A Constraint Programming Approach to the Cluster Editing Problem

Amel Mhamdi[1] **and Wady Naanaa**[2]

[1]Department of Informatics
University of Sfax
Tunisia
amel.mhamdi2@gmail.com

[2]Department of Informatics
University of Monastir
Tunisia
wady.naanaa@fsm.rnu.tn

ABSTRACT

Given a simple graph $G = (V, E)$, the problem dealt with in this paper consists in partitioning V into a disjoint union of cliques by adding or removing a minimum number of edges. The problem, which is refered to by Cluster Editing*, is NP-hard. This observation has motivated our investigation of improving the chance for finding a disjoint union of cliques in a limited amount of search. To this end, we formulate the problem in terms of a Weighted Constraint Satisfaction Problem (WCSP), a widely used framework for solving hard combinatorial problems. As search strategy, we apply a Limited Discrepancy Search coupled with a branch-and-bound algorithm, a combination which has proved to be very advantageous. We theoretically and exprimentally show that the proposed approach outperforms the most used algorithm in solving cluster editing, namely, fixed parameter tractable algorithms. An experimental clearly show that our approach is very competitive, especially on large cluster editing instances.*

Keywords: Cluster editing, Fixed parameter tractability, Cliques partition, Constraint programming, Limited discrepancy search, Algorithm engineering, Computer experiments.

2000 Mathematics subject classification: 37A20

1 Introduction

The *cluster editing* problem instance is given by a simple graph G and asks for the nearest *cluster graph* to graph G. In (Krivanek and Moravek, 1986), it has been proven that Hierarchical-Tree Clustering, of which cluster editing is a special case, is NP-hard. The problem is, referred to as *Transitive Graph Projection* or by *Zahn's problem* in (Jun, 1964), where it was first solved for specially structured graphs by C. T. Zahn. Although defined since 1964, the problem is still of great interest because it arises in the analysis of gene expression data (Hansen and

Jaumard, 2004). The input of gene expression analysis is a similarity graph, where vertices represent genes and weighted edges reflect the extent of similarity between genes. Similarity measures of gene expression profiles are useful in detecting biological relationships between species. Hence, similar expression profiles of two genes can be explained by a similar function of the corresponding proteins. Such a similarity function is used in bioinformatic to assign biological or biochemical roles to proteins, which are classified in protein families. This kinship can be used to find related family members which are related by intermediate sequences. This concept is called *transitivity* and refers to the well-known mathematical property. Similar expression profiles of organisms can be explained by a similar metabolism state which may be caused by similar environmental reasons or diseases. This explains why clustering of similar genes is vital for modern biological data analysis (Pipenbacher, Schliep, Schneckener, Schnhuth, Schomburg and Schrader, 1964).
This work is motivated by the one presented in (Gramm, Guo, Huffner and Niedermeier, 2006), which pointed out that the problem of cluster editing belongs to the class of *Fixed Parameter Tractable* (FPT) problems. Algorithms dedicated to problems in the FPT class use the desired cost as a parameter which controles the problem complexity. Indeed, the controle parameter is closely related to the time complexity of the FPT algorithms since this complexity is, in general, of the form $f(k)n^{O(1)}$, where k is a problem-specific parameter, n is the input size and $f(k)$ is a function in k.
The fixed parameter tractable (FPT) algorithm presented in (Gramm et al., 2006) was dedicated to cluster editing. The time complexity of this algorithm, is $O(2.27^k + |V|^3)$. The key idea of the latter algorithm is two easy to implement bourder tree search and an efficient polynomial-time reduction to a kernel graph of size $O(|V|^3)$.

In this paper, we explore another track in order to take advantage of the membership of cluster editing to the FPT class. We precisely resort to constraint programming (CP), a framework that has been widely employed in solving combinatorial optimization problems. The constraint programming paradigm is one of the most powerfull artificial intelligence tool which can be used in solving many real world problems (David, Precup, Petriu, Rdac and Preitl, 2013) (Zavoianu, Bramerdorfer and Amrhein, 2013) (El-Hefnawy, 2014). The cluster editing is encoded as a weighted constraint satisfaction problem (WCSP) in an easy and declarative way. The problem is, thereafter, solved by means of a branch-and-bound algorithm plugged in a limited discrepancy search. The resulting algorithm is implemented using one of the more efficient constraint solvers, namely, the TOULBAR2 solver (Sanchez, Bouveret, De Givry, Heras, Jegou, Larrosa, Rollon, Terrioux, Verfaillie and Zytnicki, n.d.).

