Algorithms for Burning Graph Families

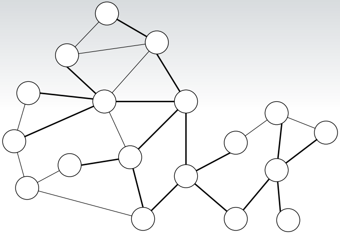
Shahin Kamali

August 6, 2021

Graph Searching in Canada (GRASCan) Workshop

Graph Burning Problem

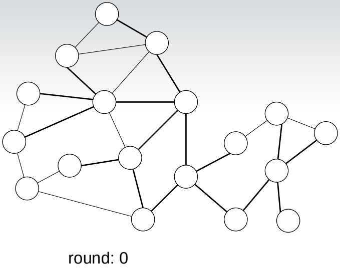
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].



round: 0

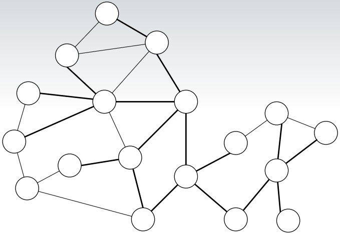
Graph Burning Problem

- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].



Graph Burning Problem

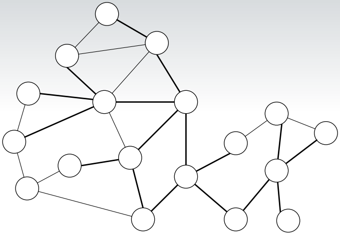
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



round: 0

Graph Burning Problem

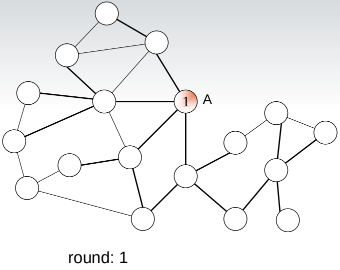
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



round: 0

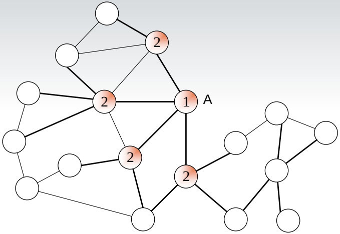
Graph Burning Problem

- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



Graph Burning Problem

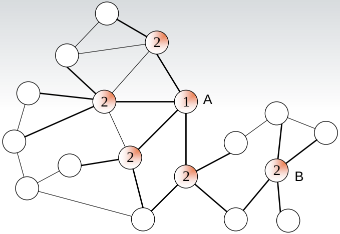
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



round: 2

Graph Burning Problem

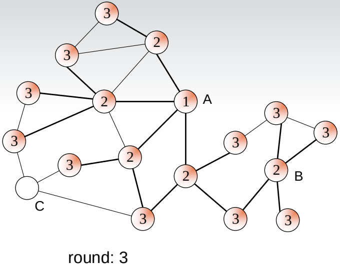
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



round: 2

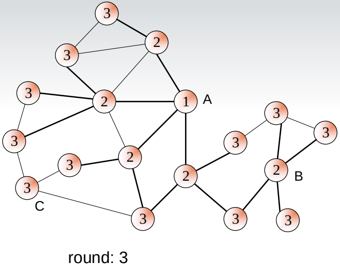
Graph Burning Problem

- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.



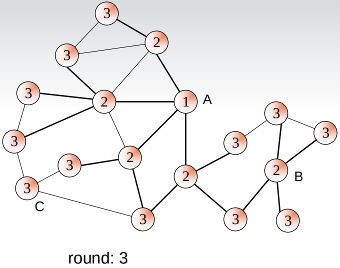
Graph Burning Problem

- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.
  - The burning completes when all vertices are on fire.



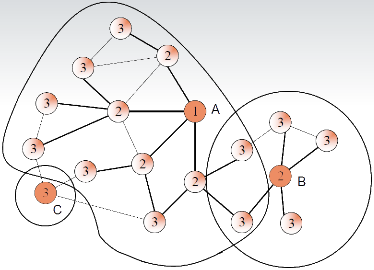
Graph Burning Problem

- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.
  - The burning completes when all vertices are on fire.
- Decision problem:
  - Can we burn G in k rounds?



Graph Burning Problem

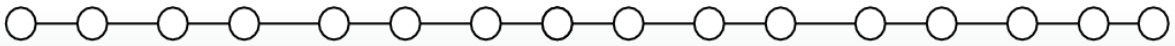
- Given an undirected graph G, the goal is to **burn** in a minimum number of **rounds** [Bonato et al., 2014].
- At each given round:
  - A new fire can be initiated at any vertex.
  - The existing fires expand to their neighboring vertices.
  - The burning completes when all vertices are on fire.



- Decision problem:
  - Can we burn G in k rounds?
  - Equivalently, can we cover the graph with "disks" of radii $0, 1, 2, \ldots, k-1$?

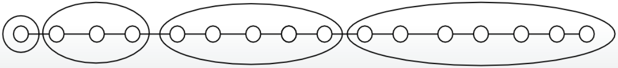
Burning Paths

- A path P_n of length n can be covered with disks of radii $0, 1, 2, \ldots, \lceil \sqrt{n} \rceil$ [Bonato et al. 2014].



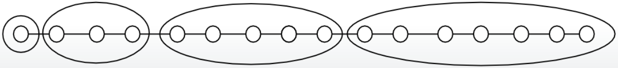
Burning Paths

- A path P_n of length n can be covered with disks of radii $0, 1, 2, \ldots, \lceil \sqrt{n} \rceil$ [Bonato et al. 2014].



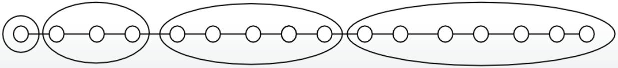
Burning Paths

- A path P_n of length n can be covered with disks of radii $0, 1, 2, \ldots, \lceil \sqrt{n} \rceil$ [Bonato et al. 2014].



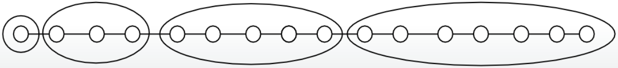
Burning Paths

- A path P_n of length n can be covered with disks of radii $0, 1, 2, \ldots, \lceil \sqrt{n} \rceil$ [Bonato et al. 2014].
- **The burning graph conjecture:** The burning number of any connected graph is at most $\lceil \sqrt{n} \rceil$ [Bonato et al. 2014].



