Towards an Efficient Handling of the Maximum Triangle Packing Problem

Youcef Abdelsadek [1, 2], Francine Herrmann [2]
[1] Department of Informatics, Systems and Collaboration
Public Research Centre-Gabriel Lippmann
Belvaux, Luxembourg
e-mail: abdelsad@lippmann.lu, otjacque@lippmann.lu

Imed Kacem [2], Benoît Otjacques [1]
[2] Laboratoire de Conception, Optimisation et Modélisation des Systèmes (LCOMS EA 7306)
University of Lorraine - Metz, France
e-mail: francine.herrmann@univ-lorraine.fr, imed.kacem@univ-lorraine.fr

Abstract—**This work addresses the problem of finding the maximum number of unweighted vertex-disjoint triangles in an undirected graph *G*. It is a challenging NP-hard problem in combinatorics and it is well-known to be APX-hard. We propose two heuristics for this problem. The first is based on local substitutions, while the second uses a surrogate relaxation of the related integer linear program which is analog to a specific knapsack problem. A computational comparison of the two heuristics using randomly generated benchmarks has shown that the first heuristic outperforms the second heuristic regarding the obtained packing solutions and the respective computation times.**

Keywords - packing problems; maximum triangle packing; heuristics; computational study.

I. INTRODUCTION

In the unweighted Maximum k-Set Packing (k-MSP) problem, a collection C of exactly k sized sets of a population P is given. The aim in the related optimization problem is to find the maximum number of pairwise disjoint sets. For example, let Π be an instance of a 2-MSP, where $C = \{\{1, 2\}, \{1, 3\}, \{3, 4\}\}$ and $P = \{1, 2, 3, 4\}$. The optimal solution maximizing the number of pairwise disjoint sets for Π is $\{\{1, 2\}, \{3, 4\}\}$. The unweighted k-MSP is a well-known NP-complete problem in the complexity theory. Even when $k = 3$, the problem remains NP-hard and can be transformed to an instance of the 3-dimensional matching problem [1]. However, it is polynomial time solvable for the well-known 2-dimensional matching problem when $k = 2$ [1].

In this paper, we will consider the Maximum Triangle Packing (MTP) problem. It can be formulated as follows. Let G be a graph such that $G = (V, E)$ where V and E respectively denote the vertex set and the edge set for the graph G. For instance Π of the 3-MSP problem, the population P represents the set V while the collection C represents all the triangles (i.e., cliques of size 3) of the graph G. Indeed, a set $S_i \in C$ which have x, y and z as elements, if and only if, (x, y), (x, z) and (y, z) belongs to the set E. The goal in this case is to find the maximum number of vertex-disjoint triangles of the graph G, the so-called MTP problem. Notice that in this work we are not addressing the case of edges-disjoint triangles because we attempt to consider the disjoint 3-Set Cover (3-SC) problem [13]. The MTP problem can be found in numerous real-world problems, including scheduling, biology [2] and extraction of test forms [3].

The remaining paper's structure is as follows. In Section 2, we quote the relevant related work of the MTP problem cases. In Section 3, we give some basics and definitions illustrating them with an example. Section 4 describes the two proposed heuristics. In Section 5, we provide an experimental comparison of the aforementioned heuristics by randomly generated MTP instances and also detail how they are built. Section 6 concludes the paper and discusses further developments of this work.

II. RELATED WORK

The interest of the scientific community regarding the MTP problem and k-MSP problem in general continues to grow almost daily. The MTP problem is NP-hard; Caprara and Rizzi [4] proved that the MTP problem is also APX-hard for graphs with a degree ≥ 4, which means that we cannot approximate the MTP problem within any given approximation ratio (i.e., nonexistence of a polynomial time approximation scheme [1]) unless P = NP. However, it is solvable in polynomial time if the maximum graph degree is at most 3. In [5], it is showed that the MTP problem admits a $(3 - (\sqrt{13} / 2) + \varepsilon)$-approximation algorithm when the maximum degree is at most 4, for any constant $\varepsilon > 0$, whereas the $((76 / 75) - \varepsilon)$-non-approximability is shown in [2]. In [6], the k-MSP problem (resp., MTP problem) admits a $((k / 2) + \varepsilon)$-approximation algorithm (resp., $((3 / 2) + \varepsilon)$-approximation algorithm). Wang, Feng and Chen [9] gave an $O(3.53^{3\kappa})$-time parameterized algorithm for the 3-MSP problem.

For the weighted case, a weight is assigned to each edge of a complete graph of size $t * 3$, the goal is to maximize the sum of edge weight selecting t vertex-disjoint triangles. Hassin and Rubinstein proposