], cf also the 8th instance in Table 5.1.

We can perturb (with values in $\mathbb{Q}$), e.g. a point $p \in A_0$ to a nearby rational point $p*$. We get a perturbed matrix $C'_{\mathbb{Q}} \in G(2; 3, 3, 3; \mathbb{Q})$ of the Cayley matrix C. The resultant is an affine invariant of a configuration or its Cayley matrix, so we can left multiply $C'_{\mathbb{Q}}$ by an invertible matrix M in block form with a 3×3 identity matrix in the upper left corner and an integer 2×2 integer matrix M' with non-zero determinant in the lower right corner, to get an integer matrix $C' = MC'_{\mathbb{Q}}$, which corresponds to the same point in the mixed Grassmanian. Then, C' is the Cayley matrix of an essential integer family $\mathcal{A}'$. We can say that $\mathcal{A}$ is then a degeneration of this new integer family $\mathcal{A}'$, which is the image of the family $\mathcal{A}'_{\mathbb{Q}} = (A_0 - \{p\} \cup \{p*\}, A_1, A_2)$ by M'.

Given a regular mixed subdivision S of $\mathcal{A}$ associated to a generic lifting vector w (i.e., w is generic among the vectors that produce the same regular subdivision), we consider the regular subdivision $S'_{\mathbb{Q}}$ that w induces on the perturbed configuration $\mathcal{A}_{\mathbb{Q}}$. We then translate $S'_{\mathbb{Q}}$ via multiplication by M' to a combinatorially equivalent regular subdivision S' of $\mathcal{A}'$. It follows from Theorem 34 that the number of facets of $N(\mathcal{R})$ cannot exceed the number of facets of $N(\mathcal{R}')$, and we conclude by Theorem 63. □

Example 13. Let us consider degeneracy when $n = 1$, i.e. points are repeated: $A_0 = \{0, 1\}, A_1 = \{0, 1, 1, 2\}$. We perturb A_1 and get $A_1^* = \{0, 1,\ 101/100,\ 2\}$. We dilate by 100 (multiply a row of the Cayley matrix) and get $B_0 = \{0, 100\}, B_1 = \{0, 100, 101, 200\}$, which span $\mathbb{Z}$. The resultant polytopes for $\mathcal{A}$, $\mathcal{B}$ are combinatorially equivalent, although the former resultant has total degree $2 + 1 = 3$, and the latter $200 + 100 = 300$.

5.6 Open problems and Extensions

Open problem 1. Prove that either $f_0 \leq 22$ or $f_1 \leq 66$. That is, we conjecture that the maximum f-vector of a 4d-resultant polytope is $(22, 66, 66, 22)$.

Open problem 2. Is it true that, for maximal f-vectors, it holds $f_0 = f_3$? Is it always true that $f_1 \geq f_2$, if $f_0 \geq 10$?

The proof of Theorem 34 should extend to high dimensions. Lemma 65 generalizes Lemma 58 in any dimension and is proven analogously. It motivates us to raise Conj. 1.

Lemma 65. *A d-dimensional resultant polytope has at most m resultant facets.*

Conjecture 1. The number of vertices of a d-dimensional resultant polytope is bounded above by

$$3 \cdot \sum_{||S||=d-1} \prod_{i \in S} \tilde{f}_0(i)$$

where S is any multiset with elements in $\{1, \ldots, d-1\}$, $||S|| := \sum_{i \in S} i$, and $\tilde{f}_0(i)$ is the maximum number of vertices of a i-dimensional $N(\mathcal{R})$.

The only bound in terms of d is $(3d-3)^{2d^2}$ [131], yielding $\tilde{f}_0(5) \leq 12^{50}$ whereas our conjecture yields $\tilde{f}_0(5) \leq 231$.

Chapter 6

Geometric predicates: algorithms and software

6.1 Introduction

Computing the sign of a determinant, or in other words evaluating a determinantal *predicate*, is in the core of many important geometric algorithms. Convex hull and regular triangulation algorithms use *orientation* predicates, the Delaunay triangulation algorithms also involve the *in-sphere* predicate. Furthermore, the computation of the value of a determinant, or in other words a determinantal *construction*, is also important in some geometric algorithms. For example, the exact volume computation of a convex polytope using either of triangulation or a sign decomposition method relies on the computation of the volume of simplices, which reduces to a determinant computation [30].

In general dimension d, the orientation predicate of $d+1$ points is the sign of the determinant of a matrix containing the homogeneous coordinates of the points as columns. In a similar way we can define the volume determinant formula of a simplex defined by $d+1$ points in general position as wells as the in-sphere predicate of $d+2$ points. In practice, as the dimension grows, a higher percentage of the computation time is consumed by these core procedures. In this thesis, we study effective algorithms and implementations for the computation of the determinantal predicates and constructions that appear in geometric computations.

We follow the exact computation paradigm presented in [138] and advocated by the Computation Geometry Algorithms Library (CGAL) [35], a state-of-the-art library for geometric computations. Note that in geometric algorithms the naive use of floating point arithmetic may lead to incorrect results [94]. In this thesis we study two scenarios regarding exactness. In the first, we provide exact predicates but not necessarily exact constructions while in the second we provide both exact predicates and exact constructions. We give a particular emphasis to exact division and division-free algorithms. Avoiding divisions is crucial when working on a ring that is not a field, *e.g.*, integers or polynomials.

Contribution. The observation is that, in a sequence of computations of determinants or signs of determinants that appear in geometric algorithms, a single computation can be accelerated by using the information from the previous computations in the sequence. A special case is the sequence of computations of the orientation predicates that appear in convex hull algorithms. The convex hull problem is probably the most fundamental problem in discrete and computational geometry. In fact, the problems of regular, Delaunay triangulations and Voronoi diagrams reduce to it by computing a convex hull in one dimension higher.

First, we propose algorithms with quadratic complexity for the determinants involved in incremental or gift wrapping convex hull algorithms and linear complexity for those involved in point location algorithms. Additionally, we nominate a variant of these algorithms that can perform computations over the integers. Second, we implement our proposed algorithms along with division-free determinant algorithms from the literature. We perform an experimental analysis of the current state-of-the-art packages for exact determinant computations along with our implementations. Without taking the dynamic algorithms into account, our experiments present a result of independent interest: they serve as a study of state-of-the art determinant algorithms and implementations.

Experiments also show that dynamic algorithms outperform all the other tested determinant implementations in almost all the cases. Moreover, we adapt our implementations to work with the convex hull package `triangulation` [22]. We carry out experiments with random and real data in medium dimensions (*i.e.*, ranging from 6 to 11 depending on the problem). We show that our implementation attains a speed-up up to 3.5 times and results in a convex hull package faster than the package `triangulation` in tested scenarios, and is a competitive implementation for exact volume computation. More interestingly, when used in point location problems in triangulations, it attains a speed-up of up to 78 times.

Previous work. There is a variety of algorithms and implementations for computing the determinant of a $d \times d$ matrix. Let us denote by $O(d^{\omega})$ the complexity of matrix multiplication. First, we consider the case where the matrix has values from a field. For $\omega > 2$, an algorithm for matrix multiplication imply an algorithm for determinant computation with the same ω [31]. The best current ω is 2.3727 [137]. Another category of algorithms is ones that apply *exact divisions* i.e. with no remainder. An application is the computation of the determinant of a matrix with integer entries using only integer arithmetic. A typical algorithm in this category is [10].

A different category is *division-free* algorithms that use no divisions at all, e.g. when the matrix values are from an abstract commutative ring. The best current ω in this category is 2.697263 [90]. Here, it is worth mentioning a family of determinant algorithms that use combinatorial approaches. They were introduced by Mahajan and Vinay [105], and are based on *clow* (closed ordered walk) sequences. Several similar methods with complexity $O(d^4)$ are surveyed in [123]. Based on the idea of clow sequences Bird introduced a simpler algorithm that uses matrix operations [20]. Its complexity is $O(dM(d))$, where $M(d)$ is the complexity of matrix multiplication. Urbańska conceived a method that uses fast matrix multiplication [41] to obtain a complexity $O(d^{3.03})$ [135]. When d is small, however, Bird's algorithm behaves better than other division-free algorithms, as it will be discussed later in the text; see Section 6.4.2.

Determinants of matrices over a ring arise in combinatorial problems [98], in algorithms for lattice polyhedra [12] and secondary polytopes [120] or in computational algebraic geometry problems [44]. A special case of the latter is resultant polytopes that have applications in polynomial system solving [14] and geometric modeling [61].

However, good asymptotic complexity does not imply good behaviour in practice for small and medium dimensions. For instance, LinBox [48], which implements algorithms with state-of-the-art asymptotic complexity, introduces a significant overhead in medium dimensions, and seems most suitable in very high dimensions (see Section 6.4.2 for details). Eigen [80] implements LU decomposition, of complexity $O(d^3)$, and seems to be suitable for low to medium dimensions. In addition, there exists a variety of algorithms for determinant sign computation [27, 1].