Burning Paths

- A path P_n of length n can be covered with disks of radii $0, 1, 2, \ldots, \lceil\sqrt{n}\rceil$ [Bonato et al. 2014].
- **The burning graph conjecture:** The burning number of any connected graph is at most $\lceil\sqrt{n}\rceil$ [Bonato et al. 2014].
  - The burning number of any connected graph is at most $\frac{\sqrt{6}}{2}\sqrt{n} \approx 1.22\sqrt{n}$ [Land and Lu, 2016].



Computational Complexity

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
  - Reduction from 3-Partition problem (an extension of 2-partition problem to 3 set).

Computational Complexity

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
  - Reduction from 3-Partition problem (an extension of 2-partition problem to 3 set).
  - The problem remains NP-hard for disjoint set of paths, trees, other graph families.

Computational Complexity

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
  - Reduction from 3-Partition problem (an extension of 2-partition problem to 3 set).
  - The problem remains NP-hard for disjoint set of paths, trees, other graph families.
  - The problem is more "interesting" when the underlying graphs are sparse.

Computational Complexity

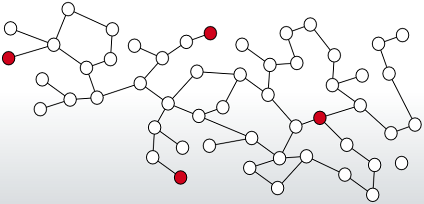
- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
  - Reduction from 3-Partition problem (an extension of 2-partition problem to 3 set).
  - The problem remains NP-hard for disjoint set of paths, trees, other graph families.
  - The problem is more "interesting" when the underlying graphs are sparse.
- It is claimed that the problem is APX-hard [Mondal et al., 2021] (no $(1+\epsilon)$-approximation exists assuming $P \neq NP$).

Approximation Algorithms

- If there are r vertices of pairwise distance $\geq 2r - 1$ in a graph G, then G cannot be burned in less than r rounds.

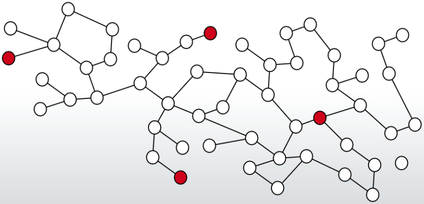
Approximation Algorithms

- If there are r vertices of pairwise distance $\geq 2r - 1$ in a graph G, then G cannot be burned in less than r rounds.



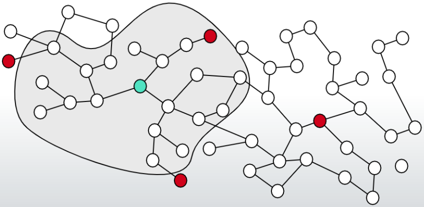
Approximation Algorithms

- If there are r vertices of pairwise distance $\geq 2r - 1$ in a graph G, then G cannot be burned in less than r rounds.
- Example: suppose there are $r = 4$ vertices of pairwise $2r - 1 = 7$ in a graph G.
  - It is not possible to cover G with 3 disks of radii 3.
  - Therefore it is not possible to cover G with 3 disks of radii $0, 1, 2$.



Approximation Algorithms

- If there are r vertices of pairwise distance $\geq 2r - 1$ in a graph G, then G cannot be burned in less than r rounds.
- Example: suppose there are $r = 4$ vertices of pairwise $2r - 1 = 7$ in a graph G.
  - It is not possible to cover G with 3 disks of radii 3.
  - Therefore it is not possible to cover G with 3 disks of radii $0, 1, 2$.



Constant Approximation Algorithm

- Define a procedure Burn-Guess(G,g) which returns:
  - Either a schedule that completes burning in at most $3g - 3$ rounds.
  - Or 'Bad-Guess', which guarantees burning cannot be complete in $g - 1$ rounds.

Constant Approximation Algorithm

- Define a procedure Burn-Guess(G,g) which returns:
  - Either a schedule that completes burning in at most $3g - 3$ rounds.
  - Or 'Bad-Guess', which guarantees burning cannot be complete in $g - 1$ rounds.
- The smallest value of g^* for which Burn-Guess returns a schedule gives a burning scheme that completes in $3g^* - 3$ while the optimal schedule will require $g^* - 1$ rounds to complete.
  - Approximation ratio of at most 3.

Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".

Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.

Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.

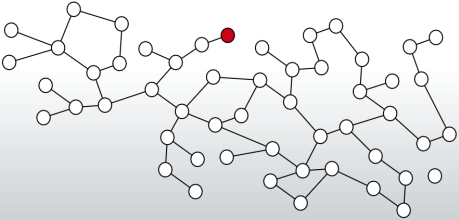
- E.g., here $g = 4$ and later we look at $g = 5$.

Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
- E.g., here $g=4$ and later we look at $g=5$.

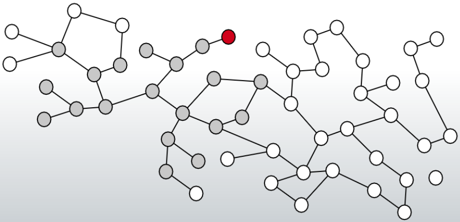
Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

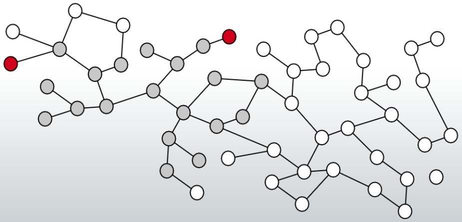
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

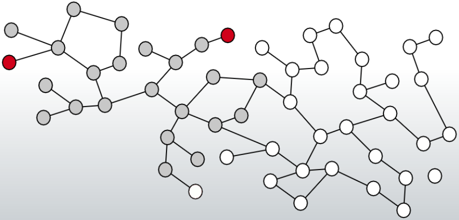
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.

- E.g., here $g=4$ and later we look at $g=5$.



