Chapter 61

Star Contraction

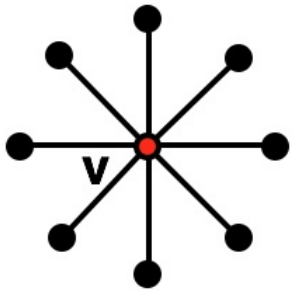
This chapter covers star partition and star contraction, an efficient and parallel graph-contraction technique for general graphs.

1 Star Partition

In an edge partition , if an edge incident on a vertex v is selected as a block, then none of the other edges incident on v can be their own block. This limits the effectiveness of the edge partition technique, because it is unable to contract graphs with high-degree vertices significantly. In this section, we describe an alternative technique, star partition, that does not have this limitation.

Definition 61.1 (Star Partition)**.** A ***star partition*** of a graph G is a partition of G where each block is vertex-induced subgraph with respect to a star graph .

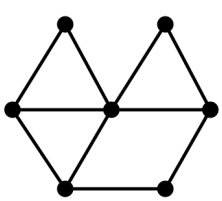
Example 61.1. Consider star graph with center v and eight satellites.



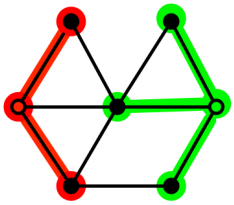
- A partition consisting of the whole graph is a star partition, where the only block is the graph itself, induced by the star graph.

- A partition where each block is an isolated vertex is a star partition, because each block is a vertex-induced subgraph of a single vertex, which is a star.

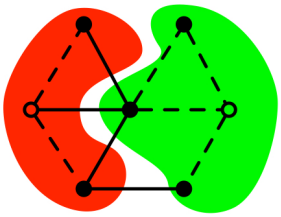
Example 61.2. Consider the graph shown below on the left. To partition this graph, we first find two disjoint stars, which are highlighted. Each star induces a block consisting of its vertices and the corresponding edges of the graph. These two blocks form a star partition the graph. Note that in a star partition, a block might not be a star.



A graph



Its two stars (highlighed).



The corresponding star partition. Solid edges are cut edges. Dashed edges are internal edges.

Constructing a Star Partition (Sequential). We can construct a star partition sequentially by iteratively adding stars until the vertices are exhausted as follows.

- Select an arbitrary vertex v from the graph and make v the center of a star.
- Attach as satellites all the neighbors of v in the graph.
- Remove v and its satellites from the graph.

Computing a Star Partition (Parallel). We can construct a star partition in parallel by making local independent decisions for each vertex, and using randomization to break symmetry. One approach proceeds as follows.

- Flip a coin for each vertex.
- If a vertex flips heads, then it becomes the center of a star.
- If a vertex flips tails, then there are two cases.
  - The vertex has a neighbor that flips heads. In this case, the vertex selects the neighbor (breaking ties arbitrarily) and becomes a satellite.
  - The vertex doesn't have a neighbor that flips heads. In this case, the vertex becomes a center.

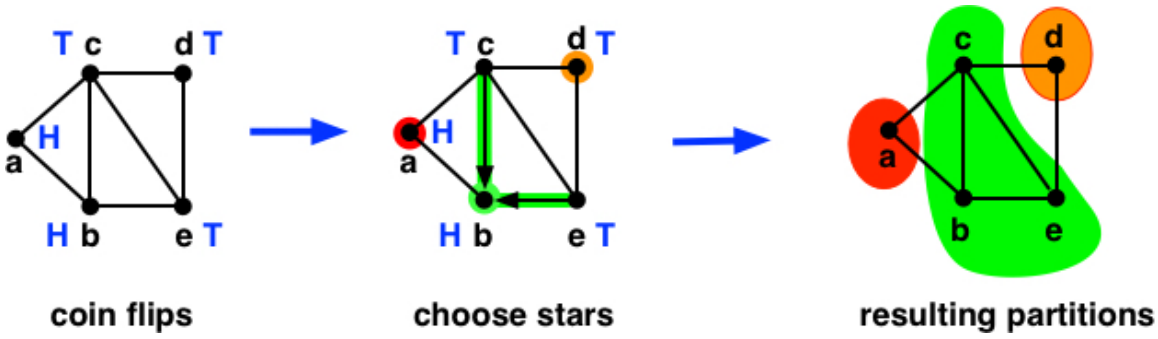
Note that if a vertex doesn't have a neighbor (it is "isolated"), then it will always become a center.

