Lösung von komplexen Aufgaben

Patrick Klitzke

Universität des Saarlandes, Saarbrücken, Deutschland

Davos Camp 2015

Worum geht es?

Im Grunde stelle ich Themen vor, die ich bei meiner Bachelorarbeit behandelt habe.

Ich werde hier sowohl die einfachen als auch die komplexeren Lösungen vorstellen.

Es geht um zwei Probleme die ich erklären werden und dann alles etwas mit C++ code verschönern werde.

Wir sind nicht alleine

Die Ergebnisse der Arbeit wurden zusammen mit Tobias Friedrich und Karl Bringmann erstellt.

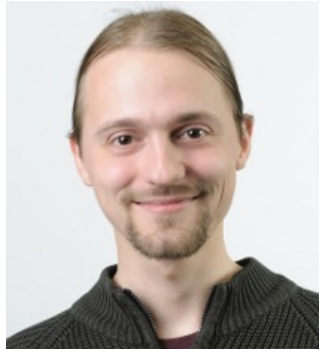


Figure : Karl Bringmann

Figure : Tobias Friedrich

Warum sollte ich zuhören?

Relevant für IOI

Der sogenannte "Convex Hull Trick" kommt ab und zu bei schweren Aufgaben vor, um die volle Punktzahl zu bekommen.

Warum sollte ich zuhören?

Relevant für IOI

Der sogenannte "Convex Hull Trick" kommt ab und zu bei schweren Aufgaben vor, um die volle Punktzahl zu bekommen.

Relevant für Uni

Ich habe damals ein Paper veröffentlicht zusammen mit Karl Bringmann und Tobias Friedrich und im Grude kannte niemand diesen Convex Hull Trick.

Warum sollte ich zuhören?

Relevant für IOI

Der sogenannte "Convex Hull Trick" kommt ab und zu bei schweren Aufgaben vor, um die volle Punktzahl zu bekommen.

Relevant für Uni

Ich habe damals ein Paper veröffentlicht zusammen mit Karl Bringmann und Tobias Friedrich und im Grude kannte niemand diesen Convex Hull Trick.

Es macht Spaß

Ich habe damals ein Paper veröffentlicht zusammen mit Karl Bringmann und Tobias Friedrich und im Grude kannte niemand diesen Convex Hull Trick. Zu guter letzt macht es immer Spaß und Freude sich mit Algorithmen zu beschäftigen. Es ist so ein bisschen so wie Knobeln.

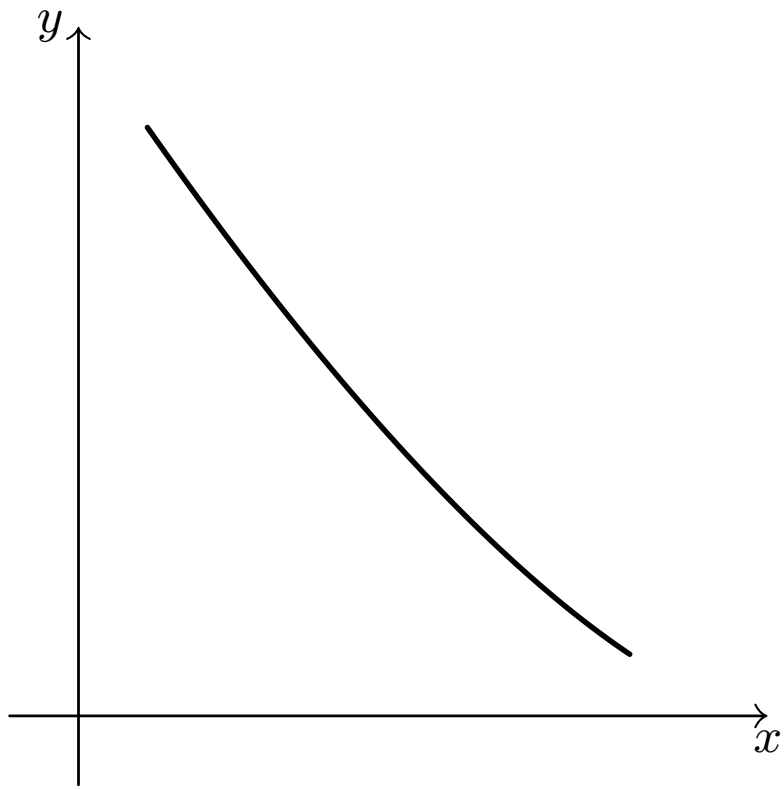
Bitte fragen Stellen unterbrechen?

Wir sind in einer kleinen Gruppe und ich möchte, dass möglichst alles verstehen und verfolgen.

Deshalb bitte sagen, falls ihr etwas nicht versteht.

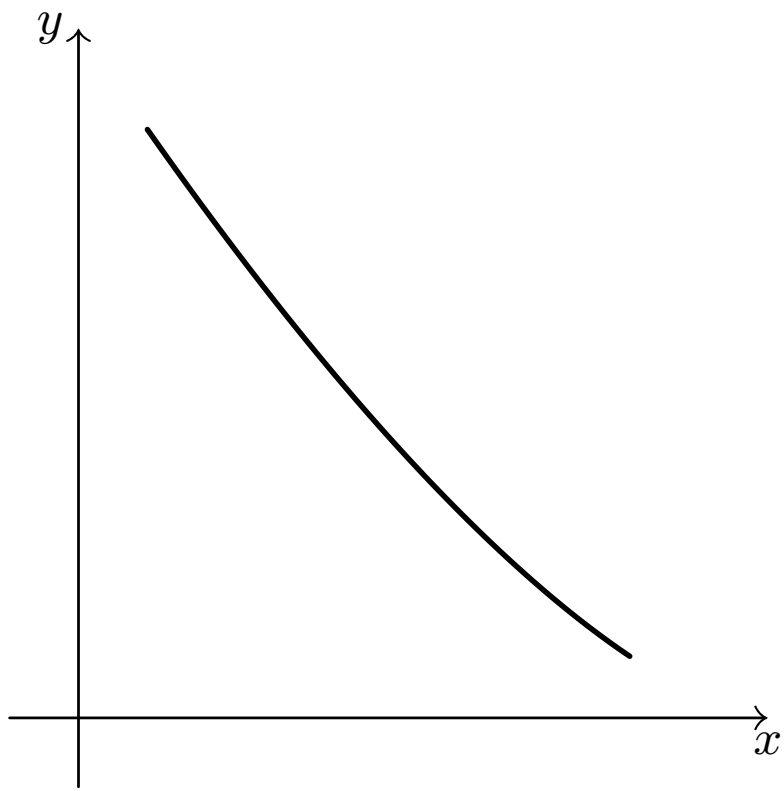
Problemstellung

Stellt euch vor ihr wollt ein Windturbine bauen. Diese soll möglichst wenig kosten, aber gleichzeitig auch möglichst effizient sein. Man kann nicht beides gleichzeitig optimieren. Man kann dies graphisch darstellen als eine funktion, wo die Effizienz auf der einen und die Kosten auf der anderen Achse dargestellt werden.



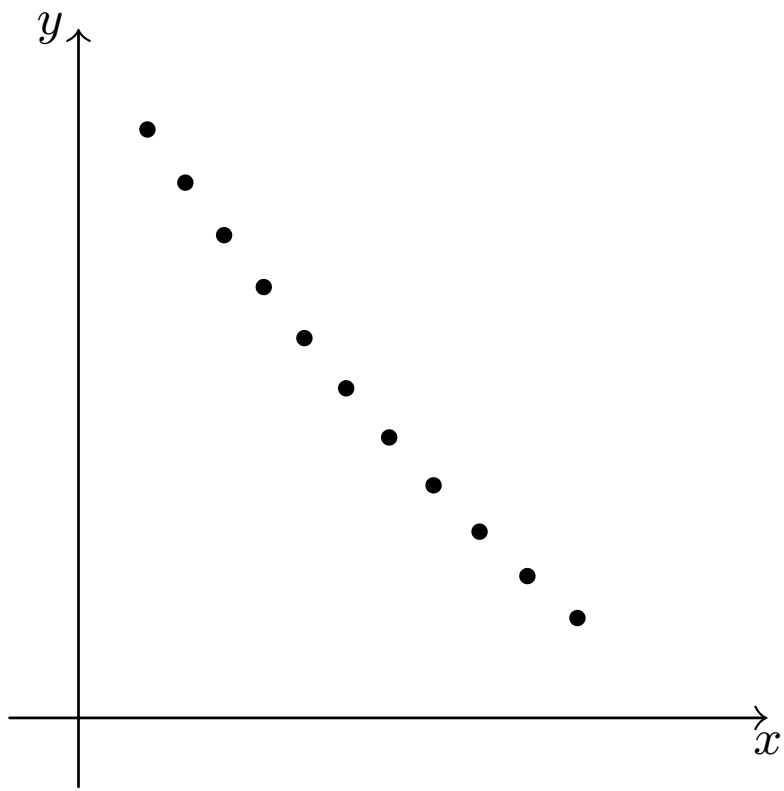
Problemstellung

Oft kennen wir diese Funktion nicht bzw ist so komplex, dass wir sie nur an einigen Stellen auswerten können, weil die Kosten sehr hoch sind. Wir bekommen dann viellecihт so 100 000 Punkte heraus. Am Ende muss aber ein Mensch entscheiden, welche Punkte wir nehmen. Deshalb wollen wir weniger Punkte (z.B 10), nur welche?



Problemstellung

Oft kennen wir diese Funktion nicht bzw ist so komplex, dass wir sie nur an einigen Stellen auswerten können, weil die Kosten sehr hoch sind. Wir bekommen dann viellecihт so 100 000 Punkte heraus. Am Ende muss aber ein Mensch entscheiden, welche Punkte wir nehmen. Deshalb wollen wir weniger Punkte (z.B 10), nur welche?



Wie sind die Resultate

Hypervolumenindikatorauswahlproblem (HYSSP)

HYPSSP kann in zwei Dimensionen wie folgt geloest werden

- $\mathcal{O}(n^3)$ [Bader, 2009]
- $\mathcal{O}(kn^2)$ [Auger, Bader, Brockhoff, and Zitzler]
- $\mathcal{O}(n \cdot (k + \log n))$ (our contribution)

Wie sind die Resultate

Hypervolumenindikatorauswahlproblem (HYSSP)

HYPSSP kann in zwei Dimensionen wie folgt geloest werden

- $\mathcal{O}(n^3)$ [Bader, 2009]
- $\mathcal{O}(kn^2)$ [Auger, Bader, Brockhoff, and Zitzler]
- $\mathcal{O}(n \cdot (k + \log n))$ (our contribution)

Epsilonindikatorauswahlproblem (EPSSSP)

EPSSSP kann in zwei Dimensionen wie folgt geloest werden:

- $\mathcal{O}(n^2 \log n)$ [Ponte, Paquete, and Figueira]
- $\mathcal{O}(n^2)$ [Vaz, Paquete, and Ponte]
- $\mathcal{O}(n \cdot \log n)$ (our contribution)

Wie sind die Resultate

Hypervolumenindikatorauswahlproblem (HYSSP)

HYPSSP kann in zwei Dimensionen wie folgt geloest werden

- $\mathcal{O}(n^3)$ [Bader, 2009]
- $\mathcal{O}(kn^2)$ [Auger, Bader, Brockhoff, and Zitzler]
- $\mathcal{O}(n \cdot (k + \log n))$ (our contribution)

Epsilonindikatorauswahlproblem (EPSSSP)

