An Efficient Two-Level Partitioning Algorithm for VLSI Circuits*

Jong-Sheng Cherng, Sao-Jie Chen
Dept. of Electrical Eng.
National Taiwan Univ.
Taipei, Taiwan, R.O.C.

Chia-Chun Tsai
Dept. of Electronic Eng.
National Taipei Univ. of Tech.
Taipei, Taiwan, R.O.C.

Jan-Ming Ho
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.

Abstract

In this paper, a new two-level bipartitioning algorithm TLP, combining a hybrid clustering technique with an iterative improvement based partitioning process, is proposed. The hybrid clustering algorithm consisting of a local bottom-up clustering technique to merge modules and a global top-down ratio-cut technique for decomposition can be used to reduce the partitioning complexity and improve the performance. To generate a high-quality partitioning solution, a module migration based partitioning algorithm MMP is also proposed as the base partitioner for the TLP algorithm. The MMP algorithm implicitly promotes the move of clusters during the module migration processes by paying more attention to the neighbors of moved modules, relaxing the size constraints temporarily during the migration process, and controlling the module migration direction.

Experimental results obtained show that the TLP algorithm generates stable and high-quality partitioning results. The TLP algorithm improves the unstable property of module migration based algorithms such as FM [6] and STABLE [3] in terms of the average net cut value. On the other hand, TLP outperforms MELO [2], GFM_t [11], and $CDIP_{LA3}$ [5] by 23%, 7%, and 10%, respectively and is competitive with hMetis [8], ML_C [1], and LSR/MFFS [4] which have generated better results than many recent state-of-the-art partitioning algorithms.

1 Introduction

In VLSI circuit design, circuits with millions of transistors are now common, conventional logic-level and physical-level design tools cannot deal with the increasing complexity of VLSI circuit design without the participation of partitioning. Currently, in submicron designs, interconnection delays tend to dominate gate delays; therefore, decomposing a circuit into subsets to minimize the number of interconnections between subsets is thus of great importance. For that purpose, a *size-constrained min-cut bipartitioning* problem is addressed in this paper as follows. Given a hypergraph $H(V, E)$ with n vertices, a vertex weighting function s: $V \rightarrow R^+$, and a lower and an upper bounds on the partition size S_m and S_M, the bipartitioning problem is to divide V into two disjoint nonempty subsets L and R, where $L \cup R = V$, with the objective of minimizing *cut*(L, R) (which is the number of hyperedges connecting modules in L and R) and subject to the size constraints $S_m \le S(L), S(R) \le S_M$ (where $S(L)$ denotes the size of subset L).

Since the size-constrained min-cut bipartitioning problem is *NP-complete* [7], an optimal solution is hard to obtain when the problem size is large and various heuristics have been developed. Kernighan and Lin [9] proposed a well-known iterative improvement based module interchange algorithm for graph partitioning, which was later improved by Fiduccia-Mattheyses (FM) algorithm [6] by employing an efficient bucket list data structure to reduce time complexity to be in linear proportion to the pin number. Since the iterative improvement based FM algorithm is very efficient, much work has sought to improve upon the FM algorithm [10] [11] [5].

On the other hand, module clustering has been shown to reduce the complexity of the partitioning problem and enhance the performance of an iterative improvement algorithm [3] [1] [8] [4]. The goal of clustering algorithms is to identify the natural *clusters* in a circuit, where a cluster is a group of highly connected modules in a circuit. One of the most important classes of clustering based partitioners is the *two-level* partitioners [3] [4]. At the first level of these partitioners, clustering techniques are first applied on the original circuit H to derive a contracted circuit H_C and then partitioning techniques are applied on H_C to derive a partitioning solution P_C. At the second level, P_C is used to obtain a new partitioning solution P_I of H and then a final partitioning solution P is thus obtained by applying partitioning techniques on H using P_I as its initial solution.

In this paper, we propose a novel clustering based partitioner *Two_Level_Partitioning* (TLP) based on the two-level methodology. TLP integrates an efficient hybird clustering technique into a module migration based partitioning algorithm MMP. The proposed hybird clustering technique tries to combine the merits of the bottom-up and top-down clustering techniques by using a local bottom-up clustering technique to merge modules and a global top-down ratio-cut technique for decomposition. The proposed iterative improvement based MMP partitioning heuristic implicitly promotes the move of an entire cluster into one of the two subsets by paying more attention to the neighbors of moved modules, relaxing the size constraints temporarily during the partitioning process, and controlling the module migration direction. An overview of the TLP partitioner is shown in Section 2.