The problem of computation of sequences of determinants has also been studied. TOPCOM [120] is the reference software for enumerating all regular triangulations of a set of points in high dimensions. It efficiently pre-computes all orientation determinants that will be needed in the computation and stores their signs. In [60], a similar problem is studied in the context of computational algebraic geometry. The computation of orientation predicates is accelerated by maintaining a hash table of computed minors of the determinants. These minors appear many times in the computation. However, the above methods study sequences of determinants that appear in the computation of several triangulations (or equivalently convex hulls) and cannot be efficiently applied to the case of a single convex hull computation.

Our main tools are the Sherman-Morrison formulas [128, 11]. They relate the inverse of a matrix after a small-rank perturbation to the inverse of the original matrix. Other applications of these formulas include solving the dynamic transitive closure problem in graphs [124] and studying the effect of new links on Google Page Rank [9].

Overview of the chapter. The chapter is organized as follows. Section 6.2 introduces the dynamic determinant algorithms and the following section presents their application to the convex hull and point location problems. Section 6.4 discusses the implementation, experiments, and comparison with other software. We end up with conclusions and future work.

6.2 Dynamic Determinant Computations

In the *dynamic determinant problem*, a $d \times d$ matrix A is given. Allowing some preprocessing, we should be able to handle updates of elements of A and return the current value of the determinant. We consider here only non-singular updates, that is, updates that do not make A singular.

The Sherman-Morrison formula [128, 11] states that

$$\left(A+wv^T\right)^{-1}=A^{-1}-\frac{(A^{-1}w)(v^TA^{-1})}{1+v^TA^{-1}w}, \tag{6.1}$$

where A a $d\times d$ matrix and v,w vectors of dimension d. Let A' be the matrix resulting from replacing the i-th column of A by a vector u. Also let $(A)_i$ denotes the i-th column of A, and e_i the vector with 1 in its i-th place and 0 everywhere else. An i-th column update of A is performed by substituting $v=e_i$ and $w=u-(A)_i$ in Equation 6.1. Then, we can write A'^{-1} as follows.

$$A'^{-1}=\left(A+(u-(A)_i)e_i^T\right)^{-1}=A^{-1}-\frac{\left(A^{-1}(u-(A)_i)\right)(e_i^TA^{-1})}{1+e_i^TA^{-1}(u-(A)_i)}. \tag{6.2}$$

If A^{-1} is computed, we compute A'^{-1} using Equation 6.2. The computation is performed as follows:

$$h_1=A^{-1}(u-(A)_i) \tag{6.3}$$

$$h_2=h_1/(1+(h_1)^i) \tag{6.4}$$

$$H_3=h_2\,(A^{-1})^i \tag{6.5}$$

$$A'^{-1}=A^{-1}-H_3 \tag{6.6}$$

where $(A)^i,(h_1)^i$ denote the i-th row of A and the i-th element h_1 respectively. The intermediate results are the d-dimensional vectors h_1,h_2 and the $d\times d$ matrix H_3. Hence, the equations 6.3, 6.4, 6.5, 6.6 are computed in d^2+d, $d+O(1)$, d^2, d^2 arithmetic operations respectively and thus $3d^2+2d+O(1)$ in total.

The matrix determinant lemma [81] states that

$$\det(A+wv^T)=(1+v^TA^{-1}w)\det(A) \tag{6.7}$$

which yields the following equation

$$\det(A')=\det\left(A+(u-(A)_i)e_i^T\right)=\left(1+e_i^TA^{-1}(u-(A)_i)\right)\det(A) \tag{6.8}$$

Using Equation 6.8 we compute $\det(A')$ in $2d+O(1)$ arithmetic operations, if $\det(A)$ is known. Equations 6.2 and 6.8 lead to the following result.

Proposition 66. [128] *The dynamic determinant problem can be solved using $O(d^\omega)$ arithmetic operations for preprocessing and $O(d^2)$ for non-singular one column updates. The preprocessing consist in the computation of A^{-1} and* $\det(A)$.

Then we show how this computation can be performed over a ring. To this end, we use the adjoint of A, denoted by A^{adj}, rather than the inverse. It holds that $A^{\mathrm{adj}} = \det(A)A^{-1}$, thus we obtain the following two equations.

$$A'^{\mathrm{adj}} = \frac{1}{\det(A)}\left(A^{\mathrm{adj}}\det(A') - \left(A^{\mathrm{adj}}(u-(A)_i)\right)\,\left(e_i^T A^{\mathrm{adj}}\right)\right) \tag{6.9}$$

$$\det(A') = \det(A) + e_i^T A^{\mathrm{adj}}(u-(A)_i) \tag{6.10}$$

The only division in Equation 6.9 is known to be exact, *i.e.*, its remainder is zero. If the computation follows the order of operations as determined by the parenthesis in Equations 6.9, 6.10 then the computation can be performed in $5d^2 + d + O(1)$ arithmetic operations for Equation 6.9 and in $2d + O(1)$ for Equation 6.10. In the sequel, we will call *dyn_inv* the dynamic determinant algorithm that uses Equations 6.2 and 6.8, and *dyn_adj* the one that uses Equations 6.9 and 6.10.

6.3 Geometric Algorithms

We introduce in this section our methods for optimizing the computation of sequences of determinants that appear in geometric algorithms. First, we utilize dynamic determinants in incremental convex hull algorithms, which is one of the basic classes of convex hull algorithms. Then, we show how this solution can be extended to point location in triangulations.

6.3.1 Definitions

Let us start with some basic definitions from discrete geometry. Let $\mathcal{A} \subset \mathbb{R}^d$ be a pointset. We define the *convex hull* of a pointset $\mathcal{A}$, denoted by conv($\mathcal{A}$), as the smallest convex set containing $\mathcal{A}$. A hyperplane *supports* conv($\mathcal{A}$) if conv($\mathcal{A}$) is entirely contained in one of the two closed half-spaces determined by the hyperplane and has at least one point on the hyperplane. A *face* of conv($\mathcal{A}$) is the intersection of conv($\mathcal{A}$) with a supporting hyperplane that does not contain conv($\mathcal{A}$). Faces of dimension 0 and $d-1$ are called *vertices* and *facets* respectively. We call a face f

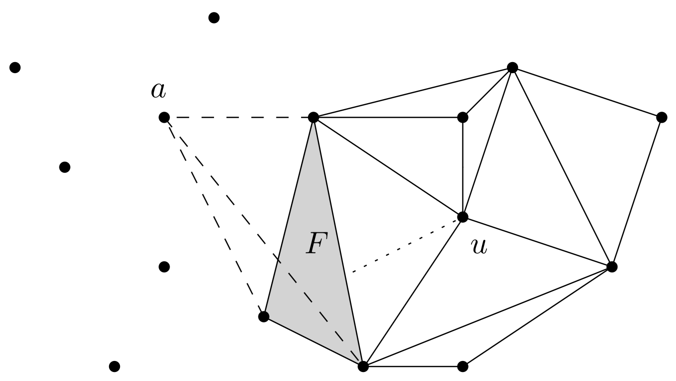


Figure 6.1: The course of an incremental convex hull algorithm in 3 dimensions.

of conv($\mathcal{A}$) *visible* from $a \in \mathbb{R}^d$ if there is a supporting hyperplane that contains f such that conv($\mathcal{A}$) is contained in one of the two closed half-spaces determined by the hyperplane and a in the other. A k-simplex of $\mathcal{A}$ is the convex hull of an affinely independent subset S of $\mathcal{A}$, where dim(conv(S)) $= k$. A *triangulation* of $\mathcal{A}$ is a collection of simplices of $\mathcal{A}$, called the *cells* of the triangulation, such that the union of the simplices equals conv($\mathcal{A}$) and every pair of simplices intersect at a common face or have an empty intersection. We define the *orientation matrix* A_C of a set C of points $\{a_1 \ldots a_{d+1}\} \subset \mathbb{R}^d$ to be the $(d+1) \times (d+1)$ matrix such that for every a_i, the column i of A_C contains $\vec{a_i}$'s coordinates as entries, where $\vec{a_i}$ is the homogeneous vector $(a_i, 1)$.

6.3.2 Incremental convex hull

For simplicity, we assume general position of $\mathcal{A}$ and present our method for the Beneath-and-Beyond (BB) algorithm [127]. However, our method can be extended to handle degenerate inputs as in [52, §8.4], as well as to be applied to more efficient incremental convex hull algorithms (e.g. [40]) by utilizing the dynamic determinant computations to answer the predicates appearing in point location (Corollary 69). A clarification of this claim is our implementation in Section 6.4 which first handles degenerate inputs in practice and second is faster compared to other software. In what follows, we use the dynamic determinant algorithm *dyn_adj*, which can be replaced by *dyn_inv* yielding a variant of the presented convex hull algorithm.

