Masterthesis

High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations

Tobias Heuer

Abgabedatum: 19.10.2012

Betreuer: Prof. Dr. Peter Sanders
Sebastian Schlag

Institut für Theoretische Informatik, Algorithmik
Fakultät für Informatik
Karlsruher Institut für Technologie

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Ort, den Datum

Zusammenfassung

Hier die deutsche Zusammenfassung.

Ich bin Blindtext. Von Geburt an. Es hat lange gedauert, bis ich begriffen habe, was es bedeutet, ein blinder Text zu sein: Man macht keinen Sinn. Man wirkt hier und da aus dem Zusammenhang gerissen. Oft wird man gar nicht erst gelesen. Aber bin ich deshalb ein schlechter Text? Ich weiß, dass ich nie die Chance haben werde im Stern zu erscheinen. Aber bin ich darum weniger wichtig? Ich bin blind! Aber ich bin gerne Text. Und sollten Sie mich jetzt tatsächlich zu Ende lesen, dann habe ich etwas geschafft, was den meisten „normalen“ Texten nicht gelingt.

Ich bin Blindtext. Von Geburt an. Es hat lange gedauert, bis ich begriffen habe, was es bedeutet, ein blinder Text zu sein: Man macht keinen Sinn. Man wirkt hier und da aus dem Zusammenhang gerissen. Oft wird man gar nicht erst gelesen. Aber bin ich deshalb ein schlechter Text? Ich weiß, dass ich nie die Chance haben werde im Stern zu erscheinen. Aber bin ich darum weniger wichtig? Ich bin blind! Aber ich bin gerne Text.

Abstract

And here an English translation of the German abstract.

I'm blind text. From birth. It took a long time until I realized what it means to be random text: You make no sense. You stand here and there out of context. Frequently, they do not even read. But I have a bad copy? I know that I will never have the chance of appearing in the. But I'm any less important? I'm blind! But I like to text. And you should see me now actually over, then I have accomplished something that is not possible in most “normal” copies.

I'm blind text. From birth. It took a long time until I realized what it means to be random text: You make no sense. You stand here and there out of context. Frequently, they do not even read. But I have a bad copy? I know that I will never have the chance of appearing in the. But I'm any less important? I'm blind! But I like to text.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Contents

List of Figures

List of Tables

Algorithmenverzeichnis

1 Introduction

1.1 Motivation

1.2 Contributions

2 Preliminaries

2.1 Definitions and Terminology

Notation and Definitions. An *undirected hypergraph* $H = (V, E, c, \omega)$ is defined as a set of n vertices V and a set of m hyperedges/nets E with vertex weights $c : V \to \mathbb{R}_{>0}$ and net weights $\omega : E \to \mathbb{R}_{>0}$, where each net is a subset of the vertex set V (i.e., $e \subseteq V$). The vertices of a net are called *pins*. We use P to denote the multiset of all pins in H. We extend c and ω to sets, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. A vertex v is *incident* to a net e if $v \in e$. $\mathrm{I}(v)$ denotes the set of all incident nets of v. The *degree* of a vertex v is $d(v) := |\mathrm{I}(v)|$. The set $\Gamma(v) := \{u \mid \exists\ e \in E : \{v, u\} \subseteq e\}$ denotes the neighbors of v. The *size* $|e|$ of a net e is the number of its pins. Nets of size one are called *single-vertex* nets. A *k-way partition* of a hypergraph H is a partition of its vertex set into k *blocks* $\Pi = \{V_1, \dots, V_k\}$ such that $\bigcup_{i=1}^{k} V_i = V$, $V_i \neq \emptyset$ for $1 \leq i \leq k$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. We call a k-way partition Π *ε-balanced* if each block $V_i \in \Pi$ satisfies the *balance constraint*: $c(V_i) \leq L_{\max} := (1 + \varepsilon)\lceil \frac{c(V)}{k} \rceil$ for some parameter ε. Given a k-way partition Π, the number of pins of a net e in block V_i is defined as $\Phi(e, V_i) := |\{v \in V_i \mid v \in e\}|$. For each net e, $\Lambda(e) := \{V_i \mid \Phi(e, V_i) > 0\}$ denotes the *connectivity set* of e. The *connectivity* of a net e is the cardinality of its connectivity set: $\lambda(e) := |\Lambda(e)|$. A net is called *cut net* if $\lambda(e) > 1$. The *k-way hypergraph partitioning problem* is to find an ε-balanced k-way partition Π of a hypergraph H that minimizes an objective function over the cut nets for some ε. Several objective functions exist in the literature [?, ?]. The most commonly used cost functions are the *cut-net* metric $\mathrm{cut}(\Pi) := \sum_{e \in E'} \omega(e)$ and the *connectivity* metric $(\lambda - 1)(\Pi) := \sum_{e \in E'} (\lambda(e) - 1)\ \omega(e)$, where E' is the set of all cut nets [?]. In this paper, we use the connectivity-metric, which accurately models the total communication volume of parallel sparse matrix-vector multiplication [?]. Optimizing both objective functions is known to be NP-hard [?]. *Contracting* a pair of vertices (u, v) means merging v into u. The weight of u becomes $c(u) := c(u) + c(v)$. We connect u to the former neighbors $\Gamma(v)$ of v by replacing v with u in all nets $e \in \mathrm{I}(v) \setminus \mathrm{I}(u)$ and remove v from all nets $e \in \mathrm{I}(u) \cap \mathrm{I}(v)$. *Uncontracting* a vertex u reverses the contraction. The two most common ways to represent a hypergraph $H = (V, E, c, \omega)$ as an undirected graph are the *clique* and the *bipartite* representation [12]. In the following, we use *nodes* and *edges* when referring to a graph representation and *vertices* and *nets* when referring to H. In the *clique* graph $G_x(V, E_x \subseteq V^2)$ of H, each net is replaced with an edge for each pair of vertices in the net: $E_x := \{(u, v) : u, v \in e, e \in E\}$. Thus the pins of a net e with size $|e|$ form a $|e|$-clique in G_x. In the *bipartite* graph $G_*(V \dot{\cup} E, F)$ the vertices and nets of H form the node set and for each net e incident to a vertex v, we add an edge (e, v) to G_*. The edge set F is thus defined as $F := \{(e, v) \mid e \in E, v \in e\}$. Each net in E therefore corresponds to a star in G_*. In both models, node weights c and edge weights ω are chosen according to the problem domain [?].

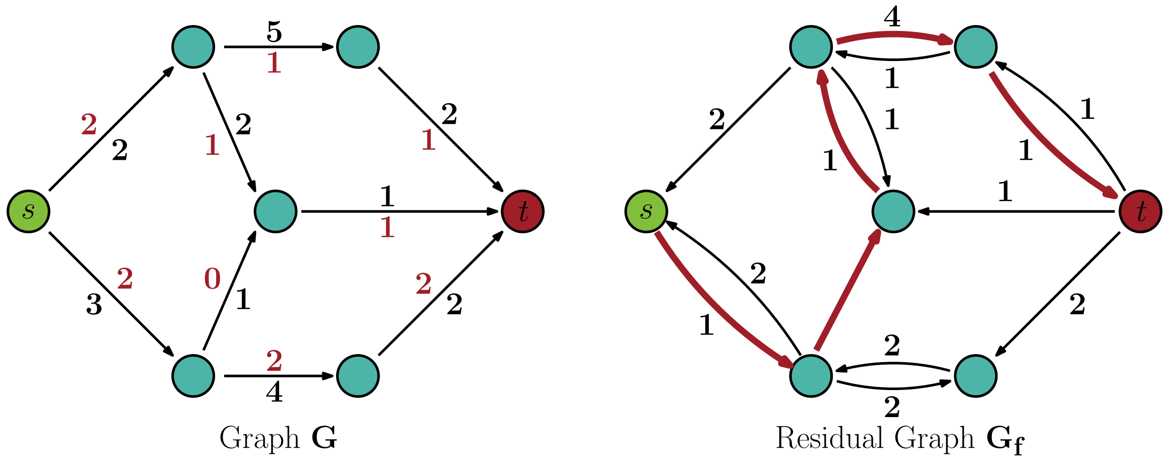

Figure 1: Figure illustrates concepts related to the maximum flow problem. A valid flow f (red values) from s to t on a graph G is shown on the left side. The corresponding *residual graph* G_f with its *residual capacities* (black values) is illustrated on the right side. The red highlighted path represents an *augmenting path* in G.

3 Related Work

3.1 Maximum Flow Problem

The first serious analysis of the maximum flow problem emerged in 1955 during a study of transportation and communication networks by Harris [9]. He formulate the problem as follows: *Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other.*
He generalizes his model of railway traffic flow to the today known maximum flow problem. Given a graph $G = (V, E, c)$ with capacity function $c : E \to \mathbb{R}_+$ and a source $s \in V$ and a sink $t \in V$. The maximum flow problem is about finding the maximum amount of flow from s to t in G. A flow is a function $f : E \to \mathbb{R}_+$, which have to satisfy the following constraints:

(i) $\forall (u, v) \in E : f(u, v) \leq c(u, v)$ (capacity constraint)

(ii) $\forall v \in V \setminus \{s, t\} : \sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$ (conservation of flow constraint)

The capacity constraint restricts the flow on an edge (u, v) by its capacity $c(u, v)$. Whereas the conservation of flow constraint ensures that the amount of flow entering a node $v \in V \setminus \{s, t\}$ is the same as leaving a node. The value of the flow is defined as $|f| = \sum_{(s,v)\in E} f(s, v) = \sum_{(v,t)\in E} f(v, t)$. A flow f is maximal, if there exists no other flow f' with $|f'| > |f|$.
Another useful construct in connection with maximum flows, is the concept of the *residual graph* G_f and the *residual capacity* r_f of a flow function f on graph G. The *residual capacity* $r_f : V \times V \to \mathbb{R}_+$ is defined as follows:

(i) $\forall (u, v) \in E : r_f(u, v) = c(u, v) - f(u, v)$

(ii) $\forall (u, v) \in E$: If $f(u, v) > 0$ and $c(v, u) = 0$, then $r_f(v, u) = f(u, v)$

For a edge $e = (u, v) \in E$ the residual capacity $r_f(u, v)$ is the remaining amount of flow which can be send over edge e. For each reverse edge $\overleftarrow{e} \notin E$ the residual capacity $r_f(\overleftarrow{e})$ is the amount of flow which is send over e. The *residual graph* $G_f = (V, E_f, r_f)$ is the network containing all $(u, v) \in V \times V$ with $r_f(u, v) > 0$. More formal $E_f = \{(u, v) \mid r_f(u, v) > 0, (u, v) \in V \times V\}$. Figure 1 illustrates all presented concepts.

3.1.1 Augmenting-Path Algorithms

An *augmenting path* $P = \{v_1, \dots, v_k\}$ is a path in G_f with $v_1 = s$ and $v_k = t$ [4]. Figure 1 illustrates such a path. Since all $(v_i, v_{i+1}) \in G_f \Rightarrow r_f(v_i, v_{i+1}) > 0$. Therefore, we can increase the flow on all edges (v_i, v_{i+1}) by $\Delta f = \min_{i \in [1,\dots,k-1]} r_f(v_i, v_{i+1})$. It can be shown that f is not a maximum flow, if an *augmenting path* exists in G_f [4].
One way to calculate a maximum flow f is to find *augmenting paths* in G_f as long as their exist one. The algorithm was established by Ford and Fulkerson [6] and consists of two phases. First, we search for an *augmenting path* $P = \{v_1, \dots, v_k\}$ from s to t, e.g. with a simple *DFS*. In the *augmentation* step, we increase the flow on each edge (v_i, v_{i+1}) by Δf and decrease the flow on each reverse edge (v_{i+1}, v_i) by Δf. If the capacities are integral, the algorithm always terminates. Since, we can find an *augmenting path* in G_f with a simple *DFS* in $\mathcal{O}(|V| + |E|)$ and increase the flow on every path by at least one, the running time of the algorithm can be bounded by $\mathcal{O}(|E||f_{max}|)$. We can construct instances, where the running time is $\mathcal{O}(|E||f_{max}|)$ or even the maximum flow $|f_{max}|$ is exponential in the problem size.
Edmond and Karp [4] improved Ford & Fulkerson algorithm by increasing the flow along an *augmenting path* of minimal length. A shortest path from s to t in a graph with unit lengths can be found by a simple *BFS* calculation. It can be shown, that the total number of *augmentations* is $\mathcal{O}(|V||E|)$. The running time of Edmond & Karps maximum flow algorithm is then given by $\mathcal{O}(|V||E|^2)$. A sample execution of the algorithm is presented in Figure 2.

3.1.2 Push-Relabel Algorithm

3.1.3 Applications

The *Max-Flow-Min-Cut*-Theorem is fundamental for many applications related to the maximum flow problem [6].

Theorem 3.1. *The value of a maximum (s,t)-flow obtainable in a graph G is equal with the weight of the minimum cutset in G seperating s and t.*

Let f be a maximum flow in a graph $G = (V, E, \omega)$ with $s \in V$ and $t \in V$. Further, let A be the set containing all $v \in V$, which are *reachable* from s in G_f. A node v is *reachable* from a node u, if there exists a path from u to v. Then the set of all cut edges between the bipartition $(A, V \setminus A)$ is a minimum-weight (s,t)-cutset [7]. A can be calculated with a simple *BFS* in G_f starting from s.
From this analogy many solutions for related problems arose. Samples are listed below:

(i) Maximum Bipartite-Matching
(ii) Minimum-Weight Vertex Seperator
(iii) Number of Edge-Disjoint Paths
(iv) Number of Vertex-Disjoint Paths

Solutions for those problems sometimes involves a transformation T of the graph G into a flow network $T(G)$, such that the *Max-Flow-Min-Cut*-Theorem is applicable. A problem important for this work is to find a minimum-weight (s,t)-vertex seperator in a graph $G = (V, E, c)$ with $c : V \to \mathbb{R}_+$.

Definition 3.1. *Let $G = (V, E, c)$ be a graph with $c : V \to \mathbb{R}_+$. $S \subseteq V$ is a vertex seperator for non-adjacent verticies $s \in V$ and $t \in V$ if the removal of S from graph G seperates s and t (s not* reachable *from t). A vertex seperator S is a minimum-weight (s,t)-vertex seperator, if for all $S' \subseteq V$ $c(S) \leq c(S')$.*

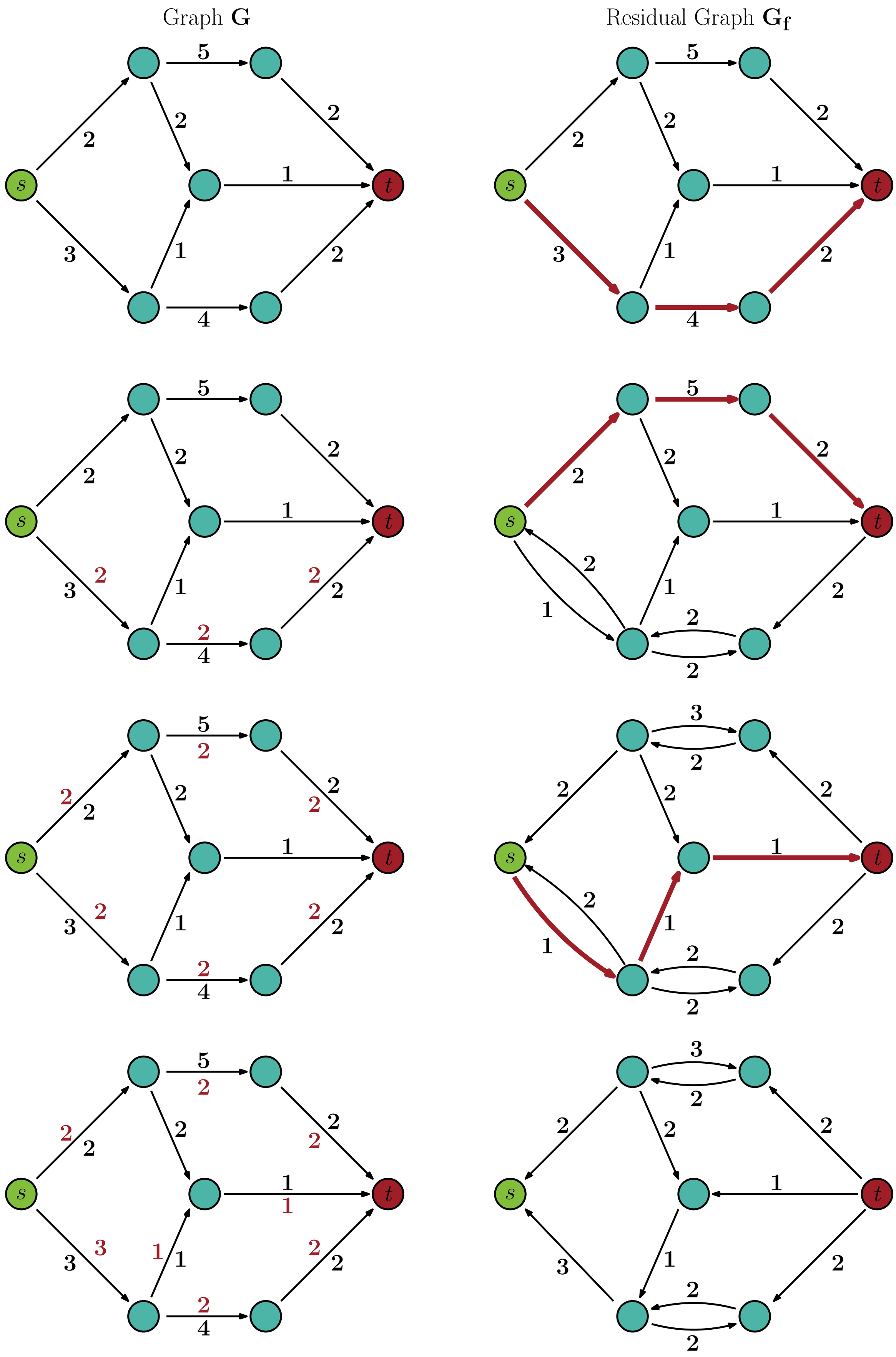

Figure 2: Sample execution of Edmond & Karps maximum flow algorithm [4]. The network G with its capacities c (black values) and flow f (red values) is illustrated on the left side. The residual graph G with its *residual capacities* r_f (black values) is presented on the right side. In each step the current *augmenting path* in G_f is highlighted by a red path.