Definition 61.2 (Isolated Vertices)**.** We say that a vertex is ***isolated*** in a graph if it doesn't have a neighbor.

Note. The parallel approach to star partition is not optimal, because it might not always create the smallest number of stars. This is acceptable for us, because we only need to reduce the size of the graph by some constant factor.

Example 61.3 (Randomized Star Partition)**.** The example below illustrates how we may partition a graph using the parallel star partition algorithm described above. Vertices `a` and `b`, which flip heads, become centers. Vertices `c` and `e`, which flipped tails, attempt to become satellites by finding a center among their neighbors, breaking ties arbitrarily. If a vertex does not have a neighbor that is a center (flipped heads), then it becomes a singleton star (e.g., vertex d).

The resulting star partition has three stars: the star with center `a` (with no satellites), the star with center `b` (with two satellites), and the singleton star `d`. The star partition thus yields three blocks, which are defined by the subgraphs induced by each star.



Algorithm 61.3 (Parallel Star Partition)**.** To specify the star-partition algorithm, we need a source of randomness. We assume that each vertex has access to a random coin flip

$$\texttt{heads}\ v : V \times \mathbb{Z} \to \mathbb{B},$$

which returns `true` if the vertex v flips heads and false otherwise for this partition.

The function *starPartition*, whose pseudo-code is given below, takes as argument a graph and a round number, and returns a graph partition specified by a set of centers and a partition map from all vertices to centers.

The algorithm starts by flipping a coin for each vertex and selecting the edges that point from tails to heads—this gives the set of edges TH. In this set of edges, there can be multiple edges from the same non-center. Since we want to choose one center for each satellite, we remove duplicates in Line 6, by creating a set of singleton tables and merging them, which selects one center per satellite. This completes the selection of satellites and their centers.

Next, the algorithm determines the set of centers as all the non-satellite vertices. To complete the process, the algorithm maps each center to itself (Line 10). These operations effectively promote unmatched non-centers to centers, forming singleton stars, and

matches all centers with themselves. Finally, the algorithm constructs the partition map by uniting the mapping for the satellites and the centers.

```
1   starPartition G = (V, E) =
2     let
3       (* Find the arcs from satellites to centers.  *)
4       TH = {(u, v) ∈ E | ¬(heads u) ∧ (heads v)}
5       (* Partition map:  satellites map to centers *)
6       P_s = ⋃_{(u,v)∈TH} {u ↦ v}
7       (* Centers are non-satellite vertices *)
8       V_c = V \ domain(P_s)
9       (* Map centers to themselves *)
10      P_c = {u ↦ u : u ∈ V_c}
11    in
12      (V_c, P_s ∪ P_c)
13    end
```

