Bayesian Optimisation of Functions over Node Combinations in Graph

Machine Learning Research Group

Department of Engineering Science
University of Oxford

Outline

Research Problem
- Functions of node combinations
- Real-world applications
- Challenges

Preliminaries
- Bayesian Optimisation
- Graph Gaussian Processes
- BO of functions on a single node

Methodology
- The combinatorial graph of node combinations
- Recursive combo-subgraph sampling
- *GraphComBO*

Real-world Experiments

Research Problem

In network analysis, we are often interested in finding a subset $\mathcal{S}$ of k nodes that leads to the maximisation of a black-box utility function $f(\mathcal{S})$ on a generic graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ over the combinatorial space $\binom{\mathcal{V}}{k}$.

In the case of maximisation, it can be expressed as:

$$\mathcal{S}^* = \arg \max_{\mathcal{S} \in \binom{\mathcal{V}}{k}} f(\mathcal{S}). \tag{1}$$

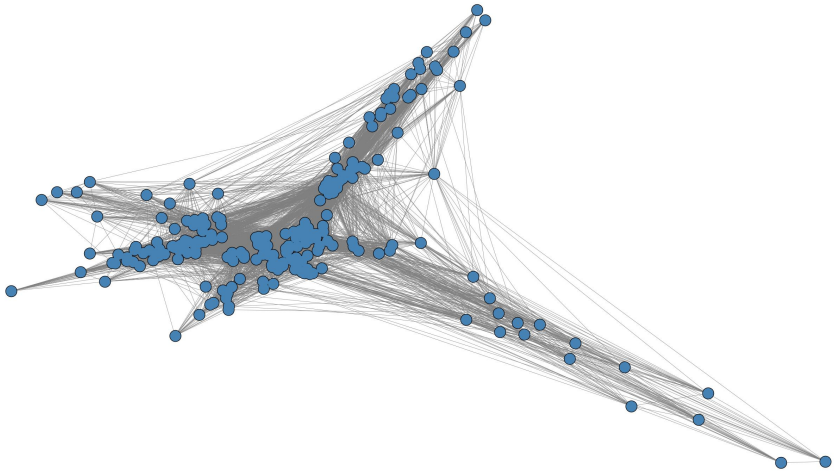
Real-world examples

Real-world applications include:

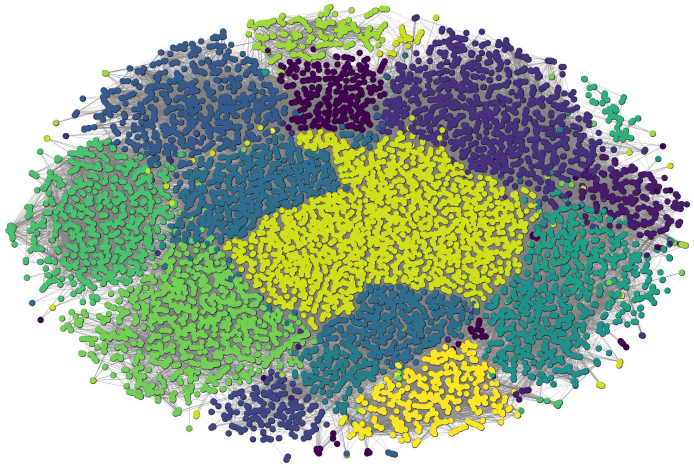
- Tracing the patient-zero in a contact network.
- Selecting target users to maximise marketing influence.
- Identifying the most vulnerable roads in a traffic network.
- Testing GNN robustness by masking graph edges.

Real-world examples

Proximity-contact network

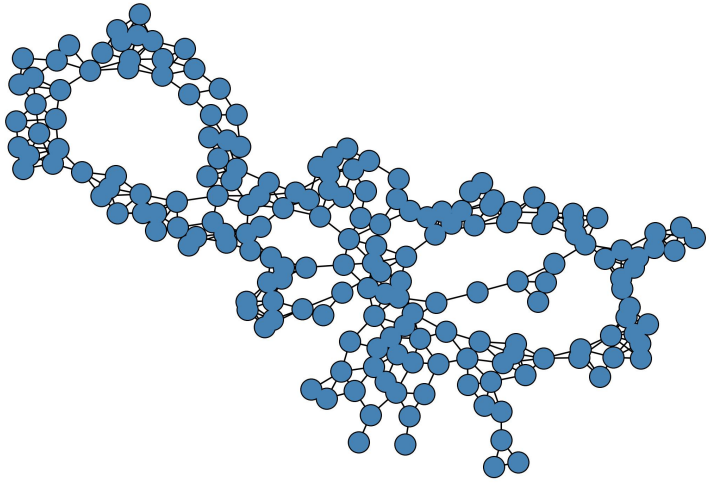


Social network



London road network

Molecule from DD-TUDataset



To optimise this utility function, however, is a non-trivial task. Challenges include:

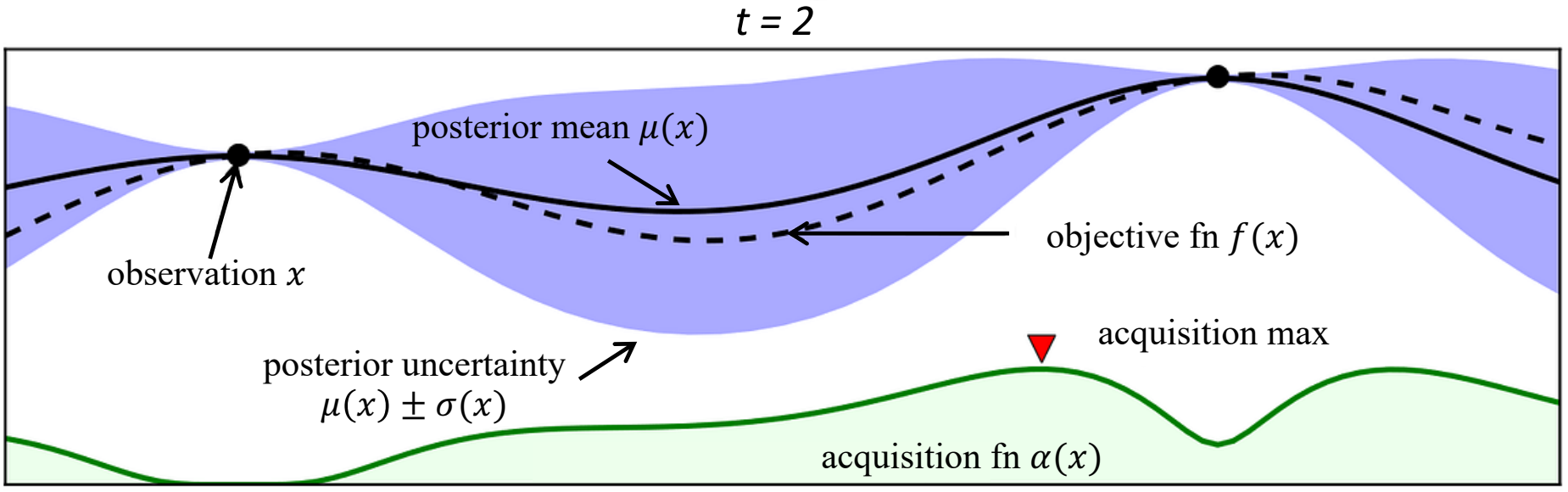
- Most conventional optimisation algorithms are designed for continuous space.
- The combinatorial search space $\binom{\mathcal{V}}{k}$ is massive and possesses an underlying graph structure.
- The original graph is not even fully observable in certain scenarios (e.g. offline social networks or contact networks).
- The objective functions are usually black-box and expensive to evaluate in many applications.

Preliminaries

BO is a gradient-free optimisation algorithm that aims to find the global optimal point x^* of a back-box function $f : \mathcal{X} \to \mathbb{R}$ over the search space $\mathcal{X}$, which, in the case of maximisation, can be written as $x^* = \arg\max_{x \in \mathcal{X}} f(x)$.