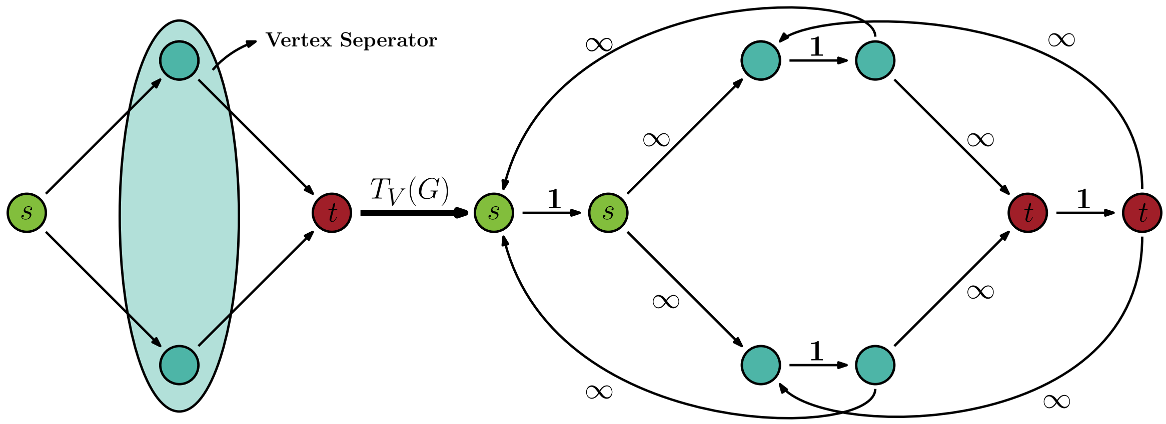

Figure 3: Illustration of the vertex seperator problem and the transformation $T_V(G)$ in which we can find a minimum vertex seperator with maximum flow computation.

We can calculate a minimum-weight (s,t)-vertex seperator with a maximum flow calculation in the following flow network (**TODO 1:** *reference*):

Definition 3.2. *Let T_V be a transformation of a graph $G = (V, E, c)$ into a flow network $T_V(G) = (V_V, E_V, c_V)$ (with $c_V : E_V \to \mathbb{R}_+$). T_V is defined as follows:*

(i) $V_V = \bigcup_{v \in V} \{v', v''\}$

(ii) $\forall v \in V$ *we add a directed edge* (v', v'') *with capacity* $c_V(v', v'') = c(v)$

(iii) $\forall (u, v) \in E$ *we add two directed edges* (u'', v') *and* (v'', u') *with capacity* $c_V(u'', v') = c_V(v'', u') = \infty$.

The vertex seperator problem and transformation $T_V(G)$ are illustrated in Figure 3. Obviously no edge between two adjacent nodes can be in a minimum-capacity (s,t)-cutset of $T_V(G)$, because for all those edges the capacity is ∞. Therefore, the cutset must consist of edges of the form (v', v''). A minimum-weight (s,t)-vertex seperator can be calculated by finding a maximum flow in $T_V(G)$, finding the minimum-capacity (s,t)-cutset with the procedure described above and then map each cut edge (v', v'') to their corresponding node v.

TODO 2: *Describing multi-source and multi-sink flow problems*

3.2 Modelling Flows on Hypergraphs

3.2.1 Model Hypergraphs as Graphs

3.2.2 Transforming Hypergraphs to Flow Networks

Let's consider the *bipartite graph* representation G of a hypergraph H [12]. Each hyperedge e is modeled as a star node with an edge to all $v \in e$. Hu and Moerder [12] introduced node capacities in G. Each hyperedge node e has a capacity equal to $\omega(e)$ and each hypernode node infinity capacity. Further, they showed that a minimum-weight (s,t)-vertex seperator in G is equal with a minimum-weight (s,t)-cutset of a hypergraph H. Finding such a seperator is a flow problem and can be calculated with the flow network $T_L(H)$ presented by Lawler [14]:

Definition 3.3. *Let T_L be the transformation of a hypergraph $H = (V, E, c, \omega)$ into a flow network $T_L(H) = (V_L, E_L, c_L)$ proposed by Lawler [14]. $T_L(H)$ is defined as follows:*

(i) $V_L = V \cup \bigcup_{e \in E} \{e', e''\}$

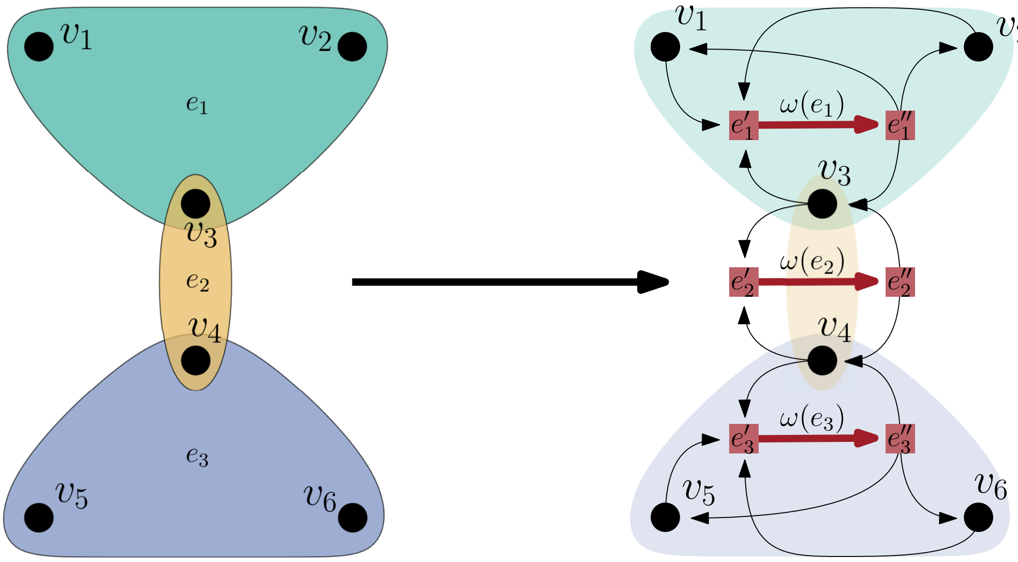

Figure 4: Transformation of a hypergraph into a equivalent flow network by Lawler [14]. Note, capacity of the black edges in the flow network is ∞.

(ii) $\forall e \in E$ *we add a directed edge* (e', e'') *with capacity* $c_H(e', e'') = \omega(e)$

(iii) $\forall v \in V$ *we add two directed edges* (v, e') *and* (e'', v), $\forall e \in I(v)$ *with capacity* $c_L(v, e') = c_L(e'', v) = \infty$.

An example of this transformation is shown in Figure 4. $T_L(H)$ is nearly equivalent to the transformation $T_V(G)$ described in Defintion 3.2 except that we do not have to split the hypernodes $v \in V$ into (v', v''). This is due to the fact, that a hypernode cannot be in a minimum-capacity (s,t)-vertex seperator, because each $v \in V$ has infinity capacity [12]. Therefore, a minimum-capacity (s,t)-cutset in $T_L(H)$ is equal to a minimum (s,t)-vertex seperator G. The resulting graph $T_L(H)$ has $|V_L| = 2|V| + |E|$ nodes and $|E_L| = 2(\bar{e}+1)|E|$ edges, where $\bar{e}$ is the average size of a hyperedge [16]. Using *Edmond-Karps* maximum flow algorithm (see Section 3.1.1) on flow network $T_L(H)$ takes time $\mathcal{O}(|V|^2|E|^2)$ [14].
A minimum-weight (s,t)-cutset of H can be found by simlpy mapping the minimum-capacity (s,t)-cutset to their corresponding hyperedges in H (see Section 3.1.3). The corresponding bipartition are all hypernodes $v \in V$ *reachable* from s in the *residual graph* of $T_L(H)$ and the counterpart are all hypernodes not *reachable* from s (**TODO 3:** *proof? reference?*).
In this thesis we often have to mix up nodes and edges of H and $T_L(H)$. If we use $v \in V_L$, there also exist a corresponding $v \in V$. v can be used in both contexts. For all $e \in E$ there exists two corresponding nodes $e', e'' \in V_L$. e' is called *incomming hyperedge node* and e'' is called *outgoing hyperedge node*. In some cases we need to treat $e', e'' \in V_L$ the same way as their corresponding hyperedge $e \in E$. E.g. $e_1' \cap e_2'$ or $e_1'' \cap e_2'$ should be the same as $e_1 \cap e_2$. However, it should be clear out of the context which terminology is used.

3.2.3 Implicit Flow Calculation on Hypergraphs

3.3 Max-Flow-Min-Cut Based Local Search on Graphs

It seems naturals to utilize maximum flow computations to improve the cut metric of a given partition of a graph. Lang and Rao [13] uses an approach, called *Max-Flow Qoutient-cut Improvement* (MQI), to improve the cut of a graph when metrics such as *expansion* or *conductance* are used. For a given bipartition $(S, \bar{S})$, they find the best improvement among all bipartitions

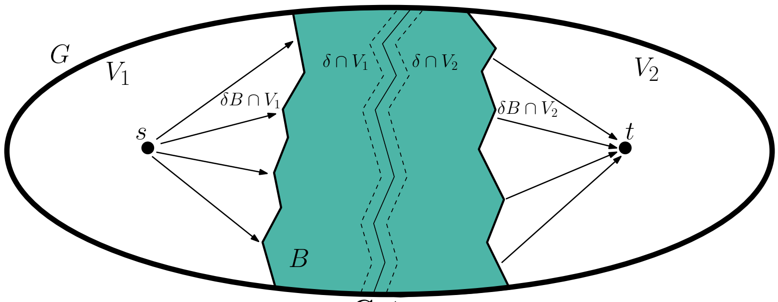

Figure 5: Illustration of setting up a flow problem around the cut of graph G [2].

$(S', \bar{S}')$ such that $S' \subset S$ by constructing a flow problem. Andersen and Lang [2] proposed a flow-based improvement algorithm, called *Improve*, which works similiar as MQI, but did not restrict the output of the partition on $S' \subset S$. However, both techniques can not guarantee that the resulting bipartition is balanced and only are applicable for $k = 2$.
Schulz and Sanders [18] integrate flow-based approaches in their *multilevel graph partitioning* framework *KaFFPa*. In general, they build a flow problem around a region B of the cut and connect the *border* of B with the source resp. sink. B is defined in such a way that the flow computation yields to a feasible cut in the original graph. Many ideas of this work are used in this thesis and adapted to hypergraphs. Therefore, we will give a detailed description of the concepts and advanced techniques to improve the cut of a graph.

TODO 4: *define expansion and conductance in preliminaries for graphs*

3.3.1 Balanced Flow-Based Bipartitioning

Let (V_1, V_2) be a balanced bipartition of a graph $G = (V, E, c, \omega)$. Further, $P(v) = 1$, if $v \in V_1$ and $P(v) = 2$, otherwise. We will now explain how to improve a given bipartition with flow computations. This technique can also be applied on a k-way partition by applying the approach on two adjacent blocks in the qoutient graph [18].
Let $\delta := \{u \mid \exists (u, v) \in E : P(u) \neq P(v)\}$ be the set of nodes around the cut of G. For a set $B \subseteq V$ we define its border $\delta B := \{u \in B \mid \exists (u, v) \in E : v \notin B\}$ and the induced subgraph $G(B) = (B, E_B, c, \omega)$ with $E_B = \{(u, v) \in E \mid u, v \in B\}$. The basic idea is to build a flow problem $G(B)$ around all cut nodes δ of G and connect all nodes in $\delta B \cap V_1$ to a source node s and all nodes in $\delta B \cap V_2$ to a sink node t.
We can construct $B := B_1 \cup B_2$ with two *Breadth First Searches* (*BFS*). One is initialized with all nodes $\delta \cap V_1$ and stops, if $c(B_1)$ would exceed $(1 + \epsilon)\frac{c(V)}{2} - c(V_2)$. The second is initialized with all nodes $\delta \cap V_2$ and stops, if $c(B_2)$ would exceed $(1 + \epsilon)\frac{c(V)}{2} - c(V_1)$. The two *BFS* only touch nodes of V_1 resp. $V_2 \Rightarrow B_1 \subseteq V_1$ and $B_2 \subseteq V_2$. The constraints for the weights of B_1 and B_2 guarantees that the bipartition is still balanced after a *Max-Flow-Min-Cut* computation. Connecting s resp. t to all border nodes $\delta B \cap V_1$ resp. $\delta B \cap V_2$ ensures that a non-cut edge not contained in $G(B)$ is not a cut edge after assigning the *Min-Cut* of $G(B)$ to G. This also yields to the conclusion that each minimum (s, t)-cutset in $G(B)$ leads to a cut smaller or equal to the old cut of G. All concepts are illustrated in Figure 5.

TODO 5: *define qoutient graph and subgraph in preliminaries*

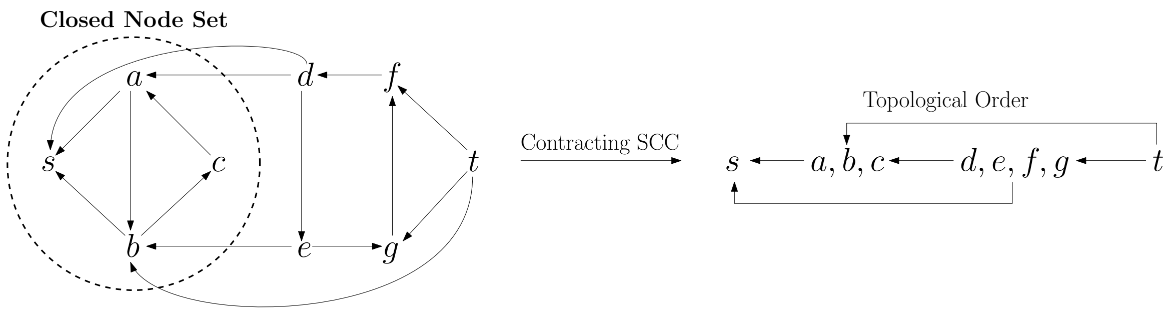

Figure 6: Nodes $C = \{s, a, b, c\}$ illustrates a *closed node set* in a graph G (left side). After contracting all *Strongly Connected Components*, we can enumerate all *closed node sets* of G by sweeping in reverse topological order to the contracted graph (right side).

3.3.2 Adaptive Flow Iterations

Sanders and Schulz [18] suggested several heuristics to improve their basic approach. If the *Max-Flow-Min-Cut* computation on $G(B)$ leads to an improvement in cut, we can apply the method described in Section 3.3.1 again. An extension of this approach is to iteratively adapt the size of the flow problem based on the result of the maximum flow computation. For this propose we define $\epsilon' := \alpha\epsilon$ for a $\alpha \geq 1$ and let the size of B depend on ϵ' rather than on ϵ. If we found an improvement on G, we increase α to $\min\{2\alpha, \alpha'\}$ where α' is a predefined upper bound for α. If not, we decrease the size of α to $\max\{\frac{\alpha}{2}, 1\}$. This approach is called *adaptive flow iterations* [18].