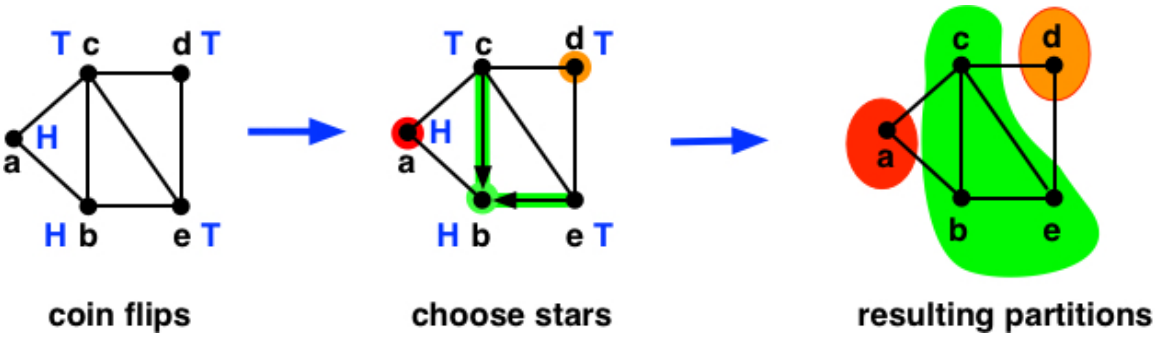
Note. Most machines don't have true sources of randomness, the function *heads* is therefore usually implemented with a pseudorandom number generator or with a good hash function.

In the algorithm, Line 6 creates a set of singleton tables and merges them. This can be implemented using sets and tables as follows.

$$\begin{array}{ll} \mathit{Set.reduce} & (\mathit{Table.union}\ (\texttt{lambda}\ (x, y)\ .\ x)) \\ & \emptyset \\ & \{\{u \mapsto v\} : (u, v) \in TH\} \end{array}$$

Note that we supply to the *union* operation a function that selects the first of the two possibilities; this is an arbitrary choice and we could have favored the second.

Example 61.4. Consider the graph below and the random coin flips.



The star-partition algorithm proceeds on this example as follows. First, it computes

$$TH = \{(\texttt{c}, \texttt{a}), (\texttt{c}, \texttt{b}), (\texttt{e}, \texttt{b})\}\,,$$

as the edges from satellites to centers. Now, it converts each edge into a singleton table, and merges all the tables into one table, which is going to become a part of the partition map:

$$P_s = \{\mathtt{c} \mapsto \mathtt{b}, \mathtt{e} \mapsto \mathtt{b}\}.$$

Note that the edge $(\mathtt{c}, \mathtt{a})$ has been removed since when uniting the tables, we select only one element for each key in the domain. Now for all remaining vertices $V_c = V \setminus domain(P) = \{\mathtt{a}, \mathtt{b}, \mathtt{d}\}$ we map them to themselves, giving:

$$P_c = \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{d} \mapsto \mathtt{d}\}.$$

The vertices in P_c are the centers. Finally we merge P and P_c to obtain the partition map

$$P_s \cup P_c = \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{c} \mapsto \mathtt{b}, \mathtt{d} \mapsto \mathtt{d}, \mathtt{e} \mapsto \mathtt{b}\}.$$

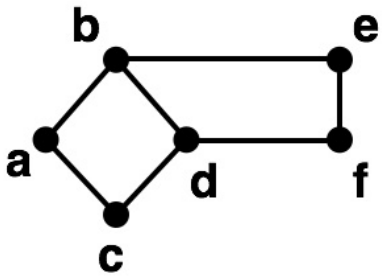
Implementation. Suppose that we are given an enumerable graph with n vertices and m edges. We can represent the graph using an edge set representation and represent the sets with sequences. This means that we have a sequence of vertices and a sequence of edges.

This representation enables a relatively clean implementation of the star-partition algorithm , as shown by the pseudo-code below. The implementation follows the pseudo-code for the algorithm but is able to compute the satellites and centers compactly by using a sequence *inject* operation. The implementation first constructs a vertex sequence V' where each vertex maps to itself. It then constructs a sequence TH of "updates" from vertices that flip heads into tails, and inject TH into the sequence of vertices V'. The resulting sequence P maps each vertex that flipped tails to a center, if the vertex has a neighbor that flipped heads. The sequence P ensures that a vertex that has flipped heads remains unaffected by the injection, e.g., if vertex i has flipped heads, then $P[i] = i$. We can thus compute set of centers by filtering over P and use the sequence P to represent the partition map for satellites and centers jointly.