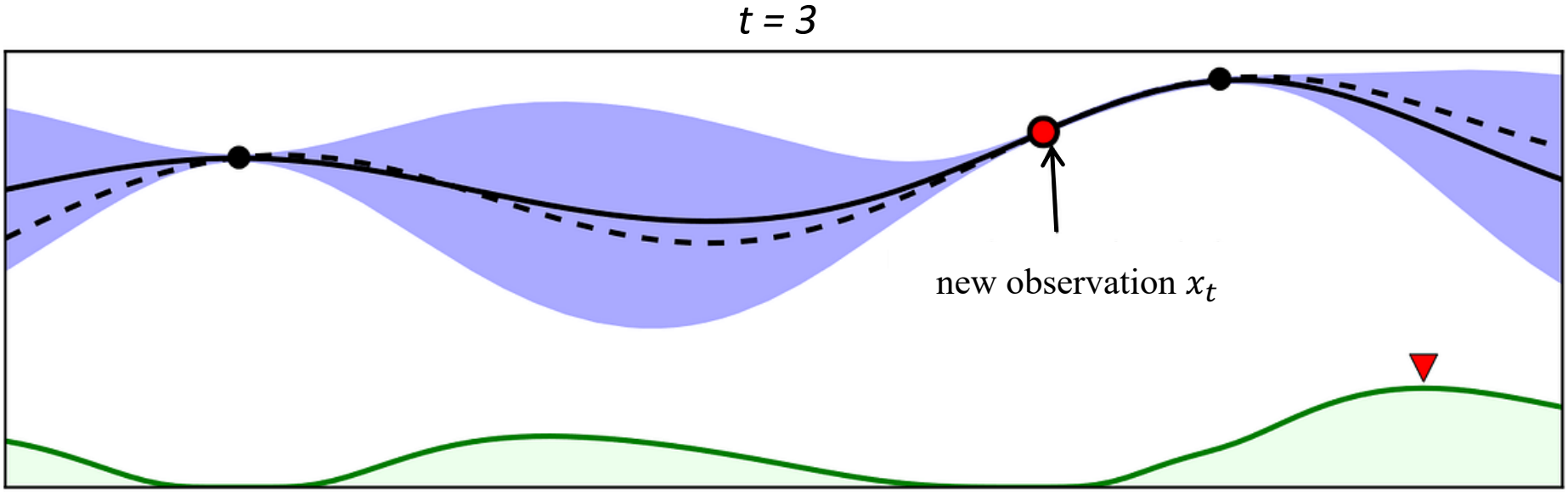
The framework consists of two steps at each iteration:

- Fitting a surrogate model.
- Maximizing an acquisition function.



Gaussian Processes surrogate:

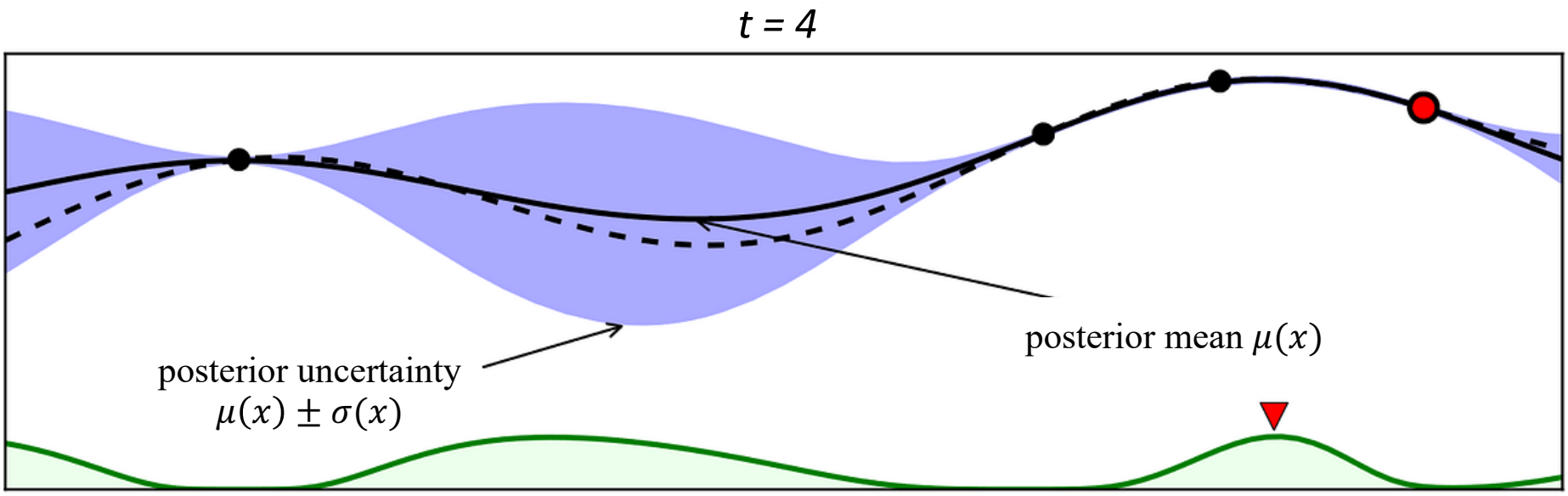
$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x')).$$



Expected Improvement acquisition:

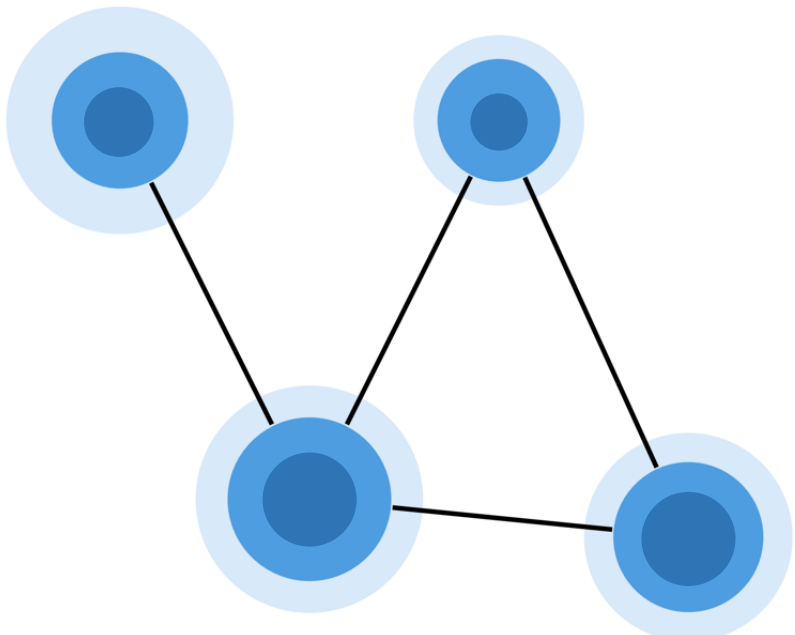
$$\mathrm{EI}_{1:t}(x') = \mathbb{E}\big[[f(x') - f(x^*_{1:t})]^+\big],$$

with $[\alpha]^+ = \max(\alpha, 0)$ and $x^*_t = \arg\max_{x_i \in x_{1:t}} f(x_i)$.


t = 4
posterior mean $\mu(x)$
posterior uncertainty
$\mu(x) \pm \sigma(x)$

What about Gaussian Processes on a graph, where the function $f(v)$ is defined on graph nodes $v \in \mathcal{V}$?

1. Consider the normalised graph Laplacian matrix: $\tilde{\mathbf{L}} = \mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$.
2. Perform eigendecomposition of the normalised Laplacian matrix $\tilde{\mathbf{L}}$: $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \cdots, \lambda_n)$ are the sorted eigenvalues and $\mathbf{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$ are their corresponding eigenvectors.



3. The covariance function $k(v_i, v_j)$ between an arbitrary node pair v_i and v_j can be formulated in the form of a regularisation function $r(\lambda_p)$ [1] defined on the eigenvalues $\{\lambda_p\}_{p=1}^n$:

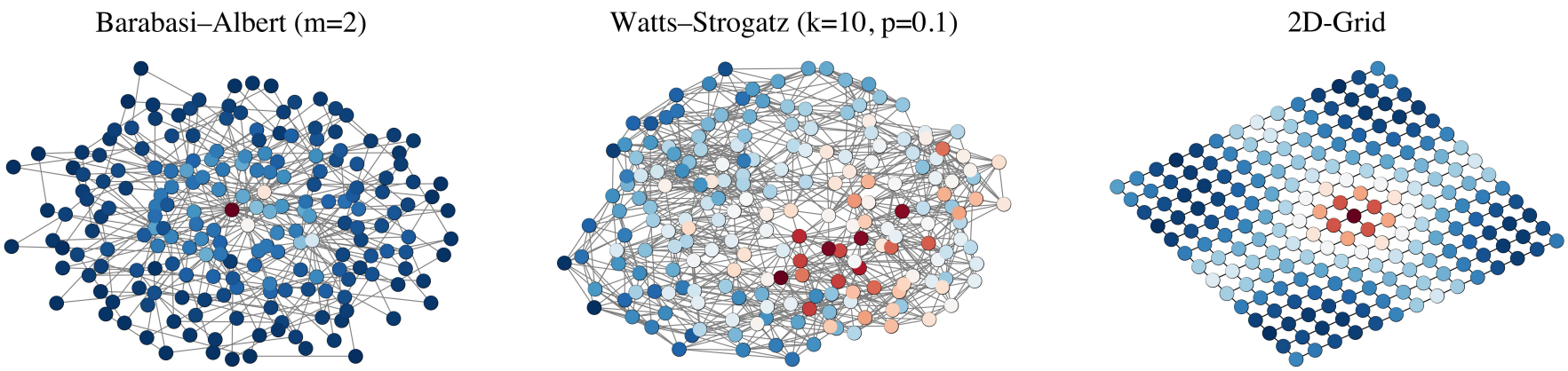
$$k(v_i, v_j) = \sum_{p=1}^{n} r^{-1}(\lambda_p)\boldsymbol{u}_p[i]\boldsymbol{u}_p[j], \tag{2}$$

where $\boldsymbol{u}_p[i]$ and $\boldsymbol{u}_p[j]$ are the i-th and j-th elements in the p-th eigenvector $\boldsymbol{u}_p$, and $r(\lambda_p)$ is some regularisation function.