3.3.3 Most Balanced Minimum Cut

Picard and Queyranne [15] showed that all minimum (s,t)-cutsets are computable with one maximum (s,t)-flow computations. To understand the main theorem and the algorithm to compute all minimum (s,t)-cutsets we need the definition of a *closed node set* $C \subseteq V$ of a graph G.

Definition 3.4. *Let $G = (V, E)$ be a graph and $C \subseteq V$. C is called a closed node set iff the condition $u \in C$ implies that for all edges $(u, v) \in E$ also $v \in C$.*

A *closed node set* is illustrated in Figure 6. A simple observation is that all nodes on a cycle have to be in the same *closed node set* per definition. Therefore, we can contract all *Strongly Connected Components* (SCC) of G with a linear time algorithm proposed by Tarjan (**TODO 6:** *reference see [18]*) and sweep to the reverse topological order of the contracted graph to enumerate all *closed node sets*. Note, if we contract all SCC of G the resulting graph is a *Directed Acyclic Graph* (DAC), therefore a topological order exists. With the Theorem of Picard and Queyranne [15] we are able to enumerate all minimum (s,t)-cuts of G with one maximum flow computation.

Theorem 3.2. *There is a* 1-1 *correspondence between the minimum (s,t)-cuts of a graph and the closed node sets containing s in the residual graph of a maximum (s,t)-flow.*

All *closed node sets* in the residual graph of G induced a minimum (s,t)-cutset on G. They can be calculated with the algorithm described above having the residual graph of G as input. The running time of the algorithm is $\mathcal{O}(|V| + |E|)$.
A common problem of the *adaptive flow iteration* approach (see Section 3.3.2) is that searching

with a large α often leads to cuts in G which violates the balanced constraints. We are able with this technique to convert a infeasible solution into a feasible by finding the *Most Balanced Minimum Cut* (MBMC) with one maximum flow computation.

TODO 7: *define contraction, strongly connected components and cycles in graph in preliminaries*

3.3.4 Active Block Scheduling

Active Block Scheduling is a *quotient graph style refinement* technique for k-way partitions [11, 18]. The algorithm is organized in rounds and executes a two-way local improvement algorithm on each pair of blocks in the *quotient graph* where at least one of both is *active*. Initial all blocks are *active*. A block becomes *inactive*, if its boundary did not change in a round. The algorithm terminates, if all blocks are *inactive*.

Fiduccia and Mattheyses [5] introduces a linear time two-way local search heuristic, called *FM*, which is fundamental for many graph partitioning algorithms. They define the gain $g(v)$ of a node $v \in V$ as the reduction of the cut metric, when moving v from its current block to its counterpart block. By maintaining the gains of the nodes in a special datastructure, called *bucket queue*, they are able to find a maximum gain node in constant time. After moving a maximum gain node they are also able to update the datastructure in time equal to the number of adjacent nodes.

The local improvement algorithm (for *Active Block Scheduling*) can either be a *FM* local search or a flow-based approach or even a combination of boths as proposed by Sanders and Schulz [18].

TODO 8: *integrate improvement table for alpha*

3.4 Hypergraph Partitioning

3.4.1 Multilevel Paradigm

3.4.2 KaHyPar - n-Level Hypergraph Partitioning

4 Optimized Approach on Modelling Flows in Hypergraphs

In Section 3.2.2 we have shown how a hypergraph H could be transformed into a flow network $T_L(H)$ such that every minimum-weight (S,T)-cutset in H is a minimum-capacity (S,T)-cutset in $T_L(H)$ [14]. However, the resulting flow network has significantly more nodes and edges than the original hypergraph. Finding a (S,T)-maximum flow is usually a very computation intensive problem. Therefore, different modelling approaches, which reduce the number of nodes and edges, can have a crucial impact on the running time of the flow algorithm.
We will present techniques to sparsify the flow network propose by Lawler. First, we will show how any subset $V' \subseteq V$ of hypernodes could be removed from $T_L(H)$ (see Section 4.1). This approach minimizes the number of nodes, but in some cases the number of edges can be significantly higher than in $T_L(H)$. But the basic idea of this technique can still be applied to remove low degree hypernodes from the *Lawler*-Network without increasing the number of edges (see Section 4.2). Additionally, we show how we can remove every hyperedge e of size 2 by an undirected flow edge between the corresponding nodes $v_1, v_2 \in e$, which further reduce the number of nodes and edges (see Section 4.3). Finally, we combine the two suggested approaches in a *Hybrid*-Network (see Section 4.4).

4.1 Removing Hypernodes via Clique-Expansion

In this Section we show how we can remove all hypernodes of $T_L(H)$. If a hypernode $v \in V$ occurs in an augmenting path P the previous node in the path must be a hyperedge node either e' or e''. Further, for all $e \in I(v)$ the capacity $c_L(v, e')$ is ∞. This leads to the conclusion, if we push flow over a hypernode v, comming from a hyperedge node, we can redirect the flow to any hyperedge node $e' \in I(v)$ during the whole maximum flow calculation, because $c_L(v, e') = \infty$. A hypernode v acts as a *bridge* between all incident hyperedges in the *Lawler*-Network. Therefore, the idea is to remove all hypernodes from $T_L(H)$ and instead inserting for all $v \in V$ a clique between all $e_1, e_2 \in I(v)$ with $e_1 \neq e_2$. In the following we will define our new network more general and show how to remove any $V' \subseteq V$.

Definition 4.1. *Let T_H be a transformation that converts a hypergraph $H = (V, E, c, \omega)$ into a flow network $T_H(H, V') = (V_H, E_H, c_H)$, where $V' \subseteq V$. $T_H(H, V')$ is defined as follows:*

(i) $V_H = V \setminus V' \bigcup_{e \in E} \{e', e''\}$

(ii) *$\forall v \in V'$ we add a directed edge (e''_1, e'_2), $\forall e_1, e_2 \in I(v)$ with $e_1 \neq e_2$ with capacity $c_H(e''_1, e'_2) = \infty$ (clique expansion).*

(iii) *$\forall e \in E$ we add a directed edge (e', e'') with capacity $c_H(e', e'') = \omega(e)$ (same as in $T_L(H)$).*

(iv) *$\forall v \in V \setminus V'$ we add for each incident hyperedge $e \in I(v)$ two directed edges (v, e') and (e'', v) with capacity $c_H(v, e') = c_H(e'', v) := \infty$ (same as in $T_L(H)$).*

An example of the transformation is shown in Figure 7. To show the correctness of $T_H(H, V')$, we need to proof that a minimum-capacity (S,T)-cutset in $T_H(H, V')$ is equal with a minimum-weight (S,T)-cutset in H. However, in the correctness proof we need a preparing lemma.

Lemma 4.1. *Let $G = (V, E, c)$ be a graph with a capacity function $c : E \to \mathbb{R}_+$. Further, let S and T be a source and sink set with $S \cap T = \emptyset$ and $\forall s \in S : \forall (s, v) \in E : c(s, v) = \infty$ and $\forall t \in T : \forall (v, t) \in E : c(v, t) = \infty$.*
For any $V' \subseteq V$ a minimum-capacity (S,T)-cutset in G is equal with a minimum-capacity (S', T')-cutset in G, where $S' = S \setminus V' \cup \bigcup_{s \in I(V' \cap S)} \{s'\}$ and $T' = T \setminus V' \cup \bigcup_{t' \in I(V' \cap T)} \{t'\}$ and $S' \cap T' = \emptyset$.

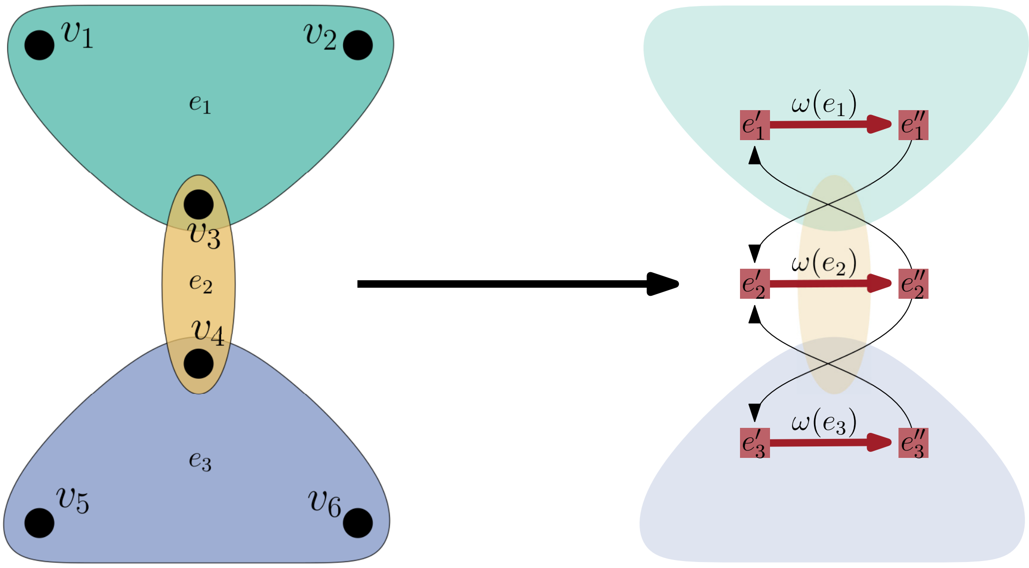

Figure 7: Transformation of a hypergraph into a equivalent flow network by removing all hypernodes. Note, capacity of the black edges in the flow network is ∞.

Proof. Let G' be the graph obtained by removing all $v \in V' \cap (S \cup T)$. If the minimum-capacity (S,T)-cutset in G is smaller than ∞, then no outgoing edge of a node $s \in S$ and no incomming edge of a node $t \in T$ can be cut, because for all those edges e the capacity $c(e) = \infty$. If $S' \cap T' = \emptyset$ every minimum-capacity (S,T)-cutset in G is equal with a minimum-capacity (S',T')-cutset in G'. Each (S,T)-cutset in G is also a (S',T')-cutset in G' and vice versa. If the minimum-capacity (S,T)-cutset in G is ∞, every cutset seperating (S,T) resp. (S',T') is a minimum (S,T)- resp. (S',T')-cutset. □

The conclusion of this lemma is, if we want to determine a minimum-capacity (S,T)-cutset in $T_L(H)$ (with $S,T \subseteq V$), we can e.g. remove any $s \in S$ resp. $t \in T$ (and even from $T_L(H)$) and instead add all incident hyperedges $e' \in I(s)$ resp. $e'' \in I(t)$ as sources resp. sinks. The resulting minimum-capacity (S',T')-cutset in $T_L(H)$ is equal with a minimum-capacity (S,T)-cutset in $T_L(H)$.

Theorem 4.1. *A minimum-weight (S,T)-cutset of a hypergraph $H = (V,E,c,\omega)$ (with $S,T \subseteq V, S \cap T = \emptyset$) is equivalent with a minimum-capacity (S',T')-cutset of the flow network $T_H(H,V') = (V_H,E_H,c_H)$ ($V' \subseteq V$), where $S' = S \setminus V' \cup \bigcup_{e \in I(V' \cap S)} \{e'\}$ and $T' = T \setminus V' \cup \bigcup_{e \in I(V' \cap T)} \{e''\}$.*

Proof. Let's consider again the bipartite graph representation $G_L = (V_L,E_L,c_L)$ of a hypergraph $H = (V,E,c,\omega)$ presented in Section 3.2.2, where for all $v \in V : c_L(v) = \infty$ and for all $e \in E : c_L(e) = \omega(e)$. A minimum-weight (S,T)-vertex seperator in G_L is equal with a minimum-weight (S,T)-cutset in H. A minimum-weight (S,T)-vertex seperator can be calculated by finding a minimum-capacity (S,T)-cutset in $T_L(H)$. Let G_H be the graph obtained by removing all $v \in V' \setminus (S \cup T)$ of G_L and insert a clique between all $e \in I(v)$. A minimum-weight (S,T)-vertex seperator in G_H can be calculated by finding a minimum-capacity (S,T)-cutset in our new network $T_H(H, V' \setminus (S \cup T))$. We will show that each vertex seperator in G_L is also a vertex seperator in G_H and vice versa. Finally, with Lemma 4.1 follows our assumption. We will denote a vertex seperator of a graph G with $V_S(G)$ and define $V'' := V' \setminus (S \cup T)$. We will show, that $V_S(G_L) = V_S(G_H)$ with the restriction $V_S(G_L) \subseteq E$ and $V_S(G_H) \subseteq E$.

Let's assume, that $V_S(G_L) \subseteq E$ is not a vertex seperator in G_H. After removing all $e \in V_S(G)$ in G_H, there exists still a path $P_H = \{v_1, \ldots, v_k\}$ with $v_1 \in S$ and $v_k \in T$ in G_H. We can extend P_H to a path P_L in G_L. We define $P_L := P_H$ and replaces every occurence of a sequence $v_i = e_1 \in E$ and $v_{i+1} = e_2 \in E$ with a triple (e_1, v, e_2) in P_L, where $v \in e_1 \cap e_2 \cap V''$ (not empty per construction). P_L not contain any vertex of $V_S(G_L)$, because we removed all $v \in V_S(G_L)$ from G_H and only hypernodes are added to P_L. P_L connects S and T in G_L, which is a contradiction that $V_S(G_L)$ is a vertex seperator in G_L.

Let's assume, that $V_S(G_H) \subseteq E$ is not a vertex seperator in G_L. After removing all $e \in V_S(G_H)$ in G_L, there exists still a path $P_L = \{v_1, \ldots, v_k\}$ with $v_1 \in S$ and $v_k \in T$ in G_L. We can extend P_L to a path P_H in G_H. We define $P_H := P_L$ and remove all $v \in P_L \cap V''$ from P_H. G_L is a bipartite graph per definition. Therefore, every path P_L in G_L is an alternating path of hypernodes and hyperedges. The predecessor and successor of a hypernode $v \in P_L \cap V''$ must be hyperedges e_1 and e_2. If $v \in V''$, then v is not contained G_H. Instead, there is a clique between all $e \in I(V) \Rightarrow (e_1, e_2)$ is contained in G_H. P_H not contain any vertex of $V_S(G_H)$, because we removed all $v \in V_S(G_H)$ from $V_S(G_L)$ and we removed only hypernodes from P_L. P_H connects S and T in G_H, which is a contradiction that $V_S(G_H)$ is a vertex seperator in G_H.

A minimum-weight (S,T)-vertex seperator in G_L and G_H only contains hyperedges, because the weight of all hypernodes in G_L and G_H is ∞. Therefore, each minimum (S,T)-vertex seperator in G_L is also a minimum-weight (S,T)-vertex seperator in G_H, because $V_S(G_L) = V_S(G_H)$. With Lemma 4.1 follows that we can calculate a minimum-weight (S,T)-vertex seperator in G_L resp. G_H by calculating a minimum-capacity (S',T')-cutset in $T_L(H)$ resp. $T_H(H,V')$. Therefore, there exists a equivalence between a minimum-weight (S,T)-cutset E_{min} of H and the following statements:

E_{min} is a minimum-…

(i) …-weight (S,T)-cutset in H

(ii) …-weight (S,T)-vertex seperator in G_L

(iii) …-capacity (S,T)-cutset in $T_L(H)$

(iv) …-capacity (S',T')-cutset in $T_L(H)$ (follows from (iii) with Lemma 4.1)

(v) …-weight (S,T)-vertex seperator in G_H

(vi) …-capacity (S,T)-cutset in $T_H(H,V'')$

(vii) …-capacity (S',T')-cutset in $T_H(H,V')$ (follows from (vi) with Lemma 4.1)

□

As a consequence of this Theorem a minimum-weight (S,T)-cutset of H can be calculated with $T_H(H,V')$ the same way as with $T_L(H)$ (see Section 3.2.2). A open problem is how to obtain the corresponding bipartition. In $T_L(H)$ all hypernodes reachable from nodes in S are part of the first and all not reachable are part of the second partition. Since we delete all nodes $v \in V'$ from $T_L(H)$ in $T_H(H,V')$ this relationship is no longer valid.