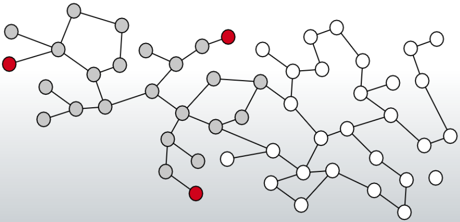
Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

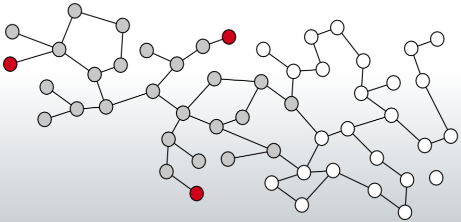
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

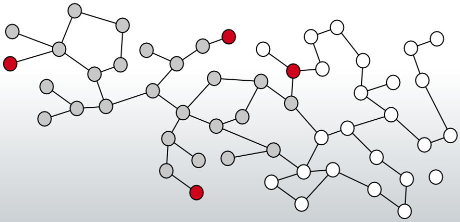
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.

- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

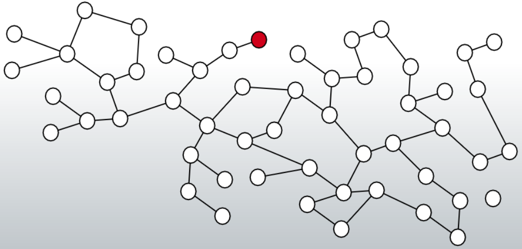
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
- E.g., here $g = 4$ and later we look at $g = 5$.



return
BAD-Guess

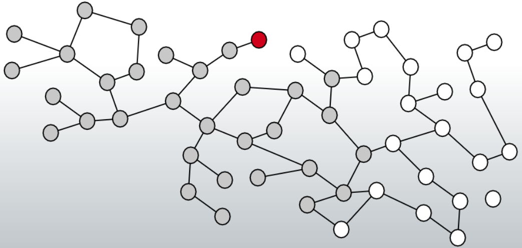
Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

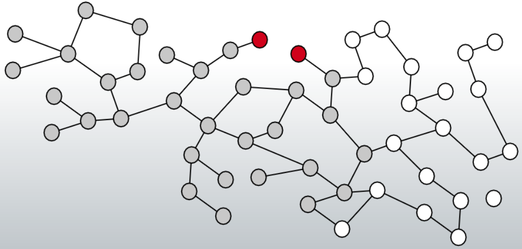
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
- E.g., here $g=4$ and later we look at $g=5$.



Burn-Guess Process

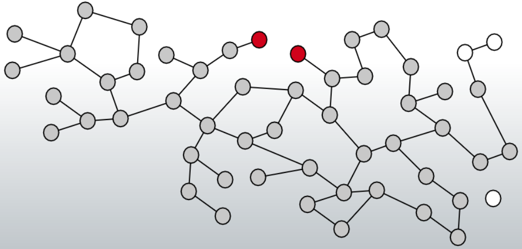
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.

- E.g., here $g = 4$ and later we look at $g = 5$.



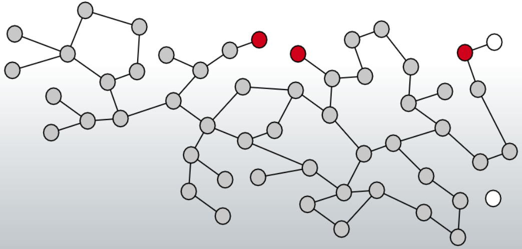
Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g-2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
- E.g., here $g=4$ and later we look at $g=5$.



Burn-Guess Process

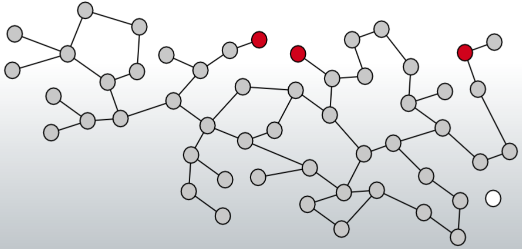
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

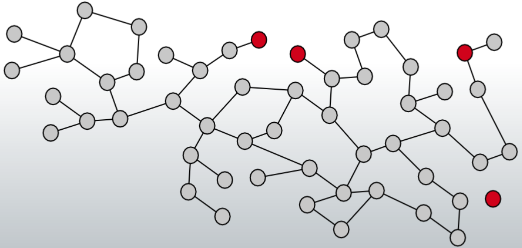
- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.

- E.g., here $g = 4$ and later we look at $g = 5$.



Burn-Guess Process

- Initially empty sets S of "centers" and L of "labeled vertices".
- Take an arbitrary unlabeled vertex u, add it to S and add all unlabeled vertices within distance $2g - 2$ of u to L.
  - If the number of centers becomes g, then return Bad-Guess.
  - If all vertices are added to L, return an arbitrary ordering of centers as the burning scheme (which completes in at most $(g-1) + (2g-2) = 3g - 3$ rounds).
- E.g., here $g = 4$ and later we look at $g = 5$.



General Graph Summary

Theorem

There is a polynomial algorithm with approximation ratio of 3 *for burning any graph* $G = (V, E)$ *[Bonato & S.K., 2019].*

- What about graph families? can we get better approximation ratio for families of graphs?

Burning Trees

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].

Burning Trees

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
- It is possible to achieve an approximation factor of 2.

Burning Trees

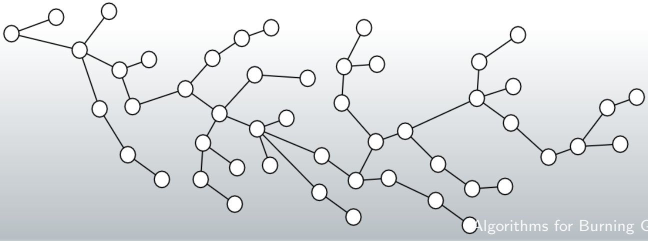
- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
- It is possible to achieve an approximation factor of 2.
- Burn-Guess-Tree (τ, g) returns either a schedule that completes in at most $2g - 2$ rounds or 'Bad-Guess', which means burning cannot complete in $g - 1$ rounds.