Choice of Kernel	Regularisation $r(\lambda_p)$	Kernel $K_n(\mathcal{V}, \mathcal{V})$
Diffusion	$\exp(\beta_p\lambda_p)$	$\sum_{p=1}^{n} \exp(-\beta_p\lambda_p)\boldsymbol{u}_p\boldsymbol{u}_p^\top$
Polynomial	$\sum_{j=1}^{\eta-1} \beta_j\lambda_p^j + \epsilon$	$\sum_{p=1}^{n} \left(\sum_{j=1}^{\eta-1} \beta_j\lambda_p^j + \epsilon\right)^{-1}\boldsymbol{u}_p\boldsymbol{u}_p^\top$
Sum-of-Inverse Polynomials	$\left(\sum_{j=1}^{\eta-1}(\beta_j\lambda_p^j + \epsilon)^{-1}\right)^{-1}$	$\sum_{p=1}^{n} \left(\sum_{j=1}^{\eta-1}(\beta_j\lambda_p^j + \epsilon)^{-1}\right)\boldsymbol{u}_p\boldsymbol{u}_p^\top$

Why not BO on a graph

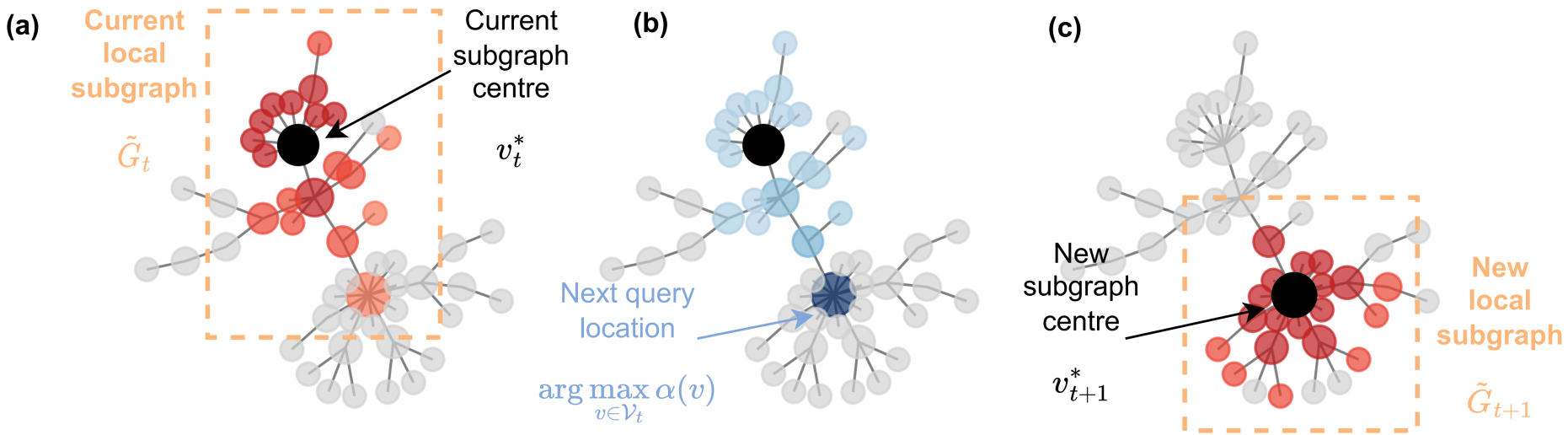
Previous work [2] investigated BO of functions on a single node.



Local modelling is required when the underlying graph is large.

BO of functions on a single node

The overall structure of Bayesian optimisation of functions defined on a single node (Wan et al., NeurIPS 2023).



BO of functions on a subset of nodes

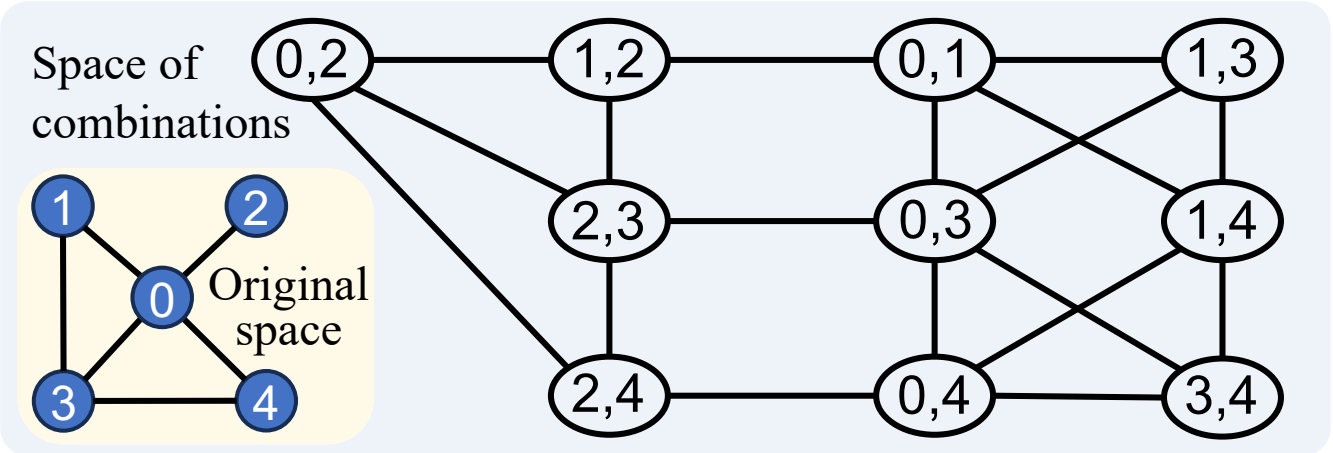
But how can we extend the settings to a subset of nodes?

The following questions need to be addressed:

1. The configuration is not just a node but a node combination.
2. The search space $\binom{|\mathcal{V}|}{k}$ is huge and inherits graph structural information, which requires local modelling with an effective approach to traverse the combinatorial space.
3. It is critical to maintain the flexibility of selecting distant nodes when considering a local context.

Methodology

The combinatorial graph of node combinations

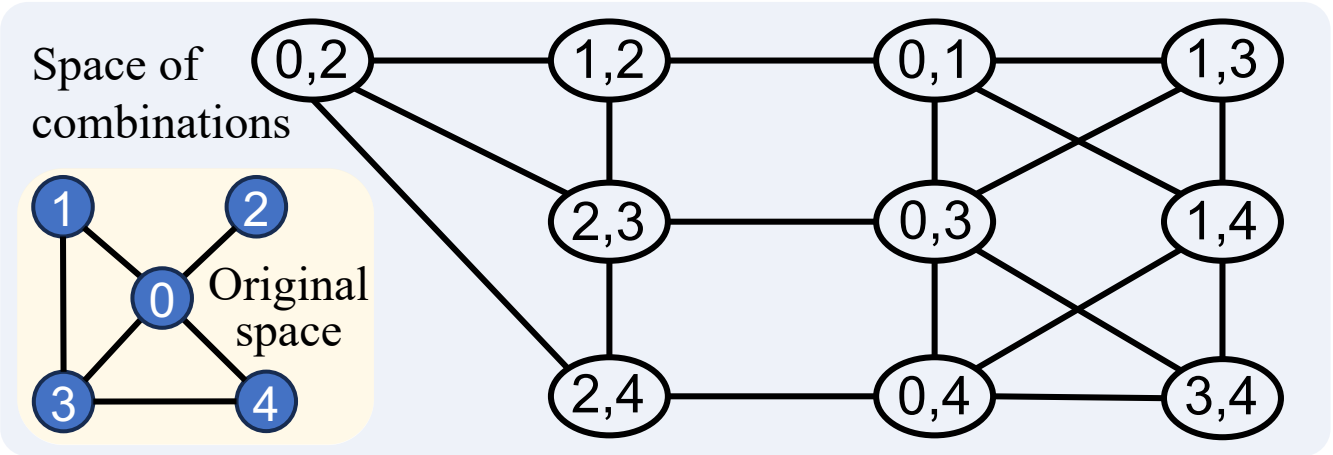


The combinatorial operation for combinations of k nodes on an underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is given by:

$$\hat{\mathcal{G}}^{<k>} = \square_{i=1}^{k} \mathcal{G}, \tag{3}$$

which leads to a combo-graph $\hat{\mathcal{G}}^{<k>} = \{\hat{\mathcal{V}}, \hat{\mathcal{E}}\}$ of size $|\hat{\mathcal{V}}| = \binom{|\mathcal{V}|}{k}$ with each combo-node $\hat{v}_i = (v_i^{(1)}, v_i^{(2)}, ..., v_i^{(k)}) \in \hat{\mathcal{V}}$ being a combination of k nodes from graph $\mathcal{G}$ without replacement.