$$
\begin{array}{l}
\mathit{starPartition}\ (G = (V, E)) = \\
\quad \mathtt{let} \\
\qquad V' = \langle\, j : 0 \le j < |V| \,\rangle \\
\qquad TH = \langle\, (u, v) \in E \mid \neg(\mathtt{heads}\ u) \wedge (\mathtt{heads}\ v) \,\rangle \\
\qquad P = \mathit{Seq.inject}\ V'\ TH \\
\qquad V_C = \langle\, j \in P \mid P[j] = j \,\rangle \\
\quad \mathtt{in}\ (V_C, P)\ \mathtt{end}
\end{array}
$$

Reminder (Edge-Set Representation). The edge set representation of a graph consists of a set of vertices and a set of edges, where each undirected edge is represented with two arcs, one in each direction.

Example 61.5. The edge-set representation of an undirected graph is shown below.



$$
\begin{aligned}
V &= \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f}\} \\
E &= \{(\mathtt{a},\mathtt{b}),(\mathtt{b},\mathtt{a}),(\mathtt{b},\mathtt{d}),(\mathtt{b},\mathtt{e}),(\mathtt{e},\mathtt{b}),(\mathtt{d},\mathtt{b}),(\mathtt{d},\mathtt{f}),(\mathtt{a},\mathtt{c}),\\
&\quad (\mathtt{c},\mathtt{a}),(\mathtt{c},\mathtt{d}),(\mathtt{d},\mathtt{c}),(\mathtt{d},\mathtt{f}),(\mathtt{f},\mathtt{d}),(\mathtt{e},\mathtt{f}),(\mathtt{f},\mathtt{e})\}
\end{aligned}
$$

1.1 Analysis of Star Partition

Theorem 61.1 (Cost of Star Partition)**.** Based on the array-based cost specification for sequences, the cost of $starPartition$ is

$$O(n+m)$$

work and

$$O(\lg n)$$

span for a graph with n vertices and m edges.

Exercise 61.1. Prove the theorem.

Number of Satellites. Let us also bound the number of satellites found by $starPartition$. Note first that there is a one-to-one mapping between the satellites and the set P_s computed by the algorithm. The following lemma establishes that on a graph with n non-isolated vertices, the number of satellites is at least $n/4$ in expectation. As we will see this means that we can use star partition to perform graph contraction with logarithmic span.

Lemma 61.2 (Number of Satelites)**.** For a graph G with $n_\bullet$ non-isolated vertices, the expected number of satellites in a call to $starPartition\ (G, i)$ with any i is at least $n_\bullet/4$.

Proof. For any vertex v, let H_v be the event that a vertex v comes up heads, T_v that it comes up tails, and R_v that $v \in domain(P)$ (i.e, it is a satellite). Consider any non-isolated vertex $v \in V(G)$. By definition, we know that a non-isolated vertex v has at least one neighbor u. So, we know that $T_v \wedge H_u$ implies R_v, since if v is a tail and u is a head v must either

join u's star or some other star. Therefore, $\mathbf{P}[R_v] \geq \mathbf{P}[T_v]\,\mathbf{P}[H_u] = 1/4$. By the linearity of expectation, the expected number of satellites is

$$\mathbf{E}\left[\sum_{v:v \text{ non-isolated}} \mathbb{I}\{R_v\}\right] = \sum_{v:v \text{ non-isolated}} \mathbf{E}[\mathbb{I}\{R_v\}]$$
$$\geq n_\bullet/4.$$

The final inequality follows because we have $n_\bullet$ non-isolated vertices and because the expectation of an indicator random variable is equal to the probability that it takes the value 1. □

2 Star Contraction

Definition 61.4 (Star Contraction)**.** ***Star contraction*** is an instance of graph contraction that uses star partitions to contract the graph.

Algorithm 61.5 (Star Contraction)**.** The pseudo-code below gives a higher-order star-contraction algorithm. The algorithm takes as arguments the graph G and two functions:

- *base* function specifies the computation in the base case, and
- *expand* function computes the result for the larger graph from the quotient graph.

