A Local Search Algorithm for Cubic Clustering

Hans Harder

TU Dresden

hans.harder@mailbox.tu-dresden.de

Overview

Notation / Sets

- k-ary subsets of V

$$\binom{V}{k} = \{\{v_1, \ldots, v_k\} \subseteq V : v_1, \ldots, v_k \text{ pairwise distinct}\}$$

- X_V is the set of partitions of V
- for $\Pi \in X_V$ and $v \in V$, then $[v]_\Pi$ denotes the set in Π that contains v (e.g. for $\Pi = \{\{1,2\},\{3\}\}$ then $[1]_\Pi = \{1,2\}$)

Notation / Random Variables

- random variables $\mathbf{x}, \mathbf{y}, \ldots$ and distributions $\mathcal{Q}, \mathcal{U}, \ldots$
- random variable "distributed as" $\mathbf{x} \sim \mathcal{Q}$
- uniform distribution over a finite set A: $\mathcal{U}(A)$
- domain of a random variable $\mathbf{x}$: $\mathrm{dom}(\mathbf{x})$ (e.g. $\mathrm{dom}(\mathbf{x}) = \mathbb{R}$)
- expected value for $\mathbf{x} \sim \mathcal{Q}$ and $f : \mathrm{dom}(\mathbf{x}) \to \mathbb{R}$ and $\mathrm{dom}(\mathbf{x})$ finite:

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{Q}}[f(\mathbf{x})] = \sum_{x \in \mathrm{dom}(\mathbf{x})} \mathcal{Q}(x) f(x)$$

Problem Statement

Input: finite set V, functions $c, c' : \binom{V}{3} \to \mathbb{R}$
Output: partition Π^* of V such that

$$\Pi^* = \operatorname*{arg\,min}_{\Pi \in X_V} \sum\nolimits_{T \in \binom{V}{3}} \ell(T, \Pi)$$

where

$$\ell(\{u, v, w\}, \Pi) = \begin{cases} c'(\{u, v, w\}) & [u]_\Pi = [v]_\Pi = [w]_\Pi \\ c(\{u, v, w\}) & [u]_\Pi, [v]_\Pi, [w]_\Pi \text{ pairwise distinct} \\ 0 & \text{otherwise} \end{cases}$$

Problem Statement

Input: finite set V, functions $c, c' : \binom{V}{3} \to \mathbb{R}$
Output: partition Π^* of V such that

$$\Pi^* = \operatorname*{arg\,min}_{\Pi \in X_V} \sum\nolimits_{T \in \binom{V}{3}} \ell(T, \Pi)$$

where

$$\ell(\{u,v,w\},\Pi) = \begin{cases} c'(\{u,v,w\}) & [u]_\Pi = [v]_\Pi = [w]_\Pi \\ c(\{u,v,w\}) & [u]_\Pi, [v]_\Pi, [w]_\Pi \text{ pairwise distinct} \\ 0 & \text{otherwise} \end{cases}$$

Intuition

- c' adds penalty/reward when u, v, w are in the same set
- c adds penalty/reward when u, v, w are in different sets

Local Search

Objectives

- defining a data structure for representing partitions
- defining a neighbourhood for every partition
- defining a search algorithm (e.g. hill climbing)

[1] if only the reduced cost is computed

Local Search

Objectives

- defining a data structure for representing partitions
- defining a neighbourhood for every partition
- defining a search algorithm (e.g. hill climbing)

Difficulties

- if V is large, computing the costs in $O(|V|^3)$ is problematic
- even runtimes in $O(|V|^2)$ are prohibitive in this case[1]
- the neighbourhood for a partition might become large as well

[1] if only the reduced cost is computed

Objective

There are many local search algorithms that work equally well.
Therefore: focus on shared difficulties.

- efficient representation (memory-wise) of partitions
- efficient transformations (from partitions to their neighbours)
- efficient enumeration of the neighbourhood

and for large V's

- efficient sampling of neighbours from the neighbourhood
- approximation of the reduced costs

Basic Idea

Indexing

- represent a partition by a mapping $\varphi : V \to \{1, \ldots, n\}$ where $|V| = n$ (this can be implemented by an array of length $|V|$)
- the *equivalence kernel*[2] of φ is the equivalence relation $\sim_\varphi$ where $u \sim_\varphi v$ iff $\varphi(u) = \varphi(v)$.
- $\sim_\varphi$ yields a partition $\Pi(\varphi)$ of V.
- we say that the *indexing* φ *induces* $\Pi(\varphi)$.

[2] see, e.g. `http://en.wikipedia.org/wiki/equivalence_relation`.

Basic Idea

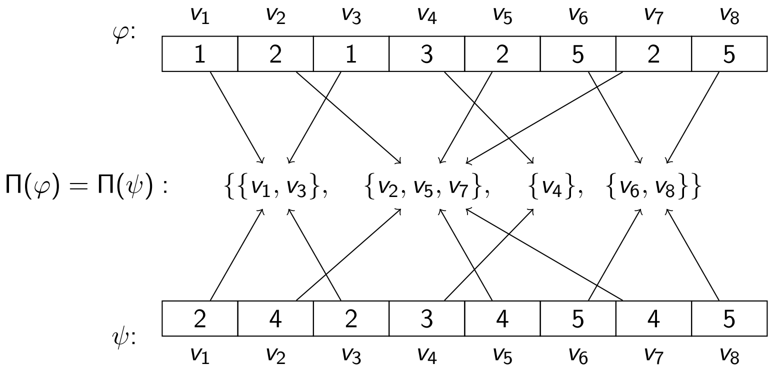
Indexing

- represent a partition by a mapping $\varphi : V \to \{1, \ldots, n\}$ where $|V| = n$ (this can be implemented by an array of length $|V|$)
- the *equivalence kernel*[2] of φ is the equivalence relation $\sim_\varphi$ where $u \sim_\varphi v$ iff $\varphi(u) = \varphi(v)$.
- $\sim_\varphi$ yields a partition $\Pi(\varphi)$ of V.
- we say that the *indexing* φ *induces* $\Pi(\varphi)$.

Pro: memory linear in $|V|$, and changing the assigned index of a $v \in V$ can be done in $O(1)$.
Con: there are multiple ways of representing the same partition through such an indexing.

[2] see, e.g. `http://en.wikipedia.org/wiki/equivalence_relation`.

Example Indexing



φ and ψ are different, but yield the same partition.

Neighbourhood of an Indexing

we define the following transformation:

$$\varphi_{v \to k}(u) = \begin{cases} k & u = v \\ \varphi(v) & u \neq v \end{cases}$$

which "moves" an element v to a (new) index k.

The neighbourhood of a partition $\Pi(\varphi)$ is defined as the set

$$Ne(\Pi(\varphi)) = \{\Pi(\varphi_{w \to k}) : w \in V, 1 \leq k \leq n, \Pi(\varphi_{w \to k}) \neq \Pi(\varphi)\}.$$

The neighbourhood of a partition $\Pi(\varphi)$ is defined as the set

$$Ne(\Pi(\varphi)) = \{\Pi(\varphi_{w\to k}) : w \in V, 1 \leq k \leq n, \Pi(\varphi_{w\to k}) \neq \Pi(\varphi)\}.$$

Open Questions. We want to find an efficient enumeration of possible "moves" $(w_1, k_1), \dots, (w_m, k_m) \in V \times \{1, \dots, n\}$ such that

- we do not enumerate "too much", i.e. $\Pi(\varphi_{w_1\to k_1}), \dots, \Pi(\varphi_{w_m\to k_m})$ are all pairwise distinct,
- all neighbours occur somewhere in this enumeration, i.e. if $\Pi \in Ne(\Pi(\varphi))$, then $\Pi = \Pi(\varphi_{w_i\to k_i})$ for a (w_i, k_i),
- we do not enumerate $\Pi(\varphi)$ itself, i.e. $\Pi(\varphi_{w_i\to k_i}) \neq \Pi(\varphi)$ for all (w_i, k_i).