The BB algorithm is initialized by computing a d-simplex of $\mathcal{A}$. At every subse-

Algorithm 4: Incremental Convex Hull $(\mathcal{A})$

```
Input   : pointset $\mathcal{A} \subset \mathbb{R}^d$
Output: convex hull of $\mathcal{A}$
sort $\mathcal{A}$ by increasing lexicographic order of coordinates, i.e., $\mathcal{A} = \{a_1, \ldots, a_n\}$;
$T \leftarrow \{a_1, \ldots, a_{d+1}\}$;
$Q \leftarrow$ facets of conv($a_1, \ldots, a_{d+1}$);
foreach $a \in \{a_{d+2}, \ldots, a_n\}$ do
    $Q' \leftarrow Q$;
    foreach $F \in Q$ do
        $C \leftarrow$ the unique $d$-face s.t. $C \in T$ and $F \in C$;
        $u \leftarrow$ the unique vertex s.t. $u \in C$ and $u \notin F$;
        $C' \leftarrow F \cup \{a\}$;
        // det($A_C$) and $A^{\mathrm{adj}}$ were computed in a previous step
        det($A_{C'}$) $\leftarrow$ (det($A_C$) after updating $u$ with $a$ using Equations 6.9, 6.10);
        if det($A_{C'}$) det($A_C$) $< 0$ then
            $T \leftarrow T \cup \{d$-face of conv($C'$)$\}$;
            $Q' \leftarrow Q' \ominus \{(d-1)$-faces of $C'\}$; // symmetric difference
    $Q \leftarrow Q'$;
return $Q$;
```

quent step, a new point from $\mathcal{A}$ is inserted, while keeping a triangulated convex hull of the inserted points. Let t be the number of cells of this triangulation. Assume that, at some step, a new point $a \in \mathcal{A}$ is inserted and T is the triangulation of the convex hull of the points of $\mathcal{A}$ inserted up to now. To determine if a facet F is visible from a, an orientation predicate involving a and the vertices of F has to be computed (Figure 6.1). That is, we have to compute the sign of the determinant of the matrix A_C, where C is the set of vertices of F union with a. If we know the adjoint and the determinant of the orientation matrix of a cell of T that contains F, this can be done by applying Equation 6.10. If F is on the boundary, this cell is unique (*e.g.* (F, u) in Figure 6.1) otherwise we arbitrarily select one of the two cells that contain F.

Algorithm 4, as initialization, computes from scratch the adjoint matrix and the determinant of the orientation matrix A_C, where C contains the vertices of the initial d-simplex. At every incremental step, it first computes the orientation predicates using the adjoint matrices and determinants computed in previous steps utilizing Equation 6.10. Second, it computes the adjoint and determinant of the orientation matrices of the new cells using Equation 6.9. By Proposition 66, this method leads to the following result.

Lemma 67. *Given a d-dimensional pointset the first orientation predicate of incremental convex hull algorithms is computed in $O(d^\omega)$ time, and all the others in $O(d^2)$ time in total $O(d^2 t)$ space, where t is the number of cells of the constructed triangulation.*

Essentially, this result improves the computational complexity of the determinants involved in incremental convex hull algorithms from $O(d^\omega)$ to $O(d^2)$ by using more space and dynamic determinant updates. Recall that $O(d^\omega)$ is the current best complexity (Section 6.1). To analyze the complexity of Algorithm 4, we bound the number of facets of Q in every step of the outer loop of Algorithm 4 with the number of $(d-1)$-faces of the constructed triangulation of conv$(\mathcal{A})$, which is bounded by $(d+1)t$. Thus, using Lemma 67, we have the following complexity bound for Algorithm 4, where we assume that $n \gg d$ to hide the preprocessing complexity $O(d^\omega)$.

Corollary 68. *Given n d-dimensional points, the complexity of BB algorithm is $O(n \log n + d^3 nt)$, where $n \gg d$ and t is the number of cells of the constructed triangulation.*

Note that the complexity of BB, without using the method of dynamic determinants, is bounded by $O(n \log n + d^{\omega+1} nt)$. Recall that t is bounded by $O(n^{\lfloor d/2 \rfloor})$ [140, §8.4], which shows that Algorithm 4, and convex hull algorithms in general, do not have polynomial complexity in n and d. The schematic description of Algorithm 4 and its coarse analysis is good enough for our purpose: to elucidate the application of dynamic determinants to incremental convex hull computation and to quantify the improvements using this method. See Section 6.4 for a practical approach to incremental convex hull algorithms using dynamic determinant computations.

In Section 6.2 we have addressed only non-singular updates. Here we show that this will not limit our method to handle degenerate cases. In a degenerate case, the determinant of an orientation matrix will be zero if the points in the orientation test span a space of dimension less than d. However, in this case, we do not have to update the adjoint or the determinant of the orientation matrix (which would be equivalent to a singular update operation) since no new cell is going to be created.

6.3.3 Point location and other geometric algorithms

The above results can be used to improve the efficiency of geometric algorithms that use convex hull computations. For *Delaunay triangulations* in $\mathbb{R}^d$ and their dual *Voronoi diagrams* one way is to be computed as the convex hull of the points lifted on the paraboloid in $\mathbb{R}^{d+1}$. For generic liftings the above construction leads to regular triangulations.

Another important geometric problem where our method could be applied is *exact volume computation*, since one of the two major classes of volume computation algorithms is based on triangulation methods [30]. To elucidate this, observe that in Algorithm 4 we can compute the volume of the polytope by summing up the volumes of all full dimensional simplices in the resulting triangulation. Indeed, the volume of a simplex is the absolute value of the determinant of its orientation matrix. The difference of an incremental convex hull and a volume computation algorithm using a triangulation method is that the former needs to evaluate determinantal predicates while the latter needs determinantal constructions.

As mentioned above, more efficient incremental convex hull algorithms (e.g. [40]) are not sorting the input points and are using *point location* methods to find the position of the point that is going to be inserted into the convex hull. It is straightforward to apply our scheme in orientation predicates appearing in *point location* algorithms, that perform orientation tests w.r.t. the facets of the triangulation. The orientation predicates queried by a point location algorithm can be computed using Equation 6.10, if the adjoint and determinant of the orientation matrices of the cells of the triangulation have been precomputed. That yields the following result.

Corollary 69. *Given a triangulation of a d-dimensional pointset computed by an incremental convex hull algorithm like Algorithm 4, the orientation predicates involved in point location algorithms that perform orientation tests w.r.t. the facets* *of the triangulation can be computed in $O(d)$ time and $O(d^2t)$ space, where t is the number of cells of the triangulation.*

6.3.4 Data structures

In this section we present how we store and retrieve the determinants and the matrices computed in the course of the geometric algorithms that we study. We will use a hash table as a data structure.

Assume that the input points are indexed as $\{a_1, \dots, a_n\}$. We use as *hash keys* the tuples of indices of the $(d-1)$-faces of the triangulation. Each $(d-1)$-face is mapped to one of the two cells (*i.e.* d-faces) of the triangulation that it belongs to. The selection between the two cells is arbitrary and does not affect the efficiency of the method. For every cell we also store the adjoint and the determinant of the matrix that corresponds to its vertices' coordinates.

In the course of geometric algorithms a given point b should be tested for orientation with respect to a hyperplane defined by points that are locally indexed as $a_1, \dots, a_d$. Querying the hash table for the tuple $(a_1, \dots, a_d)$ we obtain the adjoint and the determinant of the matrix with entries the coordinates of $a_1, \dots, a_d$ and one more point c. Thus, the requested orientation determinant is computed by updating c with b applying Equations 6.9 and 6.10.

The following 2-dimensional example illustrates our approach.

Example 14. Let $A = \{a_1 = (0,1),\ a_2 = (1,2),\ a_3 = (2,1),\ a_4 = (1,0),\ a_5 = (2,2)\}$ where every point a_i has an index i from 1 to 5. Assume we are in some step of an incremental convex hull or point location algorithm and let $T = \{\{1,2,4\}, \{2,3,4\}\}$ be the 2-dimensional triangulation of conv(A) computed so far. The cells of T are indexed using the indices of the points in $\mathcal{A}$. For each cell, the hash table will store as keys the set of indices of the 2-faces of the cell, *e.g.* for the cells $\{\{1,2,4\}$ the keys are $\{\{1,2\},\{2,4\},\{1,4\}\}$ mapping to the adjoint and the determinant of the matrix constructed by the points a_1, a_2, a_4. Similarly, $\{\{2,3\},\{3,4\},\{2,4\}\}$ are mapped to the adjoint matrix and determinant of a_2, a_3, a_4. To insert a_5 in T one should compute the orientation determinant of a_2, a_3, a_5 to determine whether the facet $\{2,3\}$ is visible from a_5 and hence should be connected to construct a new cell $\{2,3,5\}$. Similar computations are performed for the other facets. By querying the hash table for $\{2,3\}$ the adjoint and the determinant of the matrix of a_2, a_3, a_4

are returned. Then, we perform an update of the column corresponding to point a_4, replacing it by a_5 and apply Equations 6.9 and 6.10 to compute the adjoint and the determinant of the new cell. Finally, the two new keys $\{2,5\},\{3,5\}$ are added to the hash table and are mapped to the new cell $\{2,3,5\}$.