Lemma 4.2. *Let f be a maximum (S,T)-flow and A be the set of all nodes reachable from a node $s \in S$ in the residual graph of $T_L(H)$.*

$$\text{If } v \in A \Leftrightarrow \exists e \in I(v) : e'' \in A$$

Proof. If $e'' \in A$, then $v \in A$, because $c_L(e'', v) = \infty$ and $r_f(e'', v) = \infty$. Let's assume, if $v \in A$, then $\forall e \in I(v) : e'' \notin A \Rightarrow f(e'', v) = 0$. Otherwise $r_f(v, e'')$ would be greater than zero and this would imply $e'' \in A$. Each path P in the *residual graph* of $T_L(H)$ from $s \in S$ to v must be of the form $P = (\ldots, e', v)$. For at least one $e \in I(v)$ there must be a positive flow $f(v, e') > 0$, otherwise edge (e', v) is not in the *residual graph* of $T_L(H)$ $(c_L(e', v) = 0)$. There is a positive flow leaving node v, but there is no flow entering node v, because $\forall e \in I(v) : f(e'', v) = 0$. This violates the conservation of flow constraint for node v and therefore f is not a valid flow function. There must exist at least one $e \in I(v)$ with $f(e'', v) > 0 \Rightarrow r_f(v, e'') > 0 \Rightarrow e'' \in A$. □

Lemma 4.2 gives us an alternative construction technique for the minimum-weight (S,T)-bipartition of H with both networks $T_L(H)$ and $T_H(H,V')$. Regardless of the flow network, we can calculate a maximum flow on it and define the set E'', which contains all *outgoing hyperedge nodes* e'' *reachable* from a source node $s \in S$ in the *residual graph* of the flow network. Further, we define $A = \bigcup_{e \in E''} e$, then $(A, V \setminus A)$ is a minimum-weight (S,T)-bipartition of H.

4.2 Removing Low-Degree Hypernodes

The resulting flow network $T_H(H,V)$ proposed in Section 4.1 has significantly less nodes than the network $T_L(H)$ suggested by Lawler. On the other hand, the number of edges can be much larger.
Let's consider a hypernode $v \in V$. We replace v in $T_L(H)$ with a clique between all hyperedges of $I(v)$. The number of edges inserted in $T_H(H,V)$ depends on the degree of v. Every hypernode $v \in V$ induce $d(v)(d(v)-1)$ edges in $T_H(H,V)$. In $T_L(H)$ a hypernode adds $2d(v)$ edges to the network with the drawback of an additional node. A simple observation is that for all hypernodes with $d(v) \leq 3$ the inequality $d(v)(d(v)-1) \leq 2d(v)$ is satisfied. Removing such low degree hypernodes not only reduce the number of nodes, but also the number of edges.
Let $V_d(n) = \{v \mid v \in V \ \wedge \ d(v) \leq n\}$ be the set of all hypernodes with degree smaller or equal n. Then our suggested flow network is $T_H(H, V_d(3))$.

(i) **TODO 9:** *Think about removing node via Clique Expansion, e.g.* $e_1 \cap e_2 = \{v_1, v_2\}$

4.3 Removing Hyperedges via Undirected Flow-Edges

If we want to find a minimum-weight (S,T)-cutset in a graph $G = (V,E,\omega)$, we do not need to transform G into a equivalent flow network. We can directly operate on the graph with capacities $c(e) = \omega(e)$ for all $e \in E$ [6]. Hypergraphs are generalizations of graph, where an edge can consist of more than two nodes. However, a hyperedge e of size 2 can still be interpreted as a graph edge. Instead of modelling those edges as described by Lawler [14] (see hyperedge e_2 in Figure 4), we can remove all e', e'' for all $e \in E$ with $|e| = 2$ and add an undirected flow edge between $v_1, v_2 \in e$ (with $v_1 \neq v_2$) with capacity $c(\{v_1, v_2\}) = \omega(e)$.

Definition 4.2. *Let T_G be a transformation that converts a hypergraph $H = (V,E,c,\omega)$ into a flow network $T_G(H) = (V_G, E_G, c_G)$. $T_G(H)$ is defined as follows:*

(i) $V_G = V \cup \bigcup_{\substack{e \in E \\ |e| \neq 2}} \{e', e''\}$

(ii) $\forall e \in E$ with $|e| = 2$ and $v_1, v_2 \in e$ ($v_1 \neq v_2$) we add two directed edges (v_1, v_2) and (v_2, v_1) to E_G with capacity $c(v_1, v_2) = \omega(e)$ and $c(v_2, v_1) = \omega(e)$

(iii) Let $H' = (V, E', c, \omega)$ be the hypergraph with $E' = \{e \mid e \in E \wedge |e| \neq 2\}$, then we add all edges of $T_L(H')$ to E_G with their corresponding capacities.

An example of transformation $T_G(H)$ is shown in Figure 8. A hyperedge e of size 2 consists in $T_L(H)$ exactly of 4 nodes and 5 edges (see Figure 4). The same hyperedge induce 2 nodes and 2 edges in $T_G(H)$.

Theorem 4.2. *A minimum-weight (S,T)-cutset of a hypergraph $H = (V,E,c,\omega)$ (with $S, T \subseteq V, S \cap T = \emptyset$) is equal with a minimum-capacity (S,T)-cutset of the flow network $T_G(H) = (V_G, E_G, c_G)$.*

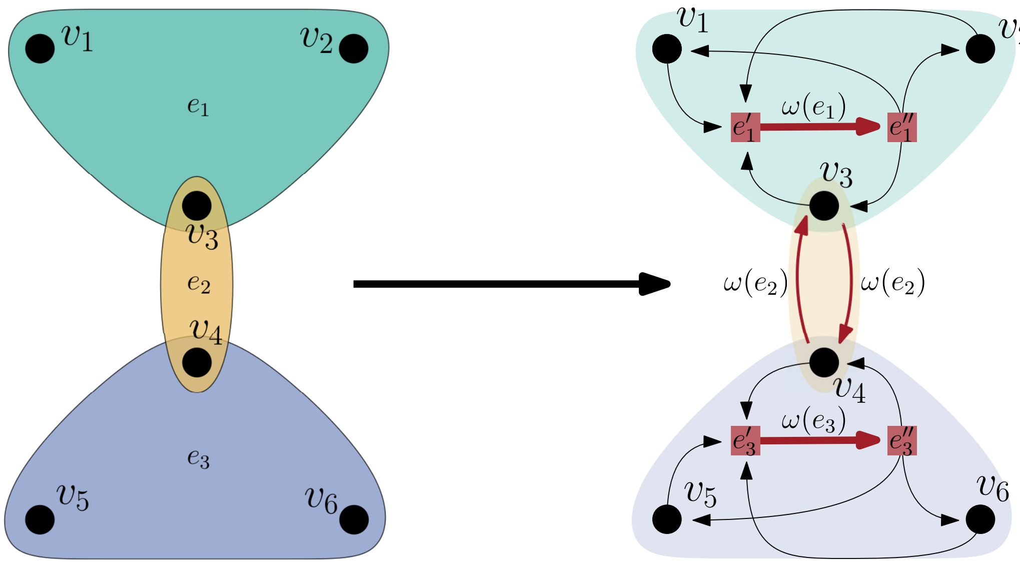

Figure 8: Transformation of a hypergraph into a equivalent flow network by inserting an undirected edge with capacity $\omega(e)$ for each hyperedge of size 2. Note, capacity of the black edges in the flow network is ∞.

Proof. We define a bijective function $\Phi : E_L \to E_G$ as follows

$$\Phi(e', e'') = \begin{cases} (e', e''), \text{if } |e| \neq 2, \\ \{v_1, v_2\}, \text{otherwise (with } v_1, v_2 \in e \text{ and } v_1 \neq v_2) \end{cases}$$

We will show that each (S,T)-cutset A_L in $T_L(H)$ is a (S,T)-cutset $\Phi(A_L)$ in $T_G(H)$ and vice versa. Per defintion $c_L(A_L) = c_G(\Phi(A_L))$ and for each (S,T)-cutset A_G in $T_G(H)$ $c_G(A_G) = c_L(\Phi^{-1}(A_G))$. Therefore, each minimum-capacity (S,T)-cutset in $T_L(H)$ must be a minimum-capacity (S,T)-cutset in $T_G(H)$ and vice versa. In the following let $E^* = \bigcup_{e \in E}\{(e', e'')\}$.
Let $A_L \subseteq E^*$ be a (S,T)-cutset in $T_L(H)$. Let's assume $\Phi(A_L)$ is not a (S,T)-cutset in $T_G(H)$ after removing all edges $e \in \Phi(A_L)$ from $T_G(H)$. Then there exists a path $P_G = \{v_1, \ldots, v_k\}$ connecting S and T in $T_G(H)$ not containing any edge $e \in \Phi(A_L)$. Let P_L be the path in $T_L(H)$ obtained by inserting edge $\Phi^{-1}(v_i, v_{i+1})$ between all $v_i = v_1 \in V$ and $v_{i+1} = v_2 \in V$ into P_G. $\Phi^{-1}(v_i, v_{i+1}) \notin A_L$, otherwise we would had removed edge (v_i, v_{i+1}) from $T_G(H)$. P_G connects S and T in $T_G(H) \Rightarrow P_L$ connects S and T in $T_L(H)$, which is a contradiction to the assumption that A_L is a (S,T)-cutset.
Let $A_G \subseteq \Phi(E^*)$ be a (S,T)-cutset in $T_G(H)$. Let's assume $\Phi^{-1}(A_G)$ is not a (S,T)-cutset in $T_L(H)$ after removing all edges $e \in \Phi^{-1}(A_G)$ from $T_L(H)$. Then there exists a path $P_L = \{v_1, \ldots, v_k\}$ connecting S and T in $T_L(H)$ not containing any edge $e \in \Phi^{-1}(A_G)$. Let P_G be the path in $T_G(H)$ obtained by removing each edge (v_i, v_{i+1}) with $v_i = e'$ and $v_{i+1} = e''$ and $|e| = 2$ from P_L. Based on the construction of $T_L(H)$ the predecessor of v_i and successor of v_{i+1} must be a hypernode. Therefore, P_G is a valid path in $T_G(H)$ connecting S and T, which not contains any edge in A_G. This is a contradiction to the assumption that A_G is a (S,T)-cutset. □

A minimum-weight (S,T)-cutset of H can be calculated in $T_G(H)$ the same way as in $T_L(H)$. Each edge (v_1, v_2) with $v_1, v_2 \in V$ in the minimum-capacity (S,T)-cutset of $T_G(H)$ can be mapped to their corresponding hyperedge with $\Phi^{-1}(v_1, v_2)$. Since their exists a one-one correspondence between the hypernodes of $T_L(H)$ and $T_G(H)$ the corresponding bipartion are all hypernodes *reachable* from a node in S and all not *reachable* from S in the *residual graph* of $T_G(H)$.

4.4 Combining Techniques in a Hybrid Flow Network

On many real world instances the average hyperedge size and hypernode degree are inversely proportional to each other. E.g., if the number of hyperedges is much greater than the number of hypernodes the average hypernode degree is usually much larger than 3. Whereas the average hyperedge size is often equal to 2. If the number of hyperedges is nearly equal to the number of hypernodes the average hypernode degree is usually smaller or equal than 3. Whereas the average hyperedge size is often much larger than 2. Of course, we can construct instances where this inversely proportional relationship can not be observed, but on real world instances we often find the described behaviour.
Currently, we have two differenct modelling approaches which either perform better on low hypernode degree instances or on small hyperedge size instances. Taking our observation from real world instances into account means that either $T_G(H)$ or $T_H(H, V_d(3))$ performs significantly better on a specific real world instance. It would be preferable to combine the two approaches into one network which performs on the most instances best.

Definition 4.3. *Let T_{Hybrid} be a transformation that converts a hypergraph $H = (V, E, c, \omega)$ into a flow network $T_{Hybrid}(H, V') = (V_{Hybrid}, E_{Hybrid}, c_{Hybrid})$, where $V' = \{v \mid v \in V_d(3) \ \wedge \ \forall e \in I(v) : |e| \neq 2\}$. $T_{Hybrid}(H, V')$ is defined as follows:*

(i) $V_{Hybrid} = V \setminus V' \bigcup\limits_{\substack{e \in E \\ |e| \neq 2}} \{e', e''\}$

(ii) *$\forall v \in V'$ we add a directed edge (e_1'', e_2'), $\forall e_1, e_2 \in I(v)$ with $e_1 \neq e_2$ with capacity $c_{Hybrid}(e_1'', e_2') = \infty$ (clique expansion).*

(iii) *$\forall e \in E$ with $|e| = 2$ and $v_1, v_2 \in e$ $(v_1 \neq v_2)$ we add two directed edges (v_1, v_2) and (v_2, v_1) with capacity $c_{Hybrid}(v_1, v_2) = \omega(e)$ and $c_{Hybrid}(v_2, v_1) = \omega(e)$*

(iv) *$\forall e \in E$ with $|e| \neq 2$ we add a directed edge (e', e'') with capacity $c_{Hybrid}(e', e'') = \omega(e)$ (same as in $T_L(H)$).*

(v) *$\forall v \in V \setminus V'$ we add for each incident hyperedge $e \in I(v)$ with $|e| \neq 2$ two directed edges (v, e') and (e'', v) with capacity $c_{Hybrid}(v, e') = c_{Hybrid}(e'', v) := \infty$ (same as in $T_L(H)$).*

In Figure 9 all explained transformations of this section are illustrated. The proof of Theorem 4.2 can be used one-to-one to show that a minimum-capacity (S', T')-cutset of $T_H(H, V')$ is equal with a minimum-capacity (S', T')-cutset of $T_{\text{Hybrid}}(H, V')$ (for definition of S' and T' see Theorem 4.1). It follows with Lemma 4.1 that this is equal with a minimum-weight (S, T)-cutset of H.
In the definition of $T_{\text{Hybrid}}(H, V')$ we prefer a hyperedge removal over a hypernode removal. E.g., if a hypernode has a degree smaller or equal than 3, we only remove it, if there is no hyperedge $e \in I(v)$ with $|e| = 2$. The reason is that a hyperedge removal always decrease the number of nodes and edges more than a hypernode removal.
The minimum-weight (S, T)-cutset of H can be calculated with the same technique described in Section 4.3. Let's define with $(A, V \setminus A)$ the corresponding bipartition. A is the union of all reachable hypernodes from S' and the union of all reachable *outgoing hyperedge nodes e''* from S' (see Section 4.1 and Lemma 4.2).

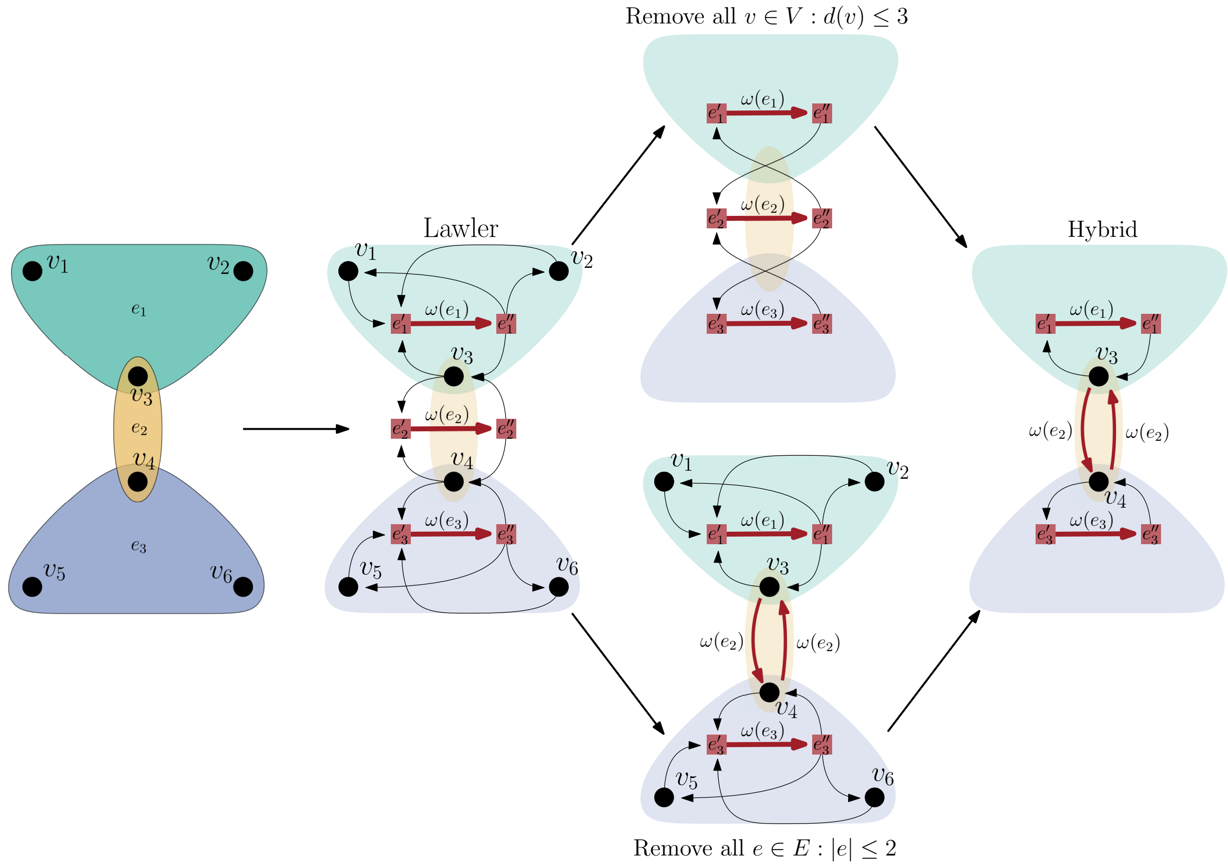

Figure 9: Illustration of all presented transformations of a hypergraph into a flow network.