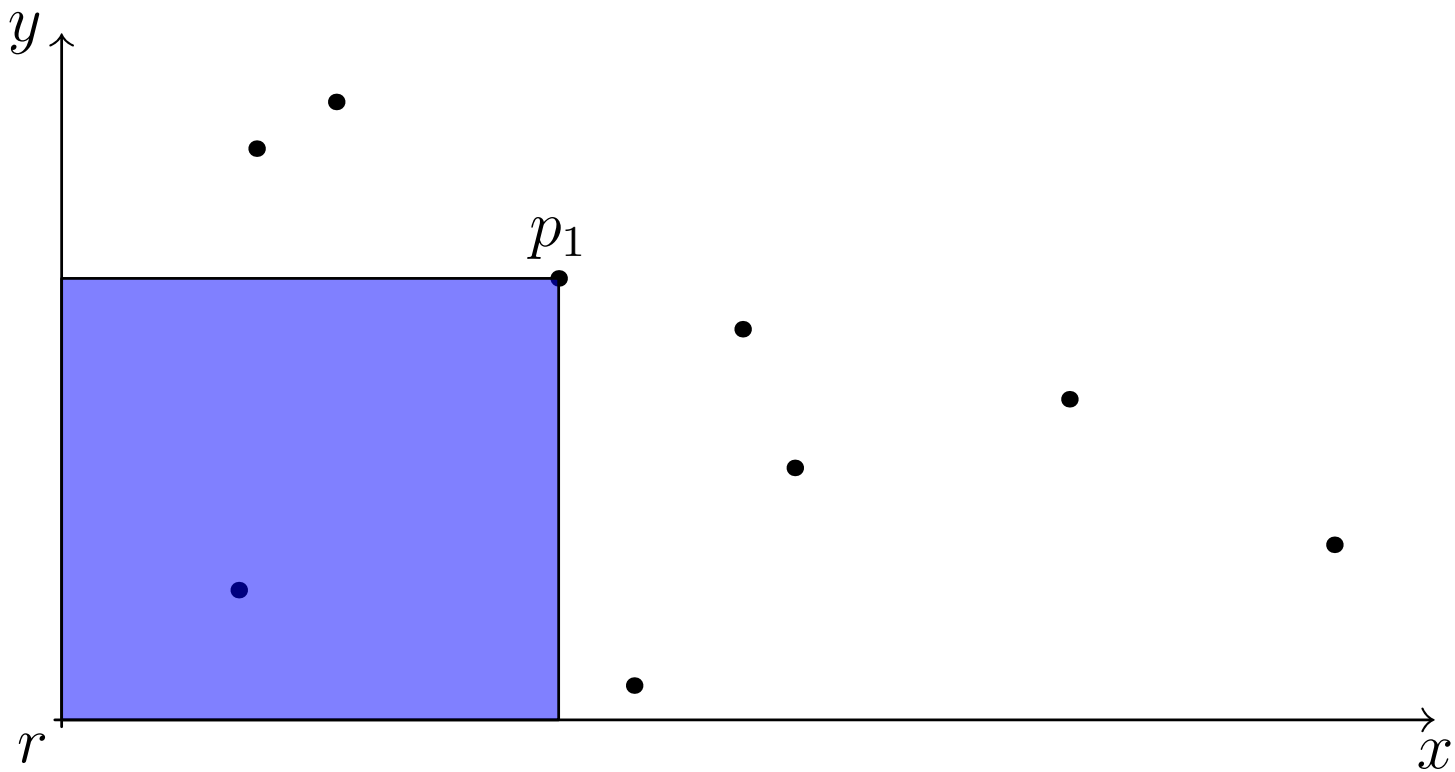
EPSSSP kann in zwei Dimensionen wie folgt geloest werden:

- $\mathcal{O}(n^2 \log n)$ [Ponte, Paquete, and Figueira]
- $\mathcal{O}(n^2)$ [Vaz, Paquete, and Ponte]
- $\mathcal{O}(n \cdot \log n)$ (our contribution)

Diese Resultate machen es möglich, größere Instanzen zu lösen.

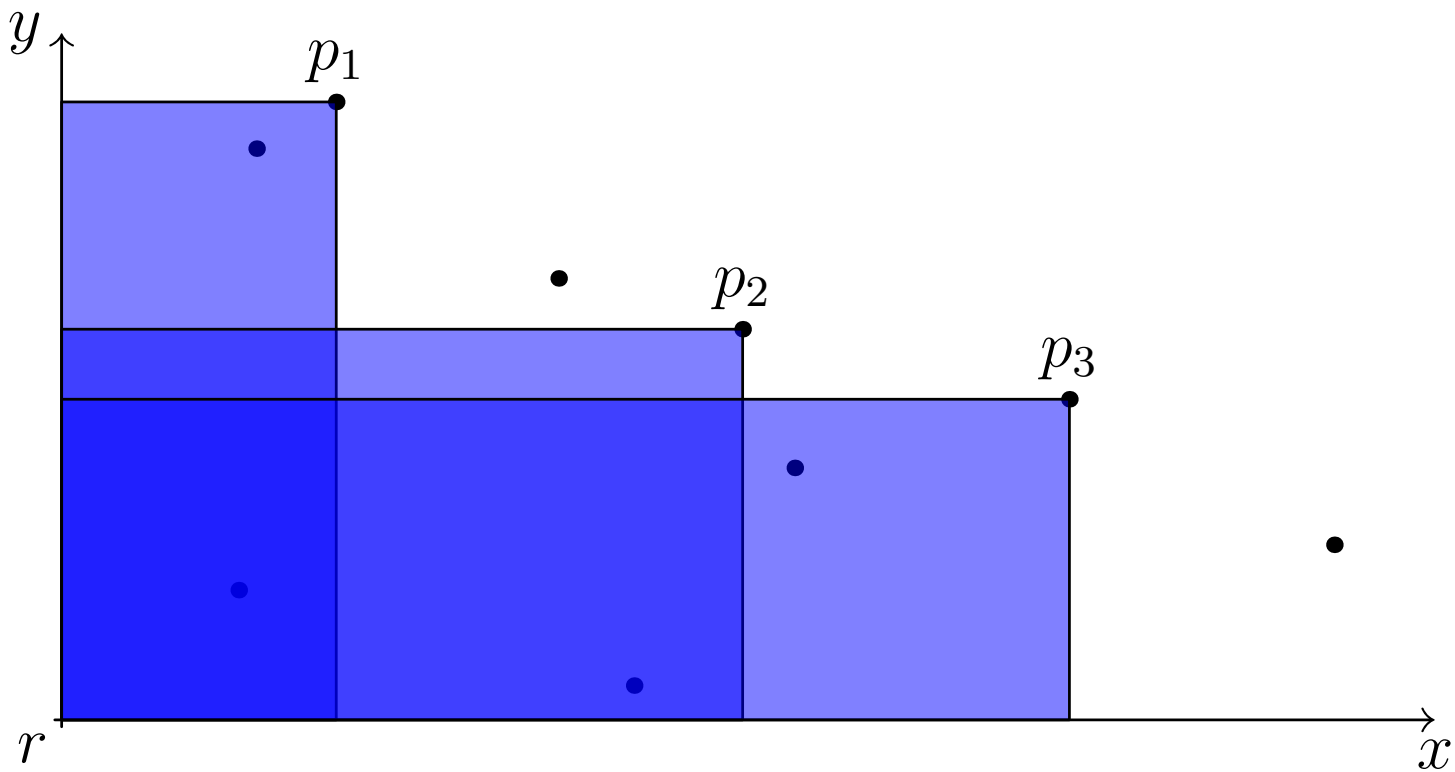
Hypervolumenindikator

Gegeben sei ein Punkt p_1 in zwei Dimensionen. Das Hypervolumenindikator von diesem Punkt ist definiert als die Fläche des Rechtecks der diesen Punkt p_1 und einen Referenzpunkt r (z.B. $r = (0, 0)$). Für mehrere Punkte ist es die Vereinigung der Rechtecke (überlagerte Flächen werden nicht mitgezählt).



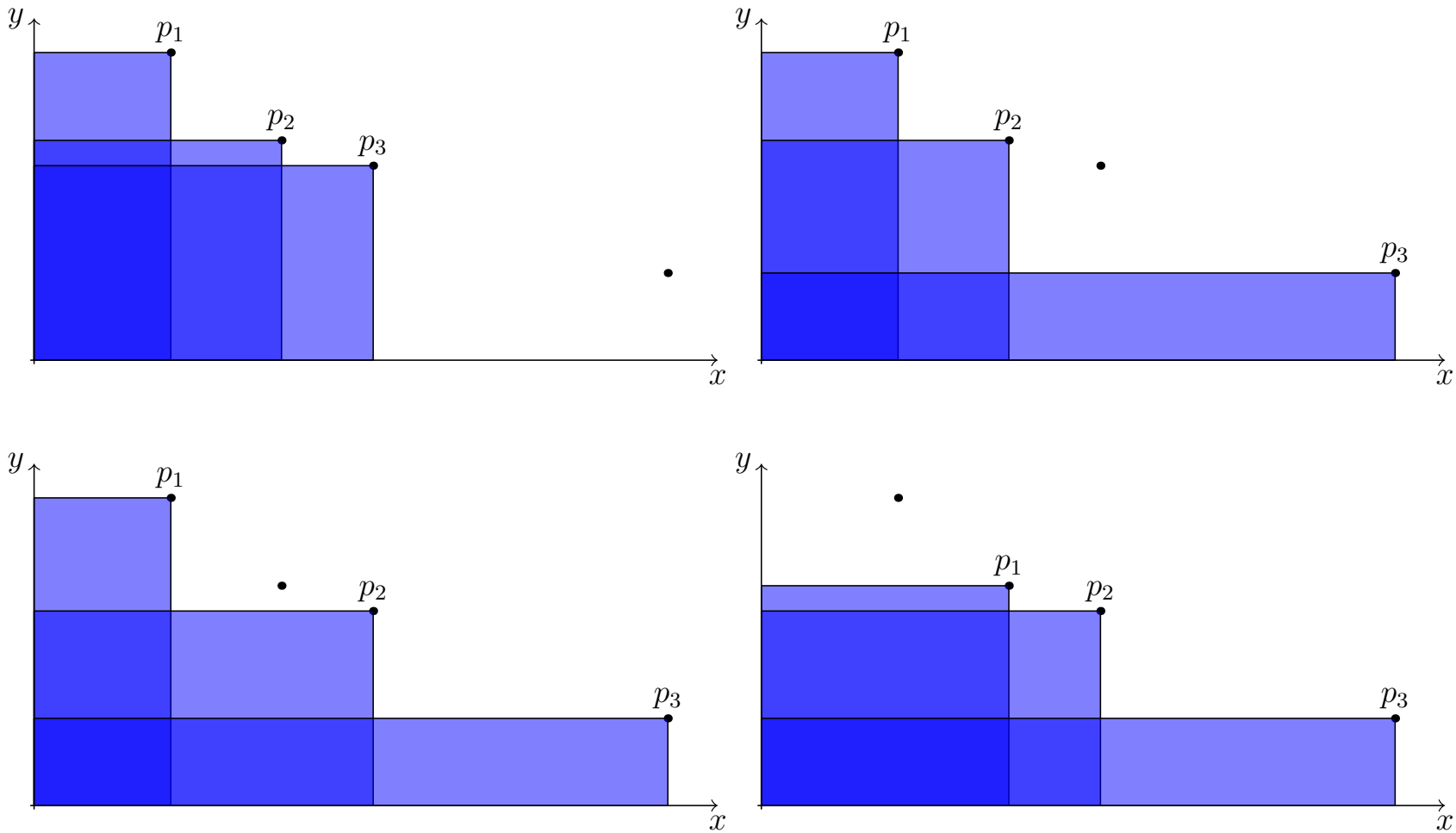
Hypervolumenindikator

Gegeben sei ein Punkt p_1 in zwei Dimensionen. Das Hypervolumenindikator von diesem Punkt ist definiert als die Fläche des Rechtecks der diesen Punkt p_1 und einen Referenzpunkt r (z.B. $r = (0, 0)$). Für mehrere Punkte ist es die Vereinigung der Rechtecke (überlagerte Flächen werden nicht mitgezählt).