6.4 Implementation and Experimental Analysis

We propose the *hashed dynamic determinants* scheme and implement it in C++. The scheme consists of efficient implementations of algorithms *dyn_inv* and *dyn_adj* (Section 6.2) and a hash table, which stores intermediate results (matrices and determinants) based on the method presented in Section 6.3. The design of our implementation is modular, that is, it can be used by either an algebraic software providing dynamic determinant algorithm implementations or by a geometric software providing fast geometric predicates and constructions (e.g. orientation, volume).

The geometric software builds the hashed dynamic determinants scheme on top of CGAL (experimental) package `triangulation`, presented in [22]. We will call this `hdch`. `Hdch` uses Eigen for initial determinant and adjoint or inverse matrix computation and Laplace determinant algorithm for dimensions lower than 6. Note that `triangulation` and [22] propose two implementations: one called `New DT` and a memory efficient variant called `Del graph`. Here, we use `New DT`, hence any reference to `triangulation` or [22] will refer to `New DT`. The package `triangulation` works on top of a CGAL-compliant d-dimensional kernel. We used a faster version of CGAL d-dimensional kernel, hacked by the authors of `triangulation` and provided as a part of the experimental package.

The `triangulation` package implements an incremental convex hull algorithm, like Algorithm 4 in Section 6.3. Their main difference is that `triangulation` does not sort the points along one coordinate but along a d-dimensional Hilbert curve and performs a fast point location at every insertion. Thus, we can take advantage of our scheme in two places: in the orientation predicates appearing in the point location procedure and in the ones that appear in the construction of the convex hull. Our implementation is independent of the data-structures used by `triangulation` and this is one feature of `hdch`. In practice, hash tables have constant insertion and retrieval times, and thus our approach does not introduce a significant overhead in computing time while remains modular. The only drawback is the overhead in space.

The hash table has been implemented using the Boost libraries [23]. To reduce memory consumption and speed-up look-up time, we sort the lists of indices that form the hash keys. We also use the *GNU Multiple Precision arithmetic library* (GMP), the current standard for multiple-precision arithmetic, which provides integer and rational types `mpz_t` and `mpq_t`, respectively.

The code is publicly available from

`http://hdch.sourceforge.net.`

We design and perform experiments with both algebraic and geometric software to quantify the efficiency of our method. The data used in the experiments are also available in the above web-page and thus all the experimental results can be reproduced.

6.4.1 Experimental setup

All experiments ran on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux. We divide our tests in four scenarios, according to the number type involved in computations:

a rationals where the bit-size of both numerator and denominator is 10000,

b rationals converted from `doubles`, that is, numbers of the form $m \times 2^p$, where m and p are integers of bit-size 53 and 11 respectively,

c integers with bit-size 10000, and

d integers with bit-size 32.

However, it is rare to find in practice input coefficients of scenarios (a) and (c). Inputs are usually given as 32 or 64-bit numbers. These inputs correspond to the coefficients of scenario (b). Scenario (d) is also very important, since points with integer coefficients are encountered in many combinatorial applications (Section 6.1).

6.4.2 Determinant computation experiments

We compare state-of-the-art software for exact computation of the determinant of a matrix. We consider LU decomposition in Eigen [80], LinBox determinant [48], applied to integers, and Maple 14 `LinearAlgebra[Determinant]`. LinBox implements state-of-the-art algorithms with the best known complexity bounds. However, their implementation usually has a big computational overhead and LinBox shows the best results only when working in high dimensions (the results of the tests of this section will corroborate this claim). LinBox provides a myriad of algorithms for computing determinants: many known dense and sparse elimination methods, the block Wiedemann algorithm [136] and an algorithm using a hybrid method mixing Chinese remaindering and last invariant factor [49]. We tested them and used for our tests the faster algorithm for our scenarios, the hybrid elimination algorithm (which is also the default in LinBox). Maple implementation chooses between Bareiss algorithm [10], Gaussian elimination [118, §2.2] and Berkowitz algorithm [16], based on the properties of the underlying algebraic structure. To test the behaviour of the class of division-free combinatorial algorithms, we choose to implement Bird's algorithm [20]. This choice may seem odd, because there are combinatorial algorithms with better complexity bounds. However, good complexity bounds are based here on fast matrix multiplication, which carries big constants in the complexity. In small to medium dimensions, which we are focusing in the present work, algorithms using fast matrix multiplication will show thus worse timings than naive multiplication algorithms. Bird's algorithm, on the other hand, leaves the choice of the matrix multiplication algorithm to the implementer. We choose to implement high-school matrix multiplication [118, §3.1] and, since Bird's algorithm operates with some rows of upper-triangular matrices, few multiplications are actually done (that is, the constant hidden in the complexity bound is very small). We also implemented another division-free algorithm, the Laplace expansion [118, §4.2]. Finally, we consider our implementations of *dyn_inv* and *dyn_adj*.

We do not consider in our tests the exact LU decomposition implemented in CGAL d-dimensional kernel [126] for two reasons. On one hand, Eigen is always around two times faster than CGAL. On the other hand, future versions of CGAL d-dimensional kernel will rely on Eigen for determinant computations. Let us mention, finally, that `triangulation` uses a hacked version of CGAL determinant. Since they compute with cartesian coordinates, the last row of the matrices is always full of ones. Thus, to compute the determinant of a matrix of size d, they create a new matrix of size $d-1$ by eliminating the last row and subtracting the last column of the original matrix to the $d-1$ first columns. Then, they compute the determinant of the new matrix, which is the same as the determinant of the original one. This method reduces the constant of the complexity of the determinant computation, but it is never faster than Eigen.

We test the above implementations in the four coefficient scenarios described above. When coefficients are integers, we can use integer exact division algorithms, which are faster than quotient-remainder division algorithms. In this case, Bird, Laplace and *dyn_adj* enjoy the advantage of using the number type `mpz_t` while the others are using `mpq_t`. The input matrices are constructed starting from a random $d \times d$ matrix, replacing a randomly selected column with a random d vector. We present experimental results of the four input scenarios in Tables 6.1–6.4. We tested a fifth coefficient scenario (rationals of bit-size 32), but do not show results here because timings are quite proportional to those shown in Table 6.1. We stop testing an implementation when it is slow and far from being the fastest (denoted by "–" in the Tables).

On one hand, without considering the dynamic algorithms, the experiments show the most efficient determinant algorithm implementation in the different scenarios described. This is a result of independent interest, and shows the efficiency of division-free algorithms in some settings. The simplest determinant algorithm, Laplace expansion, proved to be the best in all scenarios, until dimension 4 to 6, depending on the scenario. It has exponential complexity, thus it is slow in dimensions higher than 6 but it behaves very well in low dimensions because of the small constant of its complexity and the fact that it performs no divisions. Bird is the fastest in scenario (c), starting from dimension 7, and in scenario (d), in dimensions 7 and 8. It has also a small complexity constant, and performing no divisions makes it competitive with decomposition methods (which have better complexity) when working with integers. Eigen is the fastest implementation in scenarios (a) and (b), starting from dimension 5 and 6 respectively, as well as in scenario (d) in dimensions between 9 and 12. It should be stressed that decomposition methods are the current standard to implement determinant

d	Bird	CGAL	Eigen	Laplace	Maple	*dyn_inv*	*dyn_adj*
3	16.61	17.05	15.02	11.31	16.234	195.38	191.95
4	143.11	98.15	71.35	63.22	115.782	746.32	896.58
5	801.26	371.85	239.97	273.27	570.582	2065.08	2795.53
6	3199.79	1086.80	644.62	1060.10	1576.592	4845.38	7171.81
7	10331.30	2959.80	1448.60	7682.24	4222.563	–	–

Table 6.1: Determinant tests, inputs of scenario (a): rationals of bit-size 10000. Times in milliseconds, averaged over 1000 tests. We highlight the best non-dynamic algorithm and the dynamic algorithm if it is the fastest over all.

computation. Maple is the fastest only in scenario (d), starting from dimension 13. In our tests, LinBox is never the best, due to the fact that it focuses on higher dimensions. Finally, we report results of inexact computation for scenarios (b) and (d), that is, CGAL d-dimensional kernel using double-precision floating-point arithmetic (denoted by *inexact* in Tables 6.2 and 6.4). Though not comparable with the timings of exact computations, this approach does not compute the correct value of the determinant and serve as an experimental lower bound on the running time of all the above implementations. Furthermore, this experiments provide an insight of the timings one would obtain using double-precision filtered computations. Typically, these take at least twice the computing time of the shown values.