5 Using Max-Flow-Min-Cut Computations as a Local Search Strategy

We will give now a detailed description of our flow-based refinement framework. The main idea is to extract a subhypergraph $H_{V'}$ out of the original hypergraph H, which is already partitioned into k blocks. V' is choosen in such a way that it is a subset of two adjacent blocks V_i and V_j. We will show how to configure the sources S and sinks T of the corresponding flow network such that a minimum (S,T)-bipartition of $H_{V'}$ improves the connectivity metric on H (see Section 5.1). Further, we describe how the ideas of Sanders and Schulz [18] (see Section 3.3) could be adapted to work in a n-level hypergraph partitioner, called *KaHyPar* (see Section 5.2 and 5.3).

5.1 Modelling Sources and Sinks

Let $H = (V, E, c, \omega)$ be a hypergraph and $B_1 := (V_1, V_2)$ be a bipartition. $H_{V'} = (V', E_{V'}, c, \omega)$ is the subhypergraph induced by $V' \subseteq V$ with $E_{V'} = \{e \cap V' \mid e \in E : e \cap V' \neq \emptyset\}$. Further, let $E_\emptyset = \{e \cap V' \mid e \in E : e \cap V' = \emptyset\}$ be the set of all hyperedges contained in H, but not in $H_{V'}$. $T_L(H_{V'})$ (see Section 3.2.2) is the flow problem induced by $H_{V'}$ with a source set S and a sink set T. Let (V'_1, V'_2) be the minimum (S,T)-bipartition obtained by a maximum (S,T)-flow calculation on $T_L(H_{V'})$ with f as maximum flow function. We can extend the bipartition (V'_1, V'_2) of $H_{V'}$ to a bipartition $B_2 := (V_1 \setminus V' \ \cup \ V'_1, \ V_2 \setminus V' \ \cup V'_2)$ on H. Finally, we need to define the cut on hypergraph H and its subhypergraph $H_{V'}$ related to a bipartion (V_1, V_2). For this porpose we define the set of all cut hyperedges $E(V_1, V_2) := \{e \in E \mid \exists u, v \in e : u \in$

$V_1 \ \wedge \ v \in V_2\}$. We define the cut on H resp. $H_{V'}$ with

$$\omega_H(V_1, V_2) := \sum_{e \in E(V_1,V_2)} \omega(e)$$
$$\omega_{H_{V'}}(V_1, V_2) := \sum_{e \in E(V_1,V_2) \setminus E_\emptyset} \omega(e)$$

Some will be wondering about the defnition of the cut $\omega_{H_{V'}}$ of $H_{V'}$ over the cut edges of H. A cut hyperedge e of H must not necessarily be a cut hyperedge of $H_{V'}$. E.g., if $e = \{v_1, v_2\}$ with $v_1 \in V_1$ and $v_2 \in V_2$, but $v_1 \in V'$ and $v_2 \notin V'$. Then e is cut in H, but not in $H_{V'}$, because v_2 is removed from e per definition of $E_{V'}$. However, the reason that we still define e as cut hyperedge of $H_{V'}$ has to do with our problem statement, which we will define as follows:

Problem 5.1. *How do we have to define the source set S and sink set T for a subhypergraph $H_{V'}$ (with $V' \subseteq V$) and a bipartition B_1, such that after a maximum (S,T)-flow calculation (with f as maximum flow function) the resulting bipartition B_2 on H satisfy the following conditions:*

(i) $\omega_H(B_2) \leq \omega_H(B_1)$

(ii) $\Delta_H := \omega_H(B_1) - \omega_H(B_2) = \omega_{H_{V'}}(B_1) - |f| =: \Delta_{H_{V'}}$

The first condition ensures that a maximum (S,T)-flow calculation on $T_L(H_{V'})$ never decrease the cut of H. The existence of the second condition has practical reasons. First, we can simply update the cut metric via $\omega_H(B_2) = w_H(B_1) - \Delta_{H_{V'}}$, instead of summing up the weight of all cut hyperedges. Since, we have to setup the subhypergraph $H_{V'}$ before each maximum flow computation we can implicitly calculate $\omega_{H_{V'}}(B_1)$. Therefore, the cut metric can be updated after the *Max-Flow-Min-Cut* computation in constant time instead of $\mathcal{O}(|E|)$. On the other hand, we can assert the correctness of our own maximum flow algorithm. If $\Delta_H \neq \Delta_{H_{V'}}$, then with high probability our flow algorithm is incorrect. Also, the reason why we define $\omega_{H_{V'}}(V_1, V_2)$ over the cut hyperedges of H is due to the fact that the equality

$$\Delta_H := \omega_H(B_1) - \omega_H(B_2) = \omega_{H_{V'}}(B_1) - \omega_{H_{V'}}(B_2)$$

holds. If we are able to show that $|f| = \omega_{H_{V'}}(B_2)$, we simultanously show that $\Delta_H = \Delta_{H_{V'}}$.
We will now present a solution for our problem statement. First, we show how S and T can be choosen to satisfy condition (i). Afterwards, we extend S and T with additional nodes to fullfil condition (ii). Finally, we show how S and T can be modified, such that we can obtain smaller cuts on H and simultanously satisfy condition (i) and (ii) of our problem statement.
Let $V' \subseteq V$ and $\delta B = \{e \in E \mid \exists u, v \in e : u \in V' \wedge v \notin V'\}$ be the set of all *Border Hyperedges*. For a bipartition (V_1, V_2) of H, we say $v \in V_1$ is a source node of the flow network $T_L(H_{V'})$, if there exists a hyperedge $e \in \delta B$ containing v and at least one other node $u \in V_1$ with $u \notin V'$. More formal:

$$S_1 = \{s \in V' \cap V_1 \mid \exists v \notin V' : \exists e \in \delta B : v \in V_1 \ \wedge \ s, v \in e\} \quad (5.1)$$
$$T_1 = \{t \in V' \cap V_2 \mid \exists v \notin V' : \exists e \in \delta B : v \in V_2 \ \wedge \ v, t \in e\} \quad (5.2)$$

An example of a *Max-Flow-Min-Cut* computation on $H_{V'}$ with S and T as source and sink set is illustrated in Figure 10.

Lemma 5.1. *Let B_1 be a bipartition of H and $T_L(H_{V'})$ the flow network of subhypergraph $H_{V'}$ with S and T as defined in Equation 5.1 and 5.2 (with $V' \subseteq V$). Let B_2 be the bipartition obtained by a maximum (S,T)-flow computation on $T_L(H_{V'})$. Then, $\omega_H(B_2) \leq \omega_H(B_1)$.*

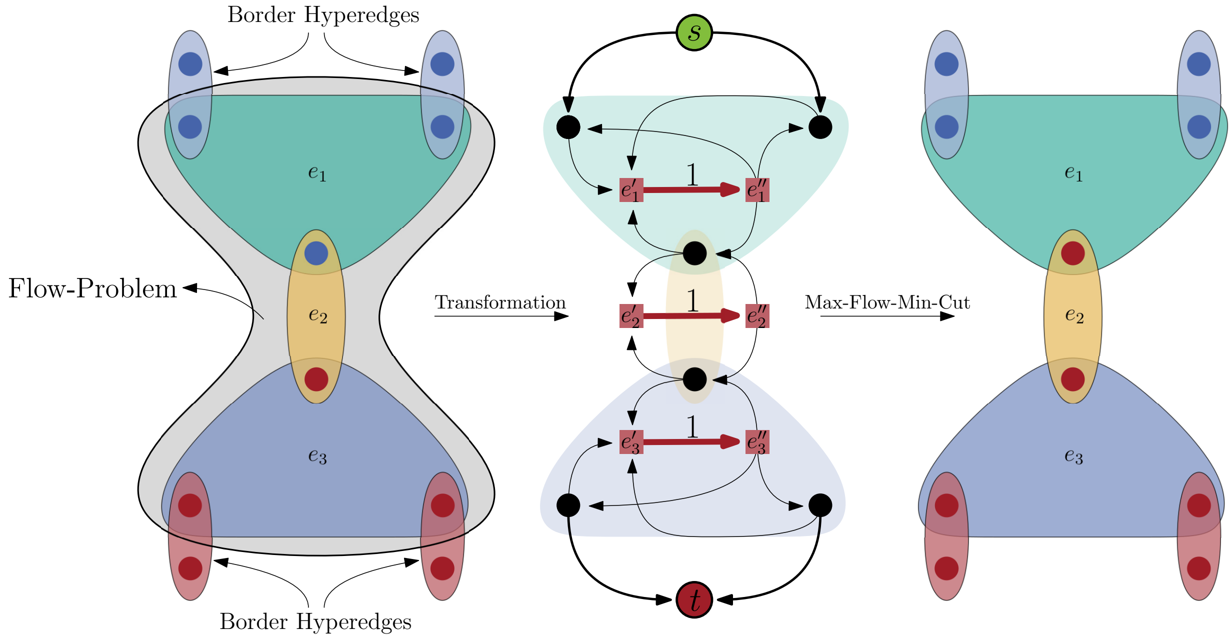

Figure 10: Example how *Border Hyperedges* are modelled as sources and sinks.

Proof. A (S,T) *Max-Flow-Min-Cut* computation on $T_L(H_{V'})$ yields to a minimum (S,T)-cutset on $H_{V'}$ [6]. Thus, for all hyperedges $e \notin \delta B \cup E_\emptyset$, which are cut in B_2, the sum of their weight must be less or equal than the sum of all cut hyperedges $e \notin \delta B \cup E_\emptyset$ of bipartition B_1. We need to show that a non-cut hyperedge $e \in \delta B$ of $B_1 = (V_1, V_2)$ cannot become a cut hyperedge of $B_2 = (V_1', V_2')$. Let $e \in \delta B$ be such a hyperedge. e must be either a subset of V_1 or V_2, otherwise e is a cut hyperedge. Let $e \subseteq V_1$, then $e \cap V' \subseteq S$ (see Equation 5.1). Defining a node $s \in S$ as source node means that it cannot change its block after a *Max-Flow-Min-Cut* computation. Therefore, $e \subseteq V_1$ and $e \subseteq V_1' \Rightarrow e$ is a non-cut hyperedge in B_2. The proof for $e \subseteq V_2$ is equivalent $\Rightarrow \omega_H(B_2) \leq \omega_H(B_1)$. □

In the next step we will show how S and T can be extended to fullfil condition (ii) of Problem 5.1. Currently, $|f| \leq \omega_{H_{V'}}(B_2)$ (without a prove). Obviously, some nodes are missing in S and T. Do understand which nodes are missing consider Figure 11. Transformation 1 illustrates our current modelling approach defined in Equation 5.1 and 5.2. The maximum flow on this network is $|f| = 1$, but the resulting minimum (S,T)-bipartition B_2 induced a cut of $\omega_{H_{V'}}(B_2) = 2$. This due to the fact that e_1 and e_3 are cut hyperedges in H, but non-cut hyperedges in $H_{V'}$. The actual cut of $H_{V'}$ is therefore 1 (instead of 2) and this is also a minimum (S,T)-cut. Transformation 2 illustrates the correct modelling approach for cut hyperedges of H which are non-cut hyperedges in $H_{V'}$. For each hyperedge $e \in \delta B$ with $e \cap V' \subseteq V_2$ and $e \setminus V' \cap V_1 \neq \emptyset$, we add the *incomming hyperedge node* e' to S. More formal:

$$S = S_1 \cup \{e' \mid e \cap V' \subseteq V_2 \ \wedge \ e \setminus V' \cap V_1 \neq \emptyset\} \tag{5.3}$$
$$T = T_1 \cup \{e'' \mid e \cap V' \subseteq V_1 \ \wedge \ e \setminus V' \cap V_2 \neq \emptyset\} \tag{5.4}$$

Lemma 5.2. *Let B_1 be a bipartition of H and $T_L(H_{V'})$ the flow network of subhypergraph $H_{V'}$ with S and T as defined in Equation 5.3 and 5.4 (with $V' \subseteq V$). Let B_2 be the bipartition obtained by a maximum (S,T)-flow computation on $T_L(H_{V'})$ with f as maximum flow function. Then, $\omega_{H_{V'}}(B_2) = |f|$ ($\Rightarrow \Delta_H = \Delta_{H_{V'}}$).*

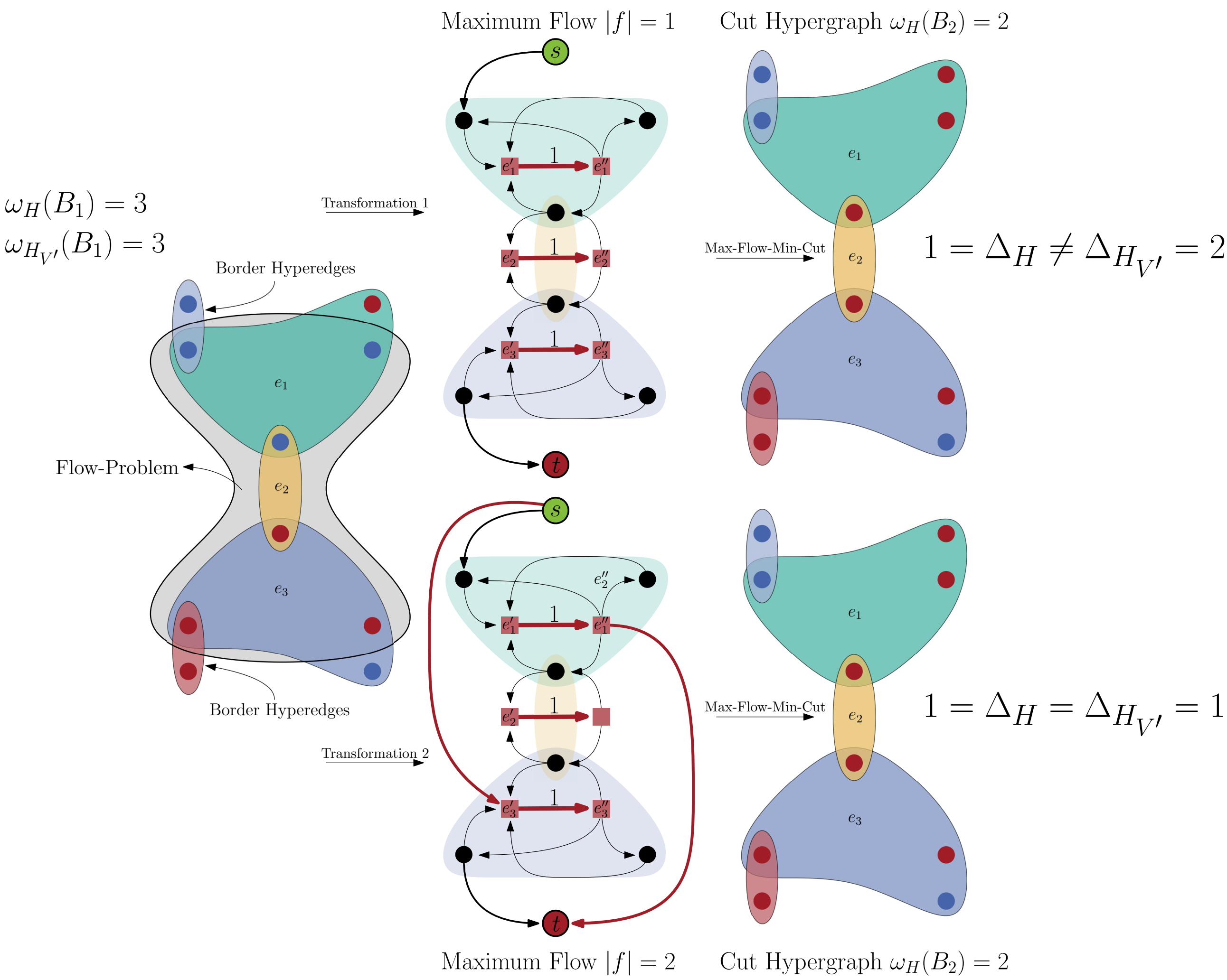

Figure 11: In this example e_1 and e_3 are cut hyperedges of the hypergraph, but not of the subhypergraph induced by the flow problem. Modelling the *outgoing* resp. *incomming* hyperedge node of e_1 resp. e_2 as sink resp. source ensures that $\Delta_H = \Delta_{H_{V'}}$.