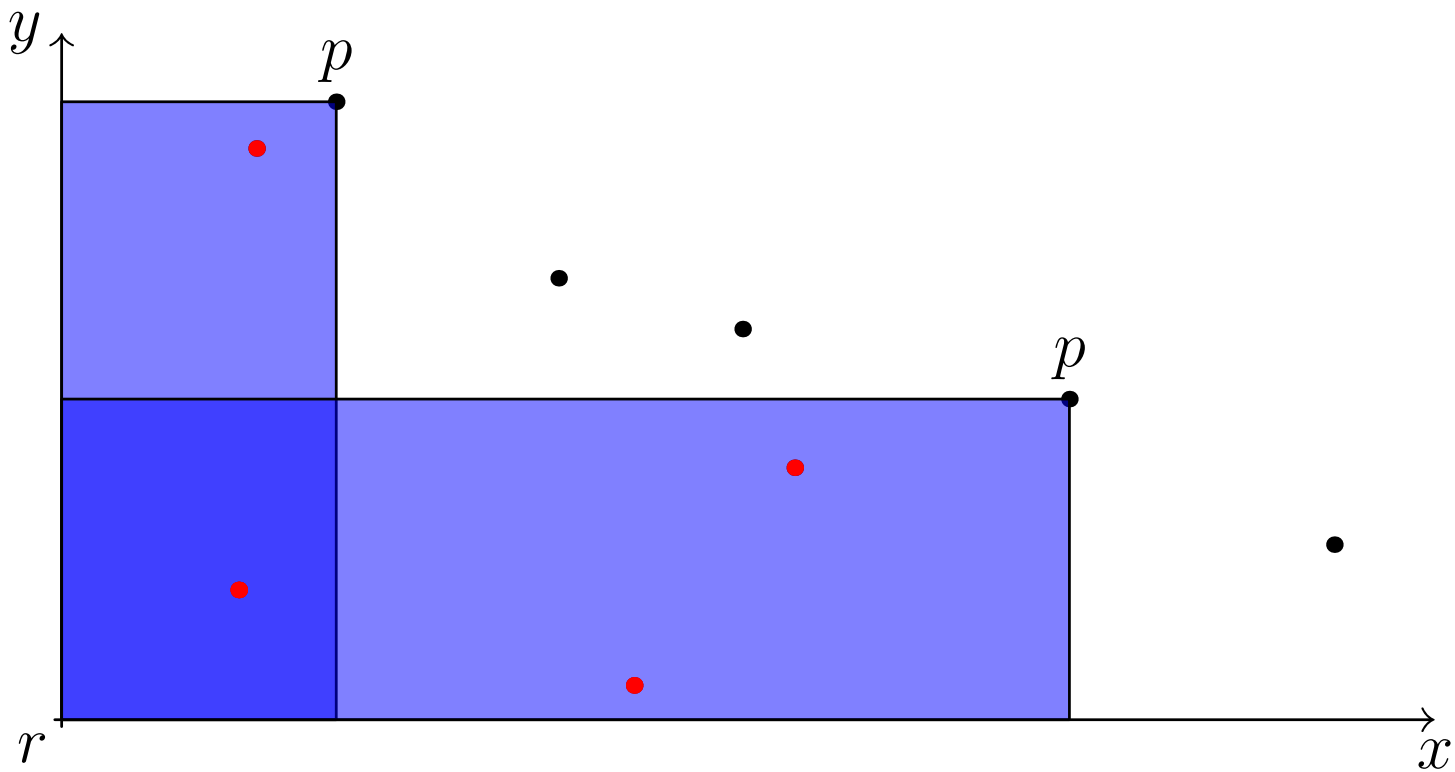
Hypervolumenindikator-Teilmengen-Auswahlproblem

Bei diesem Problem geht es darum, von k Punkten N auszuwählen, sodass sich der Hypervolumenindikator maximiert.



Vereinfachungen

Punkte sind nach steigender x Koordinate und fallender y Koordinate. Wenn ein Punkt A sowohl in der x als auch y Koordinate größer ist, dann macht es immer Sinn diesen Punkt zu nehmen.



1. Lösung: Bruteforce

Algorithmus

Wir gehen von links nach rechts durch und raten jeweils ob wir einen Punkt nehmen oder nicht. Die Laufzeit ist $\binom{n}{k} \approx n^k$

Code auf Tafel

Dynamische Programmierung

Oft kann man sich bei Bruteforce überlegen, Zwischenzustände zu speichern und so aufrufe zu sparen. Am besten ist es hier, sich Gedanken zu machen, was man genau speichern möchte.

Der Übergang zur Dynamischen Programmierung ist dann nicht allzu schwer.

dp(pos, 0) = 0
dp(pos, k) = $max_{pos<pos'}$ calc(pos, pos') + dp(pos', k-1)

Laufzeit: $O(N \cdot K \cdot N) = O(N^2 \cdot K)$ Das Einfachste ist es dies mit einer lookup table zu programmieren

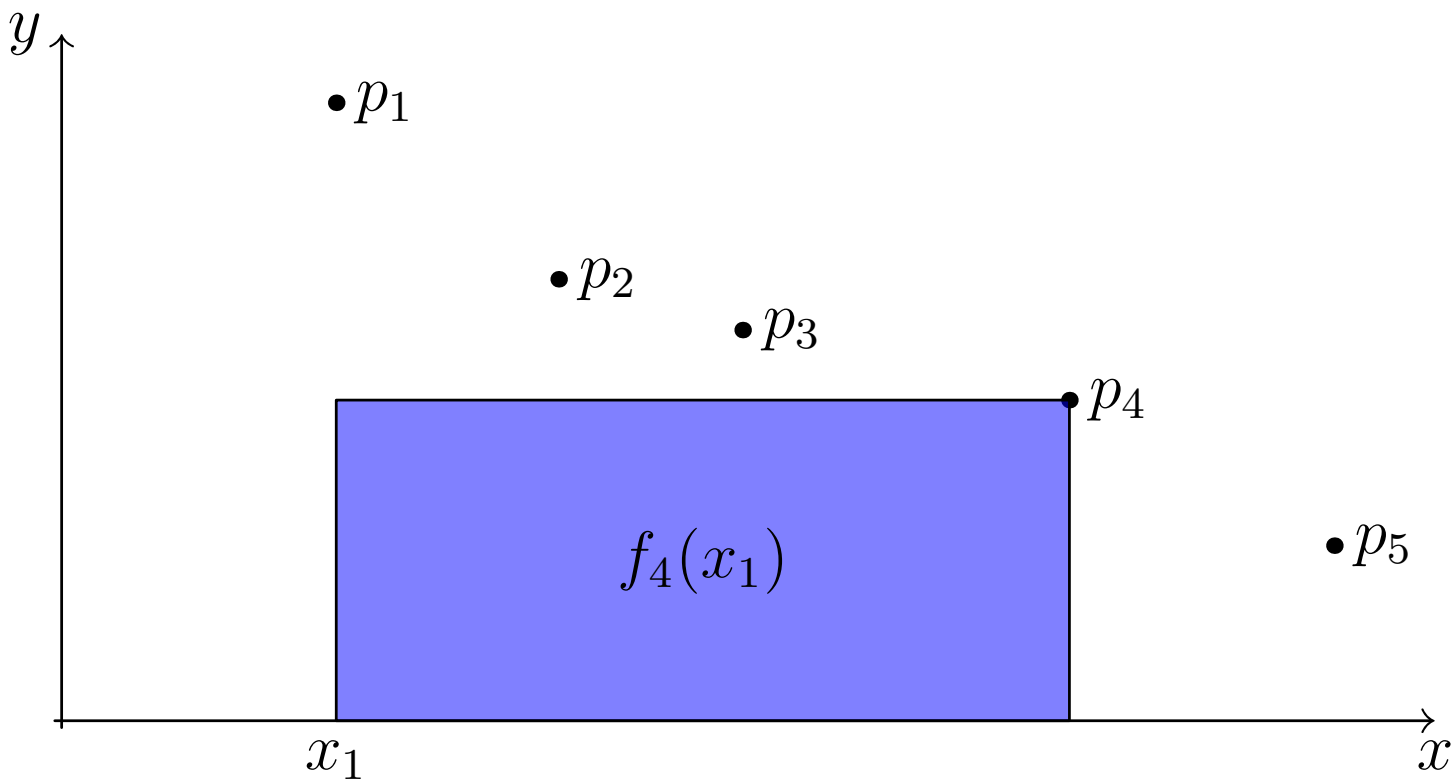
Das war der "einfache" Teil

Wenn ihr bis hierhin alles verstanden habt, dann koennt ihr bei den meisten Augaben eine gute Punktzahl herausholen. Dies sind Sachen worauf jeder kommt, das folgende wird schon etwas komplizierter.

Denkansatz Funktion

Für jeden Punkt (x_i, y_i) in der Eingabe erstellen wir eine Funktion die für eine Position x die Fläche zwischen (x_i, y_i) und $(x, 0)$ beschreibt.

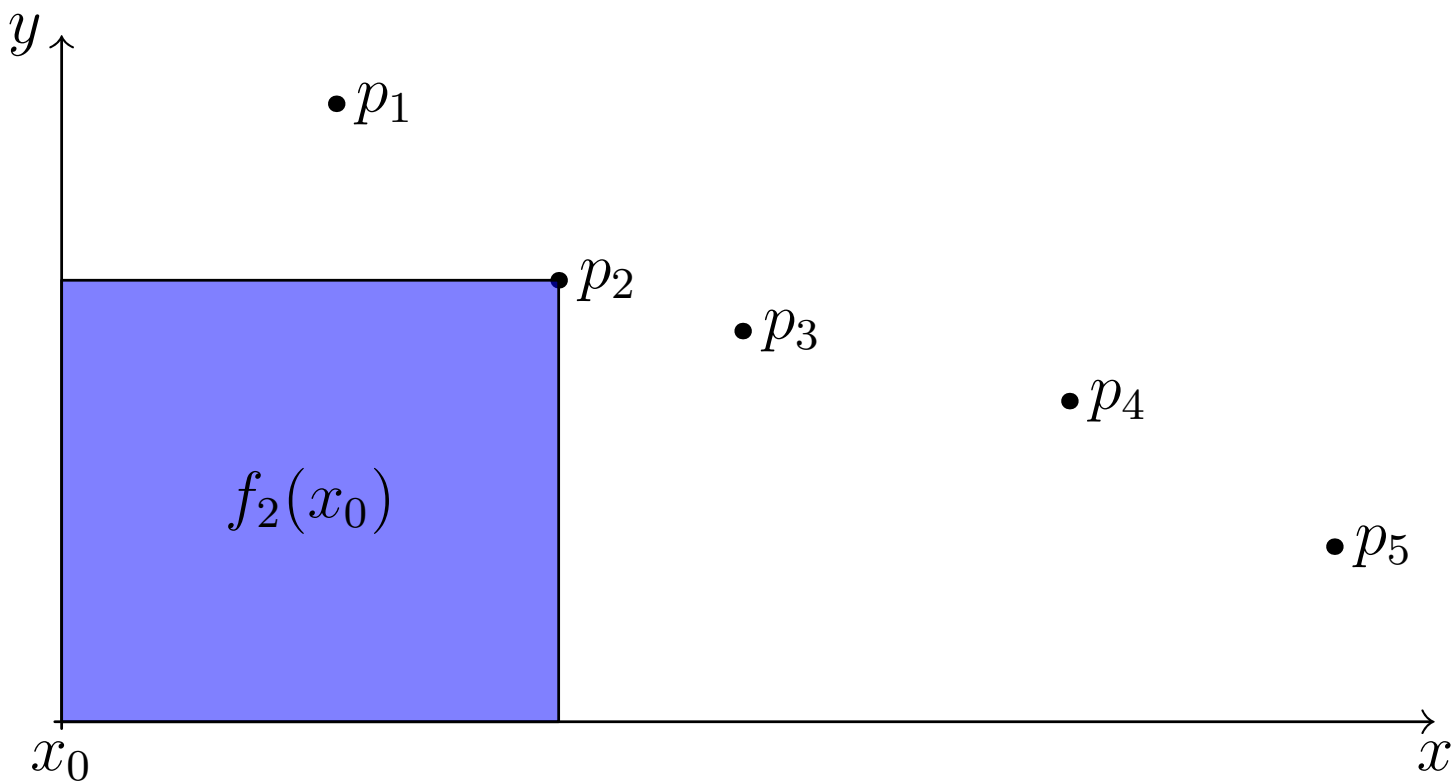
$$f_i(x) = y_i(x_i - x)$$



Denkansatz Funktion

Für jeden Punkt (x_i, y_i) in der Eingabe erstellen wir eine Funktion die für eine Position x die Fläche zwischen (x_i, y_i) und $(x, 0)$ beschreibt.