On the other hand, when dynamic determinant algorithm enter the competition, experiments show that *dyn_adj* defeats all the other algorithms in scenarios (b), (c), and (d). On each of these scenarios, there is a threshold dimension, starting from which *dyn_adj* is the most efficient, which happens because of its better asymptotic complexity. In scenarios (c) and (d), with integer coefficients, division-free performs much better, as expected, because integer arithmetic is faster than rational. In general, the sizes of the coefficients of the adjoint matrix are bounded. That is, the sizes of the operands of the arithmetic operations are bounded. This explains the better performance of *dyn_adj* over the *dyn_inv*, despite its worse arithmetic complexity.

d	Bird	CGAL	Eigen	Laplace	Maple	*dyn_inv*	*dyn_adj*	inexact
3	.013	.021	.014	.008	.058	.046	.023	.001
4	.046	.050	.033	.020	.105	.108	.042	.002
5	.122	.110	.072	.056	.288	.213	.067	.002
6	.268	.225	.137	.141	.597	.376	.102	.002
7	.522	.412	.243	.993	.824	.613	.148	.003
8	.930	.710	.390	–	1.176	.920	.210	.003
9	1.520	1.140	.630	–	1.732	1.330	.310	.004
10	2.380	1.740	.940	–	2.380	1.830	.430	.004
11	–	2.510	1.370	–	3.172	2.480	.570	.005
12	–	3.570	2.000	–	4.298	3.260	.760	.005
13	–	4.960	2.690	–	5.673	4.190	1.020	.006
14	–	6.870	3.660	–	7.424	5.290	1.360	.007
15	–	9.060	4.790	–	9.312	6.740	1.830	.008

Table 6.2: Determinant tests, inputs of scenario (b): rationals converted from `double`. Each timing (in milliseconds) corresponds to the average of computing 10000 (for $d < 7$) or 1000 (for $d \geq 7$) determinants. Highlighting as in Table 6.1. The last column of the table shows the time spent in inexact computations with double-precision floating point arithmetic, by performing an LU decomposition using the CGAL d-dimensional kernel.

d	Bird	CGAL	Eigen	Laplace	LinBox	Maple	*dyn_inv*	*dyn_adj*
3	.23	3.24	2.58	.16	132.64	.28	27.37	2.17
4	1.04	14.51	10.08	.61	164.80	1.36	76.76	6.59
5	3.40	45.52	28.77	2.02	367.58	4.52	176.60	14.70
6	8.91	114.05	67.85	6.16	–	423.08	325.65	27.97
7	20.05	243.54	138.80	42.97	–	–	569.74	48.49
8	40.27	476.74	257.24	–	–	–	904.21	81.44
9	73.90	815.70	440.30	–	–	–	1359.80	155.70
10	129.95	1358.50	714.40	–	–	–	1965.30	224.10
11	208.80	–	–	–	–	–	–	328.50
12	327.80	–	–	–	–	–	–	465.00
13	493.90	–	–	–	–	–	–	623.80
14	721.70	–	–	–	–	–	–	830.80
15	1025.10	–	–	–	–	–	–	1092.30
16	1422.80	–	–	–	–	–	–	1407.20
17	1938.40	–	–	–	–	–	–	1795.60

Table 6.3: Determinant tests, inputs of scenario (c): integers of bit-size 10000. Times in milliseconds, averaged over 1000 tests for $d < 9$ and 100 tests for $d \geq 9$. Highlighting as in Table 6.1.

d	Bird	CGAL	Eigen	Laplace	LinBox	Maple	*dyn_inv*	*dyn_adj*	inex.
3	.002	.021	.013	.002	.872	.045	.030	.008	.001
4	.012	.041	.028	.005	1.010	.094	.058	.015	.002
5	.032	.080	.048	.016	1.103	.214	.119	.023	.002
6	.072	.155	.092	.040	1.232	.602	.197	.033	.002
7	.138	.253	.149	.277	1.435	.716	.322	.046	.003
8	.244	.439	.247	-	1.626	.791	.486	.068	.003
9	.408	.689	.376	-	1.862	.906	.700	.085	.004
10	.646	1.031	.568	-	2.160	1.014	.982	.107	.004
11	.956	1.485	.800	-	10.127	1.113	1.291	.133	.005
12	1.379	2.091	1.139	-	13.101	1.280	1.731	.160	.005
13	1.957	2.779	1.485	-	-	1.399	2.078	.184	.006
14	2.603	3.722	1.968	-	-	1.536	2.676	.222	.007
15	3.485	4.989	2.565	-	-	1.717	3.318	.269	.008
16	4.682	6.517	3.391	-	-	1.850	4.136	.333	.010

Table 6.4: Determinant tests, inputs of scenario (d): integers of bit-size 32. Times in milliseconds, averaged over 10000 tests. Highlighting as in Table 6.1. The last column shows the timings using inexact arithmetic, as in Table 6.2.

6.4.3 Convex hull experiments

For the experimental analysis of the behaviour of dynamic determinants used in convex hull algorithms (Section 6.3), we experiment with four state-of-the-art exact convex hull packages. Two of them implement incremental convex hull algorithms: `triangulation` [22] implements [39] and `beneath-and-beyond (bb)` implements the Beneath-and-Beyond algorithm in `polymake` [72]. The package `cdd` [70] implements the double description method, and `lrs` implements the gift-wrapping algorithm using reverse search [6].

We design the input of our experiments parametrized on the number type of the coefficients and on the distribution of the points. The number type is either rational or integer. From now on, when we refer to rational and integer we mean scenario (b) and (d), respectively. We test three uniform point distributions:

i in the d-cube $[-100, 100]^d$,

ii in the origin-centered d-ball of radius 100, and

iii on the surface of that ball.

We perform an experimental comparison of the four above packages and `hdch`, with input points from distributions (i)-(iii) with either rational or integer coefficients. In the case of integer coefficients, we test hdch using mpq_t (hdch_q) or mpz_t (hdch_z). In this case hdch_z is the most efficient with input from distribution (ii) (Figure 6.2( ); distribution (i) is similar to this) while in distribution (iii) both hdch_z and hdch_q perform better than all the other packages (see Figure 6.2( )). In the rational coefficients case, hdch_q is competitive to the fastest package (Figure 6.3). Note that the rest of the packages cannot perform arithmetic computations using mpz_t because they are lacking division-free determinant algorithms. It should be noted that hdch is always faster than triangulation. The sole modification of the determinant algorithm made it faster than all other implementations in the tested scenarios.

At this point, it may arise the question about filtering: will our method be faster if using arithmetic filters to compute the signs of determinants? This question is answered in [22]: in the dimensions we are focusing, the truth is that simple filtering is not efficient, since it reverts too often to exact computations. [28] study this problem and propose a more complicated filtering scheme for determinant computations in higher dimensions; but it must be implemented in a layer lower than the triangulation algorithm, in this case, the d-dimensional kernel. CGAL does not, and does not plan in the near future to, implement such an algorithm.Preliminary tests with triangulation using filtered computations corroborated the need of better algorithms for computing signs of determinants in high dimensions.

triangulation would greatly benefit from this high-dimensional filtering techniques. On the other hand, implementing filtering is very difficult with our scheme.

We test the improvements of hashed dynamic determinants scheme on triangulation and their memory consumption. For input points from distribution (iii) with integer coefficients, when dimension ranges from 3 to 8, hdch_q is up to 1.7 times faster than triangulation and hdch_z up to 3.5 times faster (Table 6.5).

We carry out experiments using as input the vertices of resultant polytopes which have integral coefficients (Section 6.1). The results in Table 6.6 emphasize the utilization of the hashed dynamic determinants scheme when working with real data.

$\lvert\mathcal{A}\rvert$	d	hdch_q time (sec)	hdch_q memory (MB)	hdch_z time (sec)	hdch_z memory (MB)	triangulation time (sec)	triangulation memory (MB)
260	2	0.02	35.02	0.01	33.48	0.05	35.04
500	2	0.04	35.07	0.02	33.53	0.12	35.08
260	3	0.07	35.20	0.04	33.64	0.20	35.23
500	3	0.19	35.54	0.11	33.96	0.50	35.54
260	4	0.39	35.87	0.21	34.33	0.82	35.46
500	4	0.90	37.07	0.47	35.48	1.92	37.17
260	5	2.22	39.68	1.08	38.13	3.74	39.56
500	5	5.10	45.21	2.51	43.51	8.43	45.34
260	6	14.77	1531.76	8.42	1132.72	20.01	55.15
500	6	37.77	3834.19	21.49	2826.77	51.13	83.98
220	7	56.19	6007.08	32.25	4494.04	90.06	102.34
320	7	swap	swap	62.01	8175.21	164.83	185.87
120	8	86.59	8487.80	45.12	6318.14	151.81	132.70
140	8	swap	swap	72.81	8749.04	213.59	186.19

Table 6.5: Comparison of `hdch_q`, `hdch_z` and `triangulation` using points from distribution (iii) with integer coefficients; `swap` means that the machine used swap memory.

6.4.4 Volume computation experiments

The aforementioned packages compute the volume of the polytope, defined by the input points, as part of the convex hull computation (Section 6.3). It is important to remark that volume computation does not benefit from filtering, as algorithms using the Orientation predicate do. Due to this fact, our algorithm outperforms competitors in exact volume computation.

The last column of Table 6.6 shows the volume of resultant polytopes computed using `hdch`. Therefore, `hdch` also yields a competitive implementation (Figure 6.2) for the exact computation of the volume of a polytope given by its vertices. Computing only signs of determinants would not give, as output of the algorithm computing resultant polytopes, the volume. Table 6.7 shows the gain of using `hdch` over `lrs`, a state-of-the-art software to compute polytope volumes in low dimensions, when computing volumes of 6-dimensional polytopes.

$\lvert\mathcal{A}\rvert$	d	time(sec)			volume
		`hdch_q`	`hdch_z`	`triang`	
80	6	0.54	0.27	0.66	368986.7
100	6	0.69	0.33	0.87	108096.3
110	6	1.20	0.52	1.40	1456226058.5
125	6	1.28	0.61	1.66	66137.3
376	7	17.07	7.80	24.41	1713149926.2
414	7	23.02	10.91	32.54	82132445.9
500	7	29.40	13.05	41.22	2593047991.6
528	7	38.22	17.96	54.91	33727790.7

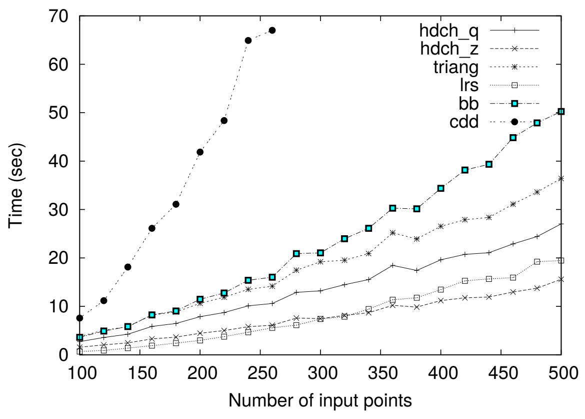
Table 6.6: Comparison of `hdch_q`, `hdch_z` and `triangulation` computing resultant polytopes.

$\lvert\mathcal{A}\rvert$	`lrs` (sec)	`hdch` (sec)
100	3.87	1.41
200	20.33	3.36
300	43.08	6.44
400	85.61	8.60
500	135.44	10.76
600	172.34	13.65
700	226.47	15.04
800	297.48	18.59
900	408.19	21.84

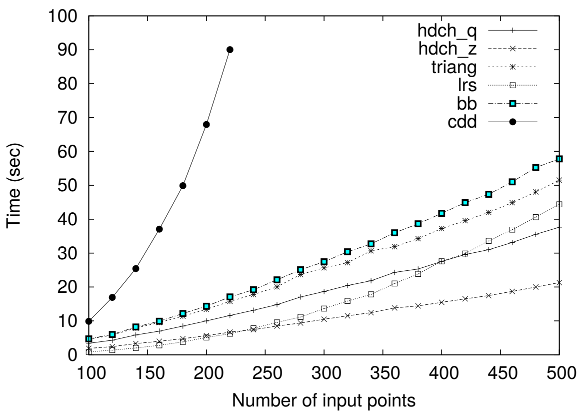
Table 6.7: Volume computation experiments; input is random points in a cube of dimension 6

6.4.5 Point location experiments

We test the efficiency of hashed dynamic determinants scheme on the point location problem in a triangulation. Given a pointset, `triangulation` constructs a triangulation of the convex hull of the pointset and a data structure that can perform point locations of new points. In addition to that, `hdch` constructs the hash table with matrices and determinants used for faster orientation computations. We perform tests with `triangulation` and `hdch` using input points uniformly distributed on the surface of a ball (distribution (iii)) as a preprocessing to build the data structures. Then, we perform point locations using points uniformly distributed inside a cube (distribution (i)). Experiments show that our method yields



(a)



(b)

Figure 6.2: Comparison of convex hull packages for 6-dimensional inputs with integer coefficients. Points are uniformly distributed (a) inside a 6-ball and (b) on its surface.

a speed-up in query time by a factor of 35 to 78 when dimension ranges from 8 to 11 using points with integer coefficients (scenario (d)) (Table 6.8).

6.4.6 Memory consumption

The main disadvantage of hdch is the amount of memory consumed, which allows us to compute up to dimension 8 (Table 6.5). One can think at this point that an intelligent memory allocation scheme could improve the performance of our algorithms. However, tests with an implementation of hdch using the Boehm-DeMers-Weiser conservative garbage collector [21] did not show improvements in computing time. This can be due to the fact that the complexity of the operations performed on the allocated numbers surpasses the complexity of the allocated space. Thus, changing the allocation scheme would not reduce significantly the computation time. This drawback can be seen as the price to pay for the obtained speed-up.

The large memory consumption of our method can be overhauled by exploiting hybrid techniques. That is, to use the dynamic determinant hashing scheme as long as there is enough memory and subsequently use the best available determinant algorithm (Section 6.4). Alternative options are to clean periodically the hash table or to use a Least Recently Used (LRU) cache to avoid storing for long

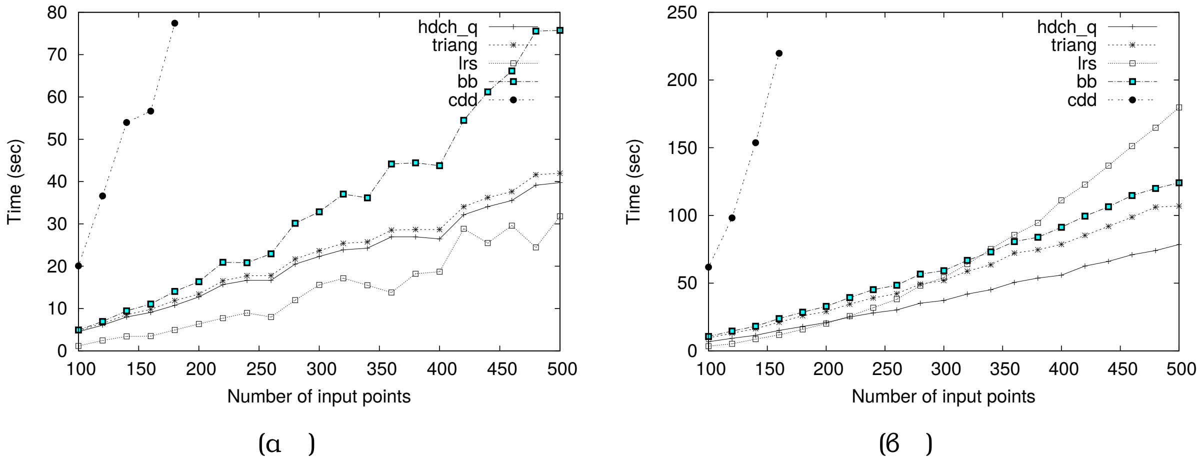


Figure 6.3: Comparison of convex hull packages for 6-dimensional inputs with rational coefficients. Points are uniformly distributed (a) inside a 6-ball and (b) on its surface.

time unused determinants and matrices. For the latter, techniques for efficiently computing determinants of matrices with more than one update, as described in [124], could be utilized.

6.5 Concluding remarks

We provide efficient determinantal predicates by utilizing the well known Sherman-Morrison formulas and describe how they can be used by algorithms that make heavy use of similar determinant computations. We also presented experimental evidences about the supremacy of these methods over state-of-the-art methods in determinant, convex hull and point location computations.

A future improvement in the memory consumption of our method could be the exploitation of hybrid memory management techniques as discussed in Section 6.4. One extension of the proposed method of this work would be the application of dynamic determinants to the *gift wrapping* (GfR) convex hull algorithms [36, 6]. Such an extension would certainly improve the memory consumption of our method.

Finally, studying the behaviour of our scheme using filtered computations, could lead to even more efficient implementations. Moreover, implementing modern algorithms for filtered computation of determinants would improve our im-

	d	$\|\mathcal{A}\|$	preproc. time (sec)	data structs. (MB)	# of cells in triangul.	query time (sec) 1K	1000K
`hdch_z`	8	120	45.20	6913	319438	0.41	392.55
`triang`	8	120	156.55	134	319438	14.42	14012.60
`hdch_z`	9	70	45.69	6826	265874	0.28	276.90
`triang`	9	70	176.62	143	265874	13.80	13520.43
`hdch_z`	10	50	43.45	6355	207190	0.27	217.45
`triang`	10	50	188.68	127	207190	14.40	14453.46
`hdch_z`	11	39	38.82	5964	148846	0.18	189.56
`triang`	11	39	181.35	122	148846	14.41	14828.67

Table 6.8: Point location time of 1K and 1000K (1K=1000) query points for `hdch_z` and `triangulation` (`triang`), using distribution (iii) for preprocessing and distribution (i) for queries and integer coefficients.

plementation, as well as its competitors.

Bibliography

[1] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, pages 197–203, 1999.

[2] P.K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry, MSRI*, pages 1–30. University Press, 2005.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51:117–122, 2008.