The remainder of the paper is organised as follow. Section 2 presents some definitions and notations and gives the necessary constraint programming background. In Section 3, fixed parameter tractable (FTP) algorithm for cluster editing is presented. In Section 4, we detail the proposed constraint-based encoding of the cluster editing problem. Section 5 described the solution algorithm used to solve our problem. Section 6 evaluates the performance of our approachs by comparing our results to the results obtained by FPT algorithms. Finally, Section

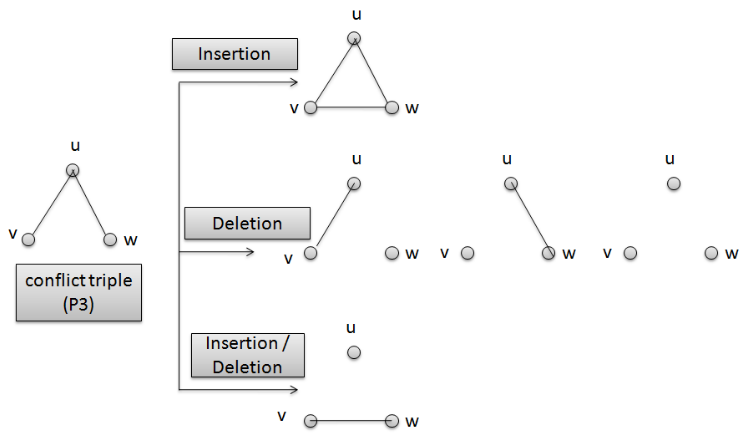

Figure 1: P_3 Graph and insertion/deletion possibilities.

7 gives a brief conclusion and directions for future works.

2 Definitions and notations

In what follows $\mathcal{P}_2(V)$ will designate the set of all pairs of finite set V.

2.1 Cluster editing (CE)

A simple graph is given by an ordered pair $G = (V, E)$, where V desingnates a finite set which contains the vertices of the graph and the elements of $E \subseteq \mathcal{P}_2(V)$ are the edges of G. In the particular case where $E = \mathcal{P}_2(V)$, the graph G is said to be *complete*. We will also use the notations $V(G)$ and $E(G)$ to respectively designate the vertices and edges of G. The *open neighborhood* $N(v)$ of vertex v is the set of vertices adjacent to v, that is, $N(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *neighborhood* of v is the set $N[v] = N(v) \cup \{v\}$. For any two graphs G and H such that $V(G) = V(H)$, we denote by $\delta(G, H)$ the number of edge modifications needed to obtain H from G, i.e.,

$$\delta(G, H) = |E(G) \setminus E(H)| + |E(H) \setminus E(G)| \tag{2.1}$$

A *cluster editing* problem instance is given by a simple graph G and asks for a *cluster graph*, i.e. an union of cliques, H such that $\delta(G, H)$ is minimum. Recall that a *clique* of a graph G is a complete subgraph of G.
Cluster graphs are equivalently defined as simple graphs that do not contain any P_3 as induced subgraph. Recall that P_3 is the path on three vertices (see Figure 2). More consicely, we can say that a cluster graph is a P_3-free graph.

2.2 Constraint programming background

In the valued CSP framework (VCSP), a valuation or cost is associated to every value combination (Schiex, Fargier and Verfaillie, 1995). The set of all possible valuations is assumed to be a totally ordered set with a minimum elements $(\bot)$ and a maximum element $(\top)$. These valuations are combined with a single monotonic binary operation $\oplus$ known as aggregation. These components can be gathered into a *valuation structure* that can be specified as follows:

Definition 2.1. A valuation structure is a tuple $S = (E, \oplus, \preceq)$, where E is a set of valuations, $\preceq$ is a total order on E and $\oplus$ is a binary commutative, associative and monotonic operator.