2 Two-Level Partitioner

The following shows the proposed partitioning heuristic TLP, which consists of two levels: (1) hybrid clustering and contracted hypergraph partitioning, and (2) unclustering and original hypergraph partitioning.

Algorithm 1 *Two_Level_Partitioner*
begin
call *Hybrid_Clustering*(H, *cluster_size*) to obtain a contracted hypergraph $H_C(V_C, E_C)$;
/* apply MMP on H_C using P_{random} as its initial solution */
call *MMP*(H_C, P_{random}) to obtain a solution $P_C(L, R)$;
call *Unclustering*(P_C, H) to obtain a solution $P_I(L, R)$;
/* apply MMP on H using P_I as its initial solution */
call *MMP*(H, P_I) to obtain a solution $P(L, R)$;
return the bipartition solution $P(L, R)$;
end

At the first level of Algorithm 1, the *Hybrid_Clustering* procedure accepts the original hypergraph H as input along with a parameter *cluster_size*, which sets a threshold to bound the size of each cluster formed in H. Any time as long as the size of a cluster in H is larger than *cluster_size*, the hybrid clustering technique is continued until every cluster has its size threshold satisfied. The contracted hypergraph H_C obtained is relatively smaller than the original hypergraph H. Empirically, the ratio of the number of vertices in the original hypergraph to the number of vertices in its corresponding contracted hypergraph is about 100. Therefore, H_C is relatively denser than H. The *MMP* partitioning procedure is subsequently applied on H_C using a random initial partitioning solution P_{random} to obtain a good

*This work was supported by the National Science Council, R.O.C., under Grants NSC87-2218-E002-033 and NSC87-2218-E027-005

partitioning solution $P_C(L, R)$ spending only short computational time. In fact, the first level tries to search a good initial partitioning solution for the second level as efficiently as possible. Using the iterative improvement partitioning techniques, although a partitioner can be devised to be less dependent on the initial partitioning solution, the initial partitioning solution is indeed crucial for the generation of high-quality solutions. Therefore, the obtained partitioning solution after the first level plays a very important role in promoting the performance of the whole two-level partitioner.

At the second level of Algorithm 1, the *Unclustering* procedure is used to uncluster each cluster formed in the *Hybrid_Clustering* procedure. This is done by simply decomposing each vertex in L (or R) of P_C into original modules and putting these modules into L (or R) of P_I, where P_I is used as the initial partitioning solution used in the subsequent *MMP* partitioning process. We describe the procedures *Hybrid_Clustering* and *MMP* in detail in the following sections

3 Hybrid Clustering Algorithm

In this section, we introduce the hybrid clustering technique used in TLP. The following is the *Hybrid_Clustering* procedure, where a bottom-up merging process and a top-down ratio-cut decomposition process are alternatively applied until a desired clustering is reached

Algorithm 2 *Hybrid_Clustering*($H(V, E)$, *cluster_size*)
begin
  /* the following *Top_Down_Clustering* procedure contains a bottom-up clustering process described in Algorithm 3 */
  call *Top_Down_Clustering*(H, *cluster_size*) to obtain a clustering Γ;
  call *Grouping*(H, Γ) to obtain a contracted hypergraph H_C;
  return the contracted hypergraph H_C;
end

Algorithm 3 *Top_Down_Clustering*($H(V, E)$, *cluster_size*)
begin
  if (($S(V) \leq$ *cluster_size*) or ($|V| = 1$))
  then return a cluster containing all modules in V;
  else begin
    call *Bottom_Up_Clustering*(H, *cluster_size*) to obtain a set of clusters $C = \{C_1, C_2, ..., C_j\}$;
    call *Grouping*(H, C) to obtain a contracted hypergraph $H_t(V_t, E_t)$;
    call *Ratio_Cut*(H_t) to generate two sub-hypergraphs H_1 and H_2;
    call *Top_Down_Clustering*(H_1, *cluster_size*) to obtain a set of clusters $CL = \{C_1, C_2, ..., C_i\}$;
    call *Top_Down_Clustering*(H_2, *cluster_size*) to obtain a set of clusters $CR = \{C_{i+1}, C_{i+2}, ..., C_k\}$;
    concatenate CL and CR to form $\{C_1, C_2, ..., C_k\}$;
    return the set of clusters $\{C_1, C_2, ..., C_k\}$;
  end;
end

