Randomized and Reactive Algorithms for Graph and Hypergraph Partitioning

Roberto Battiti

Università di Trento – Italia

May 20, 2009

Outline

Introduction

0-1 *graph bisection*, or *graph bi-partitioning* problem on graph $G = (V, E)$.

NP-hard for general graphs as well as for bipartite graphs (Garey and Johnson, 1979)
even finding good approximation solutions for general graphs or arbitrary planar graphs is NP-hard (Bui and Jones 1992)
approximation algorithms (... into two bounded but not necessarily equal-sized sets) are not practical (Leighton and Rao, 1988).

heuristics
experimental algorithmics (see physics)

The partitioning problem arises in many areas of computer science, e.g., network partitioning and VLSI circuit placement
(Dunlop and Kernighan, 1985).

- (Kernighan and Lin, 1970)
- Simulated Annealing:
  (Johnson et al., 1989), following (Kirkpatrick, Gelatt, and Vecchi, 1983) (Sechen and Sangiovanni–Vincentelli, 1986)
- GRASP: (Laguna, Feo, and Elrod, 1994)
- Genetic algorithms: ... (Rolland and Pirkul, 1992)
- Tabu Search: (Rolland, Pirkul, and Glover, 1996) (Dell'Amico and Maffioli, 1996)
- (Bui and Moon, 1996), BFS-GBA

Papers

- R. Battiti and A.A. Bertossi, Differential greedy for the 01 equicut problem, Network Design: Connectivity and Facilities Location (D.Z. Du and P.M. Pardalos, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 40 (1997), 322.
- R. Battiti, A.A. Bertossi, and R. Rizzi, Randomized Greedy Algorithms for the Hypergraph Partitioning Problem, Randomization Methods in Algorithm Design (P. Pardalos, S. Rajasekaran, and J. Rolim eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 43 (1998), 2135.
- S. L. Bezrukov and Roberto Battiti, On Partitioning of Hypergraphs, Discrete Applied Mathematics.
- R. Battiti and A. Bertossi, Greedy, Prohibition, and Reactive Heuristics for Graph- Partitioning, IEEE Transactions on Computers, vol. 48 (1999), no. 4, 361385.

Test beds and exp. setup (graphs)

eight random graphs and eight "geometric" graphs (Johnson et al., 1989), with n. vertices from 500 to 1000, 24 graphs proposed in (Bui and Moon, 1996), with n. vertices from 134 to 5252

- $Gn.d$: random, edge with probability p, such that expected vertex degree is d.
- $Un.d$: geometric, edge if and only if Euclidean distance is t or less, $d = n\pi t^2$ is the expected vertex degree.
- $breg.n.b$: random regular of degree 3, optimal bisection size b with probability $1 - o(1)$.
- $cat.n$: "caterpillar", each vertex on the spine is connected to six "legs"
- $rcat.n$: "caterpillar", each vertex has $\sqrt{n}$ legs.
- $grid.n.b$: grid, optimal bisection b.
- $w - grid.n.b$: boundaries wrapped around.

Min-Max Greedy

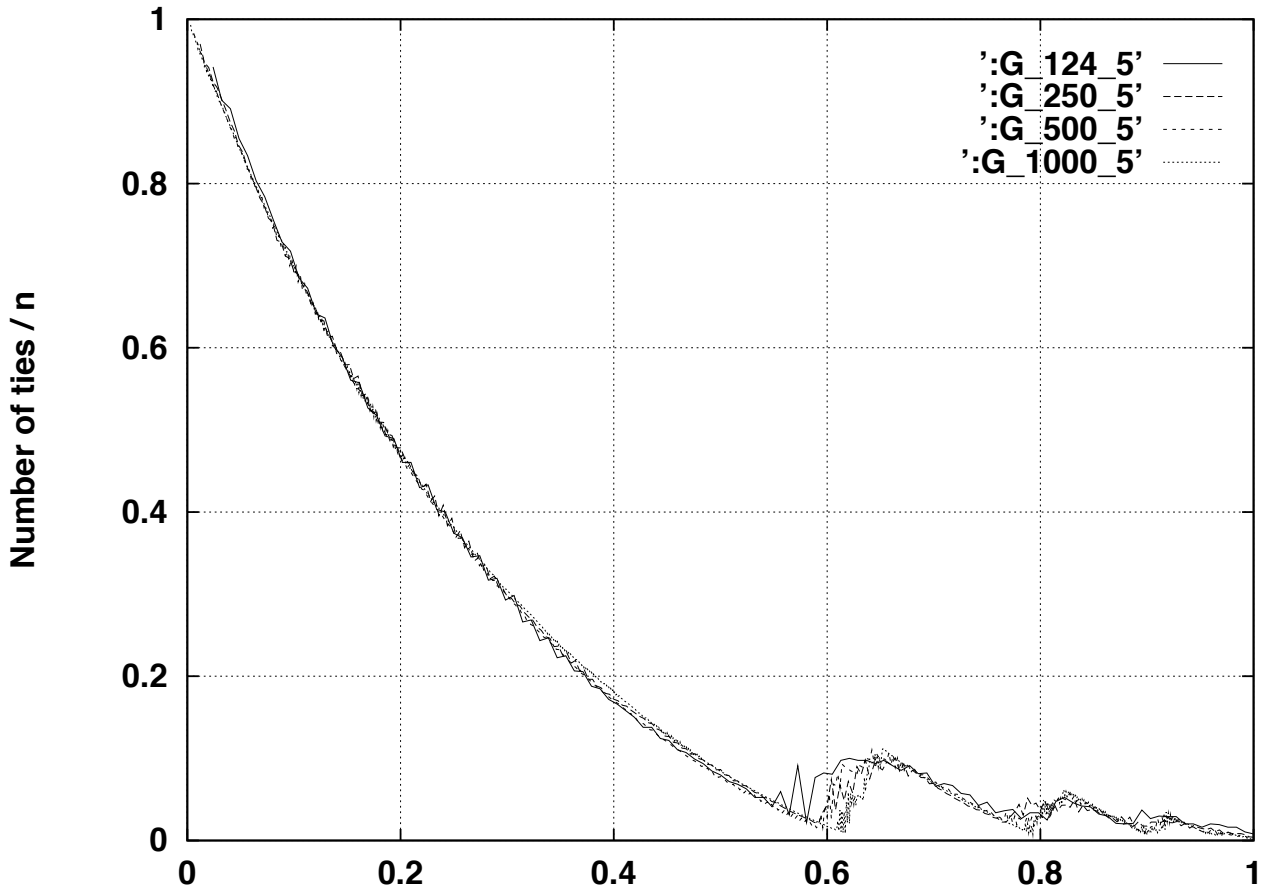
Greedy is an old technique

(Laguna, Feo, and Elrod, 1994)

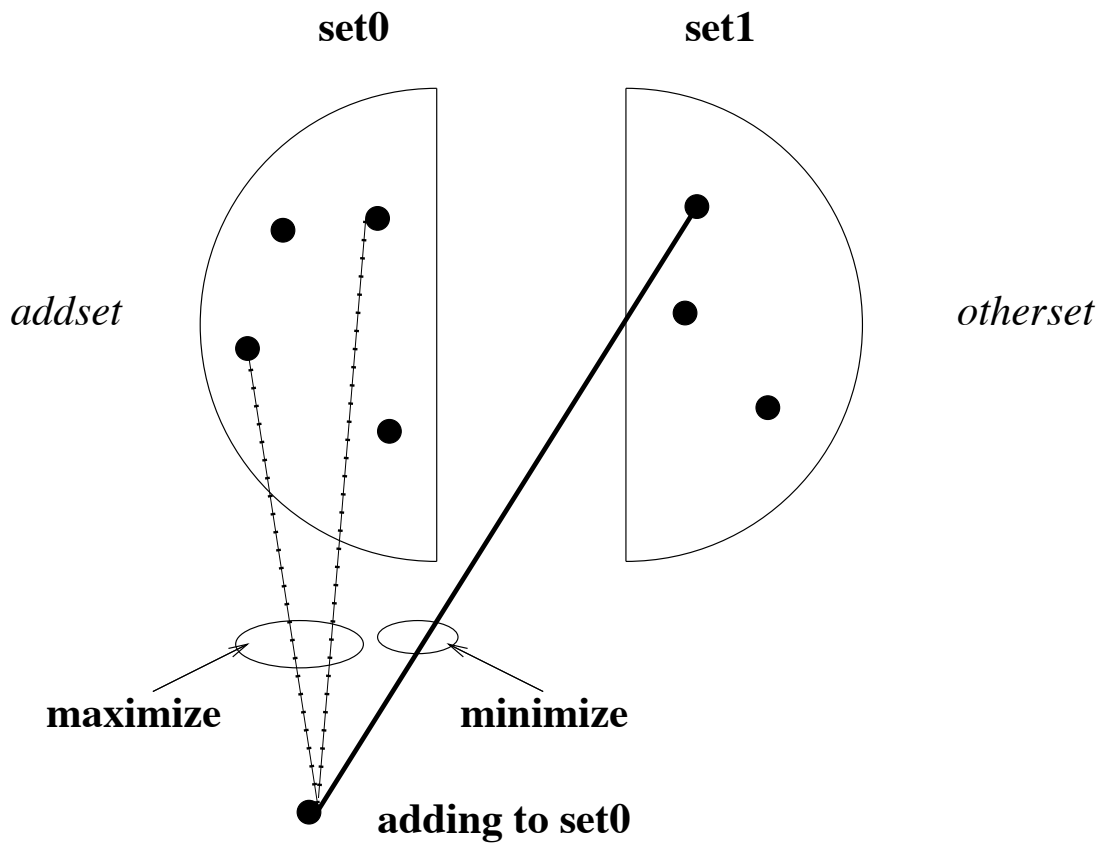
(Dell'Amico and Maffioli, 1996)

Min-Max Greedy

during most iterations, the algorithm is forced to make a random choice between a large no. of candidates, of a size comparable to that of the *tobeadded* set



Min-Max Greedy



tie–breaking rule: prefer candidates with the largest no. of edges to *addset* the more edges become internal during the initial iterations, the less the probability that one will be forced to place large no. of edges across the cut during later steps.

Min-Max Greedy

1 $in0 \leftarrow$ **random vertex** $\in \{1, ..., n\}$
2 $in1 \leftarrow$ **random vertex** $\in \{1, ..., n\} \setminus \{in0\}$
3 $set0 \leftarrow \{in0\}$
4 $set1 \leftarrow \{in1\}$
5 $tobeadded \leftarrow V \setminus \{in0, in1\}$
6 $f \leftarrow 0$
7 **if** $(in0, in1) \in E$ **then** $f \leftarrow 1$
8 $addset \leftarrow 1$
9 **while** $|tobeadded| > 0$ **do**
10 $addset \leftarrow (1 - addset)$
11 $otherset \leftarrow (1 - addset)$
12 $minedges \leftarrow \min_{i \in tobeadded} E(i, otherset)$
13 $candidates \leftarrow \{i \in tobeadded$ **such that** $E(i, otherset) = minedges\}$
14(*) $maxedges \leftarrow \max_{i \in candidates} E(i, addset)$
15(*) $candidates \leftarrow \{i \in candidates$ **such that** $E(i, addset) = maxedges\}$
16 $bestvertex \leftarrow$ **random vertex** $\in candidates$
17 $addset \leftarrow addset \cup \{bestvertex\}$
18 $f \leftarrow f + minedges$
19 $tobeadded \leftarrow tobeadded \setminus \{bestvertex\}$
20 **return** f

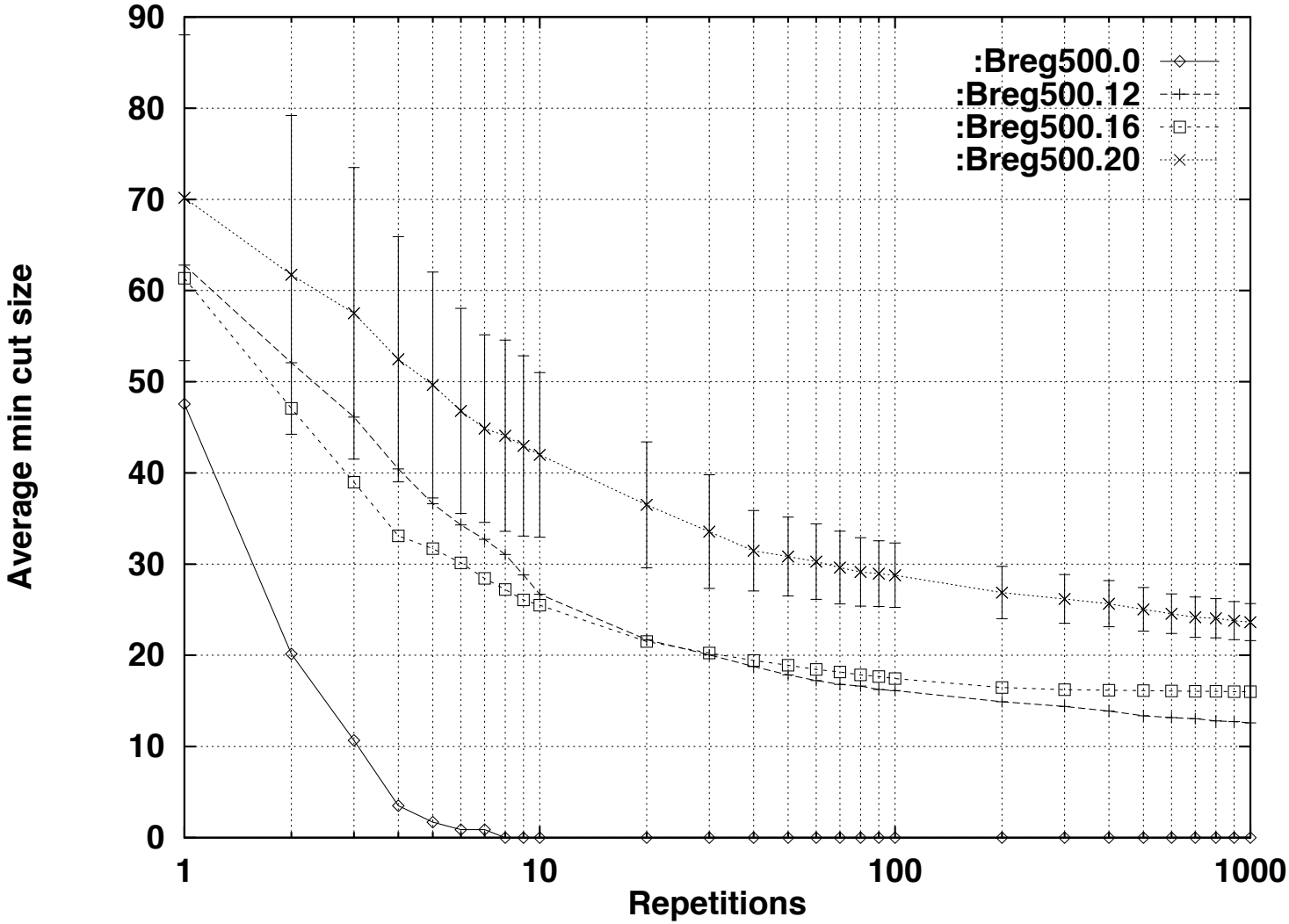
Independent repetitions of Min-Max Greedy

Min-Max Greedy is much more effective than standard greedy.
The algorithm is randomized: can repeat and take min value.

repetition

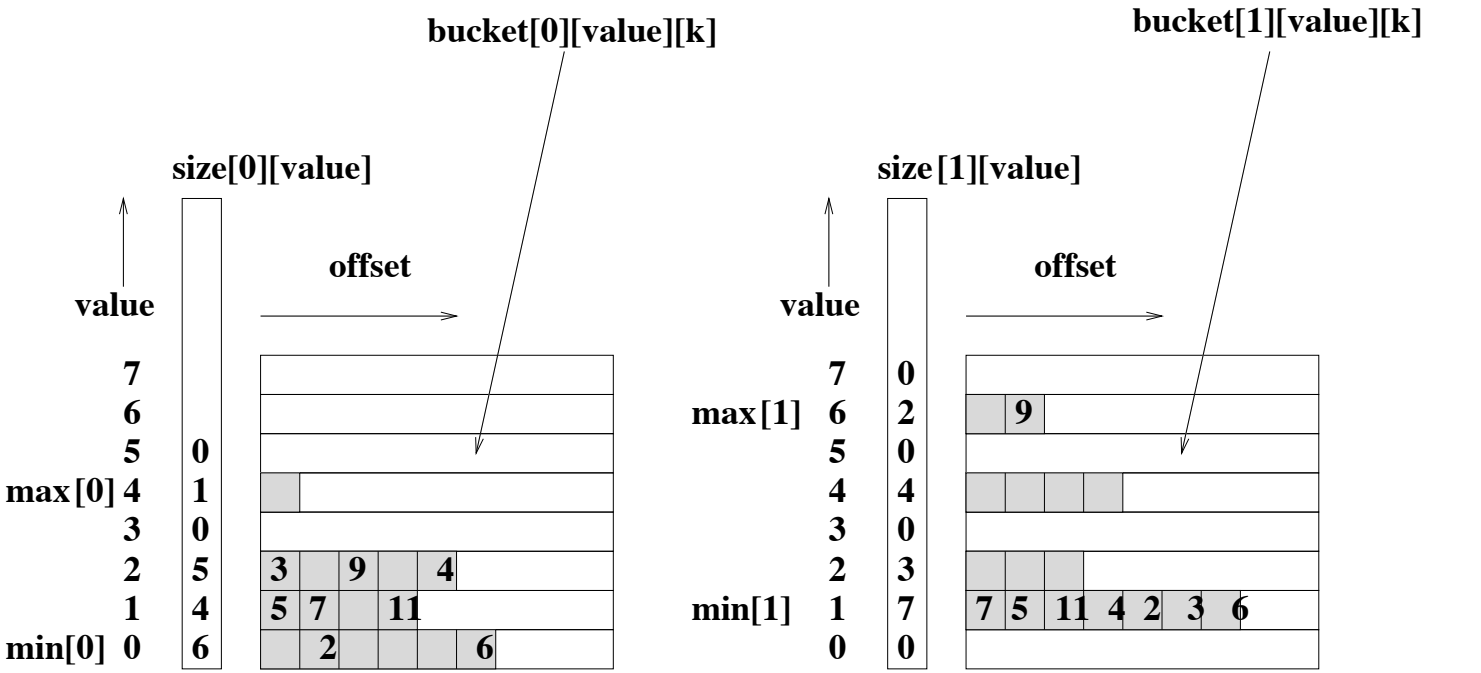
repetition of **Min-Max Greedy** easily solves most of the regular, "caterpillar", and grid graphs, in a very small CPU time.

Min-Max Greedy



Average min cut as a function of repetitions (Breg500 tasks).

Implementation of Min-Max Greedy



bucket for: number of edges to set0

bucket for: number of edges to set1

Implementation of Min-Max Greedy

- Worst-case computational complexity to update the *max* and *min* values is $O(1)$ after $Insertion(v)$, and $O(d_{max})$ after $Deletion(v)$
- For a *sequence* of at most d_{max} $Deletion(set, v)$ operations, $O(d_{max})$ is an upper bound on the complexity needed to update *min* during the entire sequence
- single randomization of the indices executed before the main loop starts

Implementation of Min-Max Greedy

Determination of the winning candidate in the "first-min" option. Naive implementation if (*) is absent:

BEST-VERTEX- "first min" implementation

```
1     otherset ← (1 − addset)
2     minedges ← min[otherset]
3     maxedges ← max[addset]
4     maxreached ← −∞
5     value ← minedges
6     for offset ← 0 to size[otherset][value] − 1 do
7         vertex ← bucket[otherset][value][offset]
8(*)      if E(vertex, addset) = maxedges then return vertex
9         if E(vertex, addset) > maxreached then
10            maxreached ← E(vertex, addset)
11            bestvertex ← vertex
12    return bestvertex
```

- during the first iterations in the main loop a very large no. of candidates can be present in *bucket*[1][*min*[1]]

Implementation of Min-Max Greedy

determination of the winning candidate in the "first-max" option:

BEST-VERTEX- "first max" implementation

```
1   otherset ← (1 − addset)
2   minedges ← min[otherset]
3   maxedges ← max[addset]
4   for value ← maxedges downto 0 do
5       for offset ← 0 to size[addset][value] − 1 do
6           vertex ← bucket[addset][value][offset]
7           if E(vertex, otherset) = minedges then return vertex
```

- for random low-density graphs and in the initial phase of the greedy construction, there is a large probability that, among the vertices connected to *set*0 with the largest no. of edges, there will be vertices not connected to *set*1

Implementation of Min-Max Greedy

- CPU time required by the "first-max" implementation of the *Min-Max Greedy* algorithm is not much larger than that for a random assignment. Average complexity of $O(|E|)$ for low-density graphs.
- notable speedup is obtained by passing from naive implementations of the
  Min-Max Greedy algorithm, to "first-max"
- speedup is larger for the lower density graphs
- speedup increases rapidly with the problem dimension

Differential Greedy

Differential Greedy

Prefer vertices that minimize the *difference* between new edges across the cut and new internal edges.

Differential Greedy

The Differential Greedy algorithm:

```
DIFF-GREEDY
1   in0 ← random vertex ∈ {1, ..., n}
2   in1 ← random vertex ∈ {1, ..., n} \ {in0}
3   set0 ← {in0}
4   set1 ← {in1}
5   if (in0, in1) ∈ E then f ← 1 else f ← 0
6   tobeadded ← V \ {in0, in1}
7   addset ← 1
8   while |tobeadded| > 0 do
9       addset ← (1 − addset)
10      otherset ← (1 − addset)
11      m ← min_{i∈tobeadded} E(i, otherset) − E(i, addset)
12      candidates ← {i ∈ tobeadded: E(i, otherset) − E(i, addset) = m}
13      bestvertex ← random vertex ∈ candidates
14      addset ← addset ∪ {bestvertex}
15      f ← f + minedges
16      tobeadded ← tobeadded \ {bestvertex}
17  return f
```

Implementation: "max-min priority queue" with buckets, *amortized cost* of $O(1)$ per iteration.

Faster, comparable, in some cases better (and different).

Prohibition-Based (Tabu) Search

Prohibition-Based (Tabu) Search

use of *prohibition* techniques and "intelligent" schemes as a complement to basic *local search* heuristics
(Glover, 1989), (Hansen and Jaumard, 1990)
denial strategy of (Steiglitz and Weiner, 1968) for TSP

(Kernighan and Lin, 1970) for GP
variable depth search

- At each KL cycle one starts from a legal configuration, and applies a *chain of* $n/2$ *tentative* 2-exchanges
- As soon as two nodes are tentatively exchanged, their membership is "frozen"
- At the end of the cycle one derives the *minimum* cut size along the chain

Prohibition-Based (Tabu) Search

Prohibition of the inverse of recent moves Fixed-TS (ex: binary strings)

$$N_A(X^{(t)}) = \{X = \mu \circ X^{(t)} \text{ such that } \mathit{LastUsed}(\mu^{-1}) < (t - T)\}$$

Relationship between Fixed-TS and KL:

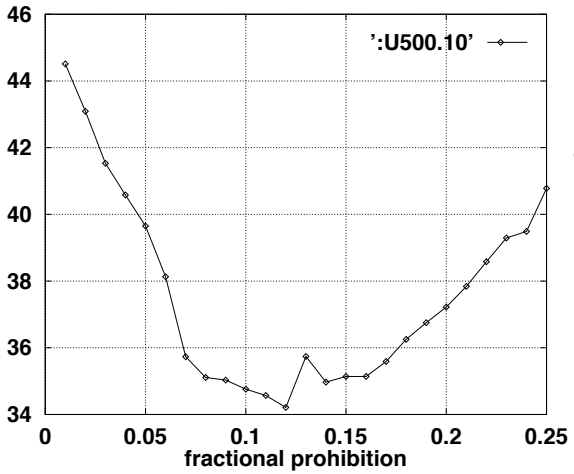
a single cycle of KL from Fixed-TS by using 2-exchanges as basic moves and by setting $T = n/2$ and $\mathit{LastUsed}(\mu) = -\infty$ at the beginning of each cycle.

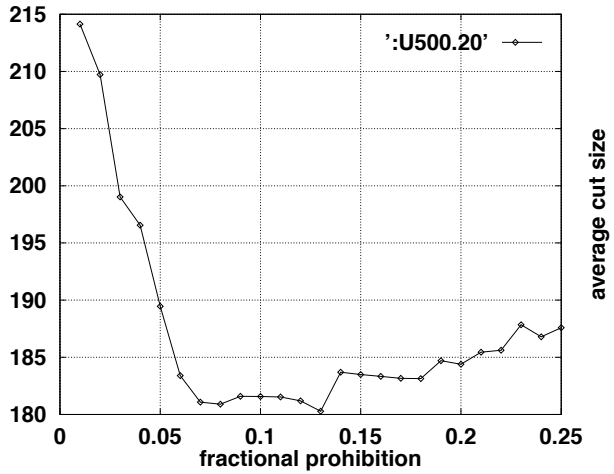
Fixed-TS for graph partitioning

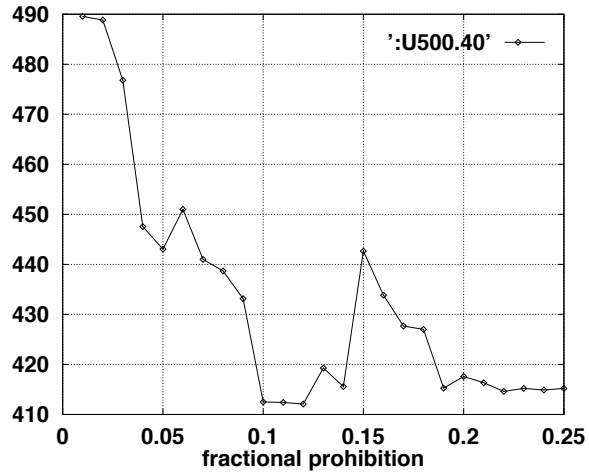
```
FIXED-TS(T_f, iterations)
1   ♢ Current X must be legal:  N_0 ≡ |{i : X(i) = 0}| =
2   n/2
3   ♢ iterations must be even
4   T ← ⌊T_f n⌋
5   for it ← 1 to iterations do
6       if N_0 ≥ n/2 then addset ← 1   else addset ← 0
7       otherset ← (1 − addset)
8       v ← BEST-MOVE(otherset)
9       addset ← addset ∪ {v}
10      otherset ← otherset \ {v}
11      LASTUSED[v] ← t
12      t ← t + 1
        if N_0 = n/2 and f < f_min then    f_min ← f

BEST-MOVE(set)
13  ♢ Searches in set for vertex to move
14  for g ← gmax[set] downto gmin[set] do
15      forall vertex ∈ bucket[set][g] do
16          if ALLOWED(vertex)   return vertex
```

cut sizes are recorded only for legal assignments (N_0 is the no. of vertices in *set*0)

Fixed-TS : effect of fixed prohibition on the performance





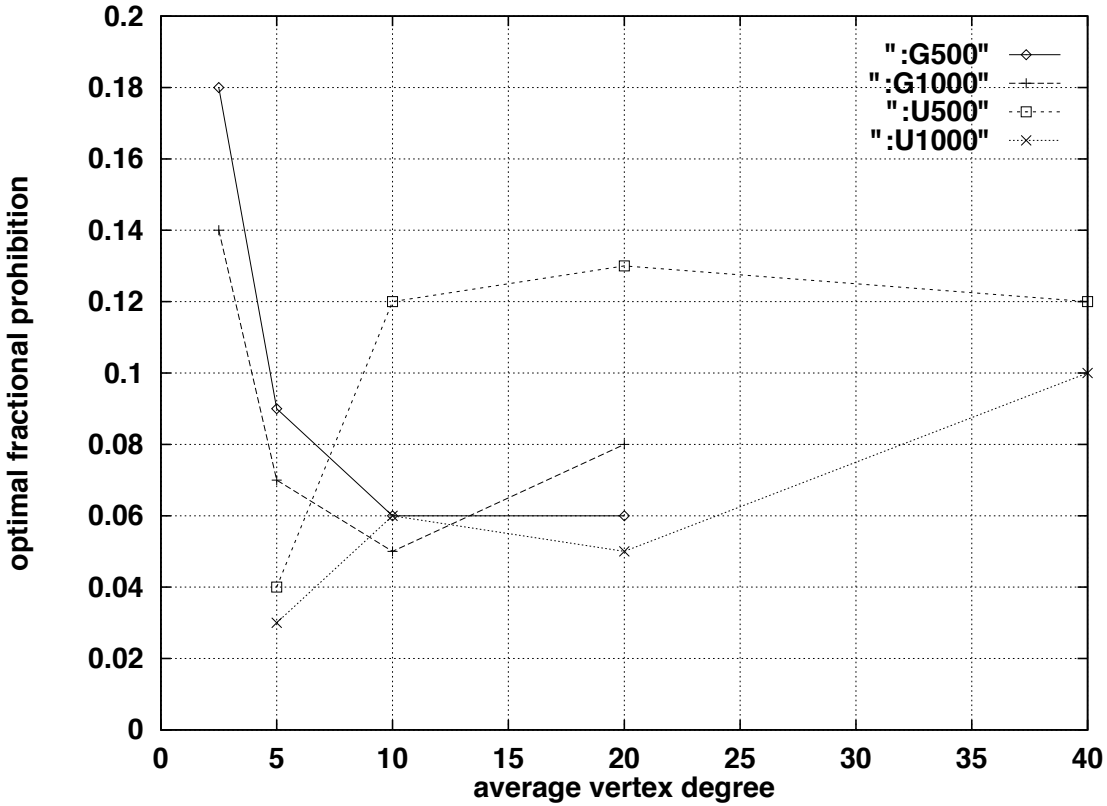


Effect of fixed prohibition on the performance:

the prohibition T has a crucial effect on the performance.

Fixed-TS : effect of fixed prohibition on the performance

Determining the appropriate prohibition for a given graph is not a trivial task



Fixed-TS (100 n steps): best fractional prohibition as a function of the average degree for the random and geometric graphs

Randomized-TS : randomized prohibition

Randomized-TS algorithm:

```
RANDOMIZED-TS(iterations, individual )
1    t ← 0
2    while t < iterations do
3        MIN-MAX-GREEDY
4        t_end ← t + individual
5        while t < t_end do
6            tmp ← RANDOM (1, 25)
7            T_f ← tmp 0.01
8            FIXED-TS(T_f, n)
```

Randomized-TS : randomized prohibition

Randomized-TS:

the randomized algorithm reach results that are *comparable to the best results* of the Fixed-TS algorithm

for some graphs with the larger densities the randomized options achieves results that are *better* than the best Fixed-TS results.

RRTS : reactive and randomized prohibition

Reactive Search Optimization (RSO) heuristics

are based on self-tuning schemes, acting while the algorithm runs *history-sensitive* task properties and local properties of the configuration space can be used for the tuning of parameters, like T

a *successful* value of T is chosen with a higher probability and kept for a longer phase.

the *successfulness* of a given T_f :

- preliminary "scoring" phase by measuring the "speed of improvement" in short runs
- (later phases) by observing whether the current value led to a new best assignment.

RRTS : randomized and reactive prohibition

```
SCORING(trials)
1   f_best ← ∞ ; elite ← ∅
2   forall T_f ∈ {0.01, 0.02, ..., 0.25} do
3       vote(T_f) ← 0
4       for try ← 1 to trials do
5           MIN-MAX-GREEDY
6           LOCAL-SEARCH
7           f_min ← f ; f_start ← f ; t_start ← t
8           do
9               FIXED-TS(T_f, 2(⌊T_f n⌋ + 1))
10              LOCAL-SEARCH
11          while (t < t_start + n/2)
12          if f_min < f_best then
13              T_{f_best} ← T_f ; f_best ← f_min
14          elite ← elite ∪ {best assignment found in the current trial}
15          vote(T_f) ← vote(T_f) + (f_start − f_min)/(t − t_start)
16  forall T_f ∈ {0.01, 0.02, ..., 0.25} do
17      if max vote(T_f) ≠ min vote(T_f) then
18          vote(T_f) ← 0.1 + 0.9(vote(T_f) − min vote(T_f))/(max vote(T_f) − min vote(T_f))
19  return T_{f_best}
```

RRTS: randomized and reactive prohibition

Reactive-Randomized-TS algorithm:

```
REACTIVE-RANDOMIZED-TS(iterations , individual )
1   T_{f_best} ← SCORING(3)
2   for i ← 1 to ⌈iterations/individual⌉ do
3       if elite ≠ ∅ then X ← extract best assignment from elite
4       else MIN-MAX-GREEDY
5       T_f ← T_{f_best}
6       t_individual ← t
7       do
8           t_start ← t
9           do
10              FIXED-TS(T_f, 2(⌊T_f n⌋ + 1))
11              LOCAL-SEARCH
12          while (t < t_start + n)
13          if t_min ≤ t_start then
14              T_f ← random in {0.01, 0.02, ..., 0.25} with probability(T_f) ∝ vote(T_f)
15      while (t < t_individual + individual)
```

RRTS : randomized and reactive prohibition

Graph	BFS-GBA			RRTS, 100 n iter.			Best	RRTS, 1000 n iter.			
	Ave	CPU	CPU_n	Ave	(Sdev.)	CPU		Min	Ave	(Sdev.)	CPU
G500.2.5	53.97	5.96	*0.46*	55.90	(1.2)	0.17	49	51	**52.06**	(0.50)	2.0
G500.05	222.13	8.09	*0.63*	**221.24**	(1.9)	0.20	218	218	**218.29**	(0.46)	2.5
G500.10	631.46	11.71	*0.92*	**629.33**	(2.0)	0.27	626	626	**626.44**	(0.59)	3.6
G500.20	1752.51	21.60	*1.70*	**1747.94**	(2.6)	0.58	1744	1744	**1744.36**	(0.66)	6.8
G1000.2.5	103.61	16.83	*1.32*	107.18	(2.3)	0.49	95	95	**98.69**	(1.01)	6.5
G1000.05	458.55	26.65	*2.09*	**458.32**	(2.4)	0.47	445	445	**450.99**	(1.43)	6.5
G1000.10	1376.37	37.05	*2.91*	**1371.47**	(3.5)	0.67	1362	1362	**1364.27**	(1.38)	9.3
G1000.20	3401.74	62.25	*4.90*	**3389.64**	(4.1)	1.10	3382	3382	**3383.92**	(1.00)	14.7
U500.05	3.65	7.54	*0.59*	**2.02**	(0.1)	0.11	2	2	**2**	(0)	1.7
U500.10	32.68	9.59	*0.75*	**26**	(0)	0.26	26	26	**26**	(0)	2.7
U500.20	179.58	11.50	*0.90*	**178**	(0)	0.47	178	178	**178**	(0)	5.3
U500.40	412.23	9.92	*0.78*	**412**	(0)	0.89	412	412	**412**	(0)	10.2
U1000.05	1.78	17.58	*1.38*	**1**	(0)	0.25	1	1	**1**	(0)	4.2
U1000.10	55.78	30.89	*2.43*	**39.76**	(0.8)	0.59	39	39	**39.03**	(0.19)	6.3
U1000.20	231.62	32.97	*2.59*	**222.27**	(1.4)	1.01	222	222	**222**	(0)	12.5
U1000.40	738.10	36.99	*2.91*	**737**	(0)	2.02	737	737	**737**	(0)	24.3

