Optim Lett (2018) 12:1959–1969
https://doi.org/10.1007/s11590-018-1273-7

AUTHOR CORRECTION

Correction to: Finding a maximum *k*-club using the *k*-clique formulation and canonical hypercube cuts

Yajun Lu[1] · **Esmaeel Moradi**[2] · **Balabhaskar Balasundaram**[1]

Published online: 7 June 2018

Correction to: Optim Lett https://doi.org/10.1007/s11590-015-0971-7

This article provides a corrigendum to "Finding a maximum *k*-club using the *k*-clique formulation and canonical hypercube cuts," published online in Optim Lett, 2015. Due to programming errors in our C++ implementations, the computational results reported in the article are incorrect. In some pathological instances, a significantly larger number of *k*-cliques that are not *k*-clubs can be detected, which can adversely affect the performance of the algorithms proposed. This corrigendum presents completely revised computational results, discussion, and conclusions that are meant to *replace* Sections 3 and 4 in the original article.

Computational results

In this section, we study the computational performance of the ITDBC and the DBC algorithms in Section 2 of the original article in solving the MkCP. The direct solution

The original article can be found online at https://doi.org/10.1007/s11590-015-0971-7.

✉ Balabhaskar Balasundaram
baski@okstate.edu

Yajun Lu
yajun.lu@okstate.edu

Esmaeel Moradi
moradie@schneider.com

1 School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK 74078, USA

2 Engineering and Research Team, Schneider National, Green Bay, WI 54311, USA

of the integer programming (IP) formulations, referred to as F1 [4] and F2 [5], serve as our comparison. All four approaches (DBC, ITDBC, F1, and F2) are implemented in C++, and Gurobi$^{\text{TM}}$ Optimizer 7.0.2 [2] is used to solve the IP formulations. All numerical experiments are conducted on a 64-bit Linux$^{®}$ compute node with dual Intel$^{®}$ Xeon$^{®}$ E5-2620 hex-core 2.0 GHz processors and 32 GB RAM. For one instance with more than 10^4 vertices, a large memory compute node with 256GB RAM is used.

We solve the MkCP for $k \in \{2, 3, \dots, 7\}$ on the test-bed from [4] using the four approaches we consider. We solve the MkCP on each connected component using the F1, F2, and DBC approaches in the decreasing order of number of vertices in the component. If the largest k-club known prior to solving the MkCP on a component is larger than the number vertices in that component, we terminate. A vertex of maximum degree in the graph and its neighbors form a feasible k-club for all $k \geq 2$, and it is used to initialize F1, F2, and DBC approaches. By contrast, ITDBC approach is initialized using the largest distance-$\lfloor \frac{k}{2} \rfloor$ neighborhood of a vertex and, in each iteration, solves the MkCP on the subgraph induced by the largest distance-k neighborhood (only if it is larger than the largest k-club known). ITDBC also uses the trim procedure, a scale reduction technique, as described in Algorithm 1 in the original article. Both DBC and ITDBC approaches implement the CHCs through the lazy cut callback procedure in Gurobi [2]. In the callback, if the k-clique encountered is also a k-club, variables are fixed to zero if the distance-k neighborhood of the corresponding vertices is smaller than the k-club detected.

F1 and F2 use Gurobi's barrier implementation to solve the root node relaxation due to potential degeneracies, while DBC and ITDBC use the Gurobi's default setting that automatically selects the solver. All approaches use the dual simplex solver at all other nodes of the branch-and-bound tree. In all approaches, we set the Gurobi objective cutoff parameter to the size of the largest k-club known prior to calling the Gurobi optimization procedure. This informs the solver that we are only interested in solutions with better objective values [2].

We impose a 1-h limit using the Gurobi parameter for limiting the solve time. We do the same with the total elapsed running time after every component is solved in F1, F2, and DBC, and after every iteration of ITDBC. However, on some instances Gurobi did not strictly enforce the time limit parameter resulting in significantly longer running times; these must be interpreted as a failure to solve to optimality within the 1-h limit. All other Gurobi parameters including numerical tolerances, branching strategies, heuristics, cuts, and the use of multiple threads were left at their default values.

We report average running times over 10 samples for each order (n) and edge density (ρ) we consider in Table 1, along with the number of instances that were not solved to optimality within 1 h. The fastest average running time for every pair of n and ρ is highlighted in bold font. On this test-bed, F1 and F2 are comparable for smaller k. For the larger values of k we often find F2 is significantly faster than F1, with some exceptions. F2 uses a slightly different variable definition compared to F1 and avoids using "big-M" parameters in the constraints, which facilitates better performance with a general-purpose solver.

By contrast, DBC and ITDBC are close in performance on most instances in this test-bed for all k values we consider. However, there are some notable exceptions where their performances are significantly different, as we identify next using (n, ρ, k) triples:

Table 1 A comparison of running times (s) averaged over 10 samples on the test-bed from [4]; fastest (on average) running times are highlighted in bold font

n	ρ	DBC	ITDBC	F1	F2	DBC	ITDBC	F1	F2
		$k = 2$				$k = 3$			
100	2%	0.06	**0.00**[a]	0.39	0.53	0.10	**0.05**	0.85	0.97
	3%	0.10	**0.00**[a]	0.69	1.04	0.18	**0.09**	2.22	5.69
	4%	**0.10**	0.02	0.91	1.19	723.85\2[b]	724.74\2	**8.55**	8.83
200	1%	0.27	**0.00**[a]	1.69	2.34	0.38	**0.04**	4.25	5.92
	1.5%	0.50	**0.00**[a]	2.96	3.72	1.16	**0.16**	12.18	24.91
	2%	0.68	**0.02**	4.19	6.01	2.78	**2.03**	68.19	67.72
300	0.5%	0.25	**0.00**[a]	1.32	1.89	0.30	**0.02**	2.93	3.49
	1%	1.20	**0.00**[a]	6.81	7.65	2.50	**0.17**	27.90	67.36
	1.5%	1.78	**0.06**	22.23	29.48	7.17	**4.15**	250.62	201.62
		$k = 4$				$k = 5$			
100	2%	0.09	**0.03**	1.40	1.34	361.32\2	45.39	**1.51**	1.55
	3%	1106.13\2	870.27\2	7.19	**7.07**	1517.91\3	1321.71\3	3.86	**3.44**
	4%	3254.11\9	3250.30\9	15.92	**8.12**	0.52	**0.31**	4.63	3.71
200	1%	0.32	**0.02**	5.82	6.72	28.02	143.14	**12.93**	20.60
	1.5%	47.53	**3.41**	30.46	43.88	3606.05\10	3605.50\10	1375.23\1	**154.78**
	2%	3610.08\10	3614.21\10	1953.97\4	**475.53**	3611.29\10	3605.07\10	1161.29\2	**120.58**
300	0.5%	0.27	**0.00**[a]	4.60	4.70	0.30	**0.07**	6.08	6.69
	1%	2.27	**0.85**	57.79	120.63	3297.15\9	3647.48\10	1434.56	**745.08**
	1.5%	3268.51\9	3279.81\9	2750.86\4	**2212.39\3**	3608.83\10	3602.99\10	3448.09\9	**1579.23\2**
		$k = 6$				$k = 7$			
100	2%	0.27	**0.19**	2.57	2.41	**0.04**	0.10	3.01	2.28
	3%	**0.06**	0.08	4.97	5.01	**0.01**	0.04	7.83	7.49
	4%	**0.01**	**0.01**	9.16	6.27	**0.01**	**0.01**	15.82	10.16
200	1%	1821.82\4	1435.23\3	42.01	**28.25**	1814.11\5	1807.72\5	98.19	**26.15**
	1.5%	3252.11\9	3261.72\9	384.22	**74.32**	1080.89\3	724.28\2	122.45	**53.57**
	2%	**0.03**	0.08	169.69	91.59	**0.03**	0.07	335.94	195.21
300	0.5%	0.24	**0.01**	7.93	8.51	0.30	**0.11**	10.54	10.84
	1%	3615.10\10	3612.63\10	3606.41\10	**1821.17\2**	2900.06\8	2913.79\8	1900.13\3	**344.29**
	1.5%	361.18\1	**66.82**	1822.41	1029.38	**0.03**	0.07	3648.80\8	2568.55

[a] An entry of 0.00 means the run took less than 9e−3 s

[b] Whenever instances are not solved to optimality under the 1-h time limit, we report average running time\#suboptimal terminations

ITDBC is, on average, significantly better than DBC on (200, 1.5%, 4), (100, 2%, 5), and (300, 1.5%, 6); DBC is better than ITDBC on (200, 1%, 5). The former are cases where the preprocessing techniques in ITDBC were very effective, while in the latter case they were not. In particular, if the number of vertices at distance k or less from the fixed vertex is very large, it would be more effective to solve that instance directly using DBC. When preprocessing is not effective, ITDBC might solve a large instance in several iterations using DBC, degrading the performance of ITDBC.