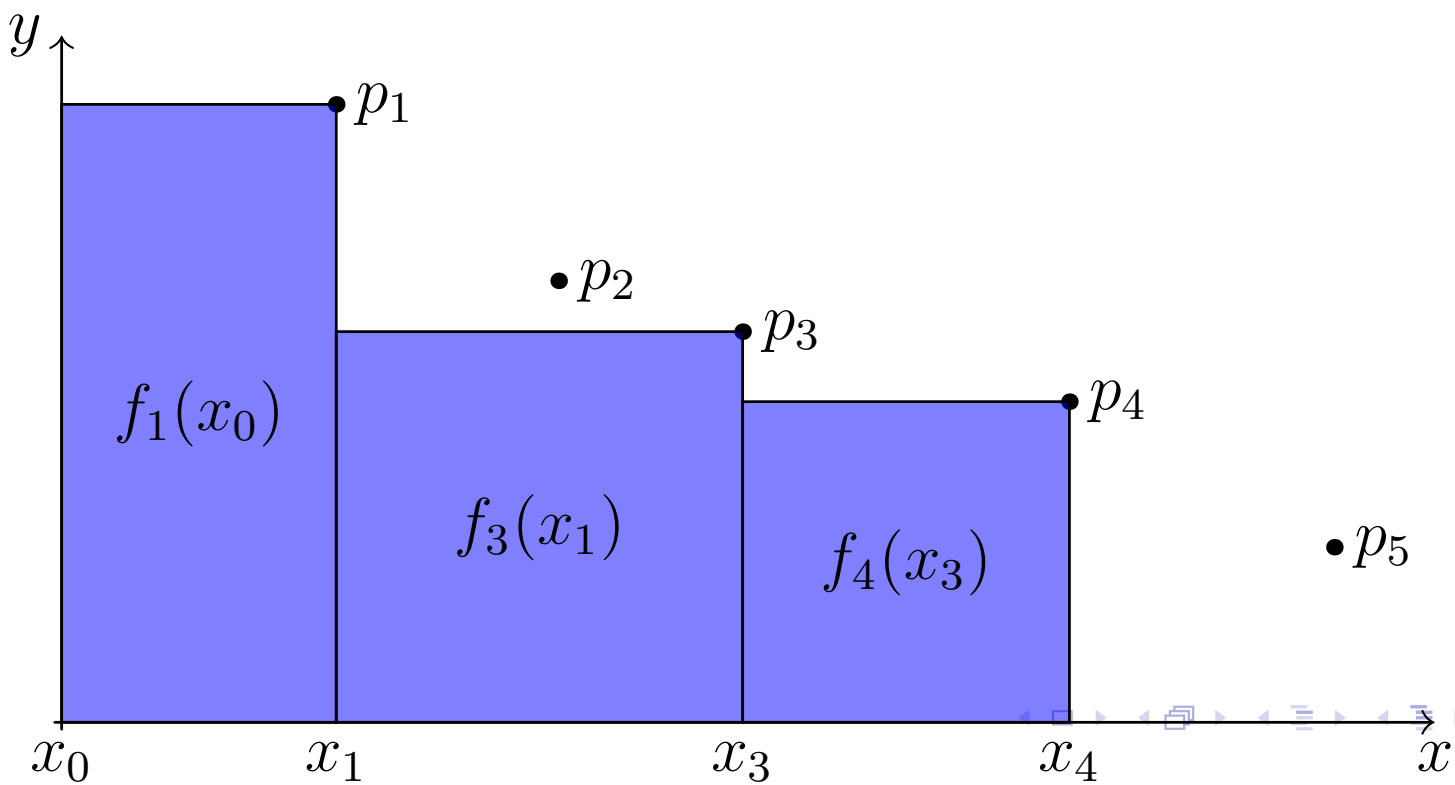
$$f_i(x) = y_i(x_i - x)$$



Denkansatz Funktion

Fuer jeden Punkt (x_i, y_i) in der Eingabe erstellen wir eine Funktion die fuer eine Position x die Flaeche zwischen (x_i, y_i) und $(x, 0)$ beschreibt.

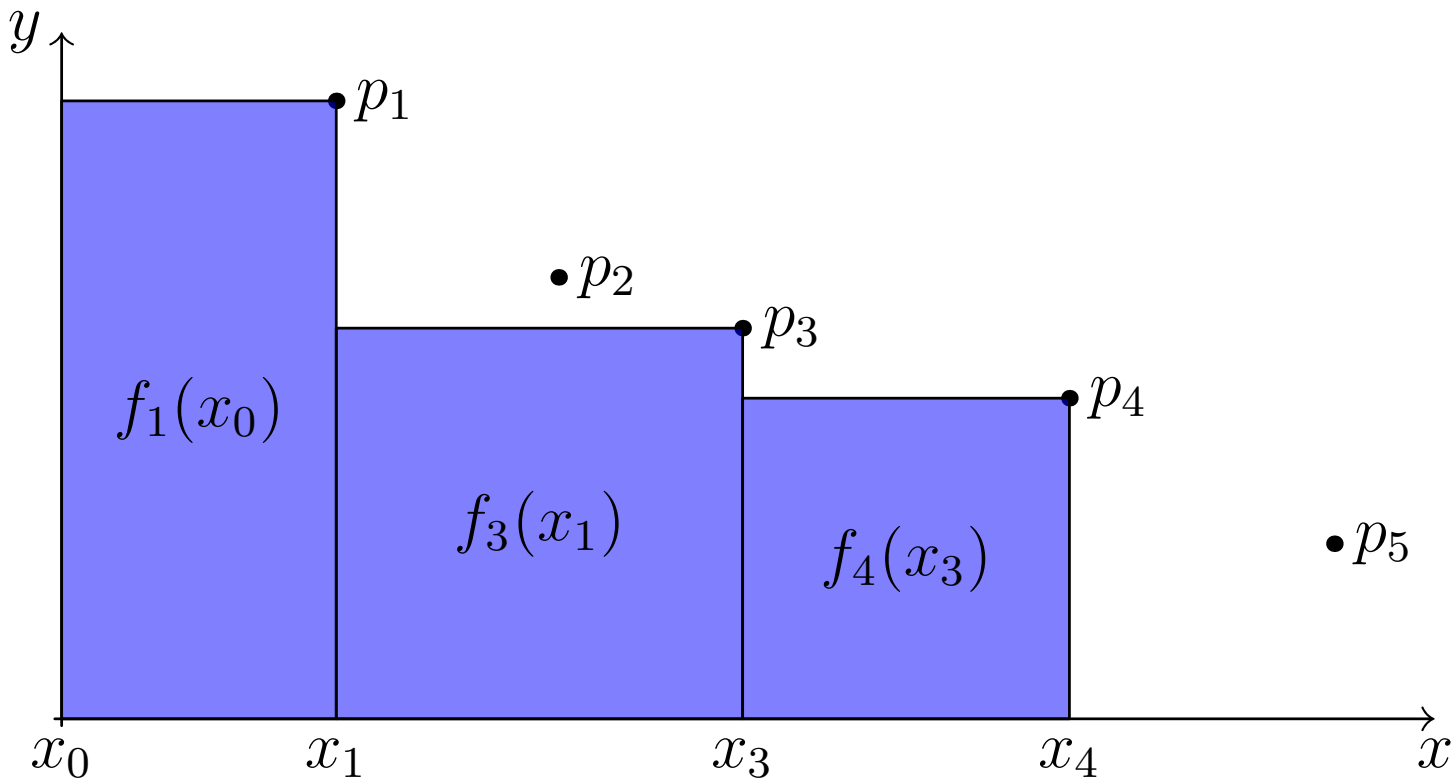
```
// simple approach
int f(Point p, int x) {
  return return p.y * (p.x-x);
}
```



Funktionsdefinition

Unser Ziel ist es folgendes zu berechnen:

$$\max_{i_1<..<i_k} f_{i_1}(x_0) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}});$$



Funktionsdefinition

Unser Ziel ist es folgendes zu berechnen:

$$\max_{i_1<..<i_k} f_{i_1}(x_0) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}});$$

Das ist aber jetzt garnicht schneller!

Umformungen

Wir wollen folgendes Berechnen:

$$\max_{i_1<..<i_k} f_{i_1}(x_0) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}});$$

Jetzt kann man umformen:

$$sol(k,\ell) = \max_{\ell<i_1<..<i_k} f_{i_1}(x_\ell) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}})$$

Umformungen

Wir wollen folgendes Berechnen:

$$\max_{i_1<..<i_k} f_{i_1}(x_0) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}});$$

Jetzt kann man umformen:

$$sol(k,\ell) = \max_{\ell<i_1<..<i_k} f_{i_1}(x_\ell) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}})$$

$$sol(k,\ell) = \max_{\ell<i_1} f_{i_1}(x_\ell) + sol(k-1, i_1)$$

Umformungen

Wir wollen folgendes Berechnen:

$$\max_{i_1<..<i_k} f_{i_1}(x_0) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}});$$

Jetzt kann man umformen:

$$sol(k,\ell) = \max_{\ell<i_1<..<i_k} f_{i_1}(x_\ell) + \sum_{j=2}^{k} f_{i_j}(x_{i_{j-1}})$$

$$sol(k,\ell) = \max_{\ell<i_1} f_{i_1}(x_\ell) + sol(k-1,i_1)$$

$$sol(k,\ell) = \max_{\ell<i_1} F_{i_1}^{k}(x_\ell)$$

Eigenschaften dieser Funktionen

Rekursive Formulierungen

Wir wollen folgendes effizient Berechnen:

$$sol(k, \ell) = \max_{\ell < i_1} F_{i_1}^k(x_\ell) = \max_{\ell < i_1} f_{i_1}(x_\ell) + sol(k-1, i_1)$$