Definition 2.2. A VCSP instance is defined by a tuple (X, D, C, S), where

- X is a finite set of variables.
- D is a set of finite domains, where $D_x \in D$ denotes the domain of $x \in X$.
- C is a set of valued constraints. Each valued constraint c is an order pair (σ, ϕ), where $\sigma \subseteq X$ is the scope of c and ϕ is a function from $\prod_{x\in\sigma} D_x$ to E.
- S is a valuation structure.

A variable $x \in X$ should be assigned a value only from its domain, D_x. If a valued constraint is defined by a function whose codomain is a subset of $\{\bot, \top\}$, then it is said to be a *hard constraint*, otherwise, it is a *soft constraint*. The arity of a valued constraint is the size of its scope. The arity of a problem is the maximum arity over all its constraints.

VCSP has many variations which mainly differ by the involved valuation structure. The choice of the more appropriate variation depends of the features of the problem to be formulated in terms of VCSP. The variation that we have used to formulate CE is Weighted CSP (WCSP). The valuation structure of a WCSP is a parametrized triple

$$S(k) \;=\; ([0..k], \oplus, \preceq) \tag{2.2}$$

where $k > 0$ is an integer, $\oplus$ is defined as $a \oplus b = \min\{k, a + b\}$ and $\leq$ is the standard order among integers (Javier and Thomas, 2004). Observe that with the valuation structure $S(k)$, we have $\bot = 0$ and $\top = k$.

3 Fixed-Parameter algorithms for Cluster Editing

In recent years, a new circle in complexity theory, namely parameterized complexity, has achieved many results in solving NP-hard problems exactly and efficiently. It is based on the observation that for many NP-hard problems, the combinatorial explosion can be confined to a small part of the input, *the parameter*.

Definition 3.1. We say that a problem is fixed parameter tractable (FPT) with respect to parameter k if there exists a solution running in $f(k)n^{O(1)}$ time, where f is a function of k which is independent of n and n denotes the overall input size.

The associated complexity class that contains all parameterized problems that are fixed-parameter tractable is designated by the FPT class. Fixed-parameter algorithms are often recursive tree search algorithms. In addition, problem reduction methods are applied to reduce the initial instance to an instance of smaller size, so that the search space is decreased. If the size of a reduced instance depends only of the parameter, the reduced instance is called problem kernel (Bocker, Briesemeister, Bui and Truss, 2011).

Definition 3.2. A kernelization algorithm for a parameterized problem is polynomial-time many-one reduction mapping an instance (I, k) to $(I^{'}, k^{'})$

1. the reduction from (I, k) to $(I^{'}, k^{'})$ is computable in polynomial time, and
2. $(I^{'}, k^{'})$ is a yes-instance iff (I, k) is a yes-instance, and
3. $|I^{'}|$, $k^{'} \leq f(k)$ for some function f only depending on k.

The function f is called the size of the kernel.

In this section, we describe an FPT algorithm for cluster editing which uses the modification cost as the fixed parameter (k). This algorithm uses the refined branching method together with the re-kernalization technique represented in (Frank, Michael, Xuemei, Sylvain, Peter and Yun, 2006).
Given a CE instance defined by a simple graph $G = (V, E)$ and a maximum modification cost k, the algorithm proposed in (Frank et al., 2006) applies two kernelization rules. The fist kernelization rule resumes simply to removing the connected components that are cliques from the original graph. The second rule applies to the neighborhood of every pair of vertices $u, v \in V$ as follows.

- If u and v have more than k common neighbors, then (u, v) has to belong to the target cluster graph. Thus, if (u, v) is not in E then it is added to E.
- If u and v have more than k non-common neighbors, then (u, v) cannot be an edge of the target cluster graph. Thus, if (u, v) is in E then it is deleted.
- If u and v have both more than k common and more than k non-common neighbors, then the given instance has no size k solution.

FPT algorithms, fixe the number of authorized modifications to a predefined value k. It proceeds by introducing edge addition/suppresion on the input graph while decrementing the parameter k at each change. It stops when the parameter k reaches 0 or when the current graph contains no induced P_3. If the algorithm succeeds to derive a solution by introducing less than k modifications then it stops. Otherwise, the fixed parameter is incremented and a new running of the FPT algorithm is launched. The search tree size explored by the refined branching strategy is $O(2.27^k)$.

In (Frank et al., 2006), authors proved that the best results are obtained by using the refined branching method interleaved with the kernelization technique.