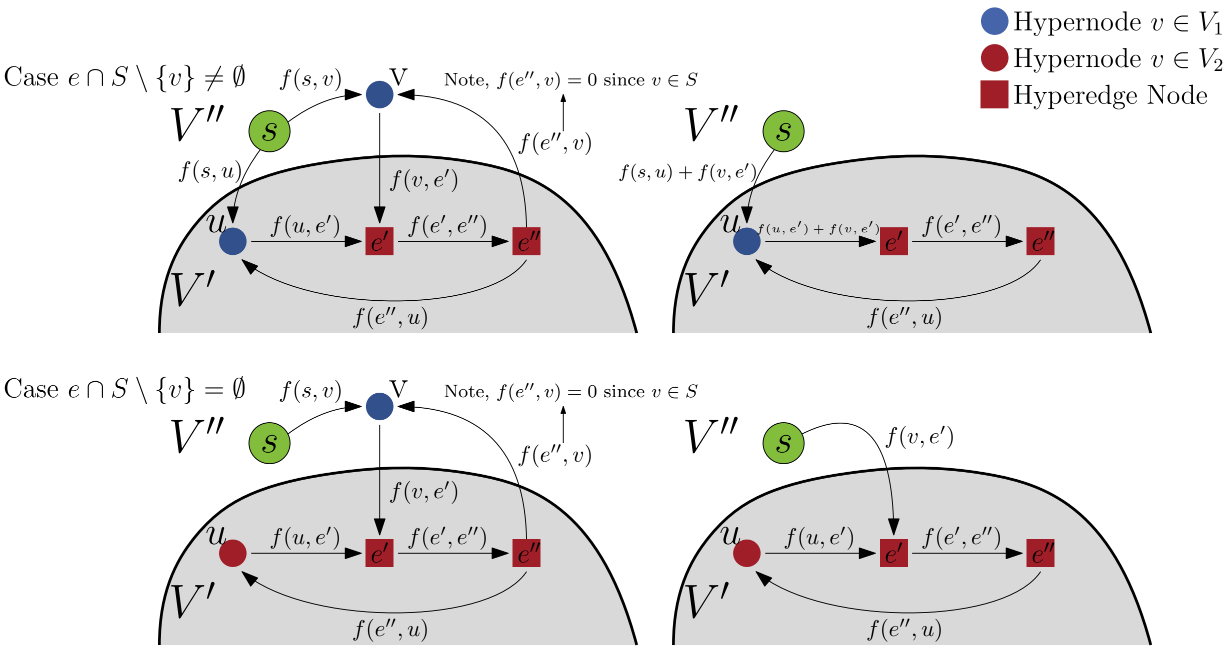

Figure 12: Illustration of the two cases presented in proof of Lemma 5.2 in order to remove a hypernode $v \in V'' \cap S$ from $T_L(H_{V' \cup V''})$.

Proof. Let $V'' = \bigcup_{e \in \delta B} e \setminus V'$ be the set of all hypernodes contained in a *border hyperedge*, but not in V'. Let $H_{V' \cup V''}$ be the subhypergraph obtained by extending $H_{V'}$ with all missing hypernodes of *border hyperedges*. We define the flow problem $T_L(H_{V' \cup V''})$ with $S' = S_1 \cup (V'' \cap V_1)$ and $T' = T_1 \cup (V'' \cap V_2)$ as sources and sinks. Further, let f' be a maximum (S',T')-flow on $T_L(H_{V' \cup V''})$ and B_2 be the corresponding minimum (S',T')-bipartition. Because all hypernodes which are part of a hyperedge in H and also in $H_{V'}$ are fully contained in $H_{V' \cup V''}$ the equality $|f'| = \omega_{H_{V'}}(B_2)$ holds. In the following we present a technique with which we can obtain a new flow network $T_L(H_{V' \cup V'' \setminus \{v\}})$ with $v \in V''$ and S'' and T'' as sources and sinks. Simultanously we map the maximum (S',T')-flow f' of $T_L(H_{V' \cup V''})$ to a maximum (S'',T'')-flow of $T_L(H_{V' \cup V'' \setminus \{v\}})$ with $|f'| = |f''|$. Applying this technique successively on all nodes $v \in V''$ will result in the flow network $T_L(H_{V'})$ with S and T as sources and sinks from Equation 5.3 and 5.4.
A hypernode $v \in V''$ is either a source or a sink. We will show how to remove a source hypernode $v \in V'' \cap S'$. At beginning we define $S'' := S'$, $T'' := T'$ and $f'' := f'$. In order to remove $v \in V''$ we have to distingush two cases based on a incident hyperedge $e \in I(v)$ of v:

$\mathbf{e \cap S \setminus \{v\} \neq \emptyset}$: Then there exists a hypernode $u \in e \cap S$ with $u \neq v$. We define $f''(u,e') = f''(u,e') + f'(v,e')$ and $f''(s,u) = f''(s,u) + f'(v,e')$.

$\mathbf{e \cap S \setminus \{v\} = \emptyset}$: In this case e must be a cut hyperedge in H, but not in $H_{V'}$, otherwise there would exist a hypernode $u \in e \cap S$ (see Equation 5.1). We define $S'' = S'' \cup \{e'\}$. Simultanously, we set $f''(s,e') = f'(v,e')$.

The two cases are illustrated in Figure 12. We can remove v from $T_L(H_{V' \cup V''})$ after applying this procedure for all $e \in I(v)$. The cases for a hypernode $v \in V'' \cap T'$ are equivalent. f'' is a valid flow function per construction and $|f'| = |f''|$. Also f'' is maximum (S'',T'')-flow on $T_L(H_{V' \cup V'' \setminus \{v\}})$, otherwise we can map a augmenting path in the residual graph $T_L(H_{V' \cup V'' \setminus \{v\}})$ to a augmenting path in $T_L(H_{V' \cup V''})$ (without a proof). We can successively remove all $v \in V''$ from $T_L(H_{V' \cup V''})$ with this method.
The resulting flow network is $T_L(H_{V'})$. For each $e \in E$ which is cut in H, but not in $H_{V'}$, we have added the corresponding *incomming hyperedge node* e' or *outgoing hyperedge node* e'' to S'' resp. T''. Therefore, S'' and T'' are equal with S and T defined in Equation 5.3 and 5.4.

Finally, the flow function f'' is a maximum (S,T)-flow on $T_L(H_{V'})$ and $|f''| = |f'| = \omega_{H_{V'}}(B_2)$ per construction. □

With our current modelling approach we are able to satisfy all conditions of our problem statement. However, sometimes we define hypernodes as source resp. sink which are unnecessary. For a explanation consider Figure 13. Hyperedge e_1 is cut in $H_{V'}$ and contains hypernodes from both blocks, which are not in the flow problem. Regardless what we do in $H_{V'}$ we can not remove e_1 from cut in H. Using our suggested source and sink modelling has as consequence that e_1 and e_2 are still cut after a *Max-Flow-Min-Cut* computation (see *Transformation 1* in Figure 13). Another approach is to define for hyperedges which are cut of $H_{V'}$ and are also of H the *incomming* resp. *outgoing hyperedge node* as source resp. sink (see *Transformation 2* in Figure 13). In our example all hypernodes of e_1 are still able to move and a *Max-Flow-Min-Cut* computation removes e_2 from cut.
To define our final source and sink set, we split the set of all *border hyperedges* into three different disjoint subsets as follows:

(i) $\delta B_1 = \{e \in \delta B \mid e \subseteq V_1 \;\vee\; e \subseteq V_2\}$
(ii) $\delta B_2 = \{e \in \delta B \mid e \cap V' \nsubseteq V_1 \;\wedge\; e \cap V' \nsubseteq V_2\}$
(iii) $\delta B_3 = \{e \in \delta B \setminus \delta B_1 \mid (e \cap V' \subseteq V_1 \;\vee\; e \cap V' \subseteq V_2\}$

δB_1 contains all non-cut *border hyperedges* of H. δB_2 contains all *cut border hyperedges* of H, which are also cut in $H_{V'}$ and δB_3 contains all *cut border hyperedges* of H, which are non-cut in $H_{V'}$.

$$S = \bigcup_{\substack{e \in \delta B_1 \\ e \subseteq V_1}} e \cap V' \;\cup \bigcup_{\substack{e \in \delta B_2 \cup \delta B_3 \\ e \setminus V' \cap V_1 \neq \emptyset}} \{e'\} \tag{5.5}$$

$$T = \bigcup_{\substack{e \in \delta B_1 \\ e \subseteq V_2}} e \cap V' \;\cup \bigcup_{\substack{e \in \delta B_2 \cup \delta B_3 \\ e \setminus V' \cap V_2 \neq \emptyset}} \{e''\} \tag{5.6}$$

Equation 5.5 and 5.6 are illustrated in Figure 14. A *Max-Flow-Min-Cut* computation on $T_L(H_{V'})$ with S and T as defined in Equation 5.5 and 5.6 satisfy condition (i) and (ii) of Problem 5.1. This can be prooven with similiar techniques used in the proof of Lemma 5.1 and 5.2. A maximum (S,T)-flow calculation yields to a minimum (S,T)-cut on $H_{V'}$. A non-cut hyperedge $e \in \delta B_1$ can not become a cut hyperedge after a *Max-Flow-Min-Cut* computation, because we still define all hypernodes $v \in e \cap V'$ as sources resp. sinks. Therefore, $\omega_H(B_2) \leq \omega_H(B_1)$. We can proof Lemma 5.2 for our new source and sink set if we adapt the conditions of the cases for a hyperedge $e \in I(v)$ based on the set δB_1, δB_2 and δB_3 where e is contained. If $e \in \delta B_1$, then there must exists a hypernode $u \in e \cap S \setminus \{v\}$ on which we apply the first case (Case 1: $e \cap S \setminus \{v\} \neq \emptyset$). For all $e \in \delta B_2 \cup \delta B_3$, we simply apply the second case (Case 2: $e \cap S \setminus \{v\} = \emptyset$). After removing all hypernodes $v \in V''$ the resulting network is $T_L(H_{V'})$ with S and T as defined in Equation 5.5 and 5.6. Further, the flow function f'' is a maximum (S,T)-flow on $T_L(H_{V'})$ with $|f''| = |f'| = \omega_{H_{V'}}(B_2) \Rightarrow \Delta_H = \Delta_{H_{V'}}$.
Finally, we want to show that for a minimum (S',T')-bipartition B_2 with S' and T' as defined in Equation 5.5 and 5.6 and a minimum (S,T)-bipartition B_3 with S and T as defined in Equation 5.3 and 5.4 calculated with flow network $T_L(H_{V'})$ the inequality $\omega_H(B_2) \leq \omega_H(B_3)$ holds. For this propose we need a preparing lemma.

Lemma 5.3. *Let $G = (V, E, c)$ be a flow network with sources S and sinks T. Further, let $S' \subseteq S$ and $T' \subseteq T$. The value of a maximum (S',T')-flow f' is less or equal than the value of a maximum (S,T)-flow f. More formal, $|f'| \leq |f|$.*

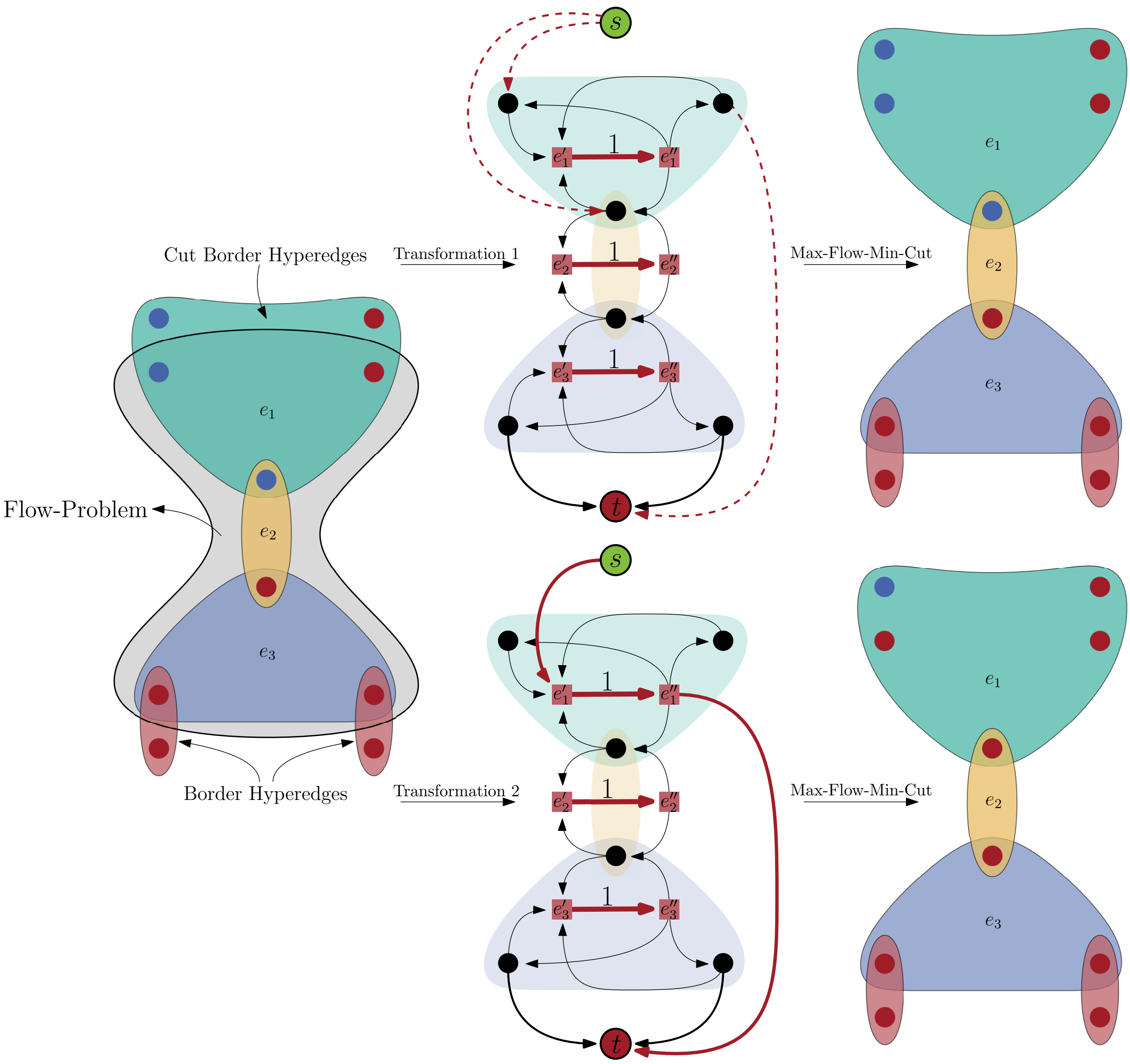

Figure 13: Example how *Cut Border Hyperedges* are modelled as sources and sinks. In this example e_1 contains node from block V_1 and V_2 not contained in the flow problem. Therefore, we can not remove e_1 from cut. Treating e_1 as a *Border Hyperedge* would result in Transformation 1. This has the consequence that we are not able to remove e_2 from cut with a *Max-Flow-Min-Cut* computation. Defining the *incomming* resp. *outgoing* hyperedge of e_1 as source resp. sinks allows the corresponding hypernodes of e_1 still to move. The consequence is that we can remove e_2 from cut with a *Max-Flow-Min-Cut* computation in Transformation 2.

Proof. Assume $|f'| > |f|$. Then, we can simply set $f = f'$, because $S' \subseteq S$ and $T' \subseteq T$. But this is a contradiction to assumption that f is a maximum (S,T)-flow on G. Therefore, $|f'| \leq |f|$. $\square$

In the following theorem, we denote with S and T the source and sink sets as defined in Equation 5.3 and 5.4 and with S' and T' the source and sink sets as defined in Equation 5.5 and 5.6.