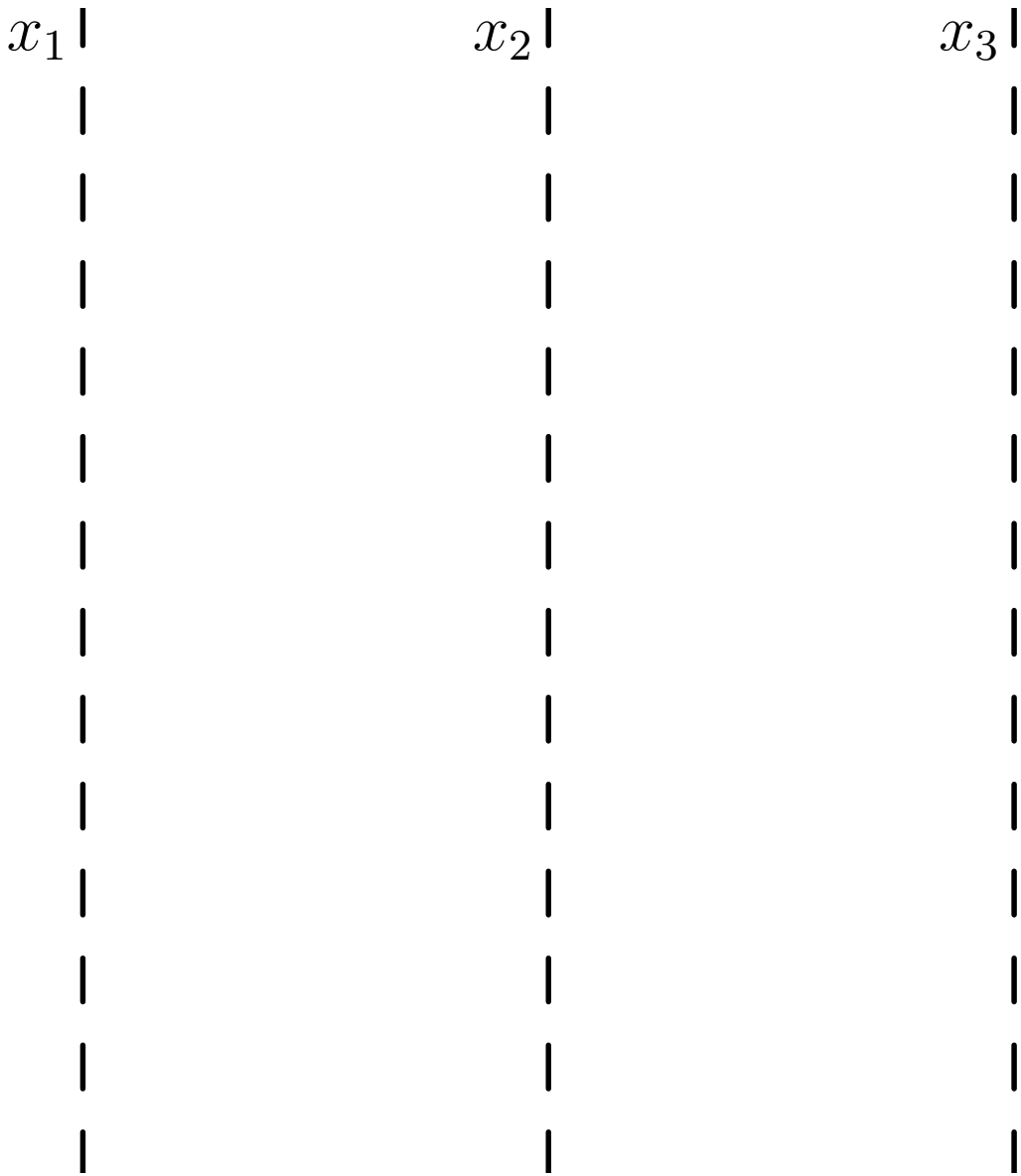
where $f_i(x) = y_i(x_i - x)$

Eigenschaften

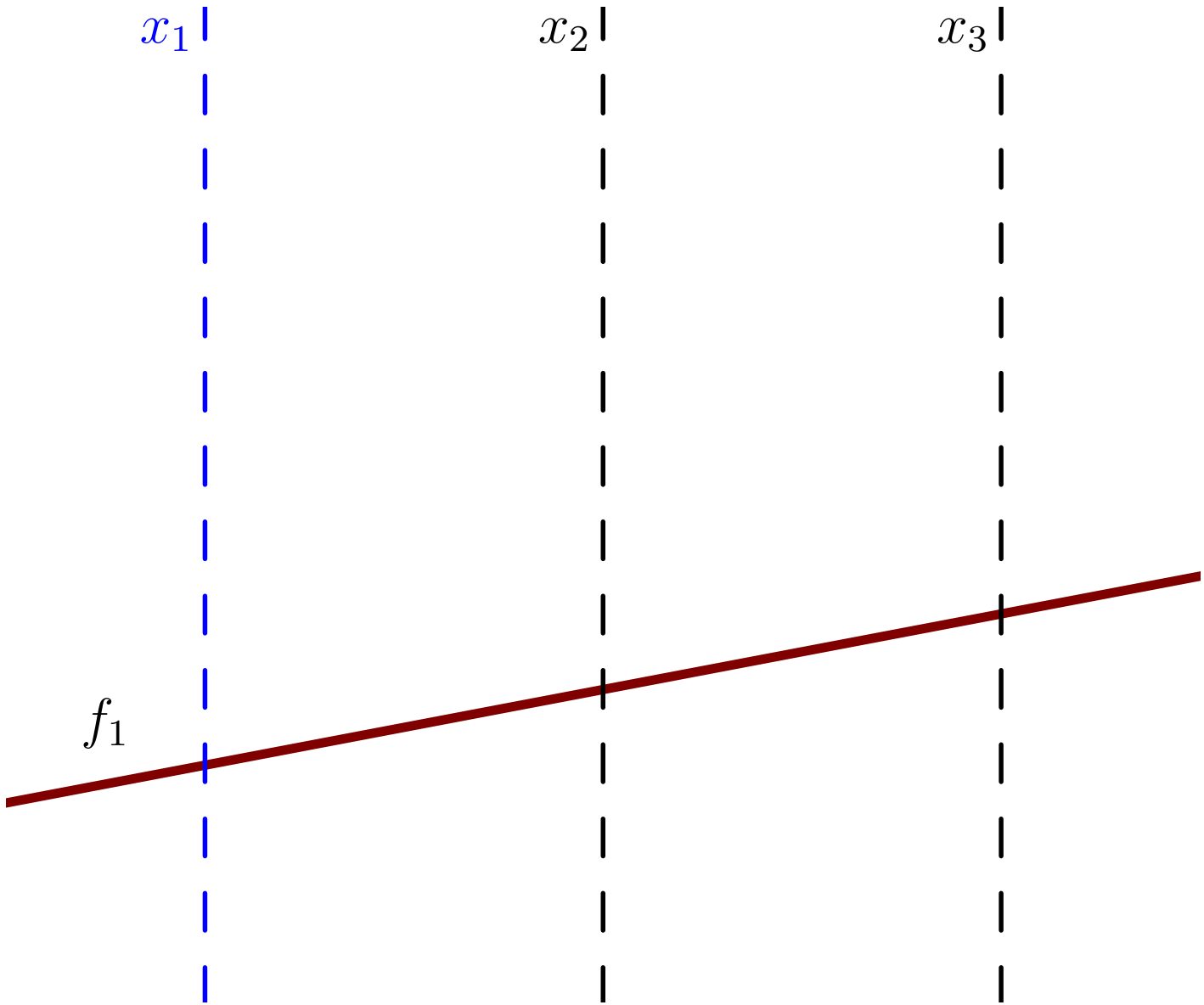
- $F_{i_1}^k, ..., F_{i_n}^k$ sind lineare Funktionen.
- $F_{i_1}^k, ..., F_{i_n}^k$ haben wachesende Steigungen wenn $y_i > y_{i+1}$

Wenn wir nun $\max_{\ell < i_1} F_{i_1}^k(x_\ell)$ für alle ℓ in $\mathcal{O}(n)$ berechnen können, dann können wir alles in $\mathcal{O}(n \cdot k)$ lösen.

