On Balanced Separators in Road Networks

Aaron Schild[1] and Christian Sommer[2]

[1] UC Berkeley aschild@berkeley.edu
[2] Apple Inc. csommer@apple.com

Abstract. The following algorithm partitions road networks surprisingly well: *(i)* sort the vertices by longitude (or latitude, or some linear combination) and *(ii)* compute the maximum flow from the first k nodes (forming the source) to the last k nodes (forming the sink). Return the corresponding minimum cut as an edge separator (or recurse until the resulting subgraphs are sufficiently small).

1 Introduction

Graph Partitioning is the well-studied problem of cutting a graph into disjoint regions of approximately equal size while minimizing the number of edges between regions. An example partition of a road network is shown as Fig. 1.

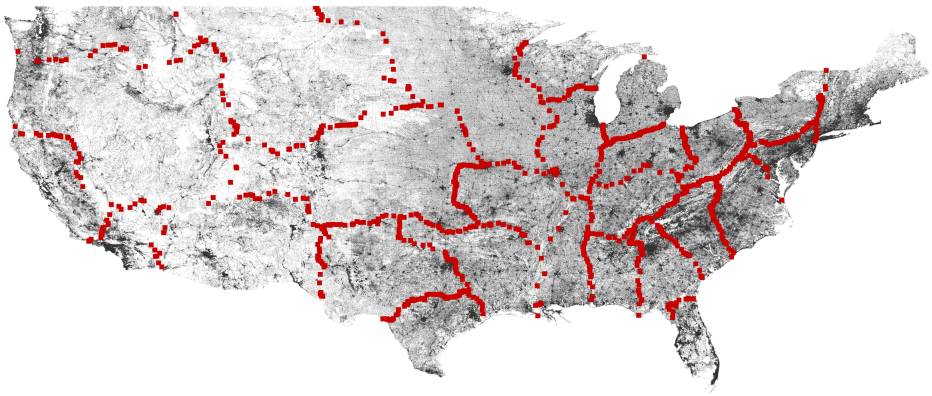

Fig. 1. Recursive bisection using our separator algorithm *Inertial Flow* (with balance 1/4) for the road network of the United States (24M nodes, 29M edges) into 27 regions by cutting a total of 1,413 edges (0.005%). The largest region contains less than 6% of the original graph.

Delling, Goldberg, Razenshteyn, and Werneck [DGRW11] discovered that road networks have remarkably small separators. Prior to our work, their patented method called *PUNCH* appeared to be the only one capable of efficiently computing these separators (*Buffoon* [SS12], another high-quality partitioner for road networks, uses *PUNCH* as a subroutine).

1.1 Problem Statement

Given a graph $G = (V, E)$, a *Graph Partition* is a partition of the vertex set V into disjoint subsets $V_0, V_1, \ldots V_{k-1}$ such that the regions (V_i, E_i) (subgraph induced by V_i) are of *roughly equal size*, and for all V_i, V_j $(i \neq j)$ the set of edges between V_i and V_j (denoted by $E(V_i, V_j)$) is *as small as possible*. A main challenge of graph partitioning is the combined objective of minimizing the cut size while keeping good balance. Various objective functions combine the two quantities. Problem variants include *balanced* k–partitioning, where partitions must satisfy $\forall i : |V_i| \leqslant (1 + \epsilon) |V| / k$ for some imbalance parameter $\epsilon > 0$, or the relaxed (and significantly easier) variant, where only $\forall i : |V_i| \leqslant r$ for some region size constraint r (as considered in this paper).

One way of obtaining such a partition is by cutting G into two pieces V_0, V_1 and then recursing on each subgraph V_0, V_1. The recursion ends when the resulting subgraphs are sufficiently small. For these bisections, there are also various objective functions.

Definition 1 (Cuts and Balanced Cuts). *Given a graph $G = (V, E)$, a* cut *is a partition of V into two disjoint subsets V_0, V_1. A b–*balanced cut *(for any $0 < b \leqslant 1/2$) is a cut such that $|V_i| \geqslant \lfloor b \cdot |V| \rfloor$ for both $i \in \{0, 1\}$.*

At each level of the recursion, the objective is to find a b–balanced cut (for some b, say $b = 1/4$) that minimizes the number of cut edges, i.e. $\min |E(V_0, V_1)|$, where $E(V_0, V_1) := \{(u, v) \in E : u \in V_0, v \in V_1\}$.

Other well-known objective functions for cuts include the *minimum st cut* and the *sparsest cut*. A *minimum st cut* is a cut minimizing $|E(V_0, V_1)|$ with the condition that s and t are separated, i.e., $s \in V_0$ and $t \in V_1$ (no balance requirements). It can be found efficiently using *maximum flow* algorithms [GR98, BK04, GHK$^+$11, Mad13]. A *sparsest cut* is a cut minimizing $|E(V_0, V_1)| / (|V_0| \cdot |V_1|)$. Sparsest cuts are hard even to approximate [KV05, CKK$^+$06].

1.2 Related Work

Theory. Various approximation algorithms for sparsest cut use maximum flow computations [KRV09, AK07, OSVV08, She09]. Roughly speaking, these algorithms iteratively refine an embedding of V by choosing source s and sink t at extremal points (of the embedding), computing st flow, followed by re-arranging V. A simplified statement of these results is that a poly-logarithmic number of carefully chosen maximum-flow computations provides a logarithmic approximation for sparsest cut (details in the corresponding papers). Previously, Lang and Rao [LR04] and Andersen and Lang [AL08] also showed how to improve cuts using maximum flow. Bui, Chaudhuri, Leighton, and Sipser [BCLS87] used maximum flow to compute bisections of regular graphs.

Some graphs are guaranteed to have small balanced cuts. For example, any planar, bounded-genus, or minor-free graph on n nodes has a balanced separator of size $O(\sqrt{n})$ [Ung51, LT79, Dji85, GHT84, And86, AST90], and recursive application yields partitions [LT79, Fre87, HKRS97, vWZA13, KMS13]. Partitions obtained by recursive bisection may be far from optimal though [ST97].

Practice. The literature on graph partitioning is vast, see e.g. [BMSW13, BMS+13] and references therein. In this brief review, we focus on recent work on partitioning road networks. Delling, Goldberg, Razenshteyn, and Werneck [DGRW11] introduce *PUNCH*, which first computes candidate cuts using maximum flows between sources and sinks chosen as follows: for a node $v \in V$, all nodes within distance $< r$ form the source, and all nodes at distance $> R$ form the sink (for two parameters $r < R$; distance can be measured in terms of BFS, shortest-path, or rank distance). These candidate cuts are then aggregated in various ways to form the final partition. Sanders and Schulz [SS11, SS12, SS13] contribute *KaFFPa[E]* and *KaHIP* (following earlier partitioners such as *Ka{SPar,PPA}*), all general-purpose partitioners, with a variant called *Buffoon* optimized for road networks. Their methods are based on the multi-level graph partitioning framework, where the input graph is first contracted, followed by a partitioning step on the smaller graph, and a refinement step to obtain a partition of the original graph. In *KaFFPa*, one of the refinement steps is called *adaptive flow iterations*, which enforces a balance constraint and computes maximum flow with source and sink chosen as BFS balls in two adjacent regions. Similar refinements using maximum flows had also been used by Boykov, Veksler, and Zabih [BVZ01].

Applications. Road network partitions can be used for applications such as shortest-path queries [Som14] or data distribution [KLSV10]. In particular, the performance of separator-based shortest-path algorithms [Fre87, Dji96, HKRS97, FR06, HSW08, DHM+09, KKS11, MS12, DGPW13] depends on the size of the separator. Most prominently, Delling, Goldberg, Pajor, and Werneck [DGPW13] recently demonstrated that separator-based methods built upon a quality partition (such as those described in their joint work with Razenshteyn [DGRW11]) are highly practical. Their method recursively partitions the graph into a multi-level partition and then, for each region and level, precomputes matrices representing shortest-path costs between boundary nodes. For each region, memory requirements are therefore proportional to the square of the number of boundary nodes, which makes the quality of the partition particularly important. Partitioning is the most time-consuming step in their preprocessing algorithm (approximately 10 minutes to compute a multi-level partition for the US road network). Dibbelt, Strasser, and Wagner [DSW14] compute metric-independent *Contraction Hierarchies* based on nested dissection, which in turn is based on recursive bisection (corresponding theory in [BCRW13]). Finding good bisections is the most time-consuming step in their preprocessing algorithm.

1.3 Contribution

Our main contribution is a simple and efficient method to find sparse balanced cuts in embedded graphs such as road networks. The method, which we call *Inertial Flow*, uses the embedding, initially sorts nodes geometrically (like the well-known *Inertial Partitioning*), and then computes a maximum flow. The corresponding minimum cut is used as the separator. *Inertial Flow* is straightforward to implement, yet its partitions are reasonably good. Our experiments

using such a straightforward implementation demonstrate that it is competitive with the state-of-the-art partitioner *PUNCH* [DGRW11]. If the *Natural Cut Heuristic* is interpreted as the heart of *PUNCH* then the objective of this paper is to describe a new heart, and not the effects of its transplantation. We speculate that, in combination with the assembly phase of *PUNCH* or *Buffoon* [SS12], partitions might improve further (particularly in terms of balance).

In addition to simplicity, another advantage of recursive bisection is that, after computing the separator tree once, it contains the information for an entire multi-level partition (see e.g. [KMS13]).

As discussed in the section on related work, various partitioners employ a maximum-flow algorithm as an important subroutine. Their main differentiator is the choice of source and sink. On one hand, when terminals consist of too few nodes, minimum cuts may be highly unbalanced. On the other hand, when terminals consist of too many nodes, the best cuts may be violated by the initial source/sink assignment. Many methods use BFS balls to assign terminals, where the choice of radii is particularly delicate: obviously, balls must not intersect, but they should also be reasonably far apart. Such kind of tuning is fairly straightforward for our method, as there is just the balance parameter b to be configured. State-of-the-art theoretical algorithms for sparsest cut first embed the graph and then refine using maximum flow. The main observation leading to our method is that a road network's embedding (which is typically provided as part of the input) may be sufficiently good to serve as the initial embedding in an analogous algorithm.

2 Inertial Flow

We present an efficient heuristic to find b–balanced cuts in road networks. For the sake of exposition, let us consider a simplified road network, defined as an undirected graph $G = (V, E)$ with an embedding $f : V \hookrightarrow \mathbb{R}^2$. We may assume that G is connected, as typically partitioning algorithms are applied to each connected component independently. Our method is rather simple as it merely applies two standard primitives: sorting and maximum flow. (The well-known *Inertial Partitioning* uses sorting, followed by sweeping, hence the name of our method.)

1. Pick a line $\ell \in \mathbb{R}^2$ and orthogonally project V onto ℓ
   (more precisely, for each vertex v, project its point in the embedding $f(v)$ onto ℓ).
2. Sort V by order of appearance on ℓ (ties broken arbitrarily but consistently).
3. Let the first $\lfloor b \cdot |V| \rfloor$ vertices (in projection order) be the *source* s, and let the last $\lfloor b \cdot |V| \rfloor$ vertices be the *sink* t.
4. Compute a maximum flow between source s and sink t.
5. Return a corresponding minimum st cut.

Key Properties

- By choice of s and t, all minimum st cuts are b–balanced.
- The running time is bounded by the time required to sort V plus the time required to compute one maximum flow in G. Computing the entire separator tree (recursive bisection) requires time proportional to sort plus $\log_{1/(1-b)} |V|$ times flow.
- A basic implementation using standard libraries is straightforward.

Choice of ℓ

The quality of the cut depends on the line $\ell \in \mathbb{R}^2$ chosen in the first step of the algorithm. Obvious choices include random lines as well as simple fixed directions such as horizontal, vertical, or diagonal. A natural heuristic is then to try multiple lines and increasing balance values and return the best cut (for some objective function that may involve balance and cut size).

Let us demonstrate the effect of ℓ on the cut using the road network of New York[3] as an example. The cut sizes range from 5 (best) to 44 edges (see Fig. 2). The choice of source and sink forces the cut to be in a corridor that, for $b = 1/4$, contains half the graph. If the source/sink assignment violates a sparse cut and the corridor is relatively dense, then *Inertial Flow* finds a suboptimal cut.

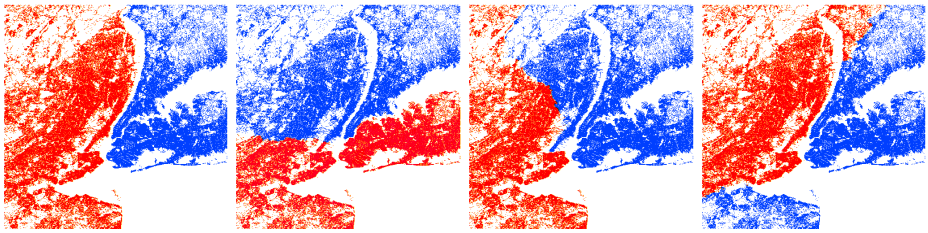

Fig. 2. The road network of New York (264K nodes) cut with balance 1/4 and four different line values. From left to right: horizontal (5 edges cut), vertical (44 edges cut), and diagonal (35 and 25 edges cut, respectively). *Inertial Flow* using horizontal sorting provides the best cut, both visually (along the Hudson) as well as in terms of the number of cut edges. The other sort orders yield comparatively large cuts as the minimum balance criterion forces unfortunate source/sink assignments violating the *Hudson cut*. Note that, compared to a typical worst-case guarantee on the order of $\sqrt{n} \approx 514$, all cuts are smaller by at least an order of magnitude.

[3] The `NY` network contains 264K nodes and 734K arcs (interpreted as 367K undirected edges). All US road networks used for experiments in this paper can be downloaded from `http://www.dis.uniroma1.it/challenge9/download.shtml`

3 Experiments

3.1 Setup

The main datasets we consider are the road networks of the United States and Europe, respectively. The `USA` graph (as used for the 9th DIMACS Implementation Challenge on Shortest Paths [DGJ08]) has 24M nodes and 58M directed arcs, which are typically interpreted as 29M undirected edges. The `EUR` graph (as made available by PTV AG, and also used in [DGJ08]) has 18M nodes and 21M edges (42M arcs).

The method used for comparison is *PUNCH* [DGRW11]. Note that *PUNCH* does not read the embedding, so *Inertial Flow* is given an unfair advantage. A main convenience of *Inertial Flow* as compared to *PUNCH* (and *Buffoon* [SS12]) is that it is straightforward to implement.

Our experiments are meant as a proof of concept, and we use a vanilla implementation (in `C++`) without any additional heuristics. For this paper, our focus is not on running times, and we also refrain from tuning parameters to experimental data. Unless indicated otherwise, balance is set to $1/4$, the lines are chosen to be horizontal, vertical, and diagonal ($\ell \in \{(1,0),(0,1),(1,1),(-1,1)\}$), and the objective function is simply to minimize the number of cut edges. The main subroutines employed are `std::sort` and maximum flow using Dinic's algorithm (augmenting paths, in the unit-capacity case computed by breadth-first search) [Din70]. Our implementation is parallel in the most obvious ways: separators for each line ℓ are computed by separate threads (with cross-notification of minimum cut upper bounds), and recursive calls are handled by a thread pool. For recursive bisections, we run 16 threads on two 2.20GHz Intel Xeon CPUs with 8 cores each. We encourage interested readers to combine *Inertial Flow* with other heuristics and/or to write more efficient implementations.

3.2 Results

Graph size vs. separator size and boundary size. Worst-case bounds for planar graphs on n nodes (and more general graph classes) guarantee the existence of a 1/3–balanced cut/separator of size $O(\sqrt{n})$ [LT79]. Recursive separation yields a partition into $O(n/r)$ regions of size $\leqslant r$ with total boundary size $O(n/\sqrt{r})$. With some more work one can obtain an *r–division* [Fre87], where each region has *worst-case* boundary $O(\sqrt{r})$. Road networks appear to have significantly smaller separators: Delling, Goldberg, Razenshteyn, and Werneck [DGRW11] compare the average boundary size to $\sqrt[3]{r}$ instead (confirmed later by Dibbelt, Strasser, and Wagner [DSW14]). We provide plots for region size vs. total boundary size in Fig. 3. For specific numbers on region size vs. total boundary size, see Table 1 and Table 2.

Running Time. As mentioned above, our main focus is not on running time. Our implementation computes multi-level partitions for `USA` and `EUR` in minutes. Specific numbers are provided in Tables 1, 2, and 3. Note that, as expected, the

initial cuts on the largest graphs are the most expensive ones. Subsequent cuts operate on smaller graphs and, by maintaining nodes in sorted order(s), do not require sorting the nodes again. For example, cutting `USA` into 2 regions requires 81 seconds (Table 3, $b = 1/4$). Recursive bisection into 6K regions takes only roughly twice as long (165.8 seconds, Table 2). Using this recursive bisection tree, reading off an entire multi-level partition is straightforward (see e.g. [KMS13]). By contrast, the *Natural Cut Heuristic* of *PUNCH* [DGRW11] depends on the target region size and is run separately for each level.

3.3 Comparison

Comparing partitions is not straightforward [BMS+14]. We compare against various partitions reported for *PUNCH* in Table 1 and observe that *PUNCH* partitions are significantly more balanced. For example, when partitioning `USA` into 27 regions as in Fig. 1, *Inertial Flow* cuts 1,413 edges with maximum region size 1.4M, while *PUNCH* cuts only 1,404 edges and obtains maximum region size 1M (2^{20}). While recursive bisection with *Inertial Flow* typically uses around 50% more regions than a perfectly balanced partition, *PUNCH* reportedly needs only about 15% more regions. For most partition granularities, the average numbers of cut edges per region are comparable.

We also compare our bisections against the optimal ones, obtained by an efficient algorithm of Delling, Fleischman, Goldberg, Razenshteyn, and Werneck [DFG+14]. Their algorithm guarantees *optimal* bisections for fairly large graphs, so comparing our method without any guarantees on optimality (only balance and running time have worst-case bounds) against their algorithm is not fair. However, we believe that the value of an optimal bisection adds an interesting perspective on cut quality (see Table 3).

Let us restate that the main advantage of *Inertial Flow* over *PUNCH* is simplicity. Another advantage is that multi-level partitions can be computed faster. As cut sizes are comparable, these advantages come at the cost of worse balance. Depending on the application, if better balance is required, a post-processing step (as in *PUNCH* or *Buffoon*) may further improve partitions.

Acknowledgments

Thanks to Ramana Idury for interesting discussions as well as contributions to the experimental framework. Thanks also to Daniel Delling and the anonymous reviewers for their feedback on earlier versions of this paper.

Graph	*PUNCH*				*Inertial Flow*, fixed r			*Inertial Flow*, target regions		
	r	regions	boundary	time	regions	boundary	time	r	boundary	time
Europe	1,024	20,129	168,767	79.7	27,129	208,280	209.5	1,378	171,064	216.0
	4,096	5,000	69,304	62.5	6,808	84,291	211.0	5,536	69,016	204.5
	16,384	1,248	28,448	61.6	1,708	34,839	214.0	22,367	28,236	194.7
	65,536	314	11,403	80.5	431	14,054	218.4	88,856	11,317	199.3
	262,144	81	4,194	106.1	106	5,275	210.5	349,449	4,246	209.2
	1,048,576	22	1,464	147.9	28	2,036	213.9	1,299,633	1,694	202.8
	4,194,304	6	371	196.6	7	573	176.3	4,861,623	461	171.8
USA	1,024	26,725	222,636	104.6	36,267	274,756	246.9	1,389	223,531	186.6
	4,096	6,643	87,762	79.9	9,000	107,170	173.2	5,570	87,193	181.8
	16,384	1,661	34,345	75.0	2,233	41,782	157.7	22,310	34,138	172.0
	65,536	418	12,767	89.9	563	15,862	166.0	87,960	12,971	168.5
	262,144	109	4,556	103.3	140	5,578	163.0	336,843	4,557	166.6
	1,048,576	27	1,504	117.6	33	1,716	148.5	1,407,053	1,413	148.3
	4,194,304	7	383	138.7	8	478	128.4	4,338,122	388	128.7

Table 1. An attempt at comparing partitions obtained by *PUNCH* and recursive bisection using *Inertial Flow*. Values for *PUNCH* were extracted from [DGRW11, Table 1 (average values)]. Each *PUNCH* average is compared to two *Inertial Flow* partitions: a partition with the same region-size constraint r, and a partition with the same number of regions. Center: when computing a partition with the same upper bounds for the maximum region size r, *PUNCH* requires fewer regions; the average number of cut edges per region is comparable. Right: when computing a partition with the same number of regions, the two partitioners cut a similar number of edges (with some *PUNCH* boundaries slightly smaller, particularly for Europe, and partitions more balanced). The running times for *PUNCH* are fairly uniform; for recursive bisection, the smaller r, the longer the computation. Note that a recursive bisection tree with regions of size at most r also contains a partition for any $r' \geqslant r$ (enabling plots like Fig. 3 with thousands of r values), hence it also contains multi-level partitions. Using the USA values in this table as an example, *Inertial Flow* simultaneously computes all 14 partitions (r = 4,338,122 through 2^{10}) in 4.1 minutes.

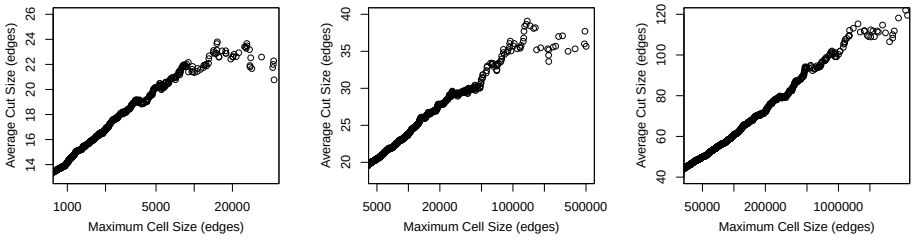

Fig. 3. Average boundary sizes for partitions with various maximum region sizes r (number of edges, logarithmic scale) for the `BAY`, `CAL`, and `USA` road networks, respectively. Worst-case results (such as those for planar graphs) guarantee average boundary sizes proportional to $r^{1/2}$. Delling, Goldberg, Razenshteyn, and Werneck [DGRW11] compare the average boundary size to $r^{1/3}$.

	CAL $\lvert V\rvert = 1.89$M			USA $\lvert V\rvert = 23.9$M			EUR $\lvert V\rvert = 18.0$M		
$\lceil \lvert V\rvert / r \rceil$	regions	boundary	time	regions	boundary	time	regions	boundary	time
2	3	53	3.3	3	140	121.9	3	276	106.2
4	6	103	4.6	6	324	114.7	7	573	135.5
8	12	215	5.8	12	648	125.9	13	1,058	157.8
16	25	441	5.6	24	1,223	120.1	26	1,867	164.3
64	99	1,437	6.0	96	4,234	165.7	98	4,990	169.2
256	387	4,418	6.3	397	12,482	141.5	399	13,280	172.2
1,024	1,561	12,957	6.1	1,593	33,197	145.5	1,592	33,228	170.5
4,096	6,326	36,129	6.9	6,307	84,274	165.8	6,321	80,332	174.7
16,384	25,543	98,232	11.1	25,401	216,078	188.6	25,208	198,433	175.3

Table 2. Recursive bisection using *Inertial Flow* (balance 1/4) on the road networks of California and Nevada (CAL), the United States (USA), and Europe (EUR), respectively, for various values of granularity (maximum region size r). Total region boundaries (cut sizes) reported correspond to the number of edges. Note that these partitions typically have around 50% more regions than necessary due to imperfect balance. Time (in seconds) corresponds to the time of recursive bisection (in particular, reading the graph and its embedding from disk is not included) as required by 16 threads (one bisection occupies 4 threads, one per slope). The variance in running times is rather substantial: even though we report the median among 11 consecutive runs, that median running time, e.g., for USA with $\lceil \lvert V\rvert / r \rceil = 64$ is slower than that for 1,024 even though only a relatively small subset of cuts is computed. The initial cuts of comparably large (sub-)graphs are the most expensive ones.

		Perfect		$b = 2/5$			$b = 1/3$			$b = 1/4$			$b = 1/5$		
Graph	$\lvert V\rvert$	Cut	Time	Cut	Bal.	Time	Cut	Bal.	Time	Cut	Bal.	Time	Cut	Bal.	Time
NY	264K	18	381	40	0.48	0.1	5	0.43	0.1	5	0.43	0.1	5	0.43	0.1
BAY	321K	18	248	28	0.48	0.2	15	0.46	0.1	12	0.46	0.2	12	0.46	0.2
COL	436K	29	2,164	27	0.43	0.2	20	0.36	0.2	14	0.32	0.3	12	0.29	0.3
FLA	1.1M	25	1,640	28	0.42	0.6	22	0.40	0.7	17	0.29	0.9	15	0.27	1.0
NW	1.2M	18	463	24	0.49	0.7	17	0.50	0.6	17	0.50	0.7	17	0.50	0.9
NE	1.5M	24	751	20	0.49	1.3	20	0.49	1.4	20	0.49	1.7	20	0.49	2.3
CAL	1.9M	32	2,658	29	0.49	2.0	29	0.47	2.4	27	0.30	2.2	26	0.30	2.5
EUR	18M	NA	NA	229	0.46	69.3	201	0.45	95.3	188	0.45	124.9	95	0.30	81.4
USA	24M	NA	NA	61	0.48	58.3	61	0.48	63.9	61	0.48	81.2	61	0.48	84.3

Table 3. Bisection of various road networks: perfectly balanced bisections were obtained by Delling, Fleischman, Goldberg, Razenshteyn, and Werneck [DFG+14, Table 4]. The balance of bisections found by *Inertial Flow* depends on the slope and the parameter b and there is no guarantee on optimality. In this table, for each $b \in \{1/5, 1/4, 1/3, 2/5\}$ we provide the minimum number of cut edges among 4 slopes. Balance is reported as the number of nodes in the smaller subgraph divided by the total number of nodes. As in Table 2, times reported are for the bisection (in seconds). When b is close to 1/2, good balance is guaranteed, but cut sizes may be significantly higher, see e.g. NY at 40 edges for $b = 2/5$, which is more than double the size of an optimal bisection. When accepting worse balance, cuts may be substantially smaller.

References

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *39th ACM Symposium on Theory of Computing (STOC)*, pages 227–236, 2007.

[AL08] Reid Andersen and Kevin J. Lang. An algorithm for improving graph partitions. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 651–660, 2008.

[And86] Thomas Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory, Series B*, 41(1):37–47, 1986.

[AST90] Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990. Announced at STOC 1990.

[BCLS87] Thang Nguyen Bui, Soma Chaudhuri, Frank Thomson Leighton, and Michael Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987. Announced at FOCS 1984.

[BCRW13] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. In *40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 93–104, 2013.

[BK04] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. Announced at EMMCVPR 2001.

[BMS+13] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *arXiv*, abs/1311.3144, 2013.

[BMS+14] David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. 2014.

[BMSW13] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering — 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*, 2013.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. Announced at ICCV 1999.

[CKK+06] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006. Announced at CCC 2005.

[DFG+14] Daniel Delling, Daniel Fleischman, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. An exact combinatorial algorithm for minimum graph bisection. *Mathematical Programming Series A*, 2014.

[DGJ08] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*. 2008.

[DGPW13] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. 2013. Announced at SEA 2011 and SEA 2013.

[DGRW11] Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato Fonseca F. Werneck. Graph partitioning with natural cuts. In *25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1135–1146, 2011.

[DHM+09] Daniel Delling, Martin Holzer, Kirill Müller, Frank Schulz, and Dorothea Wagner. High-performance multi-level routing. In *The Shortest Path Problem: 9th DIMACS Implementation Challenge*, volume 74, pages 73–92. 2009.

[Din70] Efim Abramoviq Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Doklady Akademii Nauk SSSR; translation in Soviet Mathematics Doklady*, 11(5):1277–1280, 1970.

[Dji85] Hristo Nikolov Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica. Bulgariacae mathematicae publicationes*, 11(4):369–387, 1985.

[Dji96] Hristo Nikolov Djidjev. Efficient algorithms for shortest path problems on planar digraphs. In *22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 151–165, 1996.

[DSW14] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *13th International Symposium on Experimental Algorithms (SEA)*, pages 271–282, 2014.

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Announced at FOCS 2001.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[GHK+11] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Robert Endre Tarjan, and Renato Fonseca F. Werneck. Maximum flows by incremental breadth-first search. In *19th European Symposium on Algorithms (ESA)*, pages 457–468, 2011.

[GHT84] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984. Announced as TR82-506 in 1982.

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998. Announced at FOCS 1997.

[HKRS97] Monika Rauch Henzinger, Philip Nathan Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. Announced at STOC 1994.

[HSW08] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multi-level overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13, 2008. Announced at ALENEX 2006.

[KKS11] Ken-ichi Kawarabayashi, Philip Nathan Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 135–146, 2011.

[KLSV10] Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed time-dependent contraction hierarchies. In *9th International Symposium on Experimental Algorithms (SEA)*, pages 83–93, 2010.

[KMS13] Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *45th ACM Symposium on Theory of Computing (STOC)*, pages 505–514, 2013.

[KRV09] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM*, 56(4), 2009. Announced at STOC 2006.

[KV05] Subhash Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 53–62, 2005.

[LR04] Kevin J. Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *10th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 325–337, 2004.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 253–262, 2013.

[MS12] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–222, 2012.

[OSVV08] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. On partitioning graphs via single commodity flows. In *40th ACM Symposium on Theory of Computing (STOC)*, pages 461–470, 2008.

[She09] Jonah Sherman. Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut. In *50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 363–372, 2009.

[Som14] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46:45:1–31, 2014.

[SS11] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *19th European Symposium on Algorithms (ESA)*, pages 469–480, 2011.

[SS12] Peter Sanders and Christian Schulz. Distributed evolutionary graph partitioning. In *14th Meeting on Algorithm Engineering & Experiments, (ALENEX)*, pages 16–29, 2012.

[SS13] Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *12th International Symposium on Experimental Algorithms (SEA)*, pages 164–175, 2013.

[ST97] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18:1436–1445, 1997.

[Ung51] Peter Ungar. A theorem on planar graphs. *Journal of the London Mathematical Society*, s1-26(4):256–262, 1951.

[vWZA13] Freek van Walderveen, Norbert Zeh, and Lars Arge. Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In *24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 901–918, 2013.