Burning Trees

- Finding the optimal schedule is NP-hard [Bessy et al., 2017].
- It is possible to achieve an approximation factor of 2.
- Burn-Guess-Tree (τ, g) returns either a schedule that completes in at most $2g - 2$ rounds or 'Bad-Guess', which means burning cannot complete in $g - 1$ rounds.
- An approximation factor of at most 2 is guaranteed:
  - The schedule returned by the smallest value of $g = g^*$ completes in $2g^* - 2$ rounds.
  - For $g^* - 1$, Bad-Guess is returned, which implies that the optimal scheme requires at least $g^* - 1$ rounds.

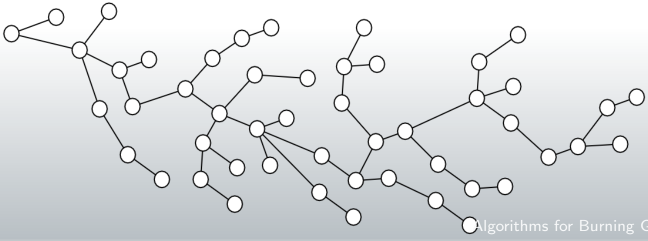
Trees

- Burn-Guess-Tree treats τ as a rooted tree:



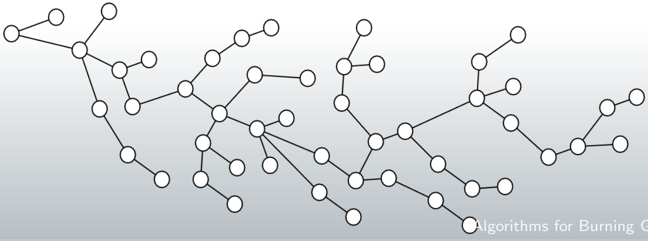
Trees

- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.



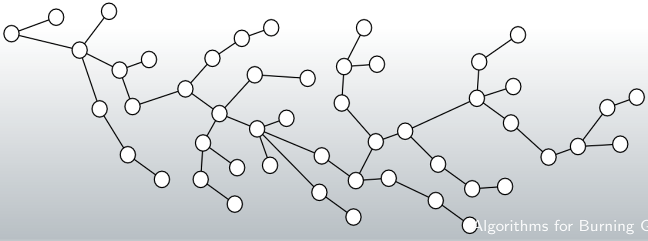
Trees

- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.



Trees

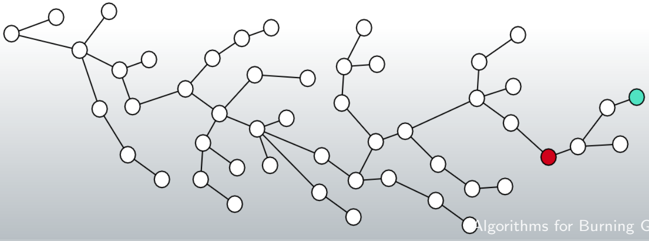
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of “terminals”, C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Trees

- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.

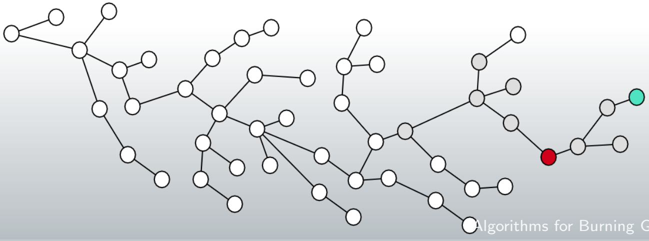
  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Trees

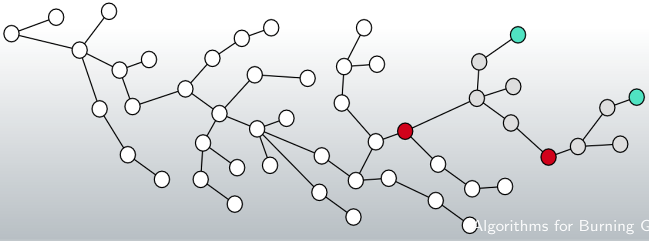
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.

  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Trees

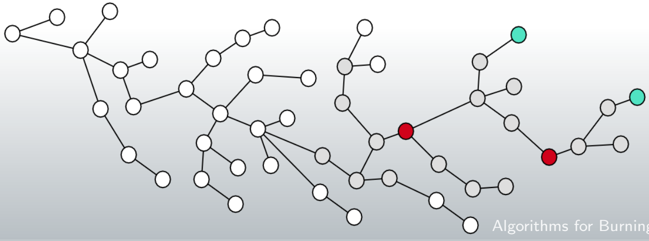
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Trees

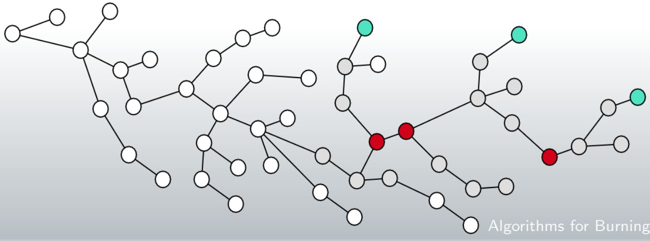
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.

  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



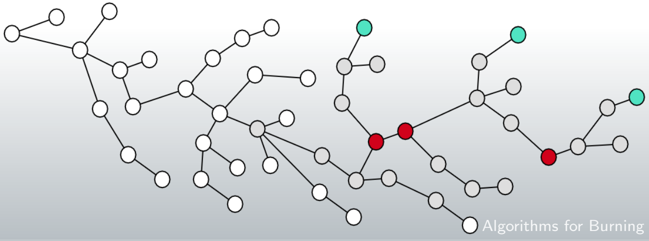
Trees

- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - Here, $g=4$ returns Bad-Guess and $g=5$ returns a schedule.



Trees

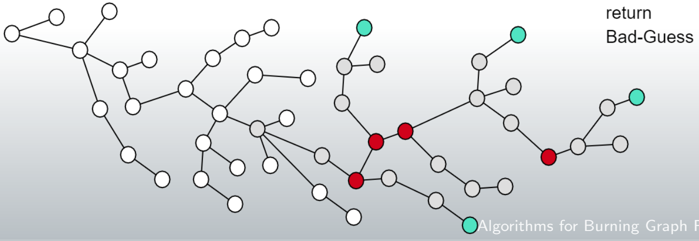
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - Here, $g=4$ returns Bad-Guess and $g=5$ returns a schedule.