RRTS : randomized and reactive prohibition

$100n$ iterations:

CPU times of RRTS are significantly lower
RRTS obtains a significantly better performance

$1000n$ iterations:

average values very close to the Best values

Conclusions (Graphs)

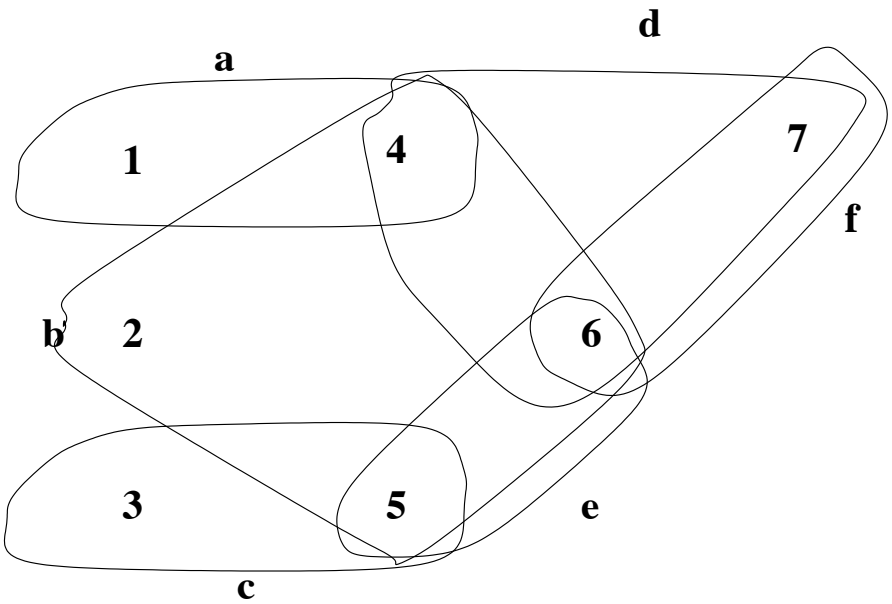
Conclusions (Graphs):

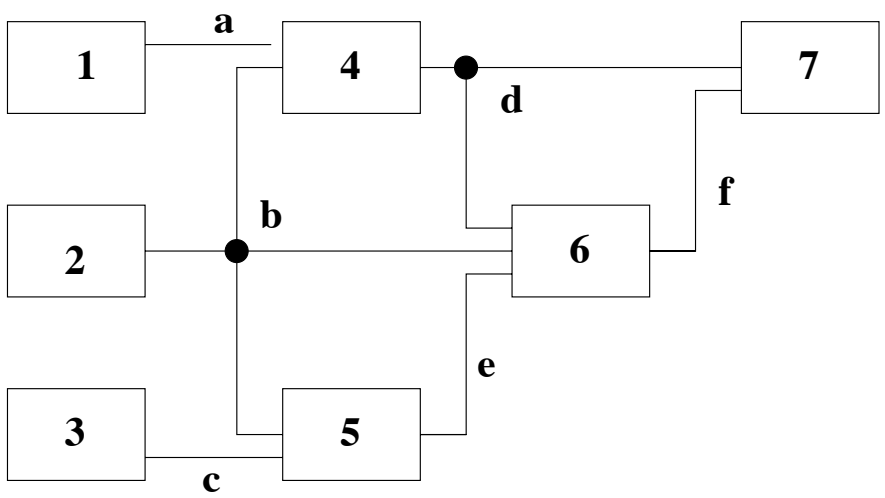
- *Min-Max Greedy* and implementations
- independent repetitions
- *prohibition–based* strategy, T
- *RRTS*
  Reactive-Randomized-TS
  for choosing T

Hypergraphs

Hypergraphs

Generalization of a graph: $H = (V, E^h)$
each hyperedge is a subset of V





Hypergraph Partitioning

Partition the vertices into k roughly equal parts, minimize cut hyperdeges

Applications:

- VLSI design (packaging, synthesis, rapid prototyping, simulation and testing)
- efficient storage of large databases, data mining

NP-hard problem, many heuristics:

- Alpert and Kahng survey (1995)
- KL ... Fiduccia – Mattheyses (1982), C. Alpert version
- Krishnamurthy "lookahead" (1984)
- Dutt and Deng PROP (1996)
- Multi-level partitioning (Karpis, Aggarval, Kumar, Shekar)

Greedy Approaches

General scheme:

First, nodes are added alternatively to Set 0 and Set 1 until sizes reaches *min_size*.
Remaining free nodes are inserted in either set.

Greedy is fast,
if **randomized** can use repetitions,
can be used as starting point

Δf **Greedy**

Choose randomly and with uniform probability between those moves leading to smallest increase of broken edges.

Greedy Approaches

Critical Edges Greedy

gain of a free node v is the no. of 1-*critical* edges containing v minus the no. of 0-*critical* edges containing v.
Insert according to max (set1) or min (set0) gain

Min-Max Greedy

Δf Greedy, break ties by considering the no. of critical edges cont. v which are not broken after inserting v.

Greedy Approaches

Δ-broken Greedy

gain[v] is the no. of edges which get broken by inserting v into Set 0 minus the no. of edges which get broken by inserting v into Set 1.

Krishnamurthy Greedy

Lookahead gain calculation scheme (used by K. in the context of *move-based* approaches, like FM).

Greedy Approaches

Probabilistic Greedy

Gain: sum over all hyperdeges cont. v

$$gain(v) = \sum_{h \ni v} gain(v; h)$$

If h is broken, or if all nodes in h are free, then $gain(v; h) = 0$. Otherwise:

$$gain(v; h) = \begin{cases} \frac{1}{2^{|h|_f - 1}} & \text{if } |h|_1 > 0 \text{ and } |h|_0 = 0 \\ -\frac{1}{2^{|h|_f - 1}} & \text{if } |h|_0 > 0 \text{ and } |h|_1 = 0 \end{cases}$$

$|h|_i = |h \cap Set\ \ i|$ (for $i = 1, 2$)
$|h|_f =$ no. of free nodes in h

Test beds (hypergraphs)

Collection by Chuck Alpert: "The Circuit Partitioning Page"

The nets are collected into four major groups:

- **Test** nets consist mainly in peculiar hypergraphs and are useful for a general testing of the algorithms.
- **Cheng's nets** were obtained from Professor C.K. Cheng at UCSD.
- **VPNR nets** were obtained by translating in netlist format some instances first represented in VPNR format.
- **Large nets** were obtained from Lars Hagen.

Nets with up to approx. 100,000 vertices, 150,000 hyperdeges

Exp. results (greedy on hypergraphs)

golem3 103,048 vertices , 144,949 hyperedges

Method	ave	(stdev)
random	81,559.5	(303.9)
Δf Greedy	32,401.6	(127.9)
Min-Max Greedy	6,810	(332.5)
Δ-boken Greedy	6,764.3	(348.7)
Critical Edges Greedy	3,479.8	(904.8)
Fiduccia - Mattheyses	3,251.1	
Prob-Greedy (1000 rep.)	**3,067.9**	(313.2)

Min. Prob-Greedy (1000 rep.): 1,775

State of the art: 1,424

(Multilevel hMETIS by Karpis et al.)