Comparing DBC/ITDBC against F1/F2 on this test-bed, we frequently find that when one or more of the samples were not solved to optimality by DBC and/or ITDBC, the average running time is worse in comparison to direct solution of the formulations. However, when the decomposition approaches are faster, they are generally much faster than directly solving the formulations. Superior performance of the DBC algorithm depends on how often the detected k-clique is also a k-club. Whenever this is not the case, we add a CHC to eliminate a k-clique that is not a k-club. The drawback of the CHC is that it eliminates only that k-clique based on which the cut is constructed. Let $\widetilde{\omega}_k(G)$ and $\bar{\omega}_k(G)$ denote the size of a maximum k-clique (the k-clique number) and k-club (the k-club number) in G, respectively. The number of k-cliques that are not k-clubs that need to be eliminated in the DBC approach is exponential in the gap $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$. This is because every subset of a k-clique is also a k-clique. These observations imply that a very large number of CHCs have to be generated on

Table 2 Average\maximum gap $\widetilde{\omega}_k - \bar{\omega}_k$ over 10 samples on the test-bed from [4]

n	ρ (%)	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
100	2	0	0	0.2\1	0.9\4	0.7\2	0.3\1
	3	0	0.3\1	2.1\5	2.6\7	0.5\2	0
	4	0	3.2\16	5.5\14	0.4\1	0	0
200	1	0	0	0	1.2\2	2.5\10	20.1\44
	1.5	0	0.3\1	0.5\2	9.5\14	4.9\7	7.1\58
	2	0	0.6\2	16.5\31	9.2\20	0.1\1	0
300	0.5	0	0	0	0	0.1\1	0.2\1
	1	0	0.3\1	0.1\1	8.1\15	17.5\24	76.1\130
	1.5	0	0.6\2	14.7\37	19.1\31	0.3\1	0

instances that have a large gap $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$. These are pathological for the DBC approach and on such instances the performance of DBC significantly degrades. This can in turn degrade the performance of ITDBC if the preprocessing techniques are not effective on such pathological instances. To highlight this issue, in Table 2 we report the average and maximum values of the gap $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$ over 10 samples for each setting of (n, ρ, k).

Table 3 lists the DIMACS clustering benchmark instances [1], many of which model real data that are also used in our study.[1] Tables 4, 5, 6, 7, 8, and 9 tabulate the results of our experiments in solving MkCP for $k \in \{2, 3 \dots, 7\}$ on this test-bed. The fastest running time is highlighted in bold font. For the purposes of our discussion, we divide these instances into two groups: group-1 includes instances with fewer than 1600 vertices, and group-2 includes the remaining instances. The two groups are separated by a line in Tables 3, 4, 5, 6, 7, 8, and 9.

On group-1, with three exceptions, we find that the decomposition approaches outperform directly solving F1/F2; consistent with our discussions on the other test-bed, these exceptions ($k = 3$ on `football` and $k = 3, 4$ on `email`) also correspond to large gaps for $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$. Interestingly, we are able to directly solve the MkCP for $k = 3$ on `football` using F1 and F2 in under 2 min, while all four approaches fail to solve the MkCP to optimality for $k = 3, 4$ on `email`.

The DBC and ITDBC approaches are comparable in terms of overall running time for most instances in group-1 for all k values we consider; we review the exceptions to this observation next. One of the exceptional instances, `email` for $k = 2$, solved much faster with ITDBC due to effective preprocessing. In fact, detailed runtime statistics reveal that the initial heuristic solution is optimal and preprocessing reduces the instance to fewer than 300 vertices from the original 1133 vertices in the three ITDBC calls in which DBC is invoked. However, the same `email` graph for $k = 5$ required 21 iterations of ITDBC, each with 1000+ vertices as preprocessing was not as effective even after the optimal solution was detected in the first iteration. Another

[1] The "challenging edge density" instances are not included in the revised computational study; instead we solve MkCP for more values of k on the DIMACS test-bed for more effective benchmarking.

Table 3 Number of vertices, edges, and edge density for DIMACS clustering benchmarks used in this study

Graph	$\|V\|$	$\|E\|$	ρ (%)
karate	34	78	13.90
lesmis	77	254	8.68
polbooks	105	441	8.08
adjnoun	112	425	6.84
football	115	613	9.35
celegans-metabolic	453	2025	1.98
email	1133	5451	0.85
polblogs	1490	16,715	1.51
netscience	1589	2742	0.22
power	4941	6594	0.05
hep-th	8361	15,751	0.05
PGPgiantcompo	10,680	24,316	4.3e−2

Table 4 The 2-clique number, the 2-club number, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_2(G)$	$\bar{\omega}_2(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	18	18	0	**0.01**	0	0.03	0.69	0.96
lesmis	37	37	0	**0.20**	0	0.03	0.41	0.46
polbooks	28	28	0	0.09	0	**0.02**	1.50	1.55
adjnoun	50	50	0	0.08	0	**0.02**	1.21	1.49
football	17	16	45	1.72	37	**0.44**	25.25	19.25
celegans-metabolic	238	238	0	1.65	0	**0.06**	30.39	40.47
email	72	72	0	15.25	0	**1.72**	268.28	318.15
polblogs	352	352	0	**16.57**	0	41.12	808.93	859.60
netscience	35	35	0	1.11	0	**0.08**	13.03	19.72
power	20	20	0	3646.81†	0	**0.24**	3652.85	4477.35†
hep-th	51	51	0	3664.07†	0	**0.85**	4544.03†	4776.69†
PGPgiantcompo	206	206	0	3815.48†	0	**0.95**	5588.86†	–

–The approach did not terminate gracefully, typically due to a memory-related crash
†The approach did not find an optimal 2-club under the 1-h time limit

exceptional instance, `polblogs`, is also an example where the ITDBC is noticeably slower than the DBC for $k = 2, 3, 4$. Detailed iteration statistics of ITDBC for this instance show that, for the aforementioned k values, multiple iterations were needed and the graph was not sufficiently reduced as a result of preprocessing. In such cases, it is more effective to solve the entire instance directly using DBC. As an illustration,

Table 5 The 3-clique number, the 3-club number or the size of the largest 3-club found, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_3(G)$	$\bar{\omega}_3(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	25	25	0	**0.03**	0	**0.03**	1.03	1.07
lesmis	58	58	0	**0.19**	0	0.20	1.19	0.97
polbooks	54	53	49	**0.97**	88,094	3639.70†	3.01	2.15
adjnoun	83	82	21	0.95	11	**0.27**	4.87	5.25
football	69	58	60,324	3603.54†	60,157	3601.70†	78.87	**65.02**
celegans-metabolic	371	371	0	0.36	0	**0.33**	323.59	163.93
email	233	$\geq$72	9404	3609.16*	13,284	3602.87*	3762.29*	4028.20*
polblogs	776	776	0	**4.74**	33	84.56	4285.86†	3755.21†
netscience	54	54	0	1.19	0	**0.16**	38.91	39.38
power	30	30	0	3647.35†	0	**0.46**	4707.34†	4747.48†
hep-th	125	$\geq$82	0	3664.84*	36,862	3645.32†	4834.78†	5359.16†
PGPgiantcompo	423	422	0	3817.36†	154	2726.54	–	–

*The approach found the 3-club reported

†The approach did not find the 3-club reported under the 1-h time limit

–The approach did not terminate gracefully, typically due to a memory-related crash

we report iteration statistics for `polblogs` in Table 10 for the case $k = 3$ that took the most number of iterations among all k values we consider for this instance.

Note that even if $\widetilde{\omega}_k(G) - \bar{\omega}_k(G) = 0$ (or nearly zero), if there are a significantly large number of optimal k-cliques that are not k-clubs, we may still have to eliminate these before finding a maximum k-club. Pertinently, even if the gap is zero and DBC does not generate any CHC, ITDBC could still require CHCs. The subgraph induced by the distance-k neighborhood of the fixed vertex, potentially a very different graph than the original graph, can result in a very different runtime behavior of the ITDBC compared to the DBC. This is reflected in the number of CHC added by the two decomposition approaches on the instances: `polbooks` when $k = 3, 4$, `polblogs` when $k = 3$, and `netscience` for $k = 5$. We also find such performance variations between different computing platforms, which is a phenomenon germane to IP solvers [3].