Trees

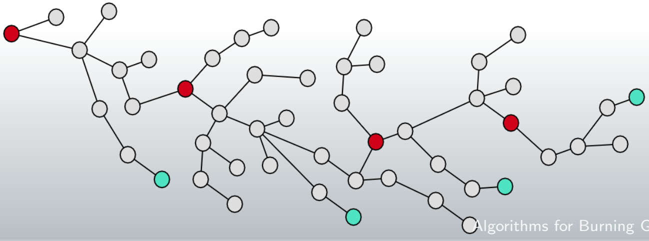
- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - When $|T| = g$, return Bad-Guess.

  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Trees

- Burn-Guess-Tree treats τ as a rooted tree:
  - Maintain sets T of "terminals", C of centers, and L of labeled vertices.
  - Take the deepest unlabeled node x, add x to T.
  - let p be the $(g-1)$-ancestor of x; add p to C and add all nodes within distance $g-1$ of p to L.
  - When $|T| = g$, return Bad-Guess.
  - When all vertices are labeled, return any ordering of C as the burning schedule. All nodes are within distance $g-1$ of g centers.
  - Here, $g = 4$ returns Bad-Guess and $g = 5$ returns a schedule.



Burning Trees Summary

Theorem

There is a polynomial algorithm with approximation ratio of 2 *for burning any tree [Bonato & S.K., 2019].*

- Open question: what is the best approximation factor attainable for trees? is it possible to get an PTAS (with approximation factor $1+\epsilon$)?

Burning Cacti

- It is possible to achieve an approximation factor of 2.75.

Burning Cacti

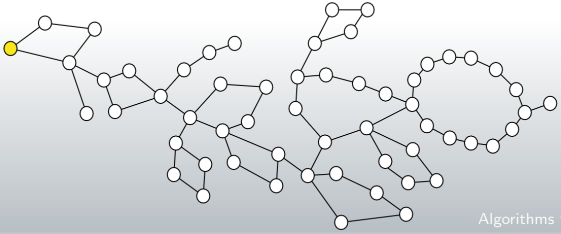
- It is possible to achieve an approximation factor of 2.75.
- Burn-Guess-Cactus(C,g) returns either a schedule that completes in at most $2.75g - 2$ rounds or 'Bad-Guess', which means burning cannot complete in $g - 1$ rounds.

Burning Cacti

- It is possible to achieve an approximation factor of 2.75.
- Burn-Guess-Cactus(C,g) returns either a schedule that completes in at most $2.75g - 2$ rounds or 'Bad-Guess', which means burning cannot complete in $g - 1$ rounds.
- Therefore, an approximation factor of at most 2.75 is guaranteed.
  - The schedule returned by the smallest value of $g = g^*$ completes in $2.75g^* - 2$ rounds.
  - For $g^* - 1$, Bad-Guess is returned, which implies that the optimal scheme requires at least $g^* - 1$ rounds.

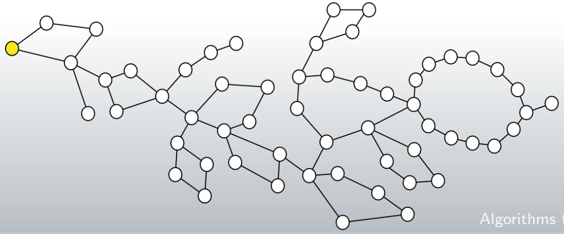
Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.



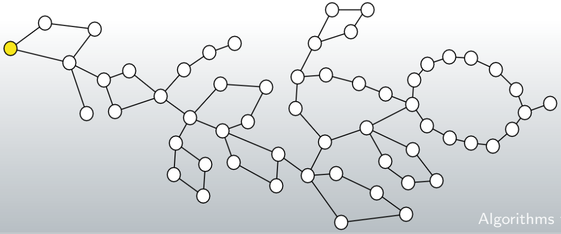
Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.



Burning Cacti

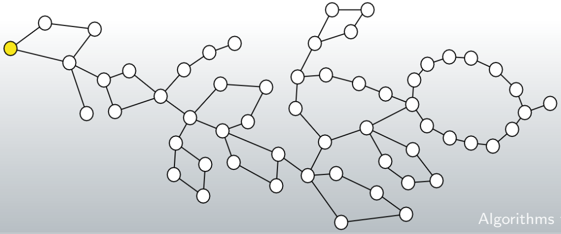
- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g - 1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g - 1$ of the path to L.

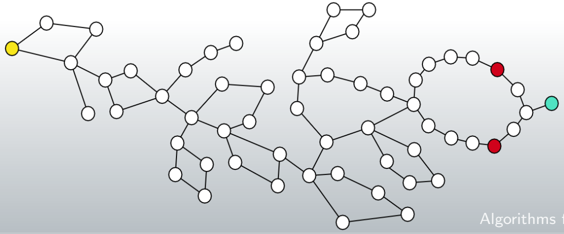
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

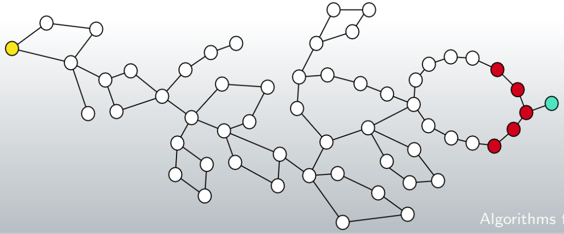
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

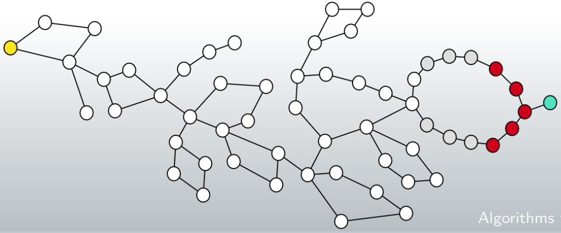
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

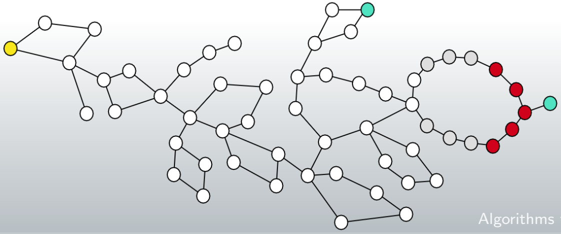
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