Exp. results: greedy + LS + TS

Use Greedy to generate starting points

file	GRE min	ave	LS min	ave	TS min	ave
baluP:0.05	3.7	97.9	3.7	73.4	**0.0**	12.1
p1:0.05	21.3	79.7	19.1	69.6	**0.0**	36.4
bm1:0.05	25.5	81.1	17.0	71.2	**0.0**	34.5
t4:0.05	41.7	135.4	33.3	123.4	8.3	76.6
t3:0.05	5.3	74.1	5.3	68.4	**0.0**	36.5
t2:0.05	18.4	60.7	17.2	57.2	**0.0**	32.5
t6:0.05	6.7	48.7	6.7	44.7	**0.0**	22.8
structP:0.05	30.3	163.7	27.3	156.0	3.0	74.7
t5:0.05	5.6	79.4	5.6	62.6	4.2	32.8
19ks:0.05	27.9	119.9	26.0	112.8	**0.0**	55.7

Comparisons with state of the art 2009

- Previous work not widely known, in some cases considered "complex"
- Multilevel schemes
- Is the multi-level overhead experimentally justified?
- What is the effect of different coarsening schemes when combined with state-of-the-art heuristics?
- Consideration of more challenging graphs

Multilevel approach

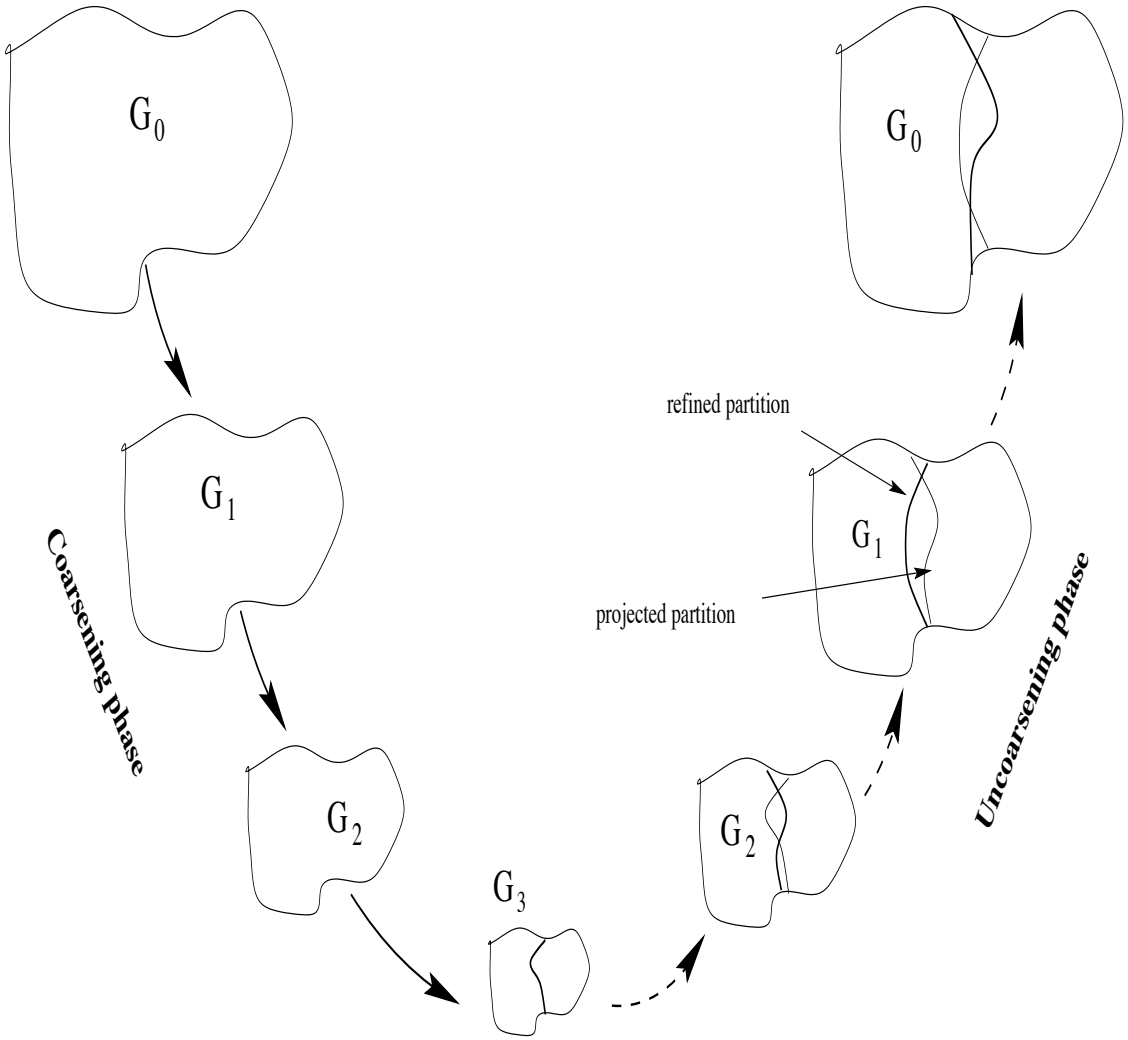
Multilevel scheme for partitioning a graph:

- **coarsening** phase
- **partitioning** phase
- **projection** phase: the projection reinforced by a refinement method is called **uncoarsening**

Why could it work

- **Learning** the graph structure before optimizing
- **Big steps**
- **Simplified fitness surface**

Multilevel approach



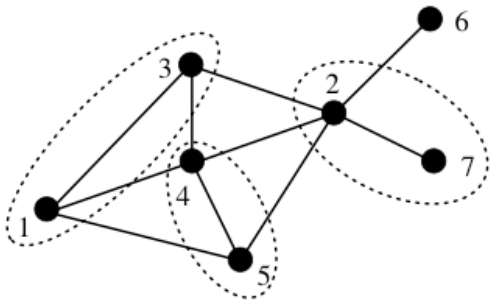
Multilevel approach

Two different coarsening schemes:

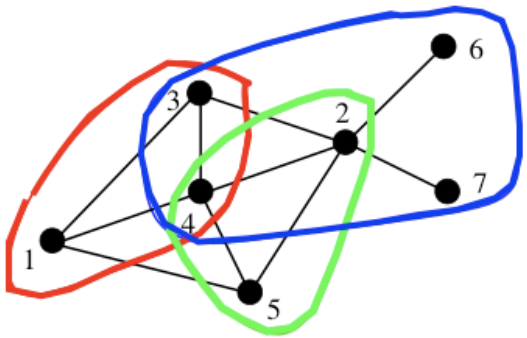
- **strict aggregation** (SAG): in the coarsening phase each vertex in the fine graph belongs to one and only one specific partition in the coarse graph

- **weighted aggregation** (WAG): each vertex can be divided into fractions and different fractions of the same vertex can belong to different partitions

SAG



WAG



Aggregation techniques in SAG

Different methods to match (aggragate) two nodes in the SAG technique:

- **Heavy Edge Matching (HEM)**: a vertex is matched with a random unmatched connected vertex
- **Modified Heavy Edge Matching (HEM*)**: a vertex u is matched with the connected vertex v that has several adjacent vertices connected to u
- **Heaviest Edge Matching (HEAV)**: the edges are ordered by decreasing weights and if the two vertices of the considered edge are unmatched, they are matched

Min Max Greedy vs Differential Greedy

Diff-Greedy is faster and obtains better results (best minimum)

graph	MIN MAX GREEDY (100.000 runs) min	avg	std.dev	time	DIFFERENTIAL GREEDY (100.000 runs) min	avg	std.dev	time
airfoil1	1,07	2,23	45,50	0,00	1,04	2,43	50,20	0,00
bcspwr09	1,00	6,27	23,50	0,00	1,00	6,53	25,40	0,00
bcsstk13	1,03	1,55	843,50	0,01	1,00	1,46	768,20	0,01
big	1,06	2,93	115,00	0,02	1,09	3,35	146,60	0,02
DEBR12	1,17	1,55	90,30	0,01	1,18	1,69	91,10	0,00
nasa4704	1,03	1,79	491,40	0,02	1,02	1,72	402,80	0,01
wave	1,12	1,81	3.309,80	0,55	1,11	1,77	3.227,60	0,35
4elt	1,06	2,93	115,00	0,02	1,09	3,35	146,60	0,02
add32	1,00	5,79	24,80	0,01	1,00	4,38	21,80	0,00
bcsstk32	1,68	3,69	4.317,50	0,26	1,57	3,34	4.043,30	0,15
brack2	1,42	6,23	2.240,10	0,20	1,56	6,19	2.231,00	0,07
crack	1,08	1,94	75,90	0,01	1,11	2,02	81,00	0,01
cs4	1,27	2,44	169,10	0,04	1,32	3,09	243,50	0,03
cti	1,02	3,30	323,70	0,03	1,01	3,31	319,10	0,02
t60k	1,89	8,51	158,80	0,07	1,66	11,24	259,30	0,04
uk	1,30	3,91	23,80	0,00	1,15	3,94	23,60	0,00
vibrobox	1,14	1,36	1.707,70	0,09	1,00	1,41	2.244,60	0,05
wing	1,49	2,81	440,10	0,14	1,60	3,88	644,80	0,08
wing_nodal	1,05	1,62	447,00	0,04	1,03	1,56	433,90	0,02
avg.	**1,204**	**3,298**	**787,50**	**0,080**	**1,186**	**3,507**	**810,76**	**0,046**

Different aggregation techniques in SAG

graph	HEM (80 runs) RRTS (*100n* iter. - different.) min	avg	std.dev	time	HEM* (80 runs) RRTS (*100n* iter. - different.) min	avg	std.dev	time	HEAV (80 runs) RRTS (*100n* iter. - different.) min	avg	std.dev	time
airfoil1	1,00	1,01	1,2	1,8	1,00	1,02	1,3	1,9	1,00	1,01	0,80	1,64
bcsstk13	1,00	1,00	0,0	11,3	1,00	1,00	0,0	12,3	1,00	1,00	0,00	13,63
big	1,01	1,02	1,3	8,1	1,00	1,02	2,3	7,9	1,01	1,02	2,90	8,23
DEBR12	1,00	1,00	1,7	2,1	1,00	1,00	1,7	2,7	1,00	1,00	2,40	2,10
nasa4704	1,00	1,00	6,2	8,9	1,00	1,00	6,2	9,3	1,00	1,00	5,50	11,24
4elt	1,01	1,02	1,3	7,9	1,00	1,02	2,3	9,3	1,01	1,02	2,90	9,02
add32	1,00	1,19	2,7	3,4	1,00	1,09	3,0	3,5	1,00	1,14	3,20	2,46
bcsstk32	1,31	1,38	489,1	236,7	1,31	1,38	489,1	239,3	1,32	1,41	425,50	521,45
brack2	1,01	1,02	3,6	155,5	1,01	1,02	3,6	166,6	1,01	1,01	3,90	190,17
crack	1,00	1,00	0,7	5,4	1,00	1,00	0,8	5,7	1,00	1,00	0,50	5,41
cs4	1,11	1,22	27,5	18,5	1,11	1,22	27,5	18,7	1,11	1,21	29,10	17,71
cti	1,00	1,05	14,4	12,2	1,00	1,05	14,4	12,4	1,00	1,04	13,10	17,64
t60k	1,28	3,53	90,3	43,4	2,54	3,81	68,3	50,6	2,82	3,92	63,40	48,47
uk	1,30	1,51	2,6	1,4	1,40	1,59	2,2	1,4	1,15	1,50	3,80	1,24
vibrobox	1,00	1,02	262,7	81,1	1,00	1,02	262,7	85,1	1,00	1,02	267,50	113,71
wing	1,15	1,20	27,0	83,2	1,15	1,19	25,3	87,5	1,13	1,18	29,60	92,32
wing_nodal	1,00	1,01	9,4	19,3	1,00	1,01	9,4	19,5	1,00	1,01	8,90	26,58
avg.	**1,069**	**1,246**	**55,4**	**41,2**	**1,148**	**1,261**	**54,1**	**43,1**	**1,150**	**1,265**	**50,8**	**63,7**

HEM technique is faster and obtains better results

RRTS

RRTS: Min-Max Greedy vs Diff-Greedy

graph	RRTS (*100n* iterations) MIN MAX GREEDY (100 runs) min	avg	std.dev	time	RRTS (*100n* iterations) DIFFERENTIAL GREEDY (100 runs) min	avg	std.dev	time
airfoil1	1,00	1,01	1,00	0,94	1,00	1,02	1,00	1,02
bcsstk13	1,00	1,00	0,00	1,87	1,00	1,00	0,00	1,81
big	1,00	1,02	2,40	4,84	1,01	1,04	6,40	5,34
DEBR12	1,00	1,00	1,70	1,48	1,00	1,01	1,30	1,42
nasa4704	1,00	1,00	7,50	2,86	1,00	1,00	7,60	2,79
4elt	1,00	1,02	2,40	5,09	1,01	1,04	6,40	4,89
add32	1,00	1,27	2,60	1,32	1,00	1,10	2,70	1,09
bcsstk32	1,35	1,44	525,70	74,82	1,33	1,41	270,30	62,36
brack2	1,00	1,02	8,60	84,73	1,01	1,02	5,80	75,38
crack	1,00	1,01	0,50	3,64	1,00	1,00	0,50	3,22
cs4	1,09	1,16	20,40	15,03	1,09	1,22	20,20	13,44
cti	1,00	1,01	9,50	7,27	1,00	1,03	13,70	6,10
t60k	2,41	3,23	48,60	53,09	2,15	3,75	85,50	46,60
uk	1,25	1,54	3,50	0,93	1,30	1,53	3,30	0,90
vibrobox	1,00	1,01	182,90	18,05	1,00	1,02	251,80	15,67
wing	1,15	1,20	26,40	82,08	1,16	1,24	55,80	80,22
wing_nodal	1,00	1,01	9,20	9,31	1,01	1,01	10,30	9,22
avg.	**1,132**	**1,233**	**50,17**	**21,609**	**1,122**	**1,261**	**43,68**	**19,498**

Diff-Greedy is faster and obtains better results (best minimum)

Conclusions (preliminary)

- Simple (repeated) greedy schemes are surprisingly effective
- Work to be done to explain why they are so successful
- When coupled with Reactive Search Optimization and prohibitions, results comparable with the state of the art, even without multi-level
- Is multilevel effective only when simple LS techniques are used?
- WAG is slower because of multi-level graph construction
- Work is continuing