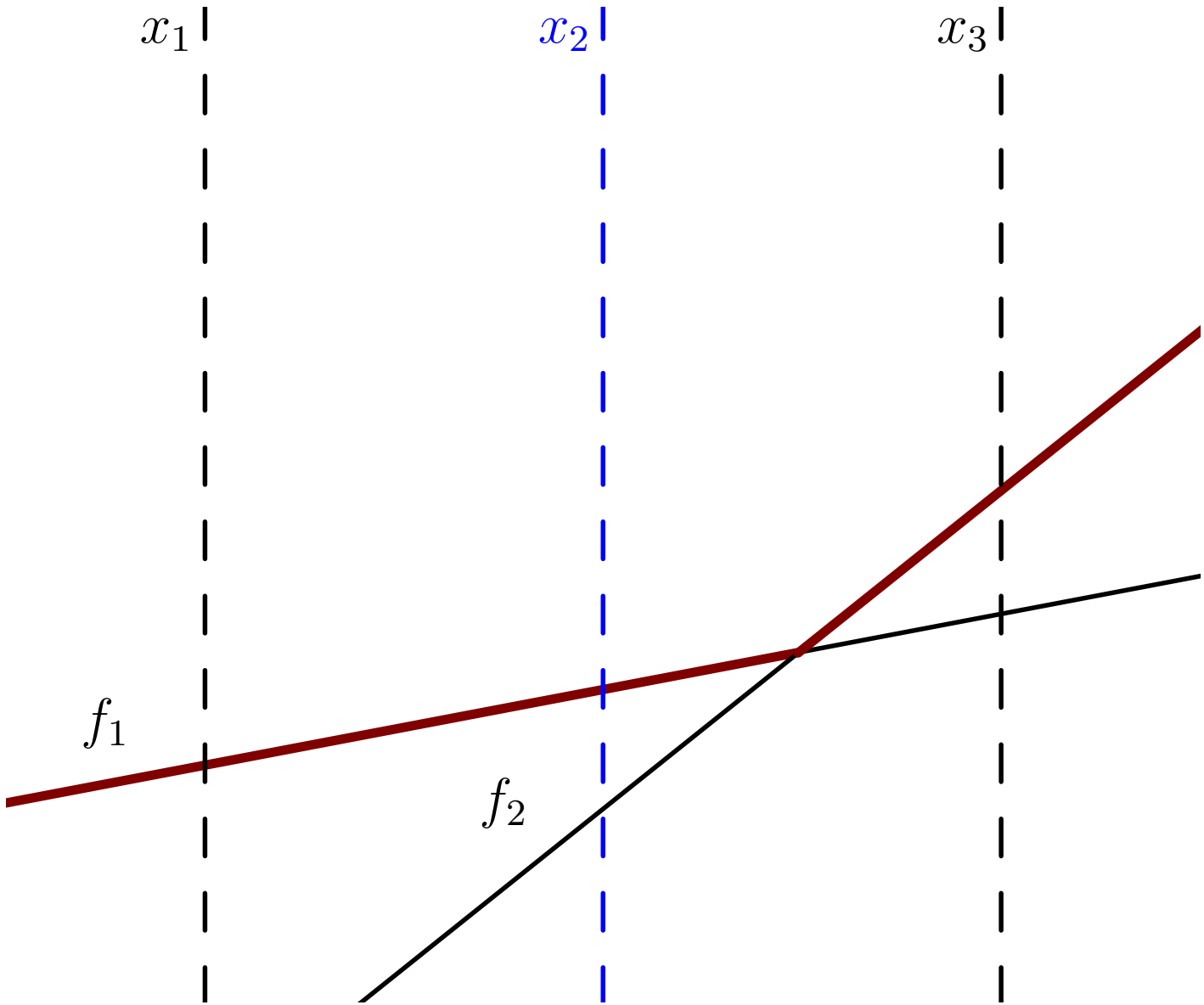
Convexe Huelle Trick



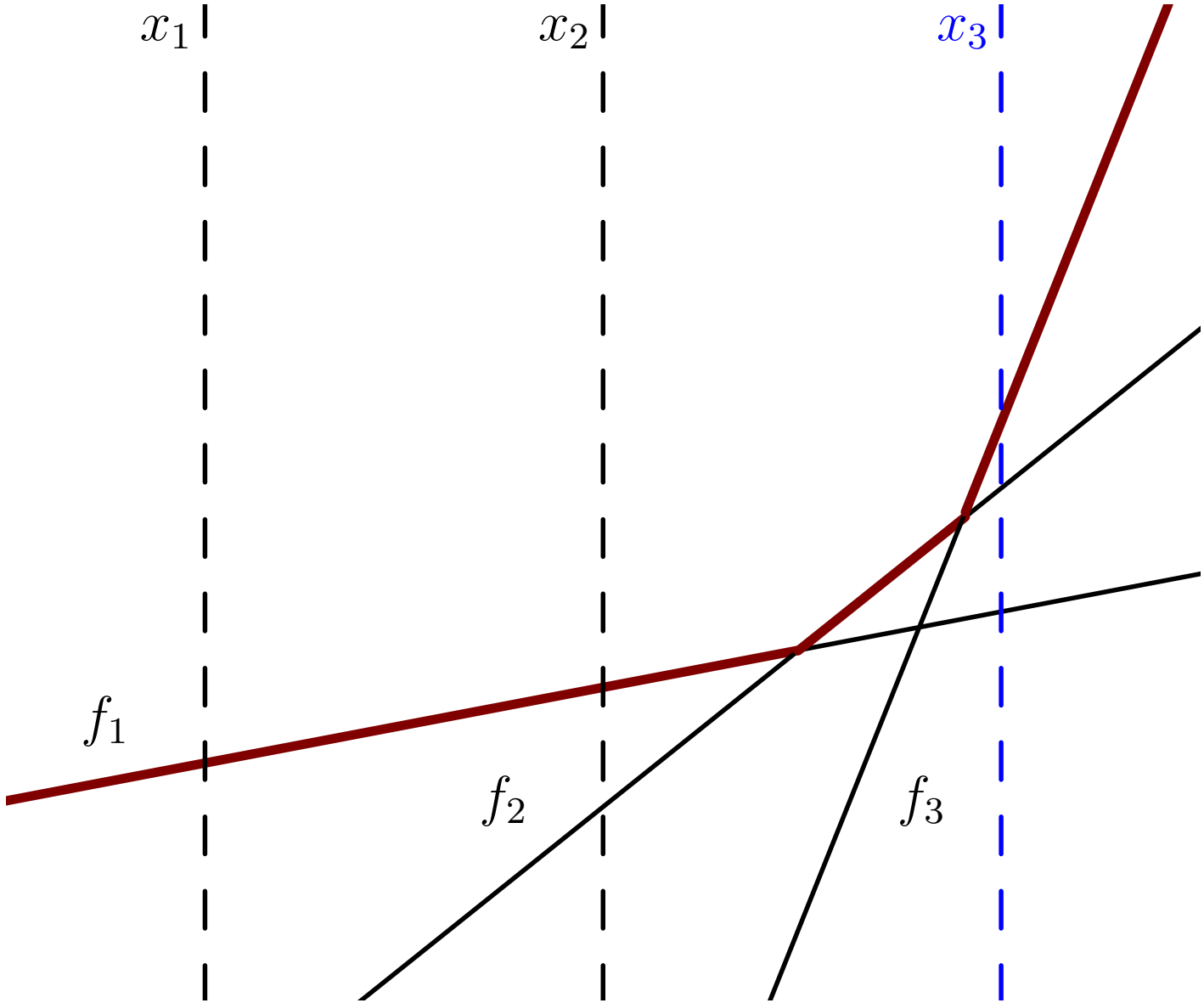
Convexe Huelle Trick



Convexe Huelle Trick



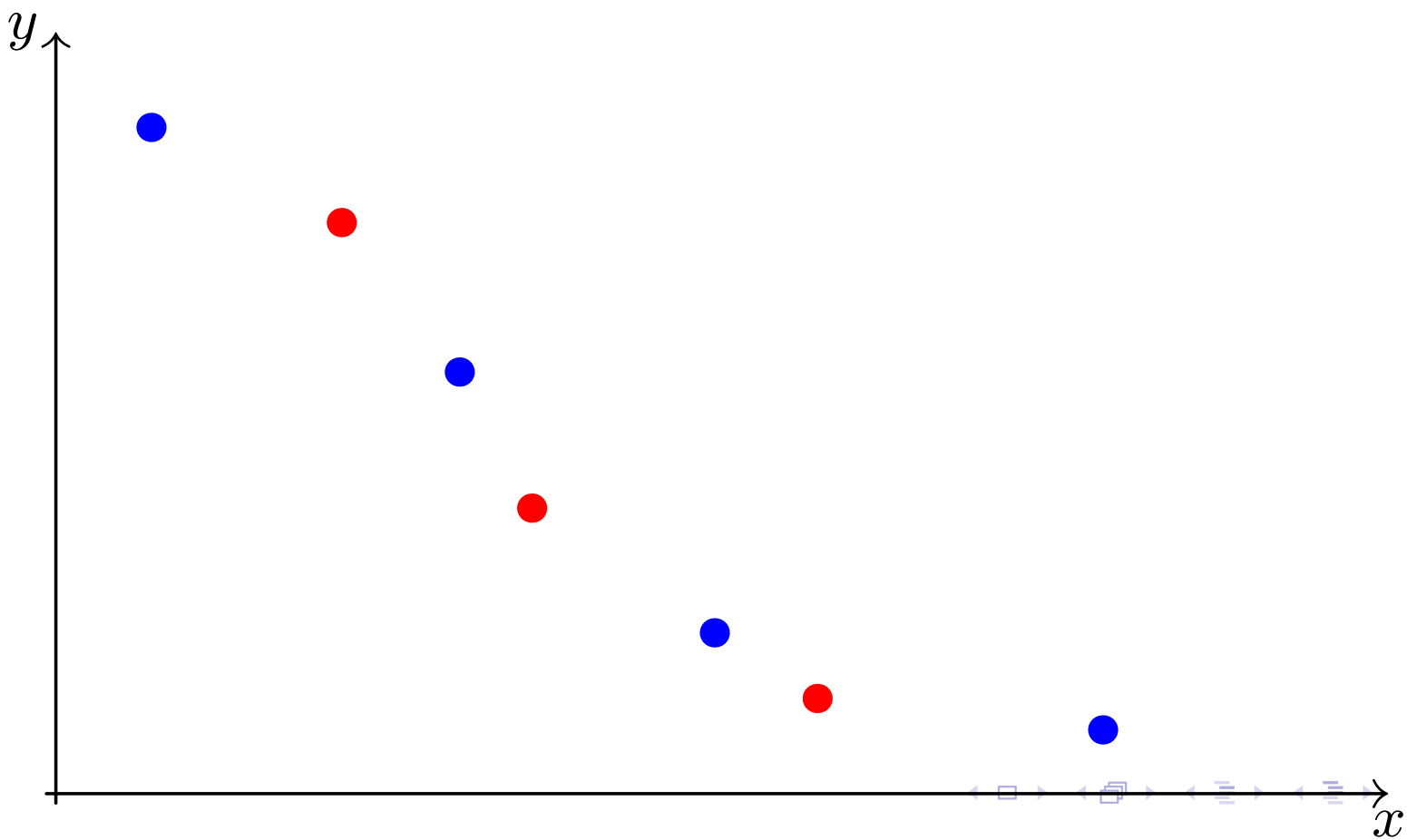
Convexe Huelle Trick



IMPLENTIERUNG

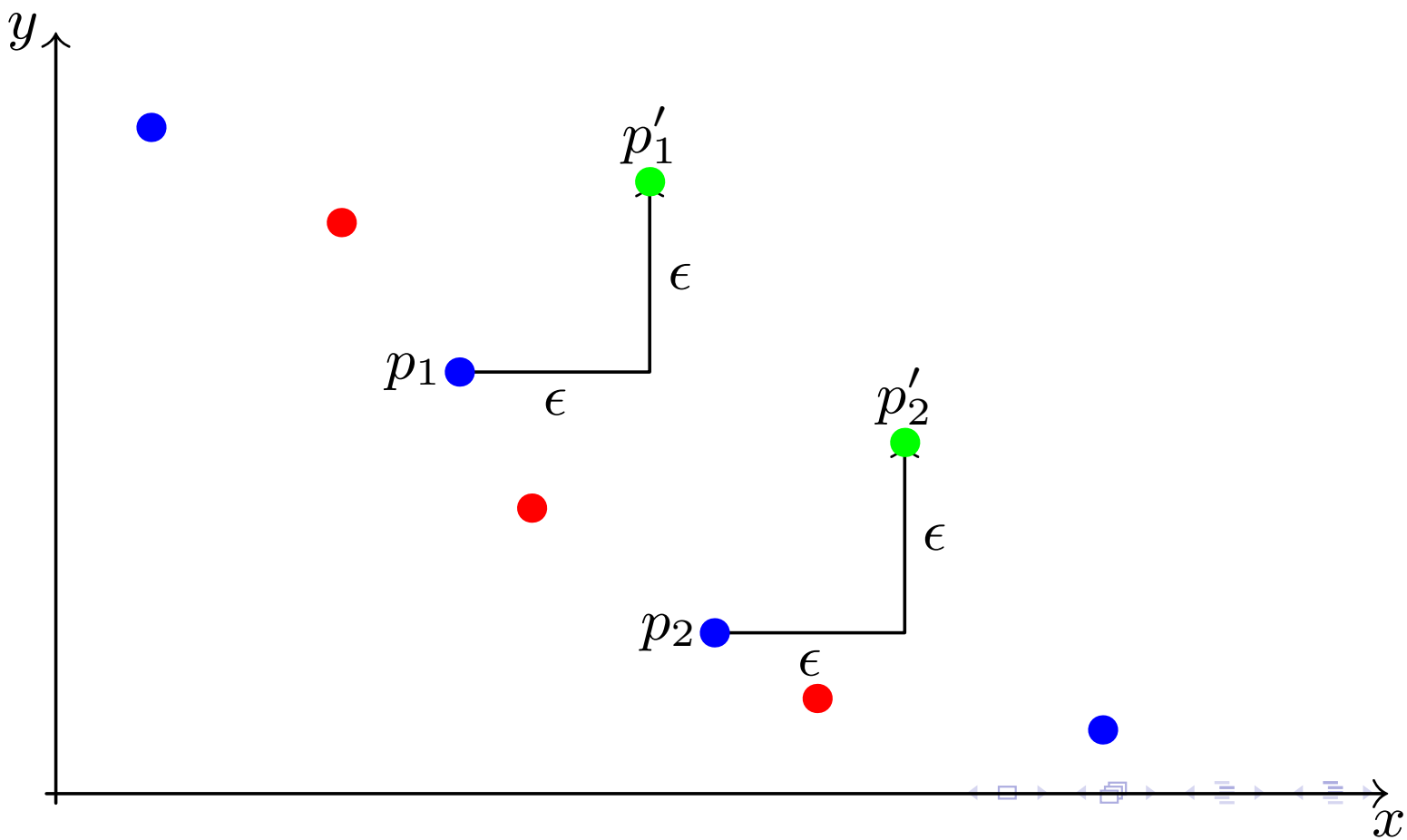
Epsilonindikator

Gegeben seinen zwei Mengen A und B und wir wollen das kleinste ε finden, sodass wenn wir k Punkte von A auswählen und sie um ε verschieben, dass die Punkte dann alle Punkte vom B dominieren (x und y koordinate groesser sind).



Epsilonindikator

Gegeben seinen zwei Mengen A und B und wir wollen das kleinste ε finden, sodass wenn wir k Punkte von A auswählen und sie um ε verschieben, dass die Punkte dann alle Punkte vom B dominieren (x und y koordinate groesser sind).



Epsilon-Test

Bevor wir das Problem lösen können, schauen wir uns folgendes Subproblem an. Nehmen wir an, wir hätten schon ein ε and k, können wir die k Punkte von A finden, sodass sie alle Punkte in B dominieren, nachdem sie um k Punkte verschoben sind ε?

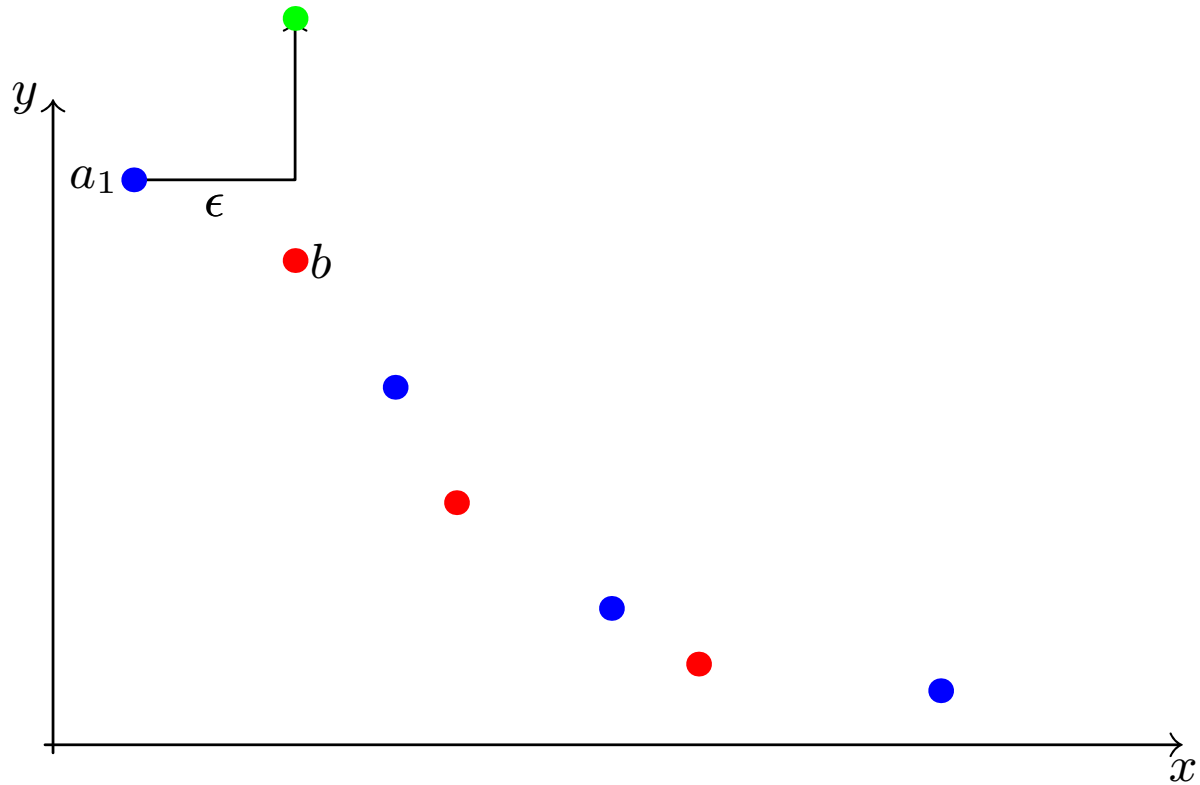
Nach diesem Test sollten wir wissen, ob das ε_{OPT} ausreicht oder es groesser ist ε

Epsilontest: Bruteforce

Das einfachste ist natuerlich wieder, alles auszuprobieren.
CODE !!