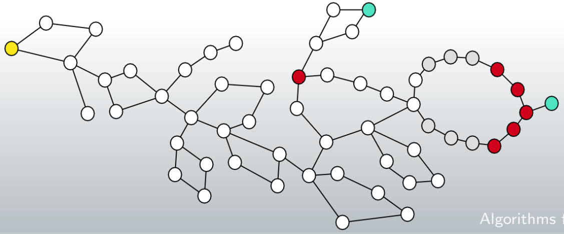
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

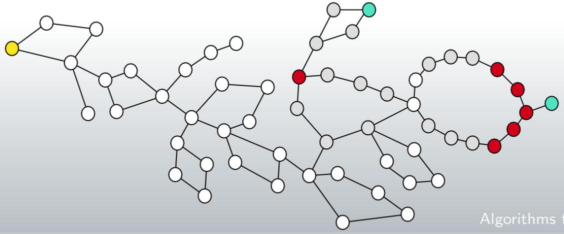
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

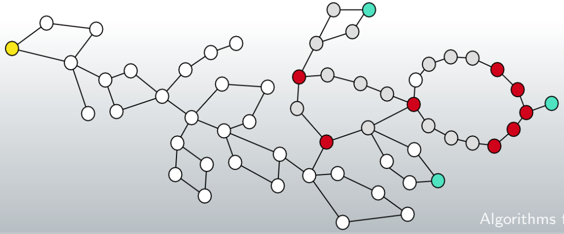
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

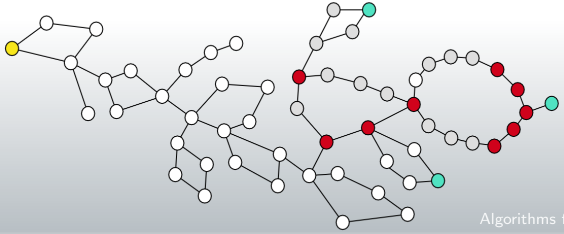
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

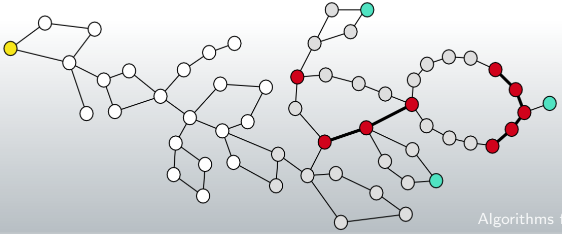
  - Here, we first look at $g = 4$ and then $g = 5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.

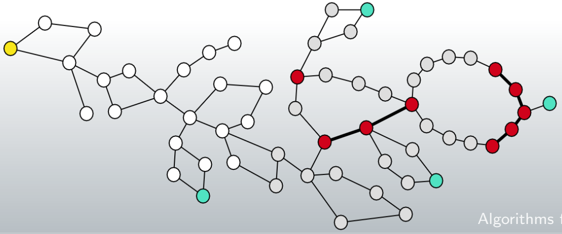
  - Here, we first look at $g=4$ and then $g=5$.



Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g-1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g-1$ of the path to L.
  - When $|T| = g$, return Bad-Guess.

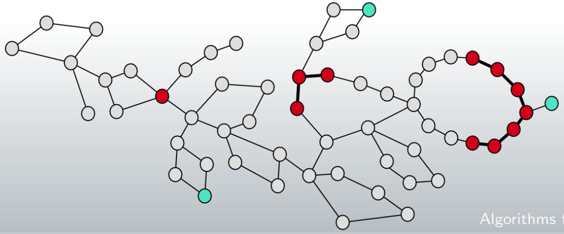
  - Here, we first look at $g = 4$ and then $g = 5$.



return Bad-Guess.

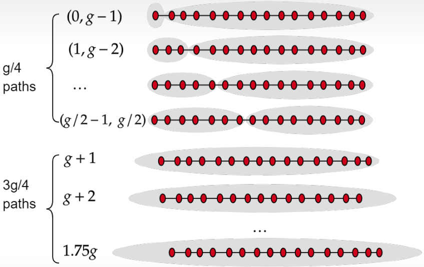
Burning Cacti

- Burn-Guess-Cactus(C,g) treats C as a rooted cactus:
  - Maintain sets T of terminals T, C **paths**, and L of labeled vertices.
  - Add the **deepest** unlabeled node x to to T.
  - There is either one or two vertices at distance $g - 1$ of x which are a part of a simple path between x and the root.
    - If there is one vertex p_1, add p to C.
    - if there are two vertices p_1, p_2, add the path between them to C.
  - Add all nodes within distance $g - 1$ of the path to L.
  - When $|T| = g$, return Bad-Guess.
  - When all vertices are marked, proceed to the next phase.
  - Here, we first look at $g = 4$ and then $g = 5$.



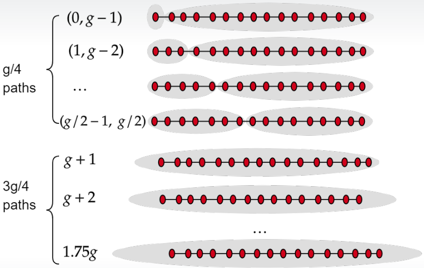
Burning Cacti

- It is possible to burn a forest C of g disjoint paths, each of length at most $2g$ nodes in at most $1.75g$ rounds.



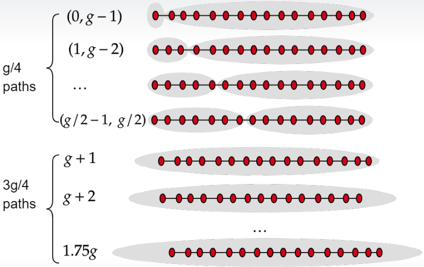
Burning Cacti

- It is possible to burn a forest C of g disjoint paths, each of length at most $2g$ nodes in at most $1.75g$ rounds.
- It is possible to burn all vertices in C in $1.75g$.



Burning Cacti

- It is possible to burn a forest C of g disjoint paths, each of length at most $2g$ nodes in at most $1.75g$ rounds.
- It is possible to burn all vertices in C in $1.75g$.
- All nodes are within distance g of one of the centers, so all vertices are burned in $1.75g + g = 2.75g$ rounds.



Burning Cacti Summary

Theorem

There is a polynomial algorithm with approximation ratio of 2.75 *for burning any cactus graph [S.K. and Shabani, 2021].*

- The main idea was to burn paths of centers instead of singular centers.
- The same idea might be applied burning other graph families (e.g., Series-Parallel graphs).

Forests of Disjoint Paths

- The burning problem is NP-hard when the input graph is a forest of disjoint paths [Bessy et al., 2017].
  - Given disks of radii $0, 1, \ldots, k-1$, it is not clear which disk should be assigned to which path.
- If there are $\Theta(1)$ disjoint paths, there is a polynomial-time algorithm that generates an optimal burning scheme [Bonato and S.K., 2019].
  - Apply a dynamic programming approach!

Forests of Disjoint Paths

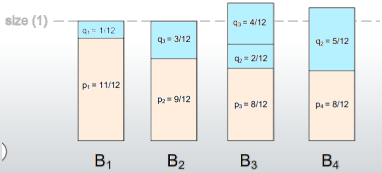
- Given any positive value ϵ, there is a fully polynomial-time approximation algorithm (FPTAS) that generates a burning scheme that completes within a factor $1 + \epsilon$ of an optimal scheme [Bonato and S.K., 2019].

Forests of Disjoint Paths

- Given any positive value ϵ, there is a fully polynomial-time approximation algorithm (FPTAS) that generates a burning scheme that completes within a factor $1 + \epsilon$ of an optimal scheme [Bonato and S.K., 2019].
  - Reduce the burning problem to the bin covering problem, and use an existing FPTAS of [Jansen and Solis-Oba, 2003] for the bin covering to get an FPTAS for the burning problem.

Forests of Disjoint Paths

- Given any positive value ϵ, there is a fully polynomial-time approximation algorithm (FPTAS) that generates a burning scheme that completes within a factor $1 + \epsilon$ of an optimal scheme [Bonato and S.K., 2019].
  - Reduce the burning problem to the bin covering problem, and use an existing FPTAS of [Jansen and Solis-Oba, 2003] for the bin covering to get an FPTAS for the burning problem.
  - **Bin covering:** "cover" a maximum number of bins of unit size with a given multi-set of items with sizes in $(0, 1]$.

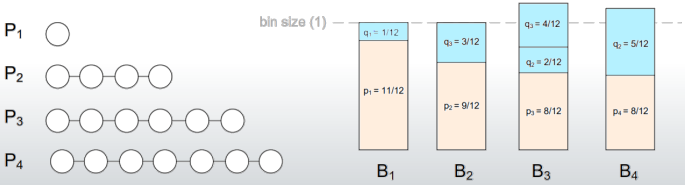


Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.

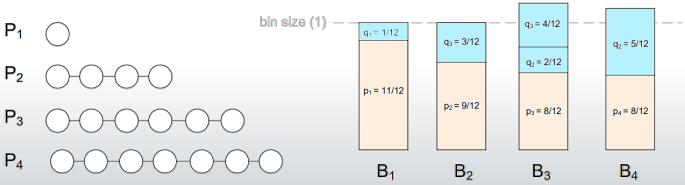
Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.
  - Think of paths as uniform "bins" that need to be "covered" by items (disks) of radii $0, 1, \ldots, k-1$.



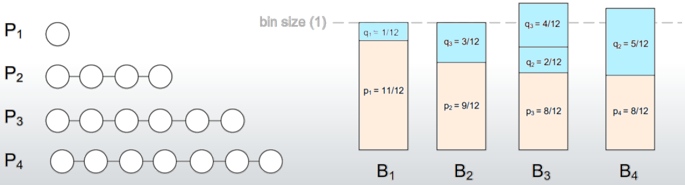
Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.
    - Think of paths as uniform "bins" that need to be "covered" by items (disks) of radii $0, 1, \ldots, k-1$.



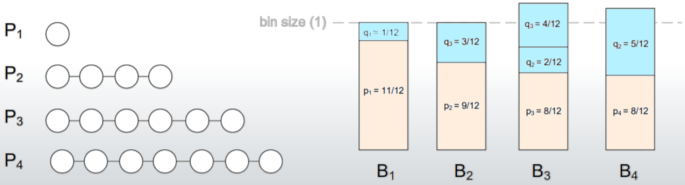
Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.
  - Think of paths as uniform "bins" that need to be "covered" by items (disks) of radii $0, 1, \ldots, k-1$.



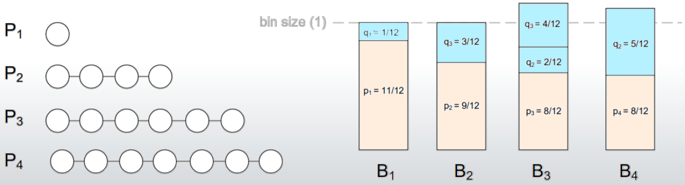
Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.
  - Think of paths as uniform "bins" that need to be "covered" by items (disks) of radii $0, 1, \ldots, k-1$.
    - Items $q_1, q_2, \ldots, q_k$ project disks (fires) of various radii to items of various sizes.



Forests of Disjoint Paths

- Reduction: Given a path forest G with b paths generate an instance of the bin covering problem such that G can be burned in k rounds iff it is possible to cover b bins.
  - Think of paths as uniform "bins" that need to be "covered" by items (disks) of radii $0, 1, \ldots, k-1$.
    - Items $q_1, q_2, \ldots, q_k$ project disks (fires) of various radii to items of various sizes.
    - Items $p_1, p_2, \ldots, p_b$ project paths of various lengths into bins of unit size.



Burning Forests of Disjoint Paths Summary

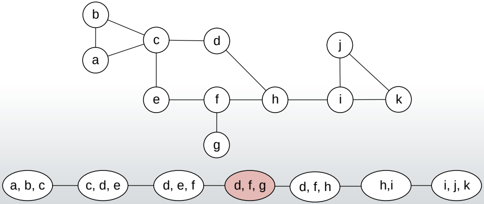
Theorem

There is a fully polynomial-time approximation scheme (FPTAS) for burning any forest of disjoint paths [Bonato and S.K., 2019].

- The complexity of the problem is settled for forests of disjoint paths.
- For what other graph families an FPTAS might be developed?