The *Bottom_Up_Clustering* procedure begins with a random permutation of the module set and then visits each in turn. To solve the clustering problem efficiently, only modules contained by the same net are considered to merge. Therefore, for a given *unmerged* module v (i.e., a module that has not yet been assigned to a cluster), the procedure first identifies all neighbors of v and then finds an unmerged module u from them that maximizes *connectivity*(v, u) (defined below). If such a u exists and the merge of v and u does not violate the cluster size constraint, form a new cluster by merging v and u; otherwise, v is assigned to its own cluster. The connectivity between v and u can be evaluated by the following equation:

$$connectivity(v, u) = connection\_strength(v, u) * size(v, u) \qquad (1)$$

where $connection\_strength(v, u) = \frac{A}{B}$ and $size(v, u) = \frac{1}{s(v)*s(u)}$. *connection_strength*($v$, u) is a measure of the connection strength between v and u. Term A represents the total contributed weight of the nets connecting v and u, and term B the minimum of the total contributed weight of the nets connecting v and the total contributed weight of the nets connecting u. *size*(v, u) is devised to encourage merging modules with smaller sizes to form clusters of uniformly sizes.

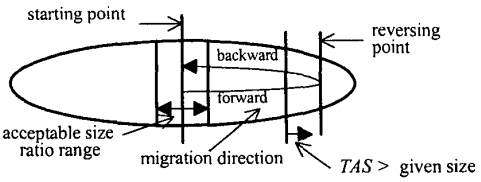

Fig. 1: *Pass* operation of the MMP algorithm.

The *Ratio_Cut* procedure, which is used to perform the top-down decomposition process by using the ratio-cut [12] as its objective, consists of three phases: *initialization phase*, *neighbor migration phase*, and *local neighbor migration phase*. In the initialization phase, a module v is randomly selected as a seed. After the seed v has been determined, v forms the subset R and the other modules form the subset L. Afterward, the procedure travels the circuit starting from v and each time moves one best candidate into R until $|L| = 1$. The decomposition giving the minimum ratio forms initial L and R for the hypergraph concerned.

Once an initial decomposition is derived, an iterative improvement technique is utilized to achieve improvement in the subsequent neighbor migration phase. In this phase, module migration is performed first from R to L and then from L to R. Considering module migration direction to be from R to L, any time the procedure selects a *neighbor* of L to be moved into L. We call a module in a certain subset X a neighbor of a subset Y if it is adjacent to at least one of the modules in Y. The best neighbor is one that can generate the best (i.e., smallest) ratio value after its movement. Each time the procedure moves one best neighbor into the destination subset L until $|R| = 1$. The new best decomposition obtained is thus used as the initial decomposition for the other module migration process from L to R.

To achieve further improvement, the result obtained from the second phase is used as a starting point for the local neighbor migration phase. The basic idea adopted in the third phase is the same as the second phase, that is, both phases repeatedly move one best neighbor into the destination subset. The difference is that in the second phase, neighbors of the destination subset are considered to be moved into the destination subset; in the third phase, however, only neighbors of the set of previously moved modules are considered for migration. In Algorithms 2 and 3, the *Grouping* procedure is simply used to construct a contracted hypergraph from an obtained clustering.

4 Module Migration Partitioning Algorithm

After clusters are formed, a partitioning algorithm must be applied to rearrange the clusters into two subsets with prespecified sizes. In this section, we develop a module migration partitioning algorithm MMP. In a circuit, some clusters exist such that the connections among modules in a cluster are denser than others, and dividing these clusters into different subsets will increase the number of nets cut. Therefore, if we can move some modules to make these straddled clusters not to be cut, the net cut number will be fewer. This concept is used as the main guideline in the MMP partitioning heuristic. An overview of the MMP algorithm is shown as follows:

Algorithm 4 *Module Migration Partitioning*($H(V, E)$, $P_{initial}(L, R)$)
begin
  for *counter* = 1 **to** *number_of_iterations* **do begin**
    randomly select a seed v from L;
    while (L is not empty) **do begin**
      if (the total accumulated size *TAS* of moved modules exceeds β* $S(L)$ and the reversing point is found) **then** break;
      move the selected module v from L to R;
      update associated information;

```
    select from L a next module v, having the strongest connection to
    previously moved modules and a maximal FM gain, to move;
  end;
  randomly select a seed v from R;
  while ((S(L)/S(R)) ≤ (S_M/S_m)) do begin
    if (current partitioning result satisfies the size constraints and
    generates a minimal net cut)
    then record this partitioning result;
    move the selected module v from R to L;
    update associated information;
    select from R a next module v, having the strongest connection to
    previously moved modules and a maximal FM gain, to move;
  end;
  scale the value of β;
  assign the current best result as a new initial partitioning P_initial;
 end; /* end of for */
 report the best partitioning result;
end
```

From the initial partitioning solution obtained in advance, MMP attempts to move the modules in a cluster from one subset to the other to improve the existing solution. This is achieved by fixing the migration direction and choosing modules to be moved with great care. This "forward" migration process is continued until the assumed cluster is entirely moved into the other side. Next, the migration direction is reversed and the modules are moved to keep the size ratio of two subsets fall into an acceptable range. We call a *pass* is done when this "backward" migration process is completed. The best feasible solution will be used to form a new starting point for the next *pass*. The above *pass* operation will be applied iteratively to improve the result. Note that we have relaxed the size constraints temporarily during a *pass* process. The basic operation of a *pass* (i.e., two while loops in Algorithm 4) is shown in Fig. 1.

We hope only modules in one cluster are moved during forward or backward migration. Thus it is undesirable to choose a module which has no connection to the previously moved modules as the next target to move. Therefore, a module v chosen for migration must have the strongest connection to previously moved modules and have a maximal FM gain (here gain means the decrement of the net cut number if v is moved into the opposite subset). An important parameter β ($0 < \beta < 1$), which controls size relaxation, is devised to determine the size of module set to be moved in the forward migration process. Empirically, to obtain good solutions, the value of β is set to be high at the beginning to allow larger clusters to be pulled first, and is gradually decreased in the following *passes* to keep smaller clusters moving.

Backward migration is essential to search acceptable size ratio results and to find better solution for each *pass*. Naturally, the change of migration direction occurs when modules in a cluster have just been moved into one side. By direct observation, when such a cluster is moved into the opposite side, the net cut number will fall into a minimal point. If we still continue the moving operation unaware, the net cut number will certainly increase. Under this consideration, when the accumulated size of moved modules is greater than the given value, we begin to keep record of the decreasing net cut number and it is time to reverse the migration direction when this value begins to increase.

In TLP, we apply the MMP algorithm five times to obtain a good result on the contracted hypergraph and once to the original hypergraph to fine tune the partitioning result. In next section, experiments on benchmarks are performed to verify the superiority of our two-level partitioning algorithm TLP.

5 Experimental Results

Our two-level algorithm TLP was coded in C language and implemented on a Sun SPARC 10 workstation. We evaluate the TLP algorithm with other algorithms by finding solutions with different *deviations*, where deviation from the exactly balanced bipartition is defined as $\delta = (S_M - S_m) / (S_M + S_m)$.

5.1 Partitioning Circuits with Actual Module Size

In this experiment, each module was given the actual area size. In Table 1, we compare the results of the TLP algorithm to those of the FM and STABLE algorithms with the size of each subset being allowed to have 1% deviation. We implemented the FM algorithm [6] and the STABLE algorithm [3]. The runs of STABLE and TLP are set to be 20. The g value of STABLE is set to be 50 based on [3]. We set the run of FM to be 500 for meaningful comparison.

In Table 1, we demonstrate that TLP performs much better than FM and STABLE in terms of both the minimal net cut number and the average net cut value when a strictly balanced bipartition is required. On average, TLP shows a 19% and 26% improvements over the best results and a 50% and 45% improvements over the average results achieved by FM and STABLE, respectively, for all the 18 test circuits. In Table 1, we also find that TLP and STABLE have the same time magnitude, and the average time used in FM is much less than others. Although FM works very fast for each run, more than 500 runs are required to derive the same solution quality as TLP. Therefore, TLP is superior to FM with the whole performance consideration.