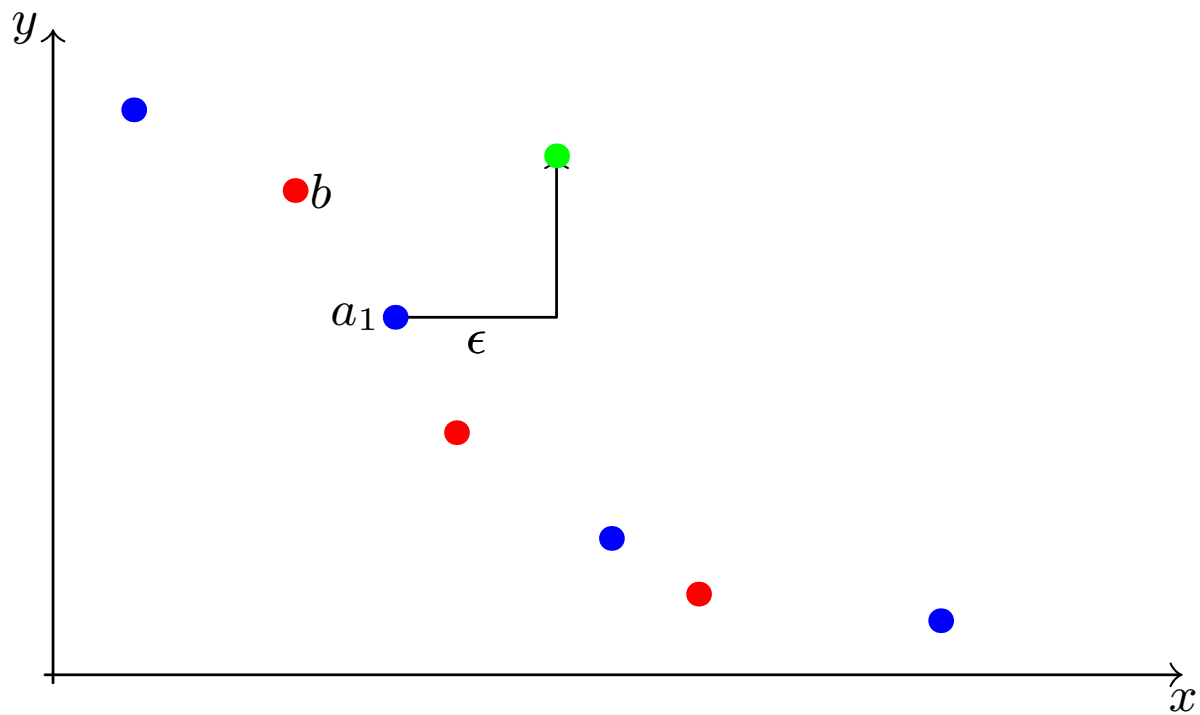
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



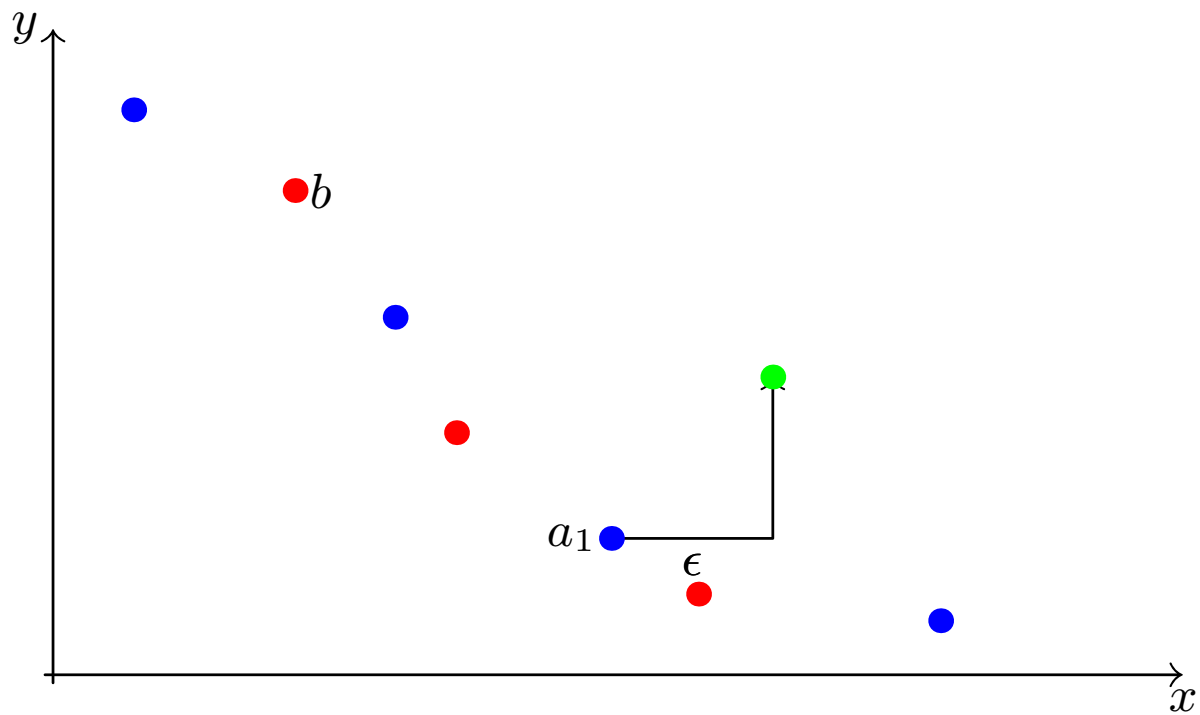
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



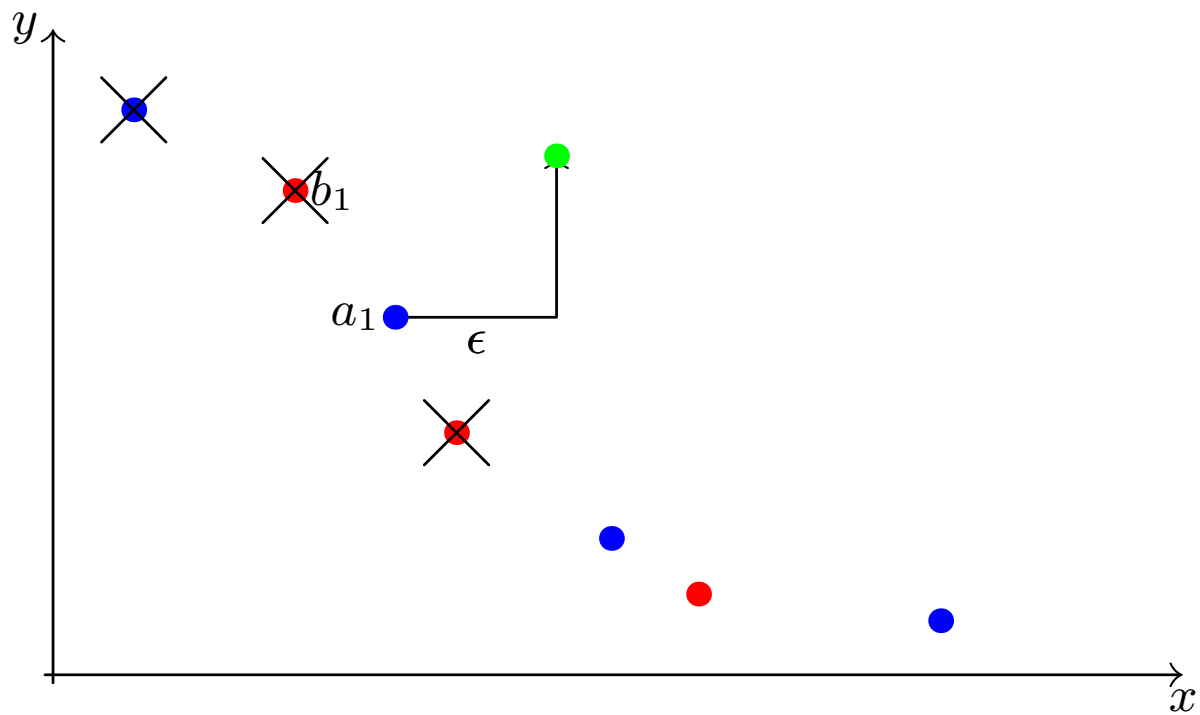
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



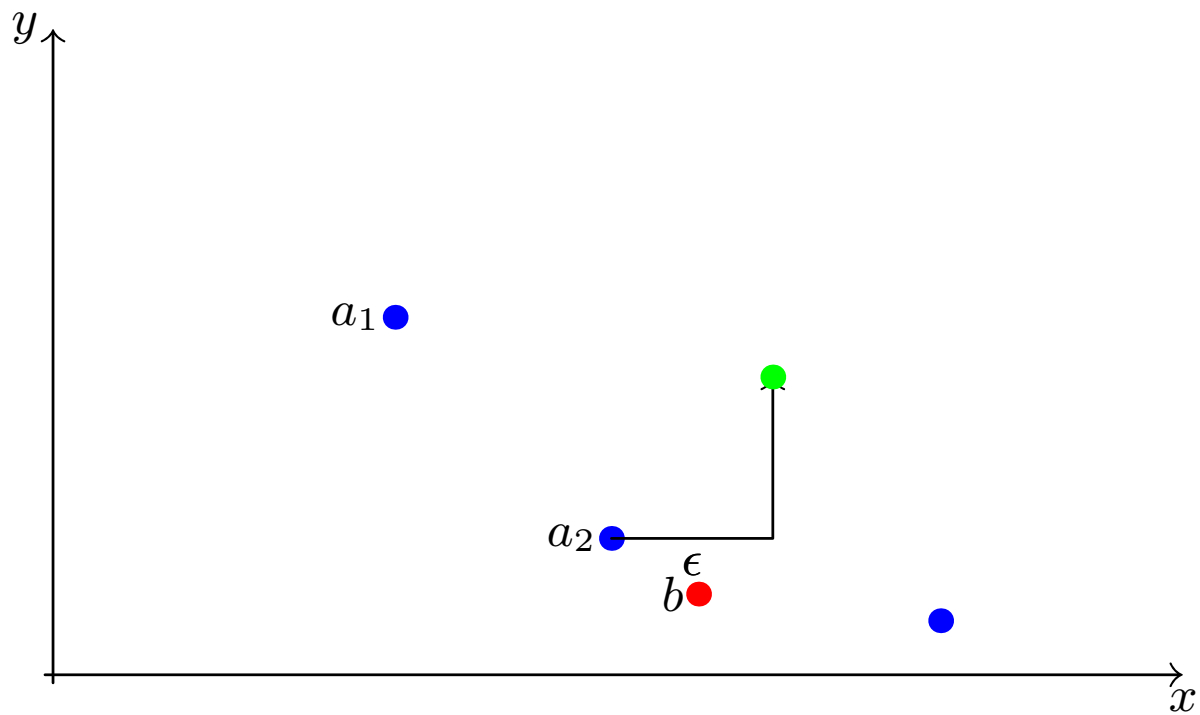
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



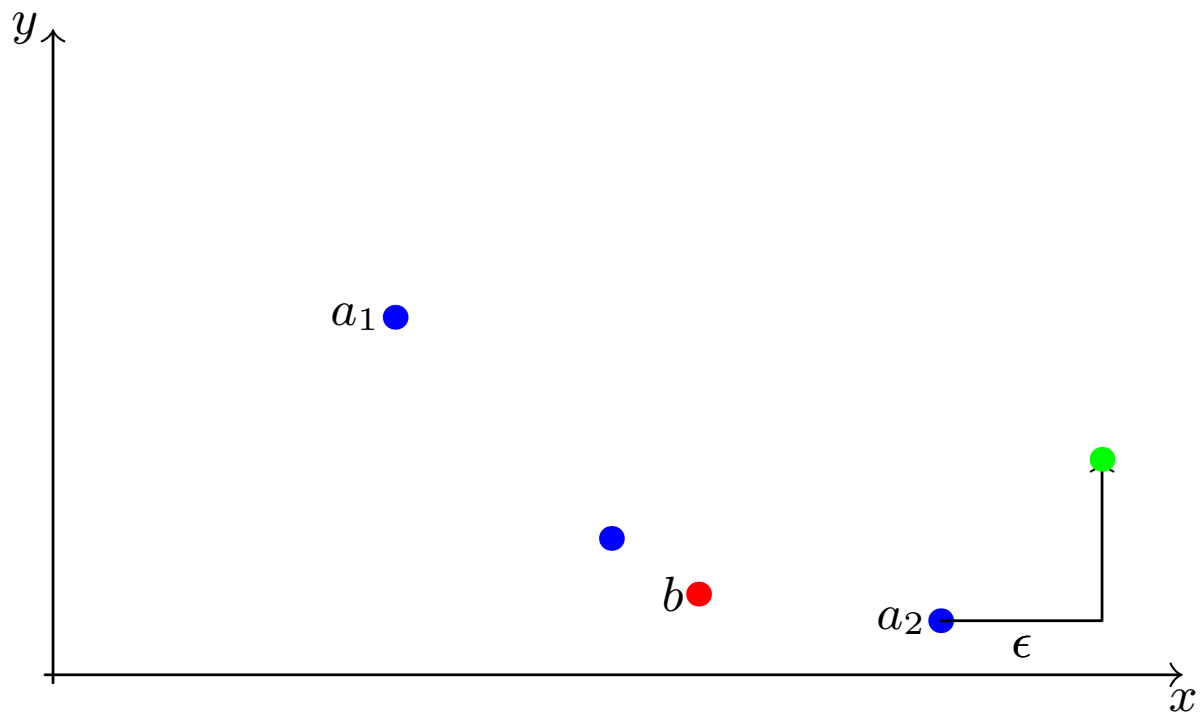
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