We close our discussion of group-1 DIMACS instances with a note on what we observed with `polbooks` when $k = 3$. This instance is not solved to optimality within the 1-h time limit by ITDBC, although it is solved to optimality by DBC, F1, and F2 in fewer than 4 s (fastest being DBC in less than 1 s). The very first iteration of the ITDBC approach that solves the problem on $N^3_G[v]$ for vertex $v = 5$ (a vertex with the largest distance-3 neighborhood fixed to be included in the solution) stymies the DBC approach in a manner the entire instance does not. Ironically, no maximum 3-club in `polbooks` contains vertex 5, as a valid upper-bound smaller than the 3-club number of `polbooks` is returned at the end of the first iteration. Furthermore, this one iteration terminates suboptimally reaching the 1-h running time limit even

Table 6 The 4-clique number, the 4-club number or the size of the largest 4-club found, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_4(G)$	$\bar{\omega}_4(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	33	33	0	0.01	0	**0.00**^{o}	1.26	1.65
lesmis	75	75	0	0.20	0	**0.00**^{o}	2.44	1.44
polbooks	68	68	0	**0.03**	1	0.06	5.11	4.05
adjnoun	107	107	0	**0.08**	0	0.23	15.71	8.43
football	115	115	0	0.06	0	**0.03**	10.87	7.82
celegans-metabolic	432	432	0	**0.19**	0	0.28	3828.89†	3084.79
email	654	$\geq$646	13,820	3630.93*	12,448	3632.23†	14,059.30†	3718.86†
polblogs	1127	1127	0	**2.86**	0	19.70	4244.67†	4165.94†
netscience	85	85	0	1.18	0	**0.03**	69.66	66.15
power	61	61	0	3646.65†	0	**0.17**	4787.24†	5091.23†
hep-th	347	$\geq$ 344	0	3660.25†	2584	3604.70*	5480.97†	5974.04†
PGPgiantcompo	1161	1161	0	3802.68†	0	**3.35**	–	–

^{o}An entry of 0.00 means the run took less than 9e−3 s
*The approach found the 4-club reported
†The approach did not find the 4-club reported under the 1-h time limit
–The approach did not terminate gracefully, typically due to a memory-related crash

Table 7 The 5-clique number, the 5-club number or the size of the largest 5-club found, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_5(G)$	$\bar{\omega}_5(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	34	34	0	**0.01**	0	**0.01**	1.86	2.15
lesmis	77	77	0	0.19	0	**0.18**	4.03	2.13
polbooks	95	95	0	**0.01**	0	0.02	9.64	5.98
adjnoun	112	112	0	**0.05**	0	0.12	54.84	30.69
football	115	115	0	**0.02**	0	**0.02**	47.92	22.50
celegans-metabolic	445	445	0	**0.15**	0	0.26	4214.41†	3617.64†
email	1003	1003	0	**1.11**	0	12.58	3859.29†	3868.84†
polblogs	1211	1211	0	**2.83**	0	4.24	4459.98†	4327.86†
netscience	117	116	417	**9.63**	1956	13.17	131.06	183.34
power	94	94	0	3646.84†	0	**0.95**	5194.36†	5484.90†
hep-th	797	$\geq$344	0	3650.96†	13	3620.74*	6114.28†	6583.29†
PGPgiantcompo	1989	$\geq$1161	0	3775.23†	13	3635.39*	–	–

*The approach found the 5-club reported
†The approach did not find the 5-club reported under the 1-h time limit
–The approach did not terminate gracefully, typically due to a memory-related crash

Table 8 The 6-clique number, the 6-club number or the size of the largest 6-club found, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_6(G)$	$\bar{\omega}_6(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	34	34	0	0.01	0	**0.00**[o]	2.56	3.10
lesmis	77	77	0	0.01	0	**0.00**[o]	6.17	2.65
polbooks	104	104	0	0.01	0	**0.00**[o]	17.52	9.24
adjnoun	112	112	0	0.19	0	**0.00**[o]	135.71	67.43
football	115	115	0	0.02	0	**0.00**[o]	116.40	56.53
celegans-metabolic	451	451	0	0.15	0	**0.05**	4036.50[†]	4312.06[†]
email	1112	1112	0	**1.03**	0	2.35	3985.88[†]	3992.93[†]
polblogs	1219	1219	0	**2.84**	0	3.99	4640.88[†]	3895.91[†]
netscience	172	172	0	0.84	0	**0.16**	814.01	99.48
power	142	142	0	3647.61[†]	0	**0.18**	5599.29[†]	5812.91[†]
hep-th	1513	$\geq$ 1268	13	3639.60[†]	948	3621.12*	6763.60[†]	7155.22[†]
PGPgiantcompo	3424	$\geq$3418	0	3744.91[†]	3447	3623.30*	–	–