Additionally, the stability of FM and STABLE is much worse than TLP when comparing the difference between the average net cut value and the minimal net cut value of each circuit. Therefore, TLP provides a more predictable behavior in net cut number than FM and STABLE when a strictly balanced bipartition is considered.

5.2 Partitioning Circuits with Unit Module Size

In this experiment, each module was given a unit area size. In Table 2, we compare the minimal net cut results of the TLP algorithm to those of some state-of-the-art partitioning algorithms with the size of each subset being allowed to have a up to 10% deviation. The partitioning results of MELO, GFM_t, $CDIP_{LA3}$, hMetis, ML_C, and LSR/MFFS were from [2], [11], [5], [8], [1], and [4], respectively.

In Table 2, on average, TLP outperforms MELO, GFM_t, and $CDIP_{LA3}$ by 23%, 7%, and 10%, respectively and is competitive with hMetis, ML_C, and LSR/MFFS which have generated better results than many recent state-of-the-art partitioning algorithms. Since a large tolerance (10% deviation) on each subset size has been allowed, all of the partitioning algorithms have more chances to search for better partitioning solutions. Consequently, different algorithms generate similar solutions for some circuits for this case with loosely balanced subset sizes.

6 Conclusion

A new two-level partitioner TLP has been developed. The hybrid clustering strategy adopted in TLP, which is the combination of a local bottom-up clustering approach and a global top-down recursive clustering approach, was shown very efficient. Also, to generate a stable and high-quality partitioning solution, a module migration based partitioner was proposed as the base partitioner for TLP. Experimental results obtained indicate that the TLP partitioner generates promising results in either the minimal net cut or the average net cut. On the other hand, the TLP partitioner also improves the unstable property of module migration based partitioners such as FM and STABLE in terms of the average net cut value.

References

[1] C. J. Alpert, J. H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 530-533.

[2] C. J. Alpert and S. Z. Yao, "Spectral partitioning: the more eigenvectors, the better," in *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 195-200.

[3] C. K. Cheng and Y. C. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 12, pp. 1502-1511, Dec. 1991.

[4] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu, "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, pp. 441-446.

[5] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 194-200.

[6] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partition," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175-181.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.

[8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in VLSI domain," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526-529.

[9] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291-307, Feb. 1970.

[10] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Computers*, vol. C-33, no. 5, pp. 438-446, May 1984.

[11] L. T. Liu, M.T. Kuo, S. C. Huang, and C. K. Cheng, "A gradient method on the initial partition of Fiduccia-Mattheyses algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 229-234.

[12] Y. C. Wei and C. K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer Aided Design*, 1989, pp. 298-301.

Table 1. Partitioning results allowing 1% deviation.

Example				FM 500 runs			STABLE 20 runs			TLP 20 runs			Improvement over (%) FM		STABLE	
	# nets	# modules	# pins	min	avg	sec/run	min	avg	sec/run	min	avg	sec/run	min	avg	min	avg
19ks	3,282	2,844	10,547	126	183.59	8.40	154	184.45	34.24	117	126.76	39.45	7	31	24	31
19kstw	3,658	3,079	11,248	97	165.19	11.71	105	151.60	41.43	89	97.80	45.38	8	41	15	36
primGA1	902	833	2,908	47	77.40	1.06	51	70.90	6.46	47	48.55	8.26	0	37	8	32
primGA2	3,029	3,014	11,219	161	286.89	6.80	169	266.85	32.95	134	140.60	28.35	17	51	21	47
primSC1	829	752	2,705	47	79.07	1.04	50	71.75	7.17	47	49.27	9.85	0	38	6	31
primSC2	2,961	2,907	10,965	172	286.56	6.88	172	249.50	32.14	133	135.80	34.70	23	53	23	46
test02	1,720	1,663	6,134	121	233.71	1.03	108	163.25	14.64	90	97.30	43.46	26	58	17	40
test03	1,618	1,607	5,807	65	129.13	2.02	71	112.20	13.71	56	58.70	20.35	14	55	21	48
test04	1,658	1,515	5,975	69	91.36	0.59	45	73.15	13.16	44	44.00	28.66	36	52	2	40
test05	2,750	2,595	10,076	62	115.34	1.05	64	92.25	24.13	51	53.06	49.00	18	54	20	42
test06	1,541	1,752	6,638	68	87.87	4.97	70	85.20	21.41	60	68.51	19.72	8	22	14	20
8,870	307	286	1,137	16	40.77	0.01	13	34.95	2.32	14	14.65	4.33	13	64	22	58
5,655	689	801	2,756	54	82.46	0.81	58	76.65	6.41	49	53.29	8.81	9	35	16	30
industry1	2,186	2,271	7,731	27	72.35	3.53	34	77.95	30.09	24	30.64	27.70	11	58	29	61
industry2	13,419	12,637	48,404	1,065	2023.47	6.08	938	1139.90	189.82	202	254.30	363.39	81	87	78	78
industry3	21,923	15,406	65,791	355	684.03	23.00	409	932.05	370.76	273	303.20	283.55	23	56	33	67
avg.small	22,124	21,918	76,231	257	648.32	173.63	501	615.65	1,744.04	214	315.87	5077.43	17	51	57	49
avg.large	25,384	25,178	82,751	320	826.31	234.66	536	711.05	1868.18	206	337.30	5617.85	36	59	62	53
Average													19	50	26	45