4 A constraint programming approach to CE

In this section, we propose an encoding of the cluster editing problem in terms of weighted constraint satisfaction problem (WCSP). The latter problem is a particular variation of VCSP obtained by using the specific valuation structure given in (2.2). In addition to the valuation structure, we should also identify the set of variables, X, the value domain of every variable, D, and finally the set of weighted constraints, C.
First, let as recall that the goal is to transform a given simple graph into a cluster graph, that is, a set of disjoint union of cliques. The transformation must involve the minimum number of edge change. To this end, the proposed CE encoding associates a boolean variable to every pair of vertices in the input graph. Hard ternary constraints will be used to forbid P_3 subgraphs to appear in the target cluster graph. The objective, which consists in limiting the modifications carried out on the input graph, is encoded using soft unary constraints.

4.1 The variables and their domain

Let $G = (V, E)$ be a simple graph. Then, every pair of vertices $\{u, v\} \subseteq V$ is associated with a boolean variable denoted by $x_{u,v}$. When $x_{u,v}$ is set to one by the WCSP solver, this means that the decision is to keep the edge $\{u, v\}$ or to add it to the target cluster graph. Otherwise, the edge $\{u, v\}$ is either removed or not added. Accordingly, the variable set X will contain $|V|(|V| - 1)/2$ boolean variables.

4.2 The constraints

Two types of constraints are used in our encoding of cluster editing: soft unary constraints and hard ternary constraints.

- The role of the soft unary constraints is to minimize the number of changes introduced on the original graph. A change is an addition of a new edge or the removal of an existing one. Each change in the original graph entails an additive unitary cost. We distinguish two situations that entails costs: The first case occurs when $\{u, v\} \in E$ and the decision is to remove that edge from the original graph by setting the associated boolean variable to 0. The second case is when $\{u, v\} \notin E$ and the decision is to add $\{u, v\}$ to the original graph by setting the associated boolean variable to 1. Observe that there are two other situations which do not entail any cost. The first one is when we decide to keep an existing edge in the graph and the second one occurs when we decide to not add a new edge (see Table 1).

- The role of the ternary hard constraints is to forbid P_3 subgraphs to appear in the target graph. Forbiding P_3 subgraphs can be achieved by enforcing the following rule to apply: If $\{u, v\} \in E$ and $\{v, w\} \in E$ then $\{u, w\} \in E$. The latter rule can be enforced by creating a ternary hard constraint for every three WCSP variables. Every triple of constrained variables must be associated to a triple of vertices of the graph. More precisely, let

	$x_{u,v}$	cost
$\{u,v\} \in E$	0	1
	1	0
$\{u,v\} \notin E$	0	0
	1	1

Table 1: The valuation function of the unary constraints

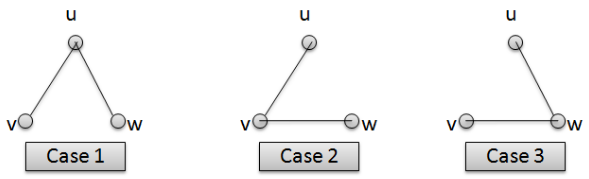

Figure 2: P3.

x, y and z be the boolean variables of the WCSP associated respectively to the pairs $\{u,v\}, \{v,w\}$ and $\{u,w\}$ of V. Then, we must have, in the encoding WCSP, a ternary constraint whose scope is the triple $\{x,y,z\}$ and whose valuation function is depicted in Table 2. As it can be seen in the latter table, there are three assignements that must not be taken simultaneously by x, y and z, these are 011, 101 and 110. For this reason, these assignements are associated with the maximum cost k. Observe that the three forbiden assignements correspond to the P_3 subgraphs that can be formed by vetices u, v and w.

xyz	cost
000	0
001	0
010	0
011	k
100	0
101	k
110	k
111	0

Table 2: The valuation function of the ternary hard constraints

example As an illustrative example, consider the graph depicted in Figure 3, which involves 7 vertices and 10 edges. The corresponding WCSP model for the cluster editing problem has 21 variables, all having boolean domains. The WCSP encoding gave rise to 21 unary constraint, as many soft unary constraints as there are variables, and 35 ternary constraints. The cost of the solution given for cluster editing is 3 since they were 3 edge modifications (add and delete).