Theorem 5.1. *Let H be a hypergraph and $H_{V'}$ be the subhypergraph induced by the subset $V' \subseteq V$. Further, B_1 is the current bipartition of H. For a minimum (S',T')-bipartition B_2 and a minimum (S,T)-bipartition B_3 obtained by a maximum (S',T')- resp. (S,T)-flow calculation on $T_L(H_{V'})$ the inequality $\omega_H(B_2) \leq \omega_H(B_3) \leq \omega_H(B_1)$ holds.*

Proof. Let $(\bar{S}',\bar{T}')$ resp. $(\bar{S},\bar{T})$ be the sets obtained by removing all *incomming* and *outgoing hyperedge nodes* e' and e'' from (S',T') resp. (S,T). It holds that $\bar{S}' \subseteq \bar{S}$ and $\bar{T}' \subseteq \bar{T}$. Afterwards, we extend the subhypergraph $H_{V'}$ with all hypernodes $V'' = \bigcup_{e \in \delta B} e \setminus V'$ and obtain subhypergraph $H_{V' \cup V''}$ with flow network $T_L(H_{V' \cup V''})$. Also we extend $(\bar{S}',\bar{T}')$ and $(\bar{S},\bar{T})$ exactly in the same way as in the proof of Theorem 5.2. With the *Max-Flow-Min-Cut*-Theorem [6] we can conclude that the cut value $\omega_{H_{V'}}(B_2)$ of a minimum $(\bar{S}',\bar{T}')$-bipartition B_2 on $H_{V'}$ is equal with the value of a maximum $(\bar{S}',\bar{T}')$-flow f' on $T_L(H_{V' \cup V''})$. The same holds for a minimum $(\bar{S},\bar{T})$-bipartition B_3 and a maximum $(\bar{S},\bar{T})$-flow f. After extending $(\bar{S}',\bar{T}')$ resp. $(\bar{S},\bar{T})$ with all hypernodes of V'' $\bar{S}' \subseteq \bar{S}$ and $\bar{T}' \subseteq \bar{T}$ still holds. With Lemma 5.3 and the *Max-Flow-Min-Cut*-Theorem follows $\omega_{H_{V'}}(B_2) = |f'| \leq |f| = \omega_{H_{V'}}(B_3)$.
We can transform $(\bar{S}',\bar{T}')$ resp. $(\bar{S},\bar{T})$ and flow network $T_L(H_{V' \cup V''})$ back to $T_L(H_{V'})$ with (S',T') resp. (S,T) as source and sink sets with the technique described in the proof of Theorem 5.2 and in the sketch of the proof for our new source and sink sets (see Equation 5.5 and 5.6). Therefore, the inequality still holds for bipartitions B_2 and B_3 obtained by a maximum (S',T')- and (S,T)-flow calculation on $T_L(H_{V'})$. Finally, it follows

$$\begin{aligned}\omega_H(B_2) &\overset{\text{Problem 5.1(ii)}}{=} \omega_H(B_1) - \omega_{H_{V'}}(B_1) + |f'| \\ &\overset{\text{Lemma 5.3}}{\leq} \omega_H(B_1) - \omega_{H_{V'}}(B_1) + |f| \\ &\overset{\text{Problem 5.1(ii)}}{=} \omega_H(B_3) \overset{\text{Problem 5.1(i)}}{\leq} \omega_H(B_1)\end{aligned}$$

$\square$

We are now able to extract a subhypergraph $H_{V'}$ out of a already bipartitioned hypergraph H and calculate a minimum (S,T)-bipartition of $H_{V'}$ with S and T as defined in Equation 5.5 and 5.6. The resulting bipartition induced a new cut on H smaller or equal than the old cut. Further, we show with our modelling technique of S and T that Δ_H can be calculated with the help of the value of a maximum (S,T)-flow computation on $T_L(H_{V'})$. Additionaly, we demonstrate that a different modelling approach of S and T which satisfy both conditions of Problem 5.1 can lead to an improved cut quality of the minimum (S,T)-bipartition on the original hypergraph H.

With the given approach we are able to optimize the cut metric of a given bipartition of a hypergraph H. We can transfer those results in order to improve a k-way partition $\Pi = (V_1, \ldots, V_k)$, if the objective is the connectivity metric. Let $V' \subseteq V_i \cup V_j$ be a subset of the hypernodes of two adjacent blocks V_i and V_j in the quotient graph. If we optimize the cut of subhypergraph $H_{V'}$ we simultanously optimize the connectivity metric of H. The reduction of the cut on $H_{V'}$ is then equal with the reduction of the connectivity on H.

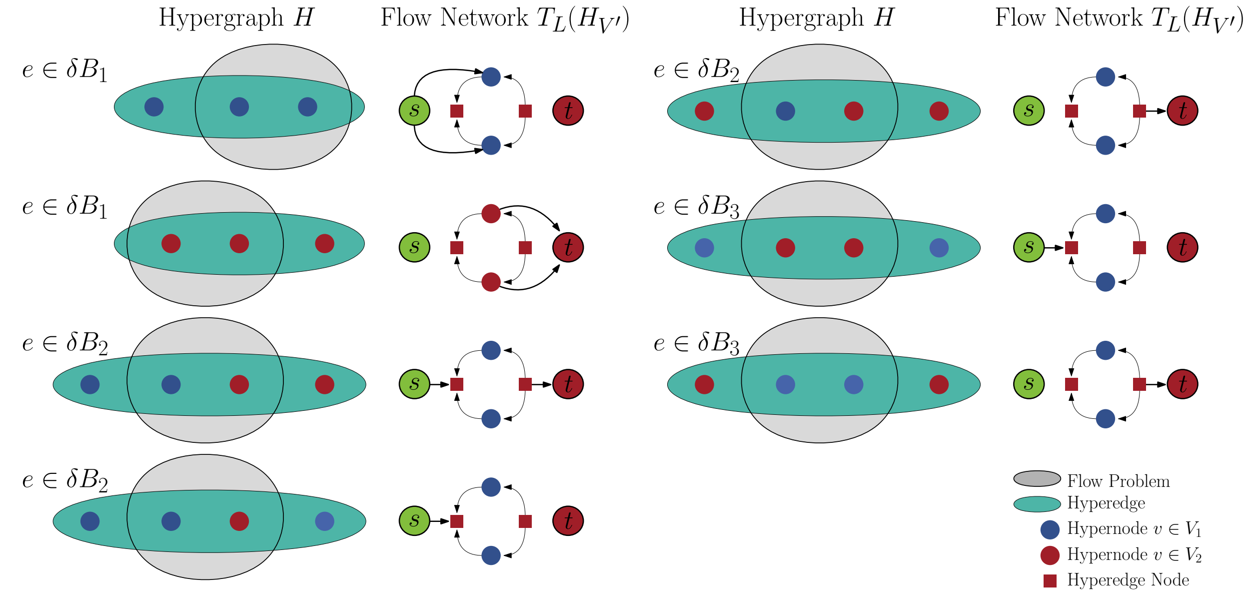

Figure 14: Illustration of source and sink set modelling defined in Equation 5.5 and 5.6.

5.2 Most Balanced Minimum Cuts on Hypergraphs

Picard and Queyranne [15] showed that all minimum (s,t)-cuts of a graph G are computable with one maximum (s,t)-flow computation by iterating through all *closed node sets* of the residual graph of G. The corresponding algorithm is presented in Section 3.3.3.
We can apply the same algorithm on hypergraphs. A minimum-capacity (s,t)-cutset of $T_L(H)$ is equal with a minimum-weight (s,t)-cutset of H. With the algorithm of Section 3.3.3 we are able to find all minimum-capacities (s,t)-cutsets of $T_L(H)$, which are also minimum-weight (s,t)-cutsets of H. The corresponding minimum-weight (s,t)-bipartitions are all *closed node sets* of the residual graph of $T_L(H)$.
However, when we use e.g. $T_H(H,V')$ (see Section 4.1) or $T_{\text{Hybrid}}(H,V')$ (see Section 4.4) as underlying flow network some hypernodes are removed from the flow problem. This a problem, if we want to enumerate all minimum-weight (s,t)-bipartitions. The solution for this problem is quite simple. After a maximum (s,t)-flow calculation on one of the two mentioned networks we insert all removed hypernodes with their corresponding edges again into the residual graph of our flow network. The maximum (s,t)-flow is still maximal, otherwise we would had found an *augmenting path* on the flow network before. We are now able to compute all minimum-weight (s,t)-bipartitions the same way as with $T_L(H)$.

5.3 A Direct K-Way Flow-Based Refinement Framework

We described how a hypergraph H could be transformed into a flow network $T_L(H)$ such that every minimum-capacity (s,t)-cutset on $T_L(H)$ is a minimum-weight (s,t)-cutset on H (see Section 3.2.2). Additionaly, we present techniques to sparsify the flow network $T_L(H)$ [14] in order to reduce the complexity of the flow problem (see Section 4). Further, we show how to configure the source and sink sets on the flow network of a subhypergraph $H_{V'}$ (with $V' \subseteq V$) such that a *Max-Flow-Min-Cut* computation improves a given bipartition of H (see Section 5.1). Finally, we are able to enumerate all minmum-weight (s,t)-cutsets of a subhypergraph $H_{V'}$ with one maximum (s,t)-flow calculation [15].
We will now present our direct k-way flow-based refinement framework which we integrated into the n-level hypergraph partitioner *KaHyPar* [10] (see Section 3.4.2). Our flow-based refinement

approach optimizes the *connectivity* metric. We used a similiar architecture as proposed by Sanders and Schulz [18] (see Section 3.3). The basic concepts of the framework are illustrated in Figure 15.
Our maximum flow calculations are embedded into an *Active Block Scheduling* refinement [11] (see Section 3.3.4). Each time we use flows to improve the connectivity metric of a given k-way partition Π we construct the quotient graph Q of Π. Afterwards, we iterate over all edges of Q in random order. For each edge (V_i, V_j) of Q we grow a flow problem around the cut of the bipartition induced by V_i and V_j. In order to do that we use two *BFS*, one only touches hypernodes of V_i and the second only touches hypernodes of V_j. The *BFS* are initialized with all hypernodes contained in a cut hyperedge of the bipartition (V_i, V_j). A pairwise flow-based refinement is embedded into the *adaptive flow iterations* strategy [18] (see Section 3.3.2) which also determines the size of the flow problem.
After we define the subhypergraph $H_{V'}$, which we use to improve the bipartition (V_i, V_j) on H, we construct one of the flow networks proposed in Section 4 with sources S and sinks T defined in Section 5.1. We implemented two maximum flow algorithms. One is a slightly modified *augmenting path* algorithm of Emdond & Karp [4] (see Section 3.1.1) and the second is the *Push-Relabel* algorithm of Goldberg & Tarjan [3, 8] (see Section 3.1.2). Since we have a *Multi-Source-Multi-Sink* problem, we can find several *augmenting paths* with one *BFS*. After we execute a *BFS* on the residual graph, we search as many as possible edge-disjoint paths in the resulting *BFS*-tree connecting a source s with a sink t. Our Goldberg & Tarjan implementation uses a *FIFO* queue and the *global relabeling* and *gap* heurisitc [3]. We do not use an external implementation of a maximum flow algorithm. Since the $I \backslash O$ of writing a flow problem to memory and reading the solution would significant slowdown the performance of our algorithm, because we have to solve an enormous number of flow problems during the *Active Block Scheduling* refinement. After determine a maximum (S, T)-flow on our flow network we iterate over all minimum (S, T)-bipartitions of $H_{V'}$ [15] and choose the *Most Balanced Minimum Cut* (see Section 3.3.3 and 5.2) according to our *balanced contraint*.
KaHyPar is a n-level hypergraph partitioner ($|V| = n$) taking the multilevel paradigm to its extreme by removing only a single vertex in every level of the hierarchy [1] (see Section 3.4.2). During the refinement step n local searches are instantiated. Therefore, using our flow-based refinement as local search algorithm on each level is not applicable, because the performance slowdown would be tremendous. On this reason we introduce *Flow Execution Policies*. One is to execute our flow-based refinement on each level i where $i = \beta \cdot j$ with $j \in \mathbb{N}_+$ and β as a predefined tunning parameter. Another approach is to simulate a multilevel partitioner with $\log(n)$ hierarchies. A flow-based refinement is then executed on each level i where $i = 2^j$ with $j \in \mathbb{N}_+$. Each policy also executes the *Active Block Scheduling* refinement on the last level of the hierarchy. In all remaining levels where no flow is executed, we can use a *FM*-based local search algorithm [1, 5, 17] (see Section 3.3.4).
An observation during the implementation of this framework was that only a minority of the pairwise refinements based on flows yields to an improvement of the connectivity metric on a hypergraph H. Therefore, we introduce several rules which might prevent unnecessary flow executions to improve the effectiveness ratio by simultanously speed up the runtime.

(i) If the cut between two adjacent blocks in the quotient graph is small (e.g. ≤ 10) we skip the flow-based refinement on these blocks except on the last level of the hierarchy.

(ii) If the value of the cut of a minimum (S, T)-bipartition on $H_{V'}$ is the same as the cut before, we stop the pairwise refinement on these blocks.

(iii) If a flow-based refinement did not lead to an improvement on two blocks in all levels of the multilevel hierarchy, we only execute flows in the first iteration of *Active Block Scheduling* on these blocks.

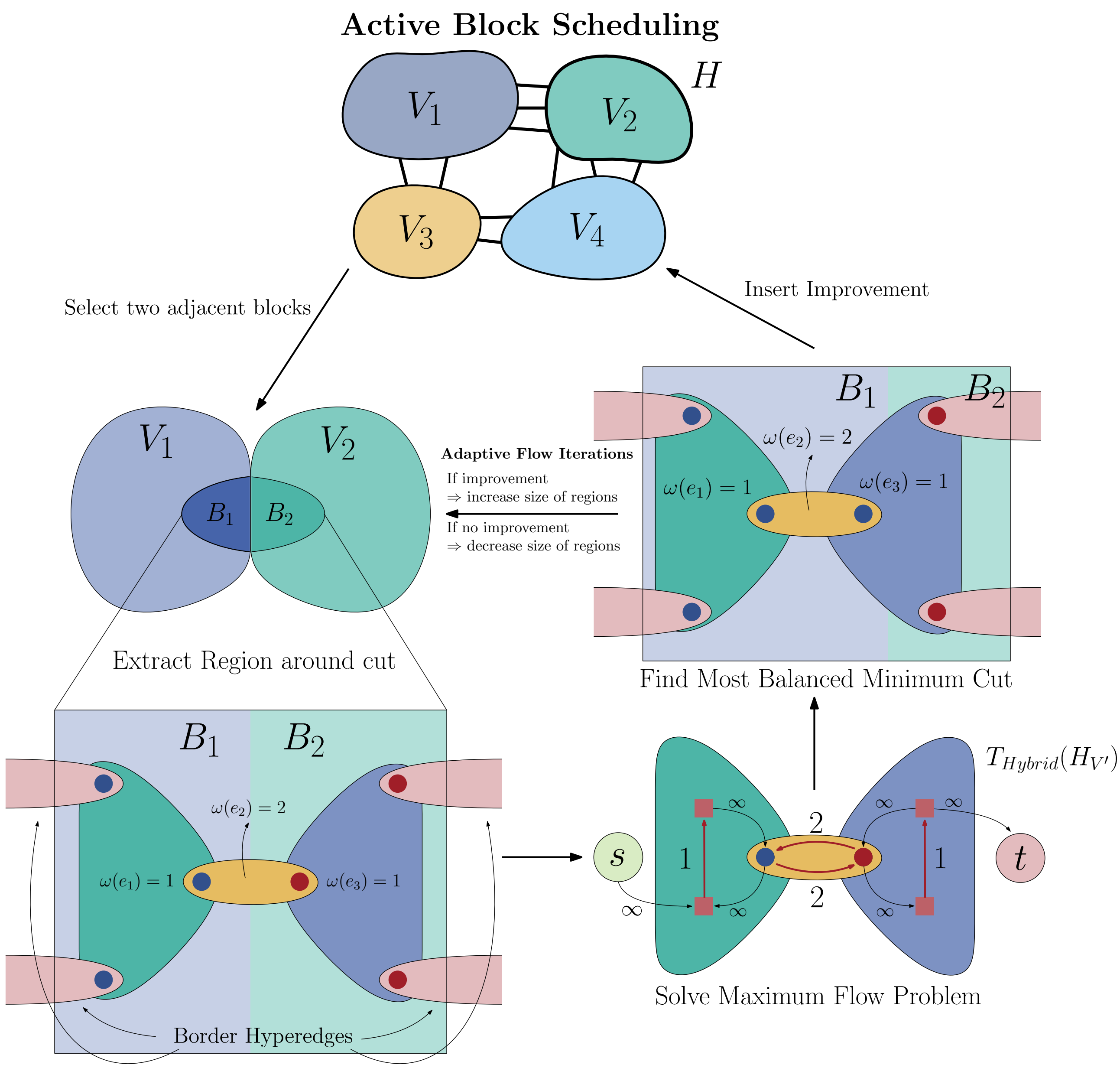

Figure 15: Illustration of our flow-based refinement framework on hypergraphs.