Table 2. Partitioning results allowing up to 10% deviation.

Example				MELO	GFM_t	$CDIP_{LA3}$	hMetis	ML_C	LSR/MFFS	TLP	Improvement over (%)					
						20 runs	20 runs	100 runs	20 runs	20 runs	MELO	GFM_t	$CDIP_{LA}$	hMetis	ML_C	LSR/MFFS
	# nets	# modules	# pins	min	min	min	min	min	min	min	min	min	min	min	min	min
19ks	3,282	2,844	10,547	119	-	105	106	106	-	110	8	-	- 5	- 4	- 4	-
primGA1	902	833	2,908	64	51	52	50	47	43	45	30	12	13	10	4	- 5
primGA2	3,029	3,014	11,219	169	139	152	145	139	119	119	30	14	22	18	14	0
test02	1,720	1,663	6,134	106	-	90	88	89	-	94	11	-	- 4	- 7	- 6	-
test03	1,618	1,607	5,807	60	-	57	58	56	-	53	12	-	7	9	5	-
test04	1,658	1,515	5,975	61	-	52	51	48	-	50	18	-	4	2	- 4	-
test05	2,750	2,595	10,076	102	-	74	71	71	-	77	25	-	- 4	- 8	- 8	-
test06	1,541	1,752	6,638	90	-	60	60	60	-	60	33	-	0	0	0	-
industry2	13,419	12,637	48,404	319	175	190	167	164	-	187	41	- 7	2	- 12	- 14	-
industry3	21,923	15,406	65,791	-	244	243	254	243	-	243	-	0	0	4	0	-
avg.small	22,124	21,918	76,231	-	-	148	130	128	127	130	-	-	12	0	- 2	- 2
avg.large	25,384	25,178	82,751	-	-	145	127	128	127	128	-	-	12	- 1	0	- 1
balu	735	801	2,697	28	28	27	27	27	27	27	4	4	0	0	0	0
bm1	903	882	2,910	48	-	52	51	47	-	47	2	-	10	8	0	-
struct	1,920	1,952	5,471	38	36	36	33	33	33	33	13	8	8	0	0	0
biomed	5,742	6,514	21,040	115	92	83	83	83	84	83	28	10	0	0	0	1
s9234	5,844	5,866	14,065	79	44	44	40	40	40	40	49	9	9	0	0	0
s13207	8,651	8,772	20,606	104	61	70	55	55	61	55	47	10	21	0	0	10
s15850	10,383	10,470	24,712	52	46	67	42	44	43	44	15	4	34	- 5	0	- 2
s35932	17,828	18,148	48,145	-	44	73	42	41	44	43	-	2	41	- 2	- 5	2
s38584	20,717	20,995	55,203	-	54	47	47	47	47	47	-	13	0	0	0	0
s38417	23,843	23,849	57,613	-	62	79	51	49	50	51	-	18	35	0	- 4	- 2
Average											23	7	10	1	- 1	0