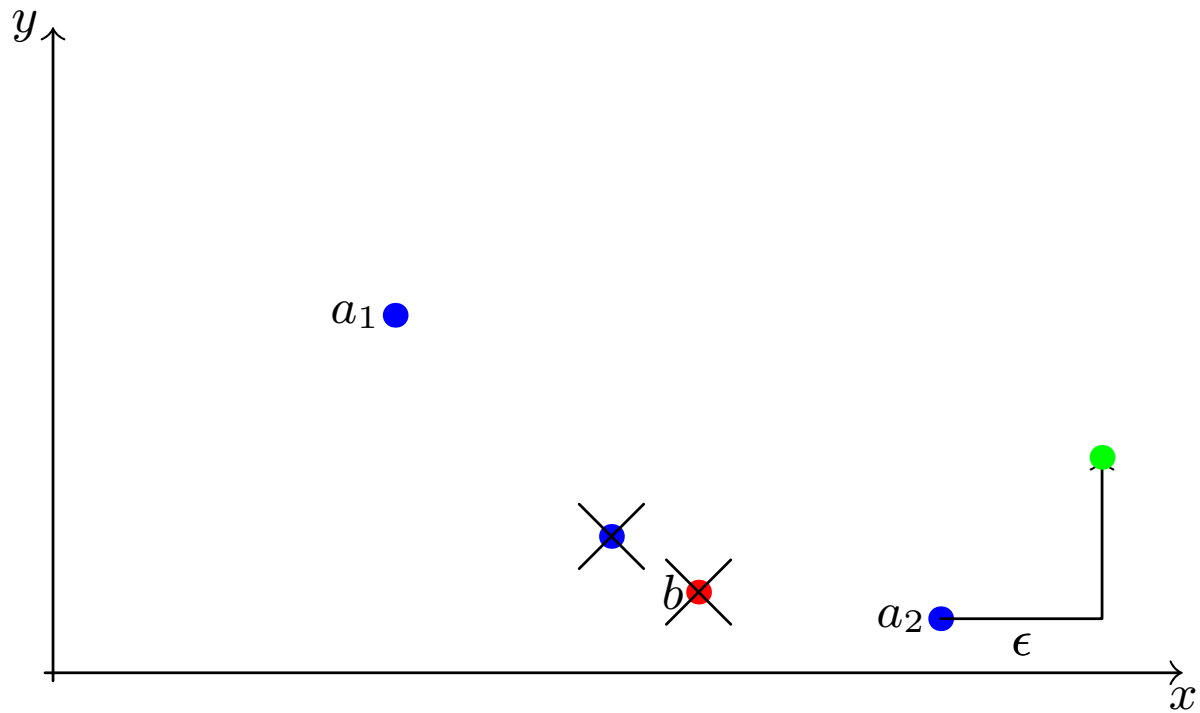
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



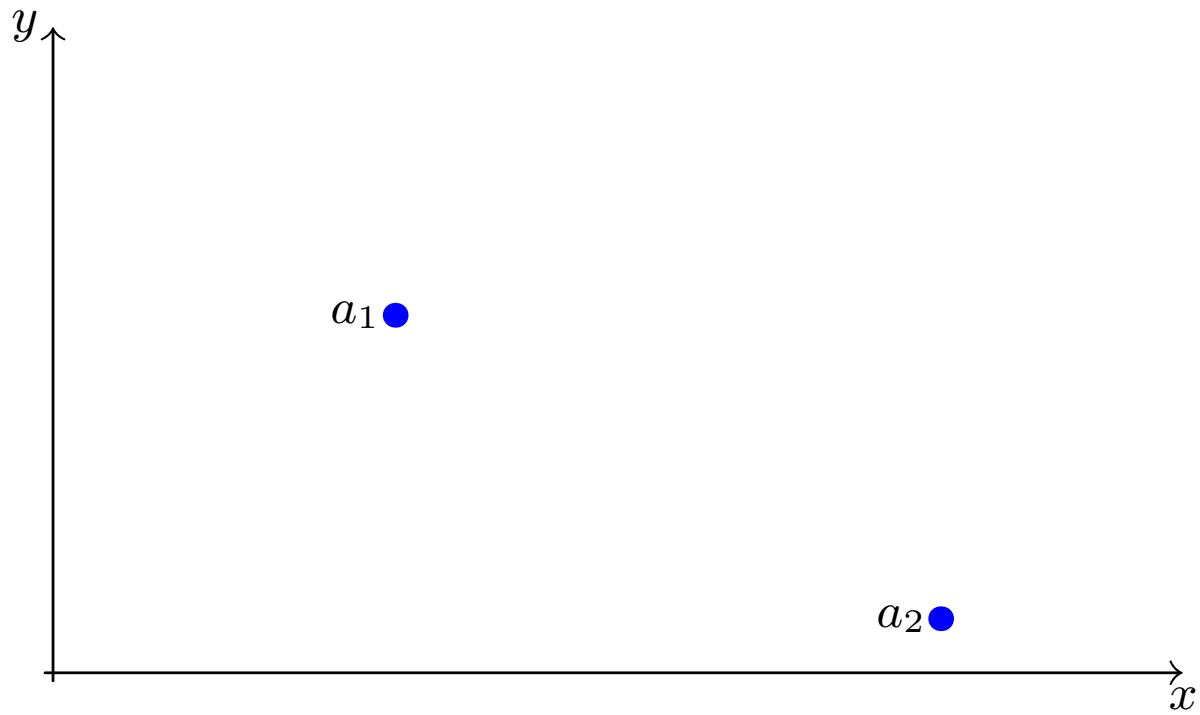
Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



Greedy Lösung

Es stellt sich heraus, dass dieses Subproblem greedy lösen kann. Wir nehmen den Punkt der am meisten rechts in A ist aber immer den ersten Punkt in B dominiert nach Verschiebung.



Greedy Lösung

Code an Tafel

Problem: Finden des optimalen epsilon ε

CODE

Problem: Finden des optimalen ε

Das optimale ε muss eine Distanz zwischen einem Punkt $a \in A$ und einem Punkt $b \in B$ sein.

Problem: Finden des optimalen ε

Das optimale ε muss eine Distanz zwischen einem Punkt $a \in A$ und einem Punkt $b \in B$ sein.

Wenn wir alle moeglichen Punkte berechnen und dann eine binäre suchen, dann haben wir eine Laufzeit von $\mathcal{O}(n^2)$.

Problem: Finden des optimalen ε

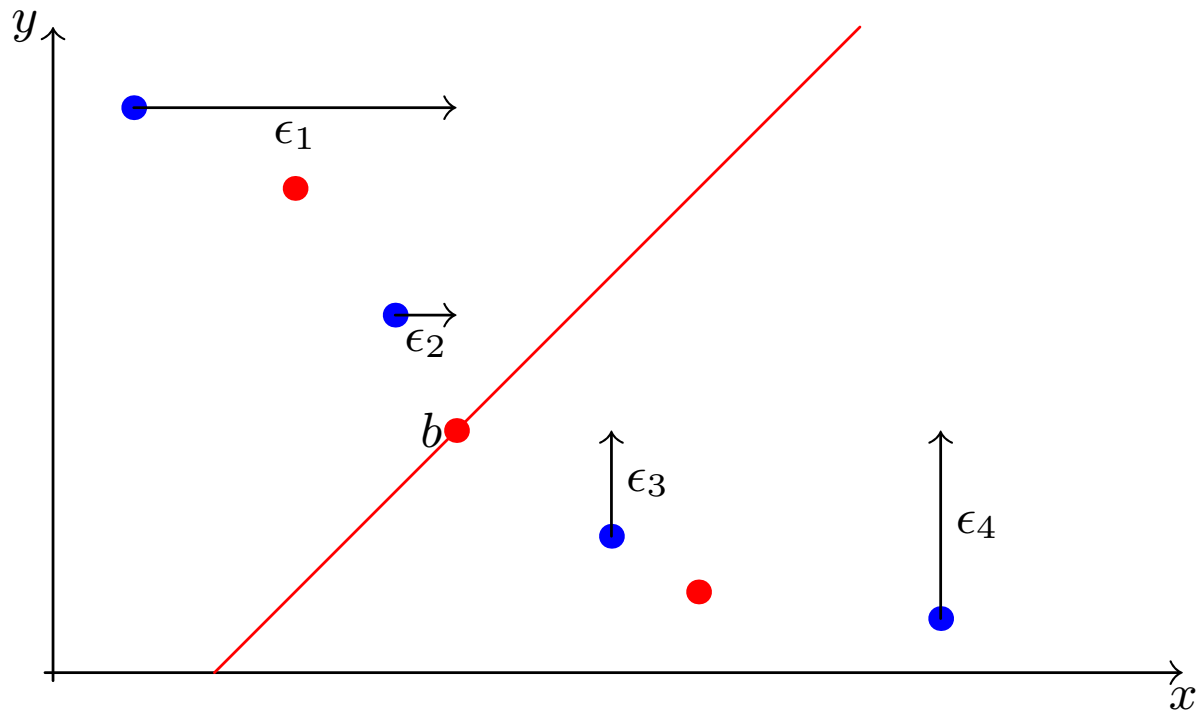
Das optimale ε muss eine Distanz zwischen einem Punkt $a \in A$ und einem Punkt $b \in B$ sein.

Wenn wir alle moeglichen Punkte berechnen und dann eine binäre suchen, dann haben wir eine Laufzeit von $\mathcal{O}(n^2)$.

Um schneller zu sein, müssen wir die Wert in einer Art und Weise speichern, sodass wir nicht alle immer auswerten müssen.

Sorted Sequences

Für jeden Punkt $b \in B$ kann man beobachten, dass alle Punkte links von b sortierte Abstände haben und dass alle Punkte rechts von B eine steigende abstände bilden.



Es stellt sich heraus, dass es reicht für jeden Punkt einfach die Anfangs und Endindex zu speichern.

Der Algorithmus

Wir erstellen zunächst die beiden Mengen inin $\mathcal{O}(|A| + |B|)$. Nachdem wir dies getan haben, tuen wir das folgende solange es mehr als eine Element in den structures gibt:

- wähle ein Zufälliges element in der Datenstruktur
- tue den ε-Test
- Wenn $\varepsilon_{OPT} \leq \varepsilon$, lösche alle $\varepsilon' > \varepsilon$
- Wenn $\varepsilon_{OPT} > \varepsilon$, lösche alle $\varepsilon' \leq \varepsilon$

Das Resultat ist ein $\mathcal{O}(n \log n)$ Algorithmus.

Datenstruktur

CODE

Einige Bemerkungen

Die Algorithmen wurden in Java und C++ verwirklicht, weil das sogenannte jMetal Framework in Java geschrieben ist. Den Code gibt es unter
`www.theinf.uni-jena.de/en/Research/Hypervolume.html`

Offene Fragen

- Kann man noch besser sein für $d = 2$?
- Kann man die NP-Härte des Problems zeigen $d > 2$?
- Können wir gute Approximationsalgorithmen für $d > 2$ finden?