[4] F. Ardila, C. Benedetti, and J. Doker. Matroid polytopes and their volumes. *Discrete & Computational Geometry*, 43(4):841–854, 2010.

[5] S. Arya, G. Dias da Fonseca, and D.M. Mount. Optimal area-sensitive bounds for polytope approximation. In *ACM Symp. on Comp. Geometry*, pages 363–372, 2012.

[6] D. Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In *Polytopes - Combinatorics and Computation*, volume 29 of *Oberwolfach Seminars*, pages 177–198. Birkhäuser-Verlag, 2000.

[7] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comp. Geom.: Theory & Applic.*, 7:265–301, 1997.

[8] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Comput. Geometry*, 8:295–313, 1992.

[9] K. Avrachenkov and N. Litvak. The effect of new links on Google PageRank. *Stoch. Models*, 22:319–331, 2006.

[10] E.H. Bareiss. Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. *Mathematics of Computation*, 22:565–565, 1968.

[11] M.S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, 1951.

[12] A. Barvinok and J.E. Pommersheim. An algorithmic theory of lattice points in polyhedra. *New Perspectives in Algebraic Combinatorics*, pages 91–147, 1999.

[13] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search. *IEEE Trans. Pattern Analysis & Machine Intelligence*, 33(2):266–278, 2011.

[14] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*. Springer-Verlag, Berlin, 2003.

[15] M. Beck and D. Pixton. The Ehrhart polynomial of the Birkhoff polytope. *Discrete & Computational Geometry*, 30(4):623–637, 2003.

[16] S.J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, March 1984.

[17] D. Bertsimas and S. Vempala. Solving convex programs by random walks. *J. ACM*, 51(4):540–556, 2004.

[18] U. Betke and M. Henk. Approximating the volume of convex bodies. *Discrete & Computational Geometry*, 10(1):15–21, 1993.

[19] L.J. Billera, P. Filliman, and B. Sturmfels. Constructions and complexity of secondary polytopes. *Advances in Math.*, 83(2):155–179, 1990.

[20] R.S. Bird. A simple division-free algorithm for computing determinants. *Inf. Process. Lett.*, 111:1072–1074, November 2011.

[21] H.-J. Boehm. Space efficient conservative garbage collection. In *Programming Language Design and Implementation*, pages 197–206. ACM, June 1993.

[22] J.-D. Boissonnat, O. Devillers, and S. Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *Proc. Annual Symp. Computational Geometry*, pages 208–216, 2009.

[23] Boost: peer reviewed C++ libraries. http://www.boost.org.

[24] E. Boros, K.M. Elbassioni, V. Gurvich, and H.R. Tiwary. The negative cycles polyhedron and hardness of checking some polyhedral properties. *Annals OR*, 188(1):63–76, 2011.

[25] D. Bremner. Incremental convex hull algorithms are not output sensitive. In *Proc. 7th Intern. Symp. Algorithms and Comput.*, pages 26–35, London, UK, 1996. Springer.

[26] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. In *Proc. Annual ACM Symp. on Computational Geometry*, pages 165–174, New York, 1998. ACM.

[27] H. Brönnimann, I.Z. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theor. Comp. Science, Spec. Issue on Real Numbers & Computers*, 210(1):173–197, 1999.

[28] Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1–2):25 – 47, 2001. 14th European Workshop on Computational Geometry.

[29] B. Büeler and A. Enge. VINCI. `http://www.math.u-bordeaux1.fr/~aenge/index.php?category=software&page=vinci`.

[30] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: A practical study. In *Polytopes: Combinatorics and Computation*, volume 29 of *Oberwolfach Seminars*, pages 131–154. Birkhäuser, 2000.

[31] J.R. Bunch and J.E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[32] M. Bussieck and M. Luebbecke. The vertex set of a 0/1-polytope is strongly P-enumerable. *Comput. Geom.: Theory & Appl.*, 11:103–109, 1998.

[33] E. Canfield and B. McKay. The asymptotic volume of the birkhoff polytope. *Online Journal of Analytic Combinatorics*, 4(0), 2009.

[34] E. Cattani, M. A. Cueto, A. Dickenstein, S. Di Rocco, and B. Sturmfels. Mixed discriminants. *Math. Z.*, 2013. to appear; also in ArXiv 2011.

[35] CGAL: Computational geometry algorithms library. http://www.cgal.org.

[36] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17(1):78–86, January 1970.

[37] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[38] K.L. Clarkson. More output-sensitive geometric algorithms. In *Proc. IEEE FOCS*, pages 695–702, 1994.

[39] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom.: Theory & Appl.*, 3:185–121, 1993.

[40] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(1):387–421, 1989.

[41] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[42] B. Cousins and S. Vempala. A cubic algorithm for computing gaussian volume. In *SODA*, pages 1215–1228, 2014.

[43] B. Cousins and S. Vempala. A Matlab implementation for volume approximation of convex bodies, 2014. `http://www.cc.gatech.edu/~bcousins/Volume.html`.

[44] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Number 185 in GTM. Springer, New York, 2nd edition, 2005.

[45] J.A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, and J. Wu. Software for exact integration of polynomials over polyhedra. *Comput. Geom.: Theory Appl.*, 46(3):232–252, April 2013.

[46] O. Devillers, 2011. Personal communication.

[47] A. Dickenstein, I.Z. Emiris, and V. Fisikopoulos. Combinatorics of 4-dimensional resultant polytopes. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 173–180, New York, NY, USA, 2013. ACM.

[48] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Linbox: A generic library for exact linear algebra. In *Proc. Intern. Congress Math. Software*, pages 40–50, Beijing, 2002.

[49] Jean-Guillaume Dumas and Anna Urbańska. An introspective algorithm for the integer determinant. In Jean-Guillaume Dumas, editor, *Transgressive Computing 2006*, pages 185–202, Granada, Spain, 2006. Copias CoCa, Madrid.

[50] M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[51] M.E. Dyer and A.M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.

[52] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[53] J. Edmonds, W.R. Pulleyblank, and L. Lovász. Brick decompositions and the matching rank of graphs. *Combinatorica*, 2(3):247–274, 1982.

[54] G. Elekes. A geometric inequality and the complexity of computing volume. *Discrete & Computational Geometry*, 1:289–292, 1986.

[55] I.Z. Emiris and V. Fisikopoulos. Efficient random walk methods for approximating polytope volume. In *Proc. Symp. Comp. Geometry*. ACM, 2014.

[56] I.Z. Emiris, V. Fisikopoulos, and B. Gärtner. Efficient volume and edge-skeleton computation for polytopes defined by oracles. In *Proc. EuroCG 2013*, Braunschweig, Germany, March 2013.

[57] I.Z. Emiris, V. Fisikopoulos, and B. Gärtner. Efficient edge-skeleton computation for polytopes defined by oracles, 2014. Submitted to journal.

[58] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. A software framework for computing newton polytopes of resultants and (reduced) discriminants. In *12th Intern. Symp. Effective Methods in Algebraic Geometry, MEGA*, Frankfurt, Germany, 2013. Poster presentation.

[59] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. In *Proc. Symp. on Comp. Geom.*, pages 179–188, 2012.

[60] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An oracle-based, output-sensitive algorithm for projections of resultant polytopes. *Intern. J. Comp. Geom. Appl., Special Issue*, 23:397–423, 2013.

[61] I.Z. Emiris, T. Kalinka, C. Konaxis, and T. Luu Ba. Implicitization of curves and (hyper)surfaces using predicted support. *Theor. Comp. Science, Special Issue on Symbolic & Numeric Computing*, 479(0):81–98, 2013.

[62] I.Z. Emiris, T. Kalinka, C. Konaxis, and T. Luu Ba. Sparse implicitization by interpolation: Characterizing non-exactness and an application to computing discriminants. *J. Computer Aided Design*, 45:252–261, 2013. Special Issue on Symposium Solid & Phys. Modeling 2012 (Dijon, France).

[63] K. Fischer, B. Gärtner, T. Herrmann, M. Hoffmann, and S. Schönherr. Bounding volumes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition, 2013.

[64] K. Fischer, B. Gärtner, S. Schönherr, and F. Wessendorp. Linear and quadratic programming solver. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition, 2013.

[65] V. Fisikopoulos and L. Peñaranda. Faster geometric algorithms via dynamic determinant computation. In Leah Epstein and Paolo Ferragina, editors, *ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 443–454. Springer Berlin Heidelberg, 2012.

[66] G. Fowler, L.C. Noll, and P. Vo. FNV hash. www.isthe.com/chongo/tech/comp/fnv/, 1991.

[67] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *J. Symbolic Computation*, 38(4):1261 – 1272, 2004.

[68] K. Fukuda. cdd and cdd+ Home Page. ETH Zürich. www.ifor.math.ethz.ch/~fukuda/cdd_home/, 2008.

[69] K. Fukuda and C. Weibel. Computing all faces of the Minkowski sum of V-polytopes. In *Canad. Conf. Comp. Geom.*, pages 253–256, 2005.