Tree Decomposition & Burning

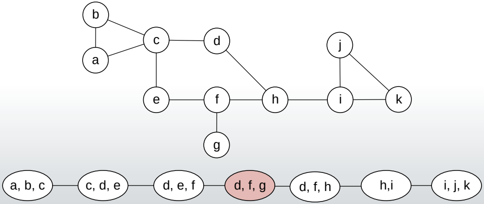
- In a Robertson-Seymour path decomposition:
  - Path-length [Dourisboure and Gavoille, 2007] is the max **distance** of vertices in any bag.
  - The graph below has path-width 2 and path-length 3.



Tree Decomposition & Burning

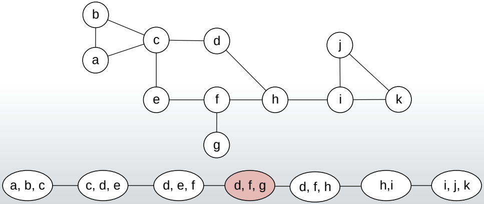
- In a Robertson-Seymour path decomposition:
  - Path-length [Dourisboure and Gavoille, 2007] is the max **distance** of vertices in any bag.
  - The graph below has path-width 2 and path-length 3.

- The burning number of a graph with **path-length** pl and diameter d is at most $\lceil\sqrt{d}\rceil + pl$ [S.K. et al., 2020].



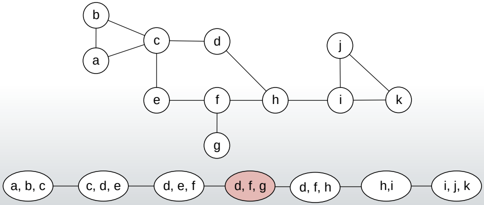
Tree Decomposition & Burning

- In a Robertson-Seymour path decomposition:
  - Path-length [Dourisboure and Gavoille, 2007] is the max **distance** of vertices in any bag.
  - The graph below has path-width 2 and path-length 3.
  - A graph has path-length 1 if and only if ??
- The burning number of a graph with **path-length** *pl* and diameter d is at most $\lceil \sqrt{d} \rceil + pl$ [S.K. et al., 2020].



Tree Decomposition & Burning

- In a Robertson-Seymour path decomposition:
  - Path-length [Dourisboure and Gavoille, 2007] is the max **distance** of vertices in any bag.
  - The graph below has path-width 2 and path-length 3.
  - A graph has path-length 1 if and only if it is an interval graph.
- The burning number of a graph with **path-length** pl and diameter d is at most $\lceil \sqrt{d} \rceil + pl$ [S.K. et al., 2020].



Burning of Graph Families

- There is an approximation algorithm with factor $1 + o(1)$ for burning any graph G of constant path-length [S.K. et al., 2020].

Burning of Graph Families

- There is an approximation algorithm with factor $1 + o(1)$ for burning any graph G of constant path-length [S.K. et al., 2020].
  - If the diameter of G is constant, we can optimally solve the problem.

Burning of Graph Families

- There is an approximation algorithm with factor $1 + o(1)$ for burning any graph G of constant path-length [S.K. et al., 2020].
  - If the diameter of G is constant, we can optimally solve the problem.
  - Otherwise, burn the graph in $\sqrt{d} + pl$ rounds, getting an approximation factor at most $\frac{\sqrt{d}+pl}{\sqrt{d}} = 2 + o(1)$.

Burning of Graph Families

- There is an approximation algorithm with factor $1 + o(1)$ for burning any graph G of constant path-length [S.K. et al., 2020].
  - If the diameter of G is constant, we can optimally solve the problem.
  - Otherwise, burn the graph in $\sqrt{d} + pl$ rounds, getting an approximation factor at most $\frac{\sqrt{d}+pl}{\sqrt{d}} = 2 + o(1)$.
- There is an approximation algorithm with factor $1 + o(1)$ for burning any graph G of constant path-length [S.K. et al., 2020].

Summary

Graph family	Apx. Factor	Details
general graphs	3	[Bonato and S.K., 2019]
trees	2	[Bonato and S.K., 2019]
cacti	2.75	[S.K. and Shabani, 2021]
forests of disjoint paths	$1+\epsilon$ (FPTAS)	[Bonato and S.K., 2019]
graphs of bounded path-length	$1+o(1)$	[S.K. et al., 2020]
graphs of bounded tree-length	$2+o(1)$	[S.K. et al., 2020]

References

Bessy, S.; Bonato, A.; Janssen, J. C. M.; Rautenbach, D.; and Roshanbin, E. (2017).
"Burning a graph is hard".
Discrete Applied Mathematics, 232, pp. 73–87.

Bonato, A.; Janssen, J. C. M.; and Roshanbin, E. (2014).
"Burning a Graph as a Model of Social Contagion".
In *Workshop of Workshop on Algorithms and Models for the Web Graph*, pages 13–22.

Bonato, A. and S.K. (2019).
"Approximation Algorithms for Graph Burning".
In *Proc. Theory and Applications of Models of Computation TAMC*, volume 11436 of *Lecture Notes in Computer Science*, pages 74–92. Springer.

Dourisboure, Y. and Gavoille, C. (2007).
"Tree-decompositions with bags of small diameter".
Discrete Mathematics, 307(16), pp. 2008–2029.

Jansen, K. and Solis-Oba, R. (2003).
"An asymptotic fully polynomial time approximation scheme for bin covering".
Theoretical Computer Science, 306(1-3), pp. 543–551.

Land, M. R. and Lu, L. (2016).
"An Upper Bound on the Burning Number of Graphs".
In *Proceedings of Workshop on Algorithms and Models for the Web Graph*, pages 1–8.

Mondal, D.; Parthiban, N.; Kavitha, V.; and Rajasingh, I. (2021).
"APX-Hardness and Approximation for the k-Burning Number Problem".
In Uehara, R.; Hong, S.; and Nandy, S. C., editors, *Proc. Algorithms and Computation - 15th International Conference*, volume 12635 of *Lecture Notes in Computer Science*, pages 272–283. Springer.

S.K.; Miller, A.; and Zhang, K. (2020).
"Burning Two Worlds".

In *Proc. SOFSEM 2020*, volume 12011 of *Lecture Notes in Computer Science*, pages 113–124. Springer.

S.K. and Shabani, M. (2021).
"Burning Cacti".
Ongoing work.