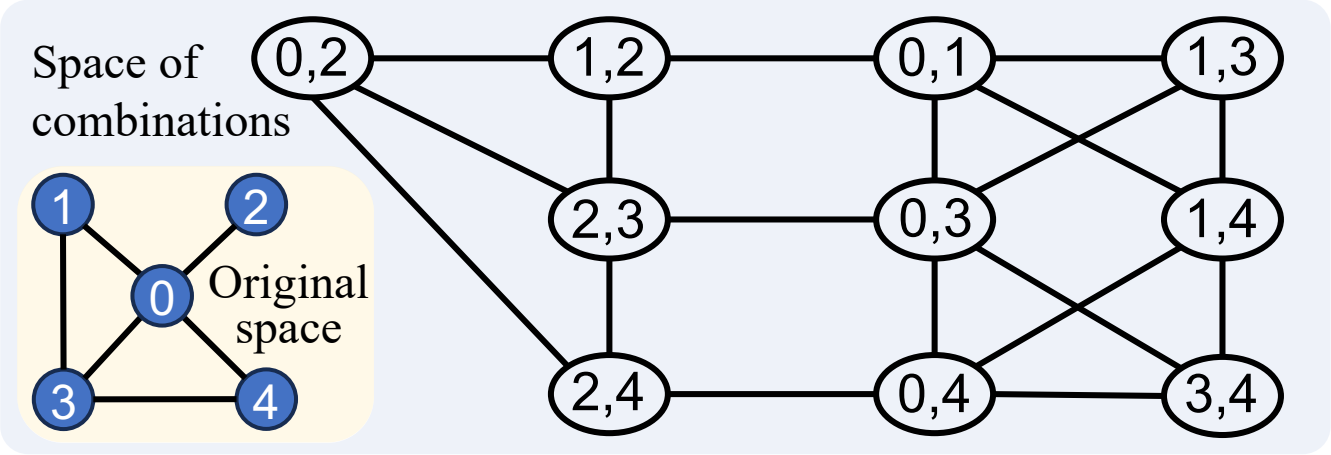
The combinatorial graph of node combinations



The combo-edges $\hat{\mathcal{E}}$ in the combo-graph are defined as follows:

Assume $\hat{v}_1 = (v_1^{(1)}, v_1^{(2)}, ..., v_1^{(k)})$ and $\hat{v}_2 = (v_2^{(1)}, v_2^{(2)}, ..., v_2^{(k)})$ are two arbitrary combo-nodes in the combo-graph $\mathcal{G}^{<k>}$, then the combo-edge $(\hat{v}_1, \hat{v}_2) \in \hat{\mathcal{E}}$ iff $\exists\, j$ such that $\forall\, i \neq j$, $v_1^{(i)} = v_2^{(i)}$ and $(v_1^{(j)}, v_2^{(j)}) \in \mathcal{E}$.

The combinatorial graph of node combinations

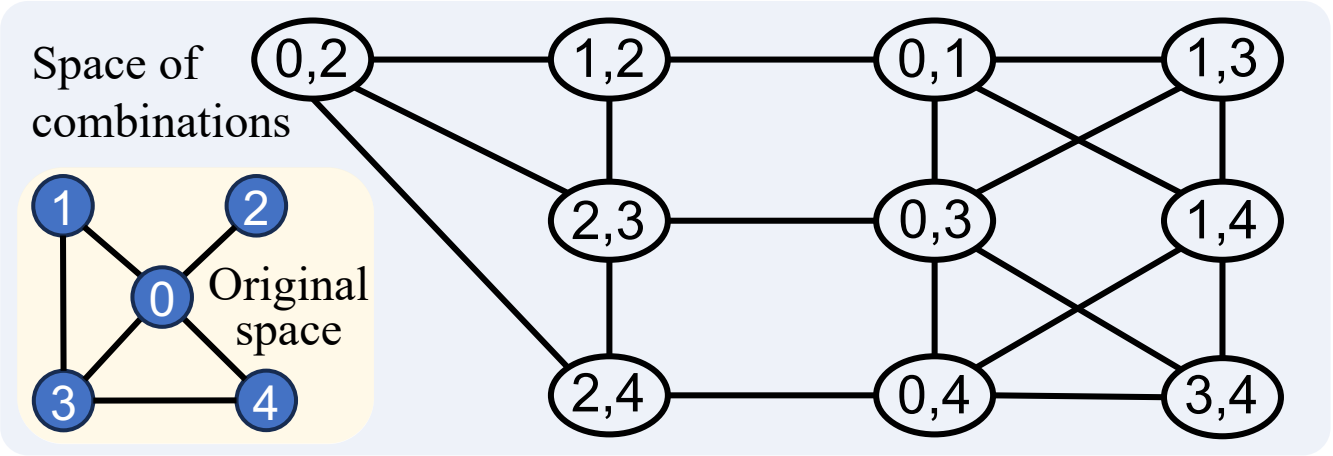


Note when $k = 1$, the problem reduces to the previous work [2] that uses BO for functions of a single node.

However, when k and $|\mathcal{V}|$ are relatively large, the combinatorial space could be huge (e.g. $\binom{1000}{8} \approx 2.4 \times 10^{19}$).

Local modelling is needed!

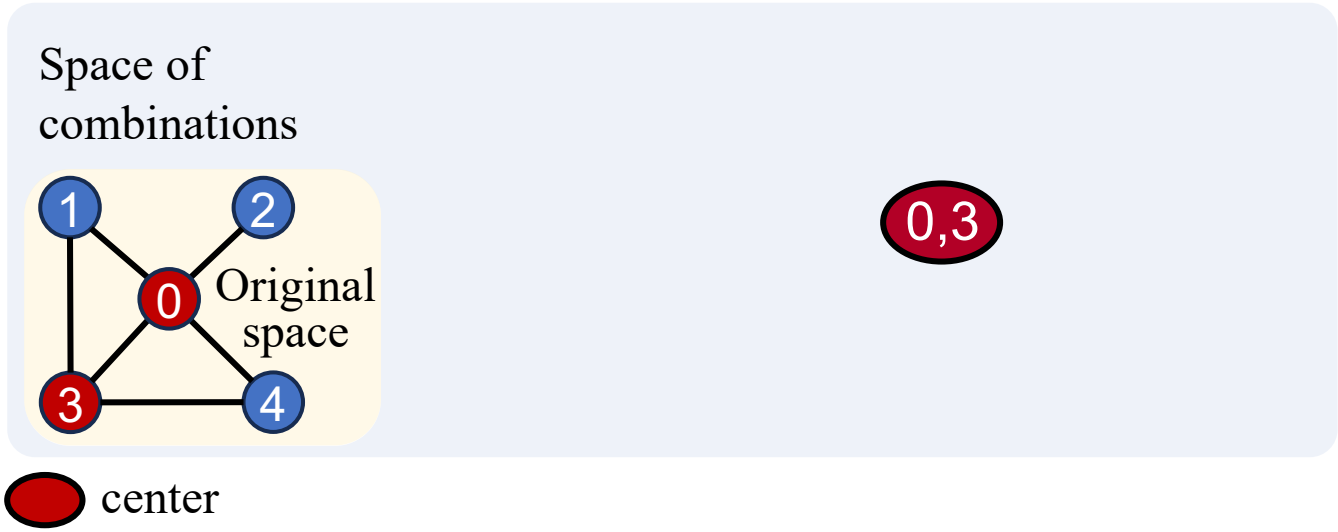
Properties of the Combo-graph



Property 1. In the proposed combo-graph, at most ℓ elements in the combination will be changed between any two combo-nodes that are ℓ-hop away.

Property 2. The degree of combo-node $\hat{v}_i$ increases linearly with k and is maximised at the combination of nodes with top k degrees: $\deg(\hat{v}_i) = \sum_{j=1}^{k} |\mathcal{N}(v_i^{(j)}) \setminus \{v_i^{(j')}\}_{j' \neq j}^{k}|$.