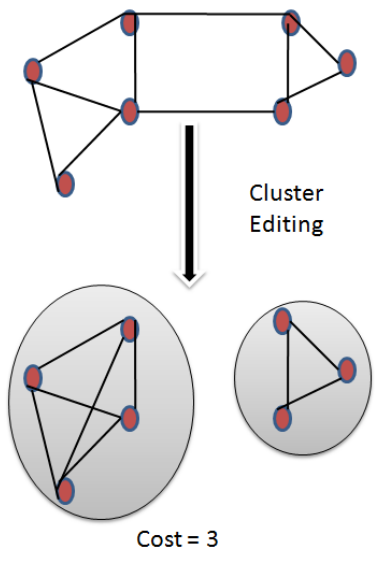

Figure 3: An example of cluster edition instance.

5 The solution algorithm

Weighted constraint satisfaction problems are typically solved by a branch-and-bound algorithm which interleaves branching with pruning and solution cost lower bound computation. The lower bound lb is used, together with an upper bound ub, to prune the search subtrees that do not contain relevent solutions based on the invariant $lb < ub$. In our case, both lower bound computation and pruning are performed by a soft arc-consistency algorithme called Full directional arc consistency (FDAC*) (Givry and Zytnicki, 2005). During search, the upper bound is usually given by the cost of the best solution found so far.
To get a more efficient solution algorithm, we plugged the branch-and-bound algorithm within a Limited Discrepancy Search (LDS) (William and Matthew, 1995). This search strategy is defined with regard to a value ordering heuristic, and the seach algorithm performs a discrepancy when it does not branch through the value indicated by the heuristic. Limited discrepancy search proceeds by allowing the discrepancies in an increasing manner. Thus, the seach path that does not involve any discrepancy, i.e. the one that follows the value ordering heuristic at every branching point, is explored first. Then those paths that involve a single discrepancy, then those that involves two discrepancy, etc...
To sum up, the proposed solution algorithm, which will be referred to by LDS-CE, performs a limited discrepancy search applying the branch-and-bound technique at each node of the search tree. The value ordering heuristic used by LDS-CE favors the value that has the lowerest unary cost. This cost is computed, during search, for every value by FDAC*. Note that LDS-CE performs a discrepancy when it assigns, to the current variable, the value that has the highest unary cost.

The maximal number of discrepancy that could be needed to solve a given CE instance is bounded as follows:

Proposition 5.1. *Given a cluster editing instance with lower bound lb and upper bound ub, the number of discrepancy δ that could be performed by LDS-CE verifies the invariant $lb + \delta < ub$.*

Proof. First, recall that the pair of values that are in the domain of every variable have respectivelly costs zero and one, depending on whether the associated edge is in the input graph or not. This fact is preserved when using FDAC* as filtering algorithm. Indeed, since the ternary costs are 0 or ub, unary costs can only evolve to ub. In such a case, the associated value is removed from the domain of the variable, which is considered as instanciated since its domain is henceforth a singleton.
On the other hand, every discrepancy consists in assigning, to the current variable, the value that has cost one. This supplementary cost is added to the current lower bound, lb, to get the new lower bound. The new bound should not reach the upper bound ub, otherwise every solution that could be derived will be irrelevent. The result follows. □

Usually, when solving a CE instance, the initial lower bound is zero. A good quality initial upper bound could be obtained using the greedy algorithm described in (Gramm et al., 2006). According to Proposition 1, the solution upper bound provides us by a limit of the number of discrepancies needed to get the optimal solution. More is true, the improved invariant $lb + \delta < ub$ could also be used during search to prune the search tree.

6 Experimental Results

In this section, we evaluate the performance of LDS-CE, which is implemented using the toulbar2 solver which is available at http://mulcyber.toulouse.inra.fr/projects/toulbar2/. LSD-CE is compared to the FPT algorithm described in Section 3.
The graphs investigated in our exprement are random graphs that have the distinction of being similar to real graphs. We used the LFR graph generator developed by santo fortunato which is available at https://sites.google.com/site/santofortunato/inthepress2. This generator uses four parameters: the number of vertices n, the average degree $\bar{\delta}$, the maximum degree Δ and the mixing parameter μ. The latter parameter is of great relevance because it controls the fraction of edges that relate dense subgraphs. If $\mu = 0$, all edges are within dense subgraphs that are not related to each other. This results in easy to solve cluster editing instances. If $\mu = 1$ then the graph does not display any structure and the associated instances are rather hard to solve. The resulting cluster editing instances are transformed into WCSP by applying the encoding described in Section 4.
We used the TOULBAR2 WCSP solver which is available at (Sanchez et al., n.d.). Both the filtering algorithm (FDAC*), the branching strategy (LSD) and the heuristic variable selection employed in this work are implemented within TOULBAR2. The evaluation criteria are the number of edge change (deletion or insertion) for the cluster editing problem and the CPU time in seconds. The FPT algorithms, with which we compared our algorithms, is implemented in C++. All programs run under Ubuntu 13.04 on a 2.0 GHz PC having 6 Go of RAM.

The results of the experiment clearly indicated that the WCSP algorithm for cluster editing outperforms the FPT algorithm on both sparse and dense graphs. Table 3 shows the details of these results. Small instances with only 50 vertices are solved within seconds, and medium graphs of size 100 are solved within minutes. We see that the WCSP approach for cluster editing is well-suited for medium-size random instances.

Parameters		FPT				WCSP MODEL			
n	m	Cost	Running time(sec.)	opt %	Unfinished	Cost	Running time(sec.)	opt %	Unfinished
10	[13-45]	6.5	0.015	50	0	6	0.007	100	0
20	[50-151]	44.6	0.25	50	1	40.4	5.852	66.67	0
30	[132-399]	118.3	4.3	40	0	109.7	1501.47	40	0
40	[121-760]	170.71	12.28	20	2	148.57	57.69	20	2
50	[240-1102]	317.4	71.6	20	4	363	109.128	20	1
60	[300-1740]	340	291	40	2	452.33	40.68	40	0
70	[235-1940]	394	205.25	25	1	390.75	226.08	25	1
80	[416-2840]	-	-	20	4	697.75	512.25	20	0
90	[460-2925]	-	-	0	3	836	256.36	40	0
100	[514-4460]	-	-	0	3	410.25	8179.516	25	1

Table 3: Results on random graphs with different numbers of nodes n, and different ranges of edge numbers m. For each n, ten random instances were generated. Costs and running times where averaged over the ten instances

We now compare the performance of FPT and WCSP using random CE instances. With the FPT algorithm, we reduced all instances using the reduction strategy described in Section 3. For each reduced instance, we apply the FPT algorithm within a running time limited to 24 hours. For average running times, we do not taken into account the time spent in solving the unfinished instances. In Table 4 we report running times of the two algorithms. Notice that the values in bold face represents optimal costs. Note that running times for FPT algorithm depend on the maximal value of parameter k which were fixed to $|E|/2$.

Using LDS-CE strategie, we were able to get suboptimal solutions for all but few of the instances. In contrast, the FPT algorithm was not able to find any feasible solution for 22 CE instances within 24 hours. More precisely, the LDS-CE were able to solve all but 5 of the instances. Note also that LDS-CE reached the optimal solution for $38 : 66\%$ of the instances in less than one hour.
For sparse graphs $(n \geq 50)$, and for most cases, the WCSP algorithm gives a better cost that the FPT algorithm (see Table 4) and for some instances $(e.g. \geq 50)$, the FPT algorithm did not finshed after two hours. On the other hand, the WCSP approach give an optimal solution if it finish in the other case the solution is not the optimal, so in every time the WCSP approach give a solution. Also the WCSP algorithm is always faster as it can be seen from Table 4 (lines 4 and 6). Note that the values in bold represents an optimal cost.

Table 5 show the weak and strong points of the proposals approaches. For some instance, the FPT can give the optimal solution faster than the WCSP approach. This result can be explained by the importance of the reduction rules which have a polynomial running time. But this result depends of the graph structure. Also for some large instance the FPT algorithm can not give solution within two hours of execution time. This can be explained by the possibility that the conditions of reduction rules are not verified.