[70] Komei Fukuda. cddlib, version 0.94f, 2008.

[71] Komei Fukuda and Christophe Weibel. A linear equation for minkowski sums of polytopes relatively in general position. *Eur. J. Comb.*, 31(2):565–573, February 2010.

[72] Ewgenij Gawrilow and Michael Joswig. polymake: a framework for analyzing convex polytopes. In G. Kalai and G.M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

[73] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. Newton polytopes of the classical resultant and discriminant. *Advances in Math.*, 84:237–254, 1990.

[74] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[75] P. Gritzmann and A. Hufnagel. On the algorithmic complexity of Minkowski's reconstruction problem. *J. London Math. Soc.*, 2:5–9, 1999.

[76] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner bases. *SIAM J. Discr. Math.*, 6(2):246–269, 1993.

[77] Peter Gritzmann and Bernd Sturmfels. Minkowski addition of polytopes: Computational complexity and applications to gro&uml;bner bases. *SIAM J. Discret. Math.*, 6(2):246–269, 1993.

[78] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer, Berlin, 2nd edition, 1993.

[79] G. Guennebaud, B. Jacob, et al. Eigen v3, 2010. `http://eigen.tuxfamily.org`.

[80] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010.

[81] D.A. Harville. *Matrix algebra from a statistician's perspective.* Springer-Verlag, New York, 1997.

[82] P. Huggins. ib4e: A software framework for parametrizing specialized lp problems. In A. Iglesias and N. Takayama, editors, *Mathematical Software (ICMS 2006)*, volume 4151 of *Lecture Notes in Computer Science*, pages 245–247. Springer, Berlin, 2006.

[83] H. Imai, T. Masada, F. Takeuchi, and K. Imai. Enumerating triangulations in general dimensions. *Intern. J. Comput. Geom. Appl.*, 12(6):455–480, 2002.

[84] U. Jaekel. A Monte Carlo method for high-dimensional volume estimation and application to polytopes. *Procedia Computer Science*, 4:1403–1411, 2011.

[85] D. James. Boost functional library. www.boost.org/ libs/functional/hash, 2008.

[86] A. Jensen and J. Yu. Computing tropical resultants. *Journal of Algebra*, 387(0):287–319, 2013.

[87] M. Joswig. Beneath-and-beyond revisited. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, Mathematics and Visualization. Springer, Berlin, 2003.

[88] M. Joswig, V. Kaibel, and F. Körner. On the k-systems of a simple polytope. *Israel J. Math.*, 129(1):109–117, 2002.

[89] G. Kalai. Rigidity and the lower bound theorem 1. *Inventiones Mathematicae*, 88:125–151, 1987.

[90] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2005.

[91] R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Rand. Struct. Algor.*, 11:1–50, 1997.

[92] M.M. Kapranov. Characterization of A-discriminantal hypersurfaces in terms of logarithmic Gauss map. *Math. Annalen*, 290:277–285, 1991.

[93] D.E. Kaufman and R.L. Smith. Direction choice for accelerated convergence in hit-and-run sampling. *Operations Research*, 46:84–95, 1998.

[94] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C.K. Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom.*, 40(1):61–78, 2008.

[95] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. *Discrete & Computational Geometry*, 39:174–190, 2008.

[96] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20(1):191–194, 1979. (Translated from) Doklady Akademii Nauk SSSR.

[97] L.G. Khachiyan. Rounding of polytopes in the real number model of computation. *Math. Oper. Res.*, 21(2):307–320, 1996.

[98] C. Krattenthaler. Advanced determinant calculus: A complement. *Linear Algebra Appl.*, 411:68, 2005.

[99] S. Liu, J. Zhang, and B. Zhu. Volume computation using a direct Monte Carlo method. In G. Lin, editor, *Computing and Combinatorics*, volume 4598 of *LNCS*, pages 198–209. Springer, 2007.

[100] J.A. De Loera, R. Hemmecke, S. Onn, U.G. Rothblum, and R. Weismantel. Convex integer maximization via Graver bases. *J. Pure & Applied Algebra*, 213(8):1569–1577, 2009.

[101] J.A. De Loera, J.A. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications, vol. 25.* Springer-Verlag, 2010.

[102] L. Lovász and I. Deák. Computational results of an $O(n^4)$ volume algorithm. *European J. Operational Research*, 216(1):152–161, 2012.

[103] L. Lovász and S. Vempala. Hit-and-run from a corner. *SIAM J. Comput.*, 35(4):985–1005, 2006.

[104] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an O*(n^4) volume algorithm. *J. Comp. Syst. Sci.*, 72(2):392–417, 2006.

[105] M. Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 730–738, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[106] T. Masada, H. Imai, and K. Imai. Enumeration of regular triangulations. In *Symp. on Comp. Geometry*, SoCG '96, pages 224–233, 1996.

[107] J. Maurer. Boost: C++ Libraries. Chapter 23. Boost Random. `www.boost.org/doc/libs/1_54_0/doc/html/boost$_$random.html`.

[108] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184, 1971.

[109] T. Michiels and R. Cools. Decomposing the secondary cayley polytope. *Discr. Comput. Geometry*, 23:367–380, 2000.

[110] T. Michiels and J. Verschelde. Enumerating regular mixed-cell configurations. *Discr. Comput. Geometry*, 21(4):569–579, 1999.

[111] D.M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching, 1997.

[112] M. Muja. Flann: Fast library for approximate nearest neighbors, 2011. `http://mloss.org/software/view/143/`.

[113] M. Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[114] S. Onn and U.G. Rothblum. Convex combinatorial optimization. *Discrete & Computat. Geometry*, 32(4):549–566, 2004.

[115] S. Onn and U.G. Rothblum. The use of edge-directions and linear programming to enumerate vertices. *J. Combin. Optim.*, 14:153–164, 2007.

[116] S. Onn, U.G. Rothblum, and Y. Tangir. Edge-directions of standard polyhedra with applications to network flows. *J. of Global Optimization*, 33(1):109–122, September 2005.

[117] S.Yu. Orevkov. The volume of the Newton polytope of a discriminant. *Russ. Math. Surv.*, 54(5):1033–1034, 1999.

[118] D. Poole. *Linear Algebra: A Modern Introduction.* Cengage Learning, 2006.

[119] Alexander Postnikov. Permutohedra, associahedra, and beyond. *Int. Math. Res. Not.*, 2009(6):1026–1106, 2009.

[120] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In *Proc. Intern. Congress Math. Software*, pages 330–340, 2002.

[121] E.A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. Symposium on Computational Geometry*, pages 390–399. ACM, 1999.

[122] F. Rincón. Computing tropical linear spaces. In *J. Symbolic Computation*, volume 51, pages 86–98, 2013.

[123] G. Rote. Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. In *Comp. Disc. Math.*, pages 119–135, 2001.

[124] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. IEEE FOCS*, pages 509–517, 2004.

[125] R. Schneider. *Convex bodies: the Brunn-Minkowski theory.* Cambridge: Cambridge University Press, 1993.

[126] M. Seel. dD geometry kernel. In *CGAL User and Reference Manual.* CGAL Editorial Board, 4.3 edition, 2013.

[127] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. Master's thesis, Dept. Comp. Sci., Univ. British Columbia, Vancouver, 1981.

[128] J. Sherman and W.J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[129] M. Simonovits. How to compute the volume in high dimension? *Math. Program.*, pages 337–374, 2003.

[130] R.L. Smith. Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.

[131] B. Sturmfels. On the Newton polytope of the resultant. *J. Algebraic Combin.*, 3:207–236, 1994.

[132] B. Sturmfels and J. Yu. Tropical implicitization and mixed fiber polytopes. In *Software for Algebraic Geometry*, volume 148 of *IMA Volumes in Math. & its Applic.*, pages 111–131. Springer, New York, 2008.

[133] H. Tangelder and A. Fabri. dD spatial searching. In *CGAL User and Reference Manual.* CGAL Editorial Board, 4.3 edition, 2013.

[134] Thorsten Theobald. On the frontiers of polynomial computations in tropical geometry. *Journal of Symbolic Computation*, 41(12):1360 – 1375, 2006.

[135] A. Urbańska. Faster combinatorial algorithms for determinant and Pfaffian. *Algorithmica*, 56:35–50, 2010.

[136] Gilles Villard. A study of coppersmith's block wiedemann algorithm using matrix polynomials. Technical report, LMC-IMAG, REPORT # 975 IM, 1997.

[137] V.V. Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM.

[138] C.K. Yap and Thomas Dubé. The exact computation paradigm. In D.-Z. Du and F.K. Hwang, editors, *Computing In Euclidean Geometry*, chapter 11, pages 452–492. World Scientific, Singapore, 1995.

[139] Y. Zheng and K. Yamane. Ray-shooting algorithms for robotics. *IEEE Trans. Automation Science & Engineering*, 10:862–874, 2013.

[140] G.M. Ziegler. *Lectures on Polytopes*. Springer, 1995.