Recursive combo-subgraph sampling

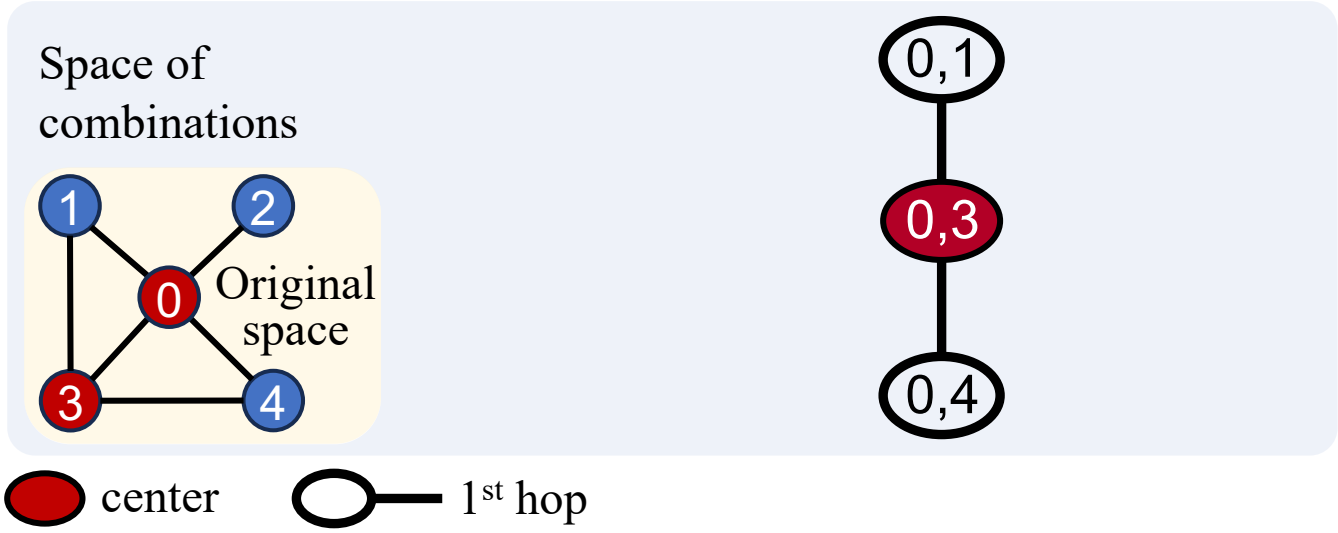


Goal: Construct an ego-subgraph at a given centre $\hat{v}^*$ of size Q with maximum hop $\ell_{\max}$ from the underlying graph $\mathcal{G}$.

Algorithm:

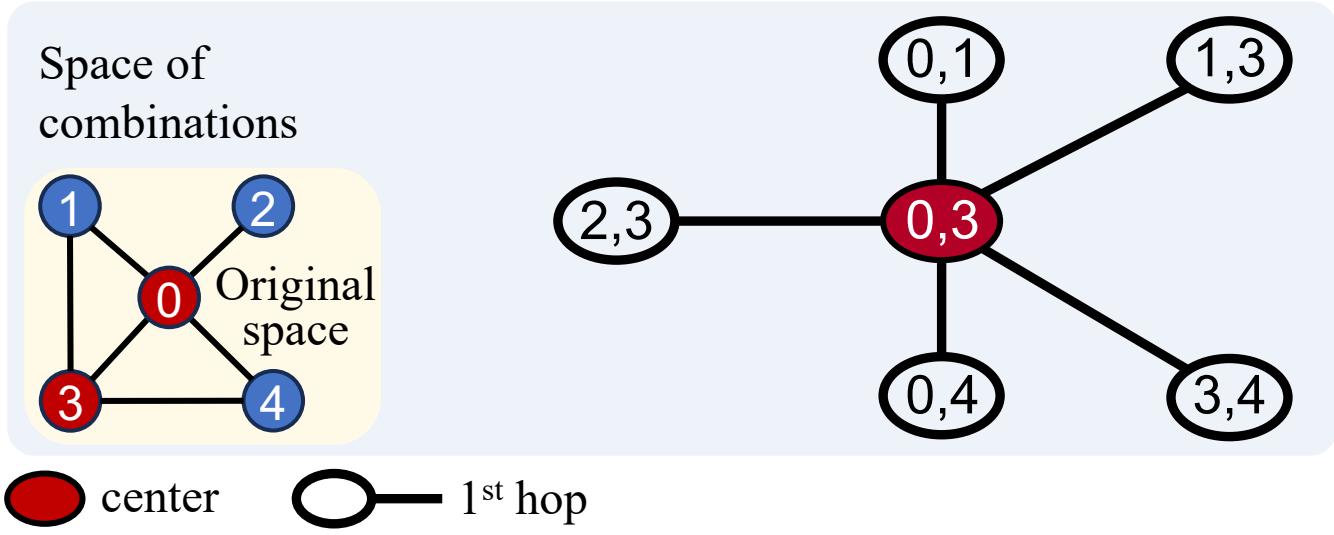
1. Initialise $\tilde{\mathcal{G}} = \{\tilde{\mathcal{V}}, \tilde{\mathcal{E}}\}$ with only $\hat{v}^*$ and then loop through neighbours of each element node $v \in \hat{v}^*$.

Recursive combo-subgraph sampling



2. If a neighbour $\mathcal{N}_i(v)$ of v is not in $\hat{v}^*$, create a new combo-node $\hat{v}'$ by substituting $\mathcal{N}_i(v)$ with v in $\hat{v}^*$, and accordingly create a combo-edge by connecting $\hat{v}'$ to $\hat{v}^*$.

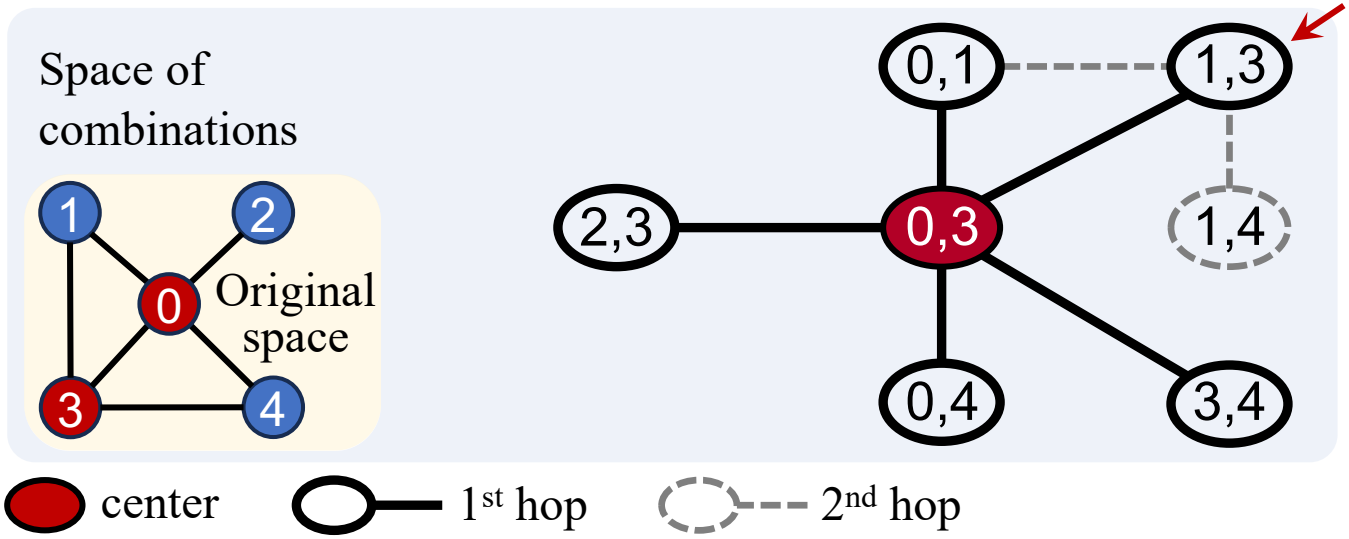
Recursive combo-subgraph sampling



2. If a neighbour $\mathcal{N}_i(v)$ of v is not in $\hat{v}^*$, create a new combo-node $\hat{v}'$ by substituting $\mathcal{N}_i(v)$ with v in $\hat{v}^*$, and accordingly create a combo-edge by connecting $\hat{v}'$ to $\hat{v}^*$.

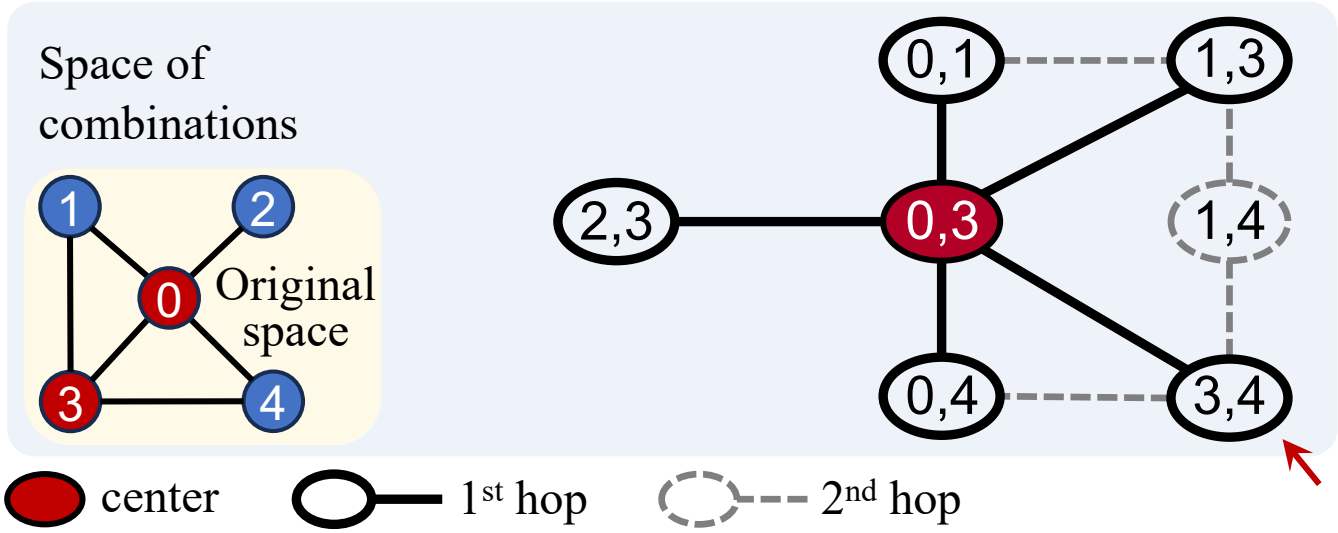
Remark: After finding the combo-neighbours of $\hat{v}^*$ at hop $\ell = 1$, $\tilde{\mathcal{G}}$ becomes a star-network at centre $\hat{v}^*$.

Recursive combo-subgraph sampling



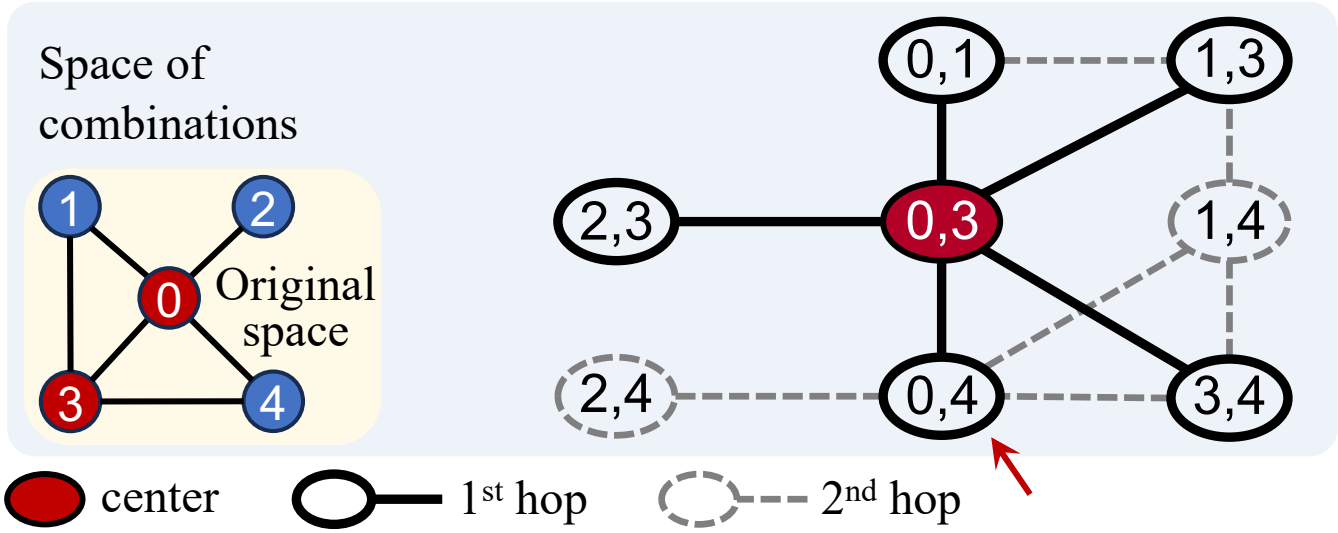
3. Repeat the above star-sampling procedures for every newly found combo-node to find their combo-neighbours at hop $\ell + 1$, which meanwhile also finds the edges among combo-nodes within hop ℓ.

Recursive combo-subgraph sampling



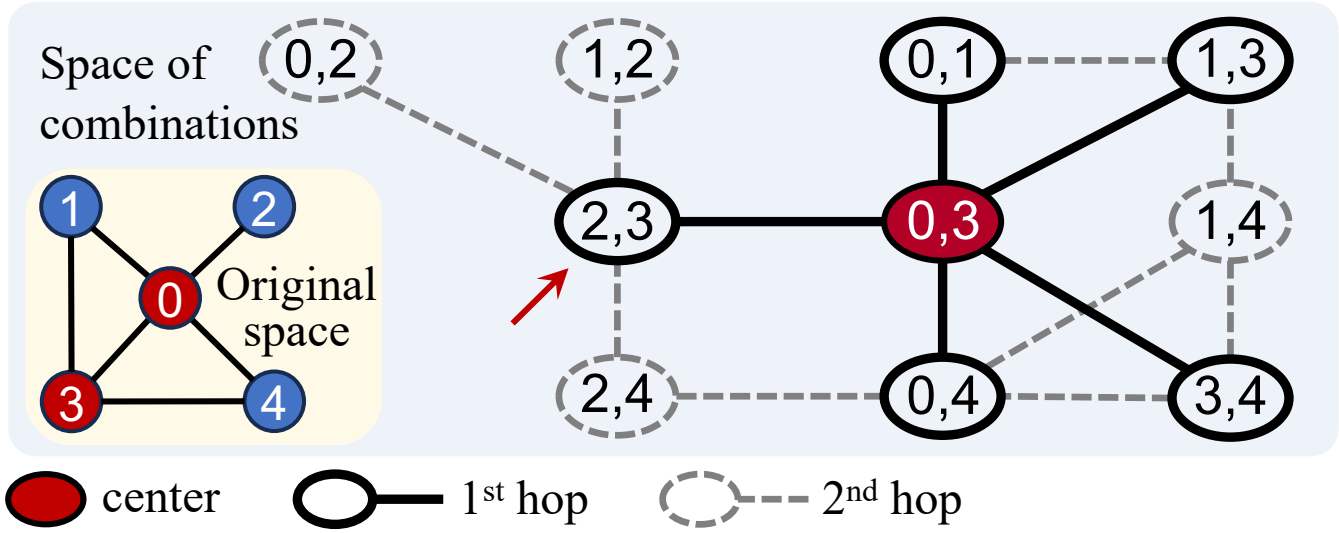
3. Repeat the above star-sampling procedures for every newly found combo-node to find their combo-neighbours at hop $\ell + 1$, which meanwhile also finds the edges among combo-nodes within hop ℓ.

Recursive combo-subgraph sampling



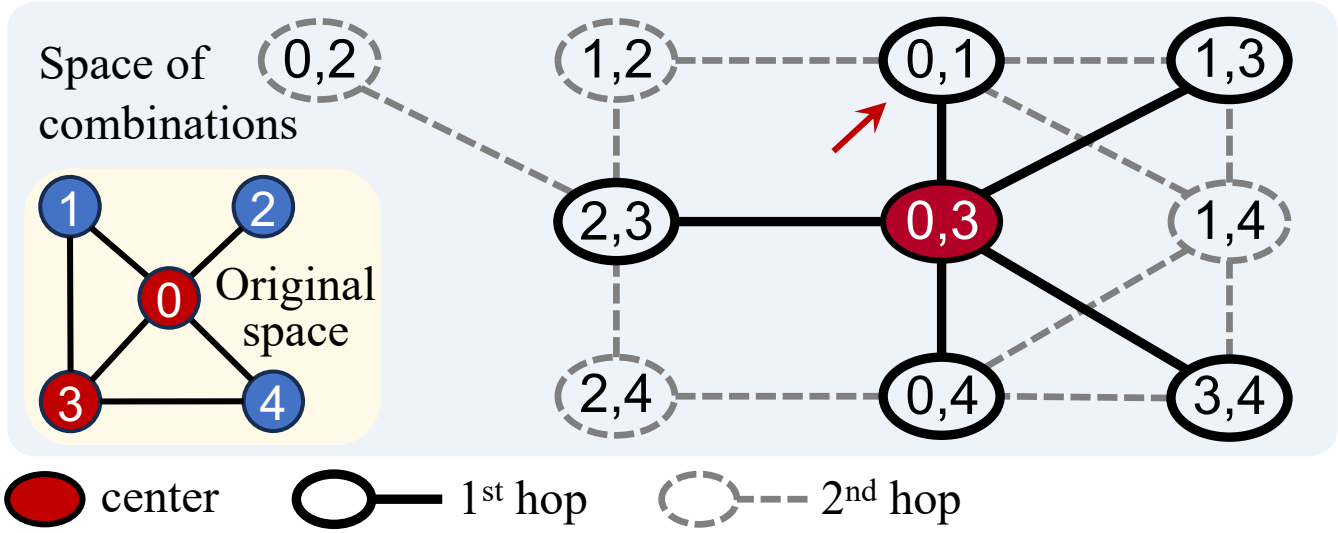
3. Repeat the above star-sampling procedures for every newly found combo-node to find their combo-neighbours at hop $\ell + 1$, which meanwhile also finds the edges among combo-nodes within hop ℓ.