Algorithm 1: MOVE-ENUMERATION

Input: Set of vertices V with indexing φ

Result: Sequence of moves $(w_1, k_1), \ldots, (w_m, k_m)$

Let $<$ be some linear order on V

Let $\mathcal{N} := \{1, \ldots, n\} \setminus \mathrm{image}(\varphi)$

forall *vertices* $w \in V$ **do**

    forall $\varphi(v) \in \mathrm{image}(\varphi) \setminus \{\varphi(w)\}$ **do**

        if $[w]_\varphi = \{w, u, \ldots\}$ *or* $[v]_\varphi = \{v, s, \ldots\}$ **then**

            enumerate $(w, \varphi(v))$

        else if $[w]_\varphi = \{w\}$ *and* $[v]_\varphi = \{v\}$ *and* $w < v$ **then**

            enumerate $(w, \varphi(v))$

    if $[w]_\varphi = \{w, u, v, \ldots\}$ *or* $([w]_\varphi = \{w, u\}$ *and* $w < u)$ **then**

        Let $k \in \mathcal{N}$

        enumerate (w, k)

Algorithm Move-Enumeration has indeed the wanted properties:

Pairwise Distinctiveness

$\Pi(\varphi_{w_i \to k_i}) \neq \Pi(\varphi_{w_j \to k_j})$ for all $1 \leq i < j \leq n$.

Completeness

for all $\Pi \in Ne(\Pi(\varphi))$, there is $1 \leq i \leq m$ such that $\Pi = \Pi(\varphi_{w_i \to k_i})$.

No self-neighbour

$\Pi(\varphi) \neq \Pi(\varphi_{w_i \to k_i})$ for all $1 \leq i \leq m$.

(proofs omitted, see report at `https://github.com/graps1/mlcv-local-search/blob/master/tex/main.pdf`).

Randomized Neighbourhood Enumeration

- there are too many neighbours: a complete sequence $(w_1, k_1), \ldots, (w_m, k_m)$ via algorithm $\textsc{Move-Enumeration}$ is in $O(|V| \cdot |\Pi(\varphi)|) = O(|V|^2)$.
- we want to select a small part of $(w_1, k_1), \ldots, (w_m, k_m)$ in order to avoid quadratically many neighbours
- how do we do this?

Randomized Neighbourhood Enumeration

- there are too many neighbours: a complete sequence $(w_1, k_1), \ldots, (w_m, k_m)$ via algorithm $\textsc{Move-Enumeration}$ is in $O(|V| \cdot |\Pi(\varphi)|) = O(|V|^2)$.
- we want to select a small part of $(w_1, k_1), \ldots, (w_m, k_m)$ in order to avoid quadratically many neighbours
- how do we do this?
  - construct r.v. $(\mathbf{w}, \mathbf{k}) \sim \mathcal{U}(\{(w_1, k_1), \ldots, (w_m, k_m)\})$
  - split sampling into two parts: $\mathbf{w} \sim \mathcal{Q}(\mathbf{w})$, then $\mathbf{k} \sim \mathcal{P}(\mathbf{k}|\mathbf{w})$ such that $\mathcal{U}(\mathbf{w}, \mathbf{k}) = \mathcal{Q}(\mathbf{w})\mathcal{P}(\mathbf{k}|\mathbf{w})$.
  - sampling from $\mathcal{Q}(\mathbf{w})$ and $\mathcal{P}(\mathbf{k}|\mathbf{w})$ can be done efficiently (see report for more details).
  - this yields an algorithm $\textsc{Random-Move-Enumeration}$ that samples N neighbours in $O(|V| \cdot N)$.

Computing the Costs

recall: we want to find a partition Π^* such that

$$\Pi^* = \operatorname*{arg\,min}_{\Pi \in X_V} \sum_{T \in \binom{V}{3}} \ell(T, \Pi).$$

Computing the Costs

recall: we want to find a partition Π^* such that

$$\Pi^* = \operatorname*{arg\,min}_{\Pi \in X_V} \sum_{T \in \binom{V}{3}} \ell(T, \Pi).$$

we can rewrite this as an expectation:

$$\begin{aligned}
\Pi^* &= \operatorname*{arg\,min}_{\Pi \in X_V} \sum_{T \in \binom{V}{3}} \ell(T, \Pi) \\
&= \operatorname*{arg\,min}_{\Pi \in X_V} \frac{1}{\left|\binom{V}{3}\right|} \sum_{T \in \binom{V}{3}} \ell(T, \Pi) \\
&= \operatorname*{arg\,min}_{\Pi \in X_V} \mathbb{E}_{\mathbf{T} \sim \mathcal{U}(\binom{V}{3})} \left[\ell(\mathbf{T}, \Pi)\right]
\end{aligned}$$

Computing the Costs

recall: we want to find a partition Π^* such that

$$\Pi^* = \arg\min_{\Pi \in X_V} \sum_{T \in \binom{V}{3}} \ell(T, \Pi).$$

we can rewrite this as an expectation:

$$\begin{aligned}
\Pi^* &= \arg\min_{\Pi \in X_V} \sum_{T \in \binom{V}{3}} \ell(T, \Pi) \\
&= \arg\min_{\Pi \in X_V} \frac{1}{\left|\binom{V}{3}\right|} \sum_{T \in \binom{V}{3}} \ell(T, \Pi) \\
&= \arg\min_{\Pi \in X_V} \mathbb{E}_{\mathbf{T} \sim \mathcal{U}(\binom{V}{3})} \left[\ell(\mathbf{T}, \Pi)\right]
\end{aligned}$$

we can approximate the costs by computing the sample mean!

Computing the Reduced Costs

if we want to compute how much a neighbour $\Pi(\varphi_{v\to k})$ improves the value of a given $\Pi(\varphi)$, we obtain:

$$\begin{aligned} J(\Pi(\varphi), \Pi(\varphi_{v\to k})) &= \mathbb{E}_{\mathbf{T}}\left[\ell(\mathbf{T}, \Pi(\varphi))\right] - \mathbb{E}_{\mathbf{T}}\left[\ell(\mathbf{T}, \Pi(\varphi_{v\to k}))\right] \\ &= \mathbb{E}_{\mathbf{T}}\left[\ell(\mathbf{T}, \Pi(\varphi)) - \ell(\mathbf{T}, \Pi(\varphi_{v\to k}))\right] \\ &= \mathbb{E}_{\mathbf{T}}\left[\delta(\mathbf{T}, \Pi(\varphi), \Pi(\varphi_{v\to k}))\right] \\ &= \mathbb{E}_{\{\mathbf{u},\mathbf{w}\}\sim\mathcal{U}(\binom{V\setminus\{v\}}{2})}\left[\delta(\{\mathbf{u}, v, \mathbf{w}\}, \Pi(\varphi), \Pi(\varphi_{v\to k}))\right] \end{aligned}$$

since $\ell(T, \Pi(\varphi)) = \ell(T, \Pi(\varphi_{v\to k}))$ whenever $v \notin T$.
$\to$ computing the reduced costs J is in $O(|V|^2)$.

Greedy Search

Algorithm 2: GREEDY-SEARCH

Input: Set of vertices V with indexing φ.
Result: Better indexing ψ

1 **while** *not* $(\textit{stopping criterion})$ **do**
2 $\quad (v_1, k_1), \ldots, (v_m, k_m) := \textsc{Move-Enumeration}(V, \varphi)$
3 $\quad (v^*, k^*) := \arg\max_{(v_i,k_i)} J(\Pi(\varphi), \Pi(\varphi_{v_i \to k_i}))$
4 $\quad$ **if** $J(\Pi(\varphi), \Pi(\varphi_{v^* \to k^*})) \leq 0$ **then**
5 $\quad\quad$ **return** φ
6 $\quad \varphi := \varphi_{v^* \to k^*}$
7 **return** φ

Complexity: line 2 returns $O(|V|^2)$ neighbours, and computing the reduced costs for each of these neighbours in line 3 is in $O(|V|^4)$. The remainder can be done in $O(1)$. The overall complexity per iteration is therefore $O(|V|^4)$.