In the base case, the graph contains no edges and the function *base* is called on vertex set.

In the recursive case, the graph is partitioned by a call to star partition (Line 6), which returns the set of (centers) super-vertices V' and P the partition map mapping every $v \in V$ to a $v' \in V'$. The set V' defines the super-vertices of the quotient graph. Line 7 computes the edges of the quotient graph by routing the end-points of each edge in E to the corresponding super-vertices in V' as specified by partition-map P. Note that the filter $P[u] \neq P[v]$. removes self edges. The algorithm then recurs on the quotient graph (V', E'). The algorithm then computes the result for the whole graph by calling the function *expand* on the result of the recursive call R.

```
1   starContract base expand (G = (V, E)) =
2     if |E| = 0 then
3       base (V)
4     else
5       let
6         (V', P) = starPartition (V, E)
7         E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
8         R = starContract base expand (V', E')
9       in
10        expand (V, E, V', P, R)
11      end
```

Theorem 61.3 (Work and Span of Star Contraction)**.** For a graph $G = (V, E)$, we can contract the graph into a number of isolated vertices in $O((|V| + |E|) \lg |V|)$ work and $O(\lg^2 |V|)$ span.

Proof structure and assumptions. For the proof, we will consider work and span separately and assume that

- function *base* has constant span and linear work in the number of vertices passed as argument, and
- function *expand* has linear work and logarithmic span in the number of vertices and edges at the corresponding step of the contraction.

Span of Star Contraction. Let $n_\bullet$ be the number of non-isolated vertices. In star contraction, once a vertex becomes isolated, it remains isolated until the final round, since contraction only removes edges. Let $n'_\bullet$ denote the number of non-isolated vertices after one round of star contraction. We can write the following recurrence for the span of star contraction.

$$S(n_\bullet) = \begin{cases} S(n'_\bullet) + O(\lg n) & \text{if} \quad n_\bullet > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Observe that $n'_\bullet = n_\bullet - X$, where X is the number of satellites (as defined earlier in the lemma about *starPartition*), which are removed at a step of contraction. Because $\mathbf{E}[X] = n_\bullet/4$, $\mathbf{E}[n'_\bullet] = 3n/4$. This is a familiar recurrence, which we know solves to $O(\lg^2 n_\bullet)$, and thus $O(\lg^2 n)$, in expectation.

Work of Star Contraction. For work, we would like to show that the overall work is linear, because we might hope that the graph size is reduced by a constant fraction on each round. Unfortunately, this is not the case. Although we have shown that star contraction can remove a constant fraction of the non-isolated vertices in one round, we have not bounded the number of edges removed.

Because removing a satellite also removes the edge that attaches it to its star's center, each round removes at least as many edges as vertices. But this does not help us bound the number of edges removed by a linear function of m, because there can be as many an n^2 edges in the graph.

To bound the work, we will consider non-isolated and isolated vertices separately. Let $n'_\bullet$ denote the number of non-isolated vertices after one round of star contraction. For the non-isolated vertices, we have the following work recurrence:

$$W(n_\bullet, m) \leq \begin{cases} W(n'_\bullet, m) + O(n_\bullet + m) & \text{if} \quad n_\bullet > 1 \\ 1 & \text{otherwise.} \end{cases}$$

This recursion solves to

$$\mathbf{E}[W(n_\bullet, m)] = O(n_\bullet + m \lg n_\bullet) = O(n + m \lg n).$$

To bound the work on isolated vertices, we note that there at most n of them at each round and thus, the additional work is $O(n \lg n)$.

We thus conclude that the total work is

$$O((n + m) \lg n).$$

Note. Consider as an example a star contraction where n and m have the following values in each round.

round	vertices	edges
1	n	m
2	$n/2$	$m - n/2$
3	$n/4$	$m - 3n/4$
4	$n/8$	$m - 7n/8$

It is clear that the number of edges does not drop below $m - n$, so if there are $m > 2n$ edges to start with, the overall work will be $O(m \lg n)$.