Recursive combo-subgraph sampling



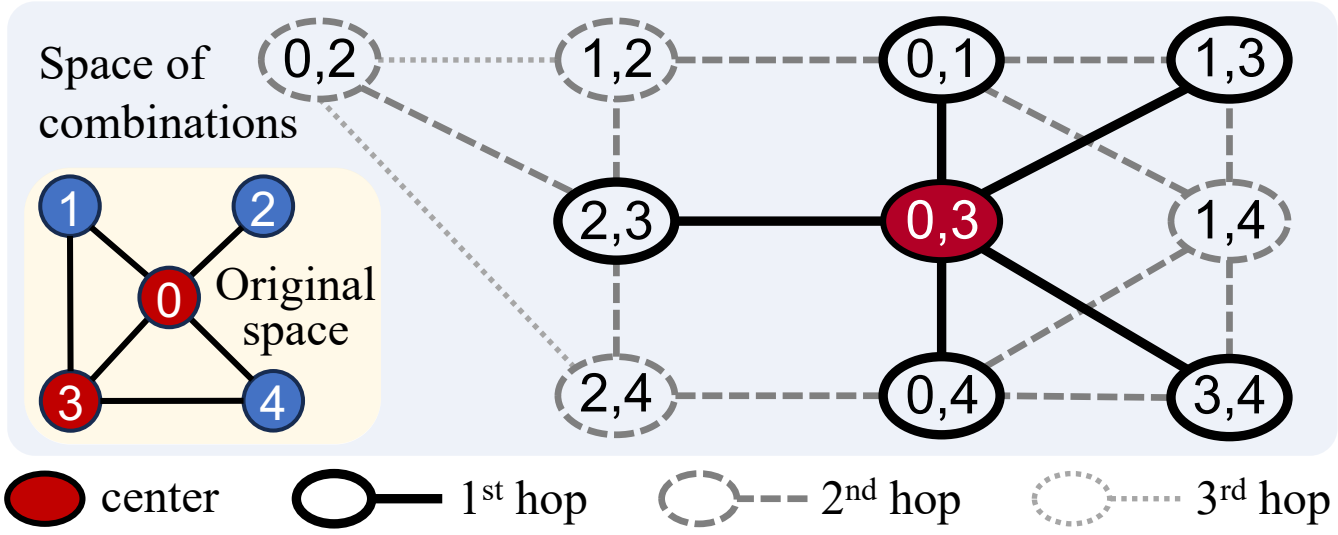
3. Repeat the above star-sampling procedures for every newly found combo-node to find their combo-neighbours at hop $\ell + 1$, which meanwhile also finds the edges among combo-nodes within hop ℓ.

Recursive combo-subgraph sampling



3. Repeat the above star-sampling procedures for every newly found combo-node to find their combo-neighbours at hop $\ell + 1$, which meanwhile also finds the edges among combo-nodes within hop ℓ.

Recursive combo-subgraph sampling

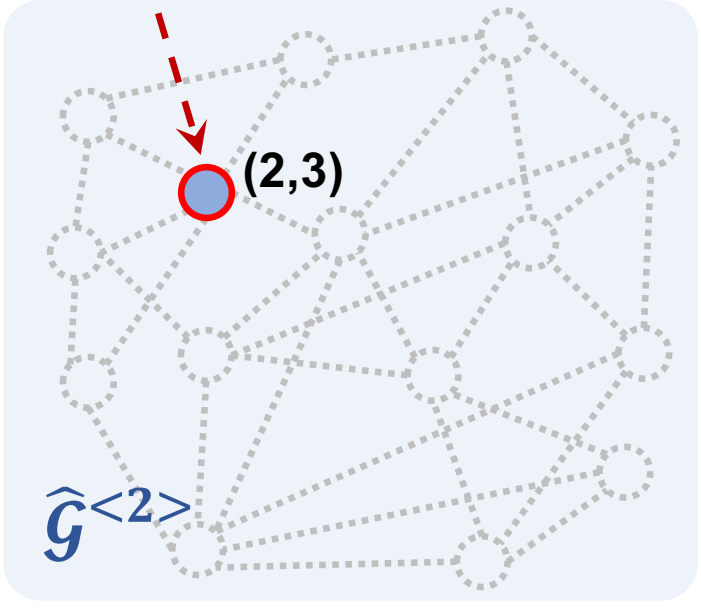


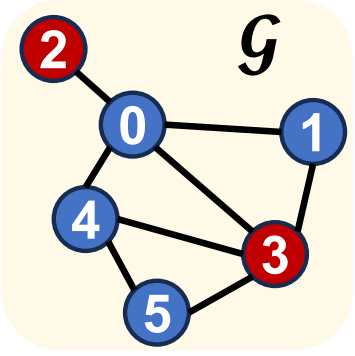
4. Stop the algorithm until the subgraph size limit Q or the maximum hop $\ell_{\max}$ is reached.

(t-1)

Initial combo-node $\hat{v}^*_{t-1}$ from restart or last query

Combinatorial Space

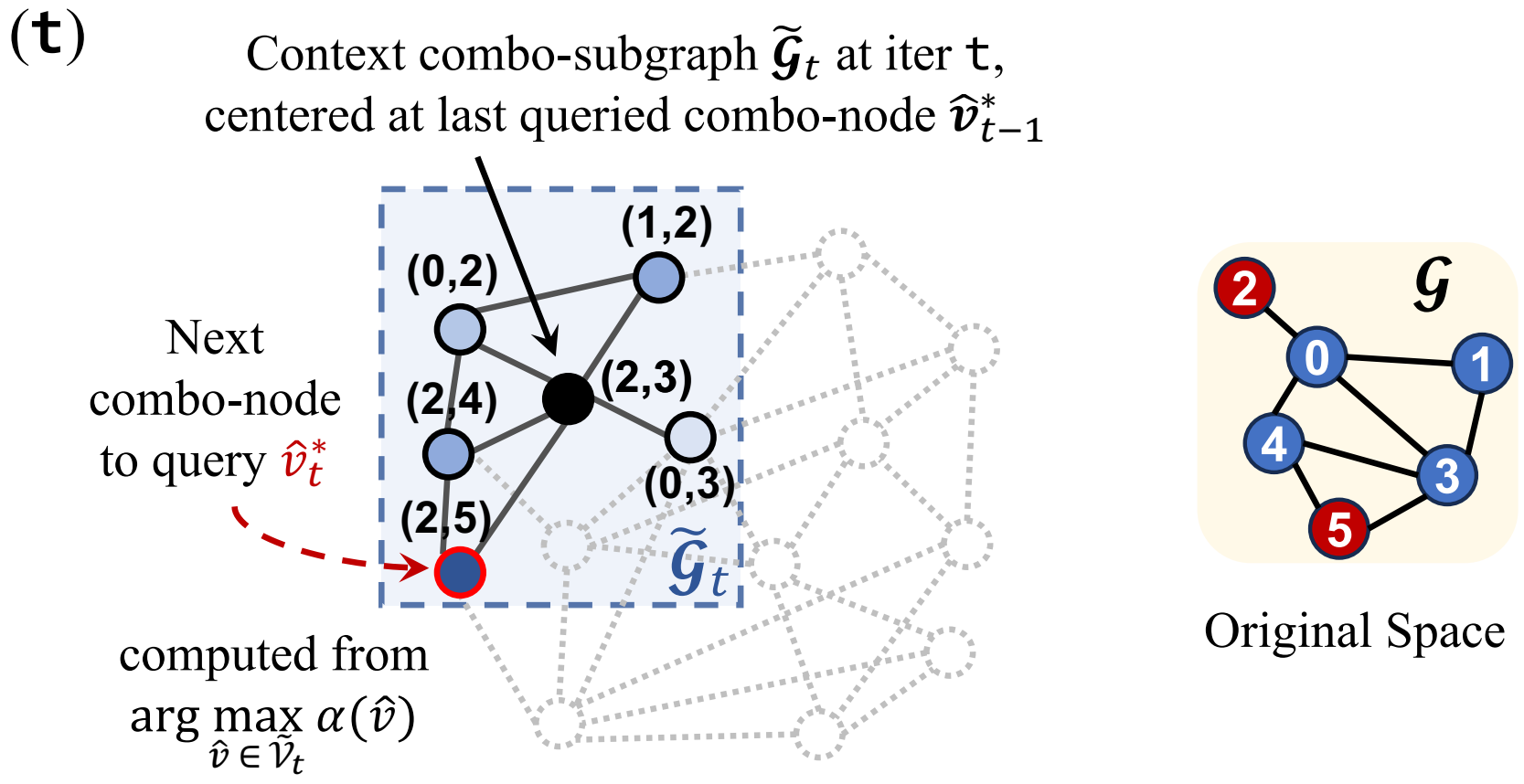




Original Space

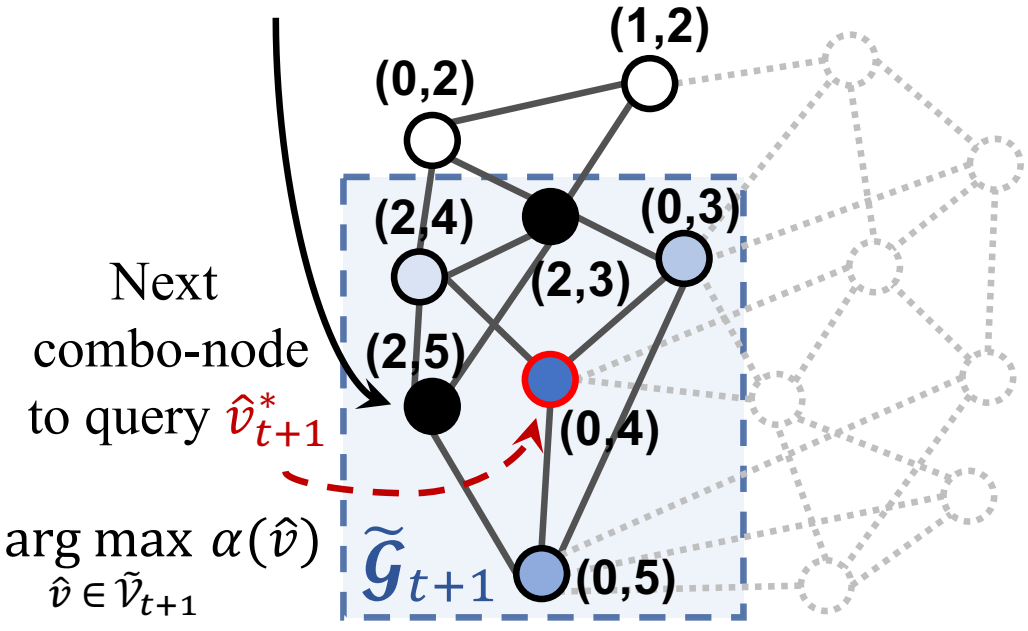
unexplored structure | explored structure | queried combo-node

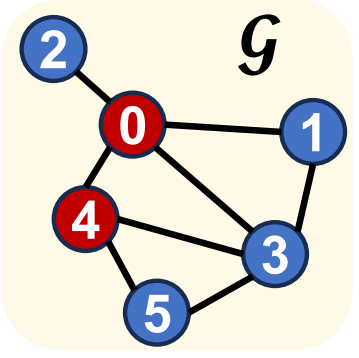
surrogate prediction | next query


(t)
Context combo-subgraph $\widetilde{\mathcal{G}}_t$ at iter t,
centered at last queried combo-node $\hat{v}^*_{t-1}$
(1,2)
(0,2)
(2,3)
(2,4)
(0,3)
(2,5)
$\widetilde{\mathcal{G}}_t$
Next
combo-node
to query $\hat{v}^*_t$
computed from
$\arg\max_{\hat{v} \in \tilde{\mathcal{V}}_t} \alpha(\hat{v})$
$\mathcal{G}$
Original Space
unexplored structure
explored structure
queried combo-node
surrogate prediction
next query

(t+1)

$\widetilde{\mathcal{G}}_{t+1}$ centered at last queried combo-node $\hat{v}_t^*$





Original Space

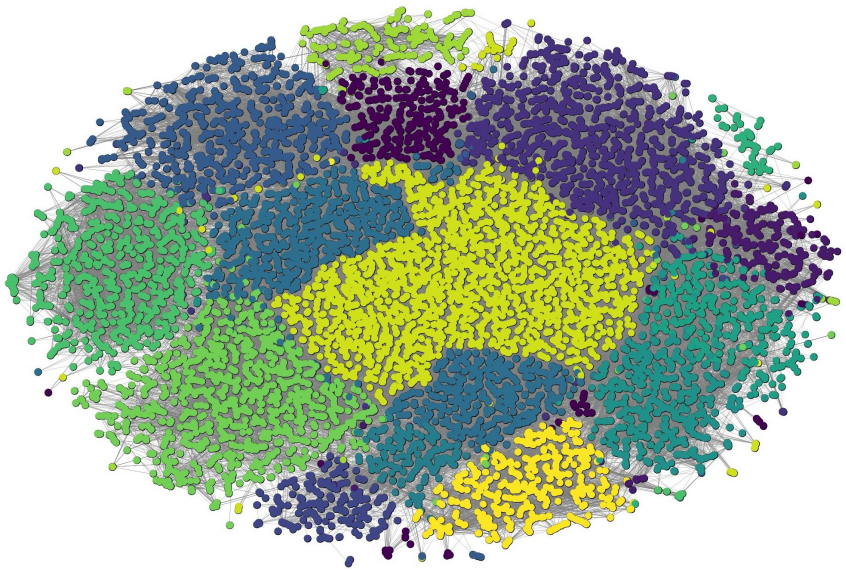
unexplored structure | explored structure | queried combo-node

surrogate prediction | next query

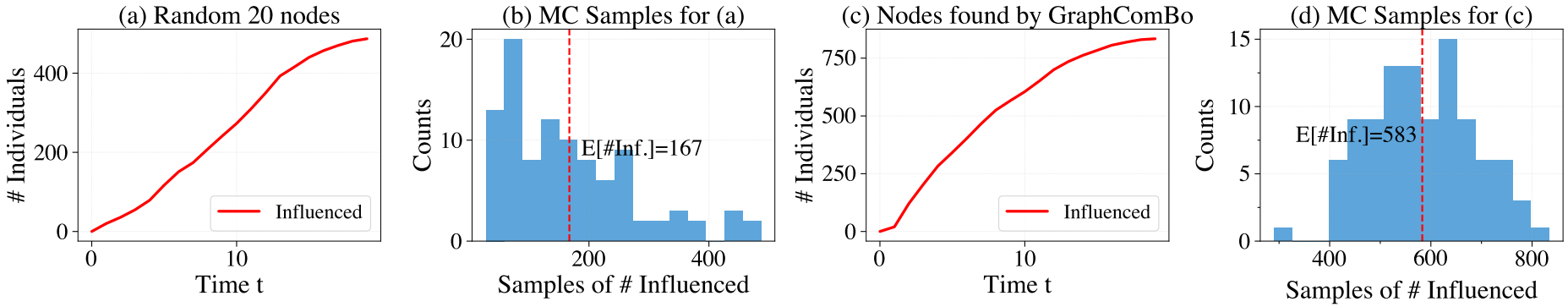
Real-world Experiments

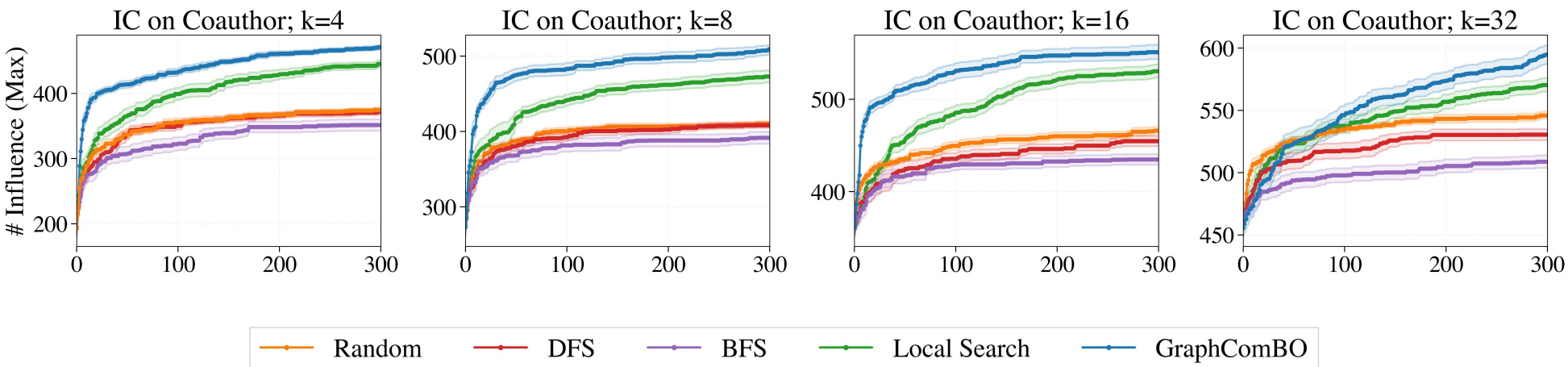
Independent cascading

- Each node can be either activated or inactivated at each step.
- Initializing with only k nodes activated.
- When node v becomes active, it is given a single chance to activate each inactive neighbour w with probability p.
- The process runs until no more activation is possible.



Influence maximisation with independent cascading





Thank You ☺

[1] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 144–158. Springer, 2003.

[2] Xingchen Wan, Pierre Osselin, Henry Kenlay, Binxin Ru, Michael A Osborne, and Xiaowen Dong. Bayesian optimisation of functions on graphs. *Advances in Neural Information Processing Systems*, 2023.