[o] An entry of 0.00 means the run took less than 9e−3 s
*The approach found the 6-club reported
[†] The approach did not find the 6-club reported under the 1-h time limit
–The approach did not terminate gracefully, typically due to a memory-related crash

Table 9 The 7-clique number, the 7-club number or the size of the largest 7-club found, the number of CHCs added, and the running time in seconds for DBC, ITDBC, F1, and F2 on the DIMACS test-bed

Graph	$\widetilde{\omega}_7(G)$	$\bar{\omega}_7(G)$	DBC		ITDBC		F1	F2
			#CHC	Time	#CHC	Time	Time	Time
karate	34	34	0	0.01	0	**0.00**[o]	3.60	3.42
lesmis	77	77	0	0.22	0	**0.00**[o]	9.81	3.59
polbooks	105	105	0	**0.01**	0	0.02	32.57	13.60
adjnoun	112	112	0	0.09	0	**0.00**[o]	329.52	115.78
football	115	115	0	0.02	0	**0.00**[o]	222.67	109.68
celegans-metabolic	453	453	0	**0.15**	0	0.45	8351.50[†]	4599.47[†]
email	1132	1132	0	**1.05**	0	1.44	4129.08[†]	4143.11[†]
polblogs	1221	1221	0	**2.86**	0	3.93	4007.61[†]	3848.14[†]
netscience	230	230	0	0.46	0	**0.38**	350.39	159.98
power	186	186	0	3644.39[†]	0	**2.49**	5911.00[†]	6195.56[†]
hep-th	2403	$\geq$1268	2	3625.58[†]	131	3625.05*	7449.08[†]	7808.35[†]
PGPgiantcompo	4870	4870	0	3700.86[†]	0	**3673.14**	–	–

[o] An entry of 0.00 means the run took less than 9e−3 s
*The approach found the 7-club reported
[†] The approach did not find the 7-club reported under the 1-h time limit
–The approach did not terminate gracefully, typically due to a memory-related crash

Table 10 Progress of the ITDBC algorithm when solving `polblogs` ($|V| = 1490$, $|E| = 16{,}715$) for $k = 3$

Iter. #	BestObj	$\|V\|$	$\|E\|$	Fixed vertex	DBC time (s)
1	352	1201	16,692	818	20.23
2	776	1092	16,439	14	0.77
3	776	1089	16,366	16	0.79
4	776	1088	16,319	22	0.74
5	776	1087	16,295	23	0.74
6	776	1085	16,209	31	0.78
7	776	1081	16,140	40	0.71
8	776	1078	16,042	55	0.75
9	776	1076	15,771	68	0.72
10	776	1075	15,749	72	0.70
11	776	1074	15,636	75	0.71
12	776	1072	15,539	89	0.74
13	776	1070	15,482	99	0.75
14	776	1068	15,354	101	0.75
15	776	1065	15,289	115	0.70
16	776	1063	15,229	117	0.70
17	776	1060	15,204	119	0.68
18	776	1057	15,108	150	0.67
19	776	1056	14,998	154	0.68
20	776	1055	14,925	155	0.72
21	776	1026	14,525	170	0.61
22	776	1017	14,410	180	0.55
23	776	1016	14,275	187	0.59
24	776	1014	14,228	189	0.58
25	776	1013	14,116	191	0.57
26	776	1011	14,042	218	0.59
27	776	1008	13,972	223	0.59
28	776	1007	13,939	238	0.60
29	776	1004	13,893	13	0.54
30	776	1003	13,845	241	0.55
31	776	1002	13,801	249	0.54
32	776	1000	13,768	276	0.58
33	776	999	13,733	278	0.54
34	776	997	13,675	204	0.55
35	776	994	13,637	113	0.54
36	776	991	13,620	132	0.56
37	776	989	13,599	144	0.53
38	776	950	13,413	24	0.33
39	776	944	13,360	18	0.32

Table 10 continued

Iter. #	BestObj	$\lvert V\rvert$	$\lvert E\rvert$	Fixed vertex	DBC time (s)
40	776	934	13,302	46	0.29
41	776	921	13,196	56	0.26
42	776	919	13,122	130	0.26

though the reduced instance only contains 104 vertices. This observation is a reminder that even smaller instances of this NP-hard problem can be extremely challenging to solve using this approach. Although the original instance is easy for DBC to solve, when we consider the subgraph induced by $N_G^3[5]$, the difficulty level is dramatically different. We verified this observation independently with the DBC code by adding the constraint $x_5 = 1$ to the original instance that solved in under 1 s, which also resulted in suboptimal termination after an hour.

We finally turn our attention to the so-called group-2 instances in the DIMACS test-bed with several thousands of vertices. An immediate observation is that the largest of these instances, `PGPgiantcompo`, crashes due to memory issues with F2 for all values of k we consider, and with F1 for nearly all values of k we consider. In general, the less memory-intensive decomposition approaches fair better. ITDBC, due to its preprocessing techniques, is very effective on all three instances when $k = 2$ and in solving `power` for all k values we consider. The next largest instance in this group, `hep-th`, with 8000+ vertices is not solved to optimality within the time limit by any of the approaches for $k = 3, \ldots, 7$. However, one of the decomposition approaches, often ITDBC, finds the best known solution for this instance. The largest instance, `PGPgiantcompo` with 10,000+ vertices, is solved to optimality for $k = 2, 3, 4, 7$ by ITDBC, and its best known solution is found by ITDBC when $k = 5, 6$.

Conclusion

Optimization Letters DOI 10.1007/s11590-015-0971-7 introduces a decomposition approach to solve the maximum k-club problem that employs the maximum k-clique problem, which is a graph-theoretic relaxation of k-club. The approach then uses a naive cutting plane, one that eliminates precisely one binary vector at a time, whenever a k-clique that is not a k-club is detected. The article also introduces an iterative version that can incorporate more effective preprocessing techniques due to the fact that in each iteration, some vertex is fixed to be included in the k-club.

In this corrigendum, a revised computational study of the proposed approaches in comparison to direct solution of integer programming formulations for the problem is presented; the corrigendum was necessitated by a coding error that was detected after the article appeared online in *Optimization Letters*. We find that the performance of the decomposition algorithms introduced in Section 2 of *Optimization Letters DOI 10.1007/s11590-015-0971-7* is heavily dependent on the gap between the size of a maximum k-clique and k-club of the graph given by $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$. When this gap is small, the decomposition algorithms generally outperform directly solving the formulations; in this case, the performance gains are very significant for larger graphs and values of k. If this gap is 10 or more, performance of the decomposition algorithms deteriorates significantly, unless preprocessing techniques drastically reduce the instance. This behavior is a direct consequence of the nature of the cutting planes added—specifically, the elimination of one k-clique at a time.

It is encouraging to note that for most of the larger instances with 1000+ vertices from the DIMACS test-bed based on real data, across all values of k we consider, at least one of the decomposition approaches is significantly faster than directly solving the formulations. Barring some exceptions noted in our computational study, the effectiveness of the decomposition approaches when the gap $\widetilde{\omega}_k(G) - \bar{\omega}_k(G)$ is small, especially on large-scale real-life instances warrants further investigation into the addition of stronger cutting planes.

Acknowledgements The authors would like to thank Austin Buchanan and Hosseinali Salemi from the School of Industrial Engineering and Management at Oklahoma State University for pointing out the error in our original computational results and for several subsequent discussions that were extremely helpful to the authors. The authors would also like to thank the journal co-editors for allowing us the opportunity to rectify our errors. The computational experiments reported in this article were conducted at the Oklahoma State University High Performance Computing Center.

References

1. DIMACS: Graph Partitioning and Graph Clustering: Tenth DIMACS Implementation Challenge. http://www.cc.gatech.edu/dimacs10/index.shtml (2012). Accessed Feb 2015
2. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual (2017). http://www.gurobi.com
3. Lodi, A., Tramontani, A.: Performance Variability in Mixed-Integer Programming, chap. 1, pp. 1–12. INFORMS Tutorials in Operations Research. INFORMS, Maryland (2013). https://doi.org/10.1287/educ.2013.0112
4. Veremyev, A., Boginski, V.: Identifying large robust network clusters via new compact formulations of maximum k-club problems. Eur. J. Oper. Res. **218**(2), 316–326 (2012)
5. Veremyev, A., Prokopyev, O.A., Pasiliao, E.L.: Critical nodes for distance-based connectivity and related problems in graphs. Networks **66**(3), 170–195 (2015)