P arameters		Running time(sec.)		Cost	
n	m	FPT	WCSP MODEL	FPT	WCSP MODEL
60	300	-	3135	-	246
60	750	-	5899	-	648
70	1340	-	-	-	-
70	1940	303	32	**465**	**465**
80	416	-	127	-	223
80	2840	1	17.56	**320**	**320**
90	460	-	375	-	363
90	2160	-	141	-	1905
100	1014	-	12.3	-	534
100	4460	-	35.2	-	**490**

Table 4: Results on random graphs with different numbers of nodes n, and different ranges of edge numbers m.

approach	pretreatment	condition proposed	Unfinished
FTP	reduction rules	yes	22
WCSP	filtring algorithm	no	5

Table 5: The strong point of the WCSP and FPT approachs

7 Conclusion

In this paper, we presented a constraint programming approach to *cluster editing problem*. Using The proposed approach combines the Limited Discrepancy Search strategy and branch and bound in order to improve the ability to derive suboptimal solutions with good quality in reasonable running time. We also implemented the algorithm proposed in (Gramm et al., 2006) and compared it with ours. We have shown both theoretically and exprimentally that the proposed algorithm is an effective means for solving cluster editing. The results obtained by our algorithm are clearly competitive especially on large instances. Since it was able to solve unweighted Cluster Editing instances with several hundred edge modifications in a matter of seconds. The experimental results clearly proved the practical usefulness of our approach and constitutes a huge improvement over the work reported in (Gramm et al., 2006), where unweighted instances with 50 edge modifications required several minutes of computation. In addition, Our approach is flexible and extensible in that it can easily encode and handle new constraints in order to cope with other variations of cluster editing. This work can be extended to solve many other variations of cluster edition, especially those involving weighted graphs.

References

Bocker, S., Briesemeister, S., Bui, Q. and Truss, A. 2011. Going weighted: Parameterized algorithms for cluster editing, *Theoretical Computer Science* **410**: 5467–5480.

David, R., Precup, R., Petriu, E., Rdac, M. and Preitl, S. 2013. Gravitational search algorithm-based design of fuzzy control systems with a reduced parametric sensitivity, *Inf. Sci.* **247**: 154–173.

El-Hefnawy, N. 2014. Solving bi-level problems using modified particle swarm optimization algorithm, *International Journal of Artificial Intelligence* **12**: 88–101.

Frank, D., Michael, A. L., Xuemei, L., Sylvain, P., Peter, S. and Yun, Z. 2006. The cluster editing problem: Implementations and experiments, *IWPEC* pp. 13–24.

Givry, S. and Zytnicki, M. 2005. Existential arc consistency: Getting closer to full arc consistency in weighted csps, *In Proc. of the 19 th IJCAI*, pp. 84–89.

Gramm, J., Guo, J., Huffner, F. and Niedermeier, R. 2006. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation., *ciac* pp. 108–119.

Hansen, P. and Jaumard, B. 2004. Cluster analysis and mathematical programming, *Math. Program.* **79**: 191–215.

Javier, L. and Thomas, S. 2004. Solving weighted {CSP} by maintaining arc consistency, *Artificial Intelligence* **159**: 1 – 26.

Jun, C. Z. 1964. Approximating symmetric relations by equivalence relations., *J. Soc. Ind. Appl. Math.* **12**: 840–847.

Krivanek, M. and Moravek, J. 1986. Np-hard problems in hierarchical-tree clustering, *Acta Informatica* **23**: 311–323.

Pipenbacher, P., Schliep, A., Schneckener, S., Schnhuth, A., Schomburg, D. and Schrader, R. 1964. Proclust: improved clustering of protein sequences with an extended graph-based approach., *Bioinformatics* pp. 182–191.

Sanchez, M., Bouveret, S., De Givry, S., Heras, F., Jegou, P., Larrosa, J., a. N. S., Rollon, E., a. S. T., Terrioux, C., Verfaillie, G. and Zytnicki, M. n.d.. Max-csp competition 2008: toulbar2 solver description. in proceedings of the third international csp solver competition, 2008.

Schiex, T., Fargier, H. and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems, *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pp. 631–637.

William, D. and Matthew, L. 1995. Limited discrepancy search, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 607–613.

Zavoianu, L., Bramerdorfer, S. and Amrhein, K. 2013. Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives, *Engineering Applications of Artificial Intelligence* **26**(8): 1781–1794.