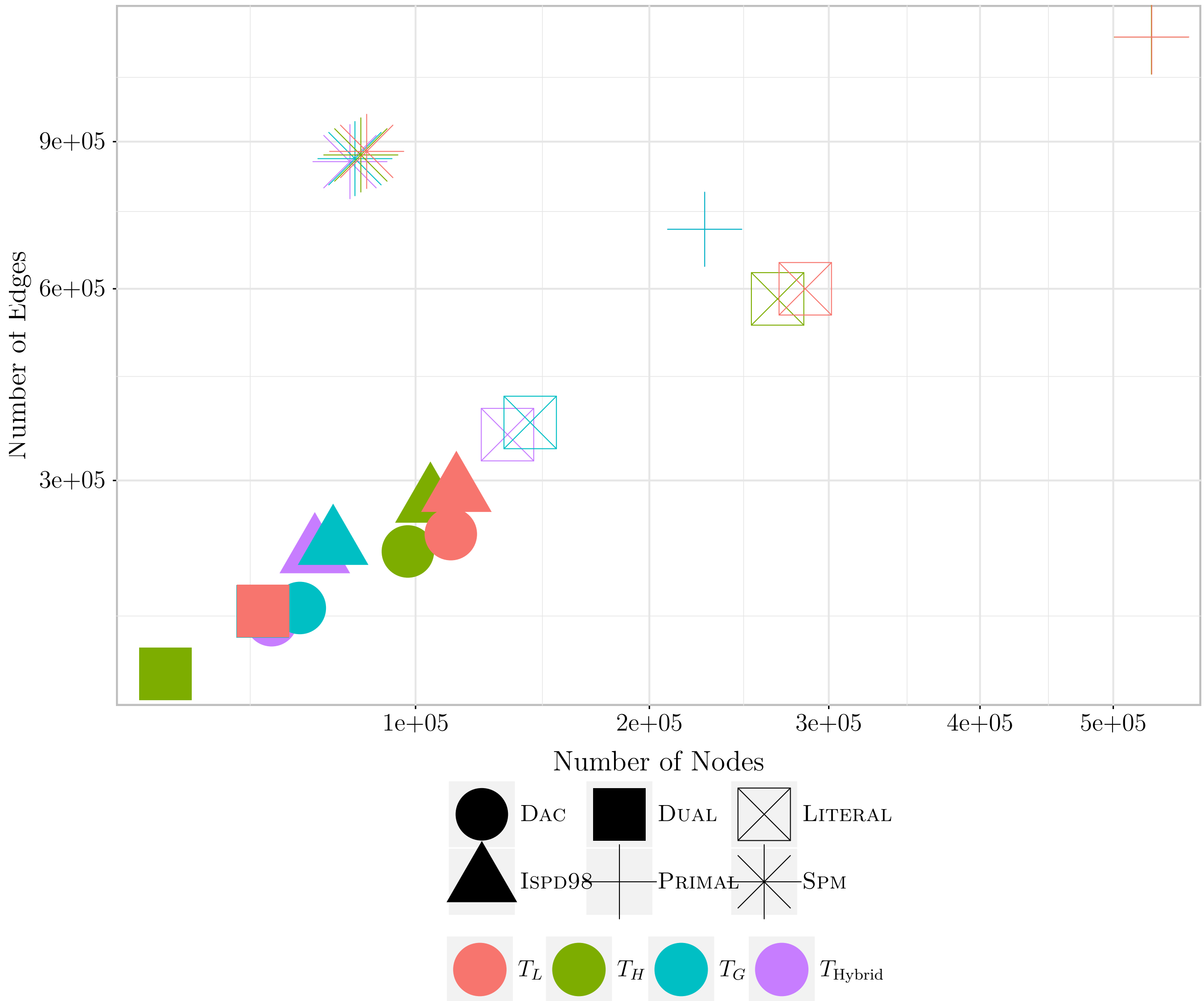

Figure 16: Comparison of the number of nodes and edges of the resulting flow graph for flow problems with size $|V'| = 25000$ for the different flow networks and instance types.

6 Experimental Results

6.1 Flow Algorithms and Networks

(i) Node and Edge Distribution for different networks

(ii) Running time of flow algorithms and flow networks for different instance sizes and types

(iii) Summary Table of running times on all instances

6.2 Flow Configuration

(i) For $k = 2$ on the benchmark subset and $\alpha = \{1, 2, 4, 8, 16\}$ test variants: (F = *flow*, M = *most balanced minimum cut*, C = *cut border hyperededges*)

a) (+F,-C,-M,-FM)

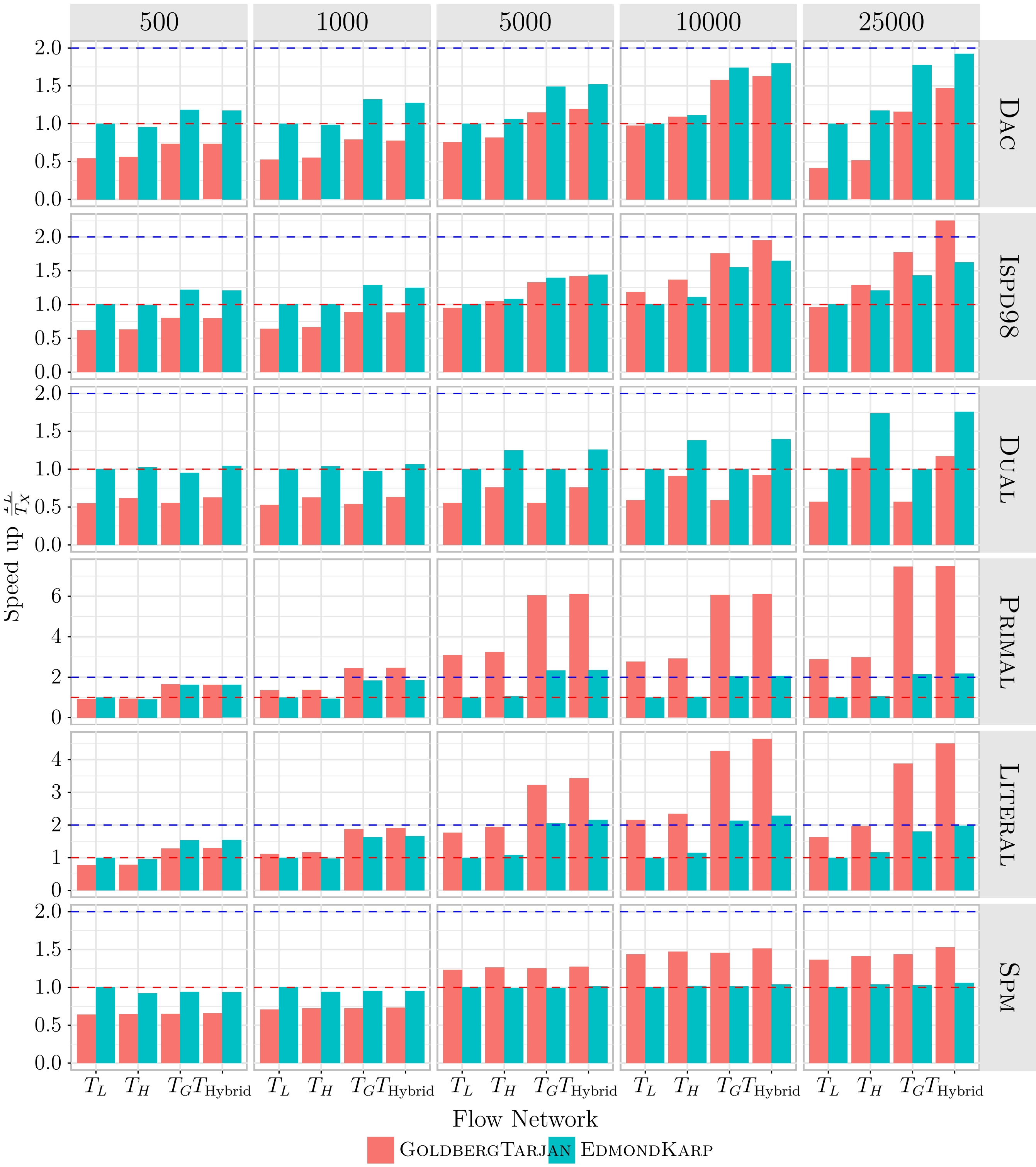

Figure 17: Speed up of our flow algorithms and networks relative to EdmondKarp on T_L for different instance sizes and types. The red dashed line indicates the (EdmondKarp, T_L) implementation and the blue dashed line indicates a speed up by a factor of 2.

Instance	$\lvert V' \rvert$	GoldbergTarjan T_{Hybrid} $t[ms]$	T_G $t[\%]$	T_H $t[\%]$	T_L $t[\%]$	EdmondKarp T_{Hybrid} $t[\%]$	T_G $t[\%]$	T_H $t[\%]$	T_L $t[\%]$
All	500	0.91	+2.24	+24.93	+29.35	**−25.39**	−24.3	−6.68	−11.53
	1000	1.95	+3.65	+26.19	+32.95	**−13.99**	−12.36	+10.81	+7.51
	5000	**13.71**	+8.63	+29.39	+43.11	+27.03	+35.33	+73.97	+86.31
	10000	**30.54**	+12.57	+36.15	+54.62	+47.93	+61.72	+100.41	+123.31
	25000	**67.96**	+23.36	+52.12	+87.8	+53.25	+77.85	+100.95	+138.8

Table 1: Running time comparison of maximum flow algorithms on different flow networks. Note, all values in the table are in percentage relative to GoldbergTarjanon flow network T_{Hybrid}. In each line the fastest variant is marked bold.

b) (+F,+C,-M,-FM)
c) (+F,+C,+M,-FM)
d) (+F,+C,+M,+FM)

6.3 Flow Heuristics

(i) For all k on the benchmark subset test variants:
 a) Without any speed up heuristic
 b) than add one heuristic and show quality is equal and time is less

(ii) Show which quality is possible with *constant* flow execution policy

6.4 Final Flow Refiner

(i) Compare final configuration of flow refiner against sea config on the full benchmark set

Variant	(+F,-C,-M,-FM)		(+F,+C,-M,-FM)		(+F,-C,+M,-FM)	
α'	Avg.[%]	$t[s]$	Avg.[%]	$t[s]$	Avg.[%]	$t[s]$
1	-20.02	12.44	-15.48	12.94	-19.69	12.63
2	-14.61	15.16	-10.5	16.07	-14.17	15.77
4	-8.99	19.92	-5.98	21.22	-8.22	21.2
8	-4.96	28.71	-3.22	30.73	-3.37	31.25
16	-2.58	47.35	-1.52	50.89	-0.34	52.19
Ref.	(-F,-C,-M,+FM)		6373.88	13.73		

Variant	(+F,+C,+M,-FM)		(+F,+C,+M,+FM)	
α'	Avg.[%]	$t[s]$	Avg.[%]	$t[s]$
1	-15.26	13.29	0.14	14.99
2	-10.12	16.93	0.36	16.93
4	-5.08	23.01	0.67	20.76
8	-1.64	33.72	1.25	28.65
16	0.51	56.39	1.87	46.17
Ref.	(-F,-C,-M,+FM)		6373.88	13.73

Table 2: Not final experiments still running

7 Conclusion

References

[1] Yaroslav Akhremtsev, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Engineering a direct k-way hypergraph partitioning algorithm. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 28–42. SIAM, 2017.

[2] Reid Andersen and Kevin J Lang. An algorithm for improving graph partitions. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 651–660. Society for Industrial and Applied Mathematics, 2008.

[3] Boris V Cherkassky and Andrew V Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[4] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[5] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. ACM, 1988.

[6] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[7] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

[8] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[9] TE Harris and FS Ross. Fundamentals of a method for evaluating rail net capacities. Technical report, RAND CORP SANTA MONICA CA, 1955.

[10] Tobias Heuer and Sebastian Schlag. Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[11] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. Engineering a scalable high quality graph partitioner. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[12] T. C. Hu and K. Moerder. Multiterminal Flows in a Hypergraph. In T.C. Hu and E.S. Kuh, editors, *VLSI Circuit Layout: Theory and Design*, chapter 3, pages 87–93. IEEE Press, 1985.

[13] Kevin Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *IPCO*, volume 4, pages 325–337. Springer, 2004.

[14] Eugene L Lawler. Cutsets and partitions of hypergraphs. *Networks*, 3(3):275–285, 1973.

[15] Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Combinatorial Optimization II*, pages 8–16, 1980.

[16] Joachim Pistorius and Michel Minoux. An improved direct labeling method for the max–flow min–cut computation in large hypergraphs and applications. *International Transactions in Operational Research*, 10(1):1–11, 2003.

[17] Laura A Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[18] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *ESA*, volume 6942, pages 469–480. Springer, 2011.

Instance	$\lvert V' \rvert$	GoldbergTarjan T_{Hybrid} $t[ms]$	T_G $t[\%]$	T_H $t[\%]$	T_L $t[\%]$	EdmondKarp T_{Hybrid} $t[\%]$	T_G $t[\%]$	T_H $t[\%]$	T_L $t[\%]$
All	500	0.91	+2.24	+24.93	+29.35	**−25.39**	−24.3	−6.68	−11.53
	1000	1.95	+3.65	+26.19	+32.95	**−13.99**	−12.36	+10.81	+7.51
	5000	**13.71**	+8.63	+29.39	+43.11	+27.03	+35.33	+73.97	+86.31
	10000	**30.54**	+12.57	+36.15	+54.62	+47.93	+61.72	+100.41	+123.31
	25000	**67.96**	+23.36	+52.12	+87.8	+53.25	+77.85	+100.95	+138.8
Dac	500	0.34	−0.36	+30.14	+34.98	−37.61	**−38.08**	−23.12	−26.56
	1000	0.8	−1.7	+41.18	+47.43	−38.94	**−41.19**	−20.88	−22.17
	5000	5.2	+4.11	+46.02	+58.5	**−21.35**	−19.79	+12.55	+19.6
	10000	10.67	+3.2	+48.92	+66.83	**−9.41**	−6.44	+46.23	+63
	25000	31.43	+26.81	+186.2	+255.32	**−23.53**	−17.16	+25.16	+47.29
Ispd98	500	0.48	−0.58	+26.23	+28.54	−33.85	**−34.5**	−19.55	−20.14
	1000	1.11	−0.8	+32.35	+37.47	−29.32	**−31.59**	−11.91	−11.88
	5000	7.06	+6.65	+35.1	+49.35	**−1.67**	+1.64	+31.03	+41.91
	10000	**16.33**	+10.97	+42.54	+64.68	+18.38	+25.84	+75.19	+95.09
	25000	**75.01**	+26.26	+73.85	+132.06	+37.85	+56.79	+85.28	+124.01
Dual	500	0.3	+12.37	+0.99	+13.6	**−40.36**	−34.35	−39.13	−37.67
	1000	0.6	+16.87	+0.83	+18.38	**−40.93**	−35.35	−39.47	−37.18
	5000	3.2	+37.54	+0.21	+37.78	**−39.66**	−23.77	−39.17	−24.01
	10000	5.78	+55.72	+1.21	+55.86	**−34.01**	−7.81	−33.3	−8
	25000	14.71	+105.19	+2.15	+105.88	**−33.35**	+17.43	−32.59	+17.28
Primal	500	1.85	**−0.73**	+73.92	+76.03	+0.86	+0.17	+79.92	+63.57
	1000	**3.9**	+0.15	+77.48	+81.23	+33.02	+33.57	+160.43	+145.98
	5000	**29.8**	+0.84	+88.23	+96.71	+160	+162.28	+481.91	+510.71
	10000	**45.94**	+0.69	+109.75	+120.04	+195.68	+197.69	+487.6	+511.93
	25000	**174.32**	+0.21	+151.07	+159.04	+243.77	+248.81	+609.44	+648.46
Literal	500	0.86	+0.72	+63.65	+67.45	**−16.1**	−15.41	+35.63	+29.41
	1000	**1.92**	+1.64	+64.51	+71.46	+15.13	+17.07	+95.07	+90.72
	5000	**12.31**	+6.15	+76.65	+94.2	+59.04	+66.99	+216.7	+243.13
	10000	**29.75**	+8.55	+97.28	+115.37	+102.47	+117.45	+302.93	+363.17
	25000	**64.4**	+15.75	+128.34	+175.78	+126.59	+148.78	+286.31	+349.43
Spm	500	1.46	+0.35	+1.22	+2.47	−29.92	−30.42	−28.84	**−34.57**
	1000	3.09	+1.45	+1.14	+3.28	−23.32	−22.94	−22.17	**−26.89**
	5000	**25.81**	+1.79	+1.09	+3.26	+26.02	+28.55	+28.61	+27.43
	10000	**74.81**	+3.78	+2.48	+5.38	+45.86	+49.36	+48.77	+51.06
	25000	**107.6**	+6.67	+8.56	+12.07	+44.39	+48.88	+47.68	+52.96

Table 3: Running time comparison of maximum flow algorithms on different flow networks. Note, all values in the table are in percentage relative to Goldberg-Tarjan on flow network T_{Hybrid}. In each line the fastest variant is marked bold.