Greedy Search with Sampling

Algorithm 3: Greedy-Search with Sampling

Input: Set of vertices V with indexing φ, neighbourhood sample size N, objective sample size M

Result: Better indexing ψ

1 **while** *not (stopping critierion)* **do**

2 $\quad (v_1, k_1), \ldots, (v_N, k_N) := \textsc{Random-Move-Enumeration}(V, \varphi, N)$

3 $\quad$ Sample $\{u_{i,1}, w_{i,1}\}, \ldots, \{u_{i,M}, w_{i,M}\}$ from $\binom{V \setminus \{v_i\}}{2}$ for $1 \leq i \leq N$

4 $\quad (v^*, k^*) := \arg\max_{(v_i, v_i)} \frac{1}{M} \sum_{j=1}^{M} \delta(\{u_{i,j}, w_{i,j}, v_i\}, \Pi(\varphi_i), \Pi(\varphi_{v_i \to k_i}))$

5 $\quad$ **if** $\varphi_{v^* \to k^*}$ *is an improvement over* φ **then**

6 $\quad\quad \varphi := \varphi_{v^* \to k^*}$

7 **return** φ

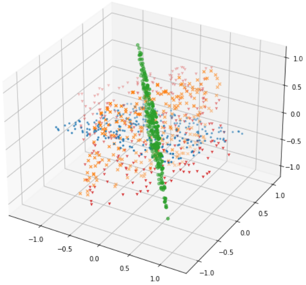
Complexity: line 2 is in $O(|V| \cdot N)$, line 3 and 4 are both in $O(M \cdot N)$, and line 5 and 6 are doable in constant time. This yields a complexity of $O(N \cdot (M + |V|))$ per iteration.

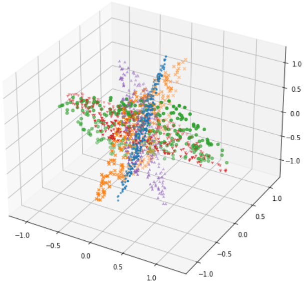
Scenario

given: set of points in $\mathbb{R}^3$ that are sampled with noise from 2-5 random planes going through the origin (without knowing from which plane a point stems).
goal: figure out original set of planes by partitioning the points.

Scenario

given: set of points in $\mathbb{R}^3$ that are sampled with noise from 2-5 random planes going through the origin (without knowing from which plane a point stems).
goal: figure out original set of planes by partitioning the points.
example:





Scenario

implemented cost-structure: define c to be 0 everywhere and define

$$c'(\{u, v, w\}) = \mathrm{distance}(\mathrm{plane}(\{u, v, w\}), (0, 0, 0)) - \tau$$

where τ is some small positive constant.

Scenario

implemented cost-structure: define c to be 0 everywhere and define

$$c'(\{u, v, w\}) = \mathrm{distance}(\mathrm{plane}(\{u, v, w\}), (0, 0, 0)) - \tau$$

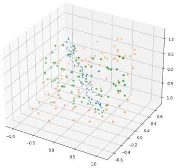
where τ is some small positive constant. This yields the following intuition:

- if the distance between the plane formed by $\{u, v, w\}$ to the origin is larger than τ, u, v, w probably don't belong to the same partition (plane) $\rightarrow$ positive costs.
- if the distance is smaller respectively, they probably belong to the same partition (plane) $\rightarrow$ negative costs.

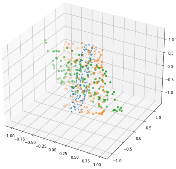
Implementation

- implemented in Python 3,
- distances are computed in batches using `numpy`,
- visualization using `matplotlib`,
- processing of experimental results using `pandas`

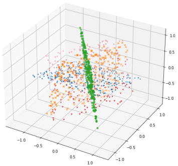
Datasets



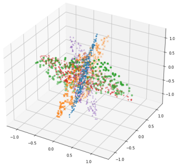
(a) Dataset 1, 3×80 points ($|V| = 240$)



(b) Dataset 2, 3×150 points ($|V| = 450$)



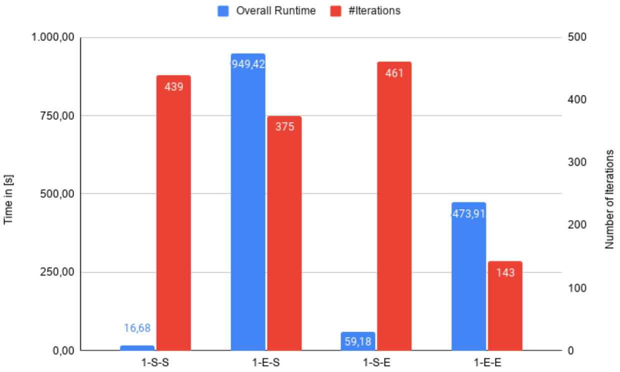
(c) Dataset 3, 4×250 points ($|V| = 1000$)



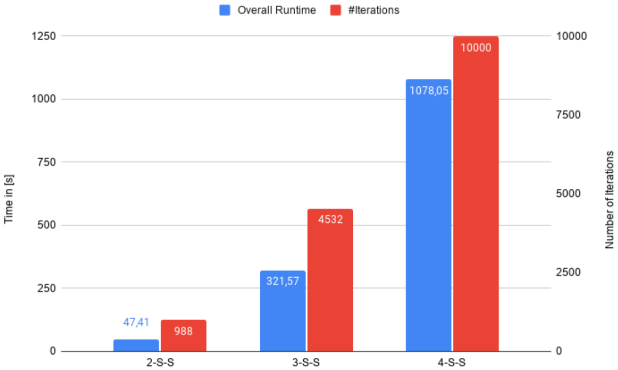
(d) Dataset 4, 5×300 points ($|V| = 1500$)

Comparisions:

- greedy search vs. greedy search with sampling
- greedy search with sampling on growing problem size



run	dataset	N	M	**or** [s]	**iter**	**or**/**iter** [s]
1-S-S	1	40	5000	16.68	439	0.038
1-E-S	1	–	5000	949.42	375	2.532
1-S-E	1	40	–	59.18	461	0.128
1-E-E	1	–	–	473.91	143	3.314



run	dataset	N	M	**or** [s]	**iter**	**or**/**iter** [s]
2-S-S	2	50	5000	47.41	988	0.048
3-S-S	3	50	10000	321.57	4532	0.071
4-S-S	4	50	15000	1078.05	10000	0.108

Conclusion.

- investigation of a fitting data structure:
  - can be stored in $O(|V|)$
  - elements can be moved from one index to another in $O(1)$
  - comparing the equivalence of two indexings φ, ψ (i.e. whether their induced partitions are the same) is doable in $O(|V|)$[3]
- proposed algorithms for enumerating the neighbourhood (randomly)
- viewing (reduced-)costs as expected values allowed for efficient approximations

[3]see my report for more details.

Conclusion.

- investigation of a fitting data structure:
  - can be stored in $O(|V|)$
  - elements can be moved from one index to another in $O(1)$
  - comparing the equivalence of two indexings φ, ψ (i.e. whether their induced partitions are the same) is doable in $O(|V|)$[3]
- proposed algorithms for enumerating the neighbourhood (randomly)
- viewing (reduced-)costs as expected values allowed for efficient approximations

Further Research.

- extension to other local search algorithms (e.g. tabu-search, simulated annealing)
- investigate scaleability (e.g. through a more efficient implementation in another programming language)
- investigate how something similar could be carried out to other data structures, e.g. the disjoint-set data structure.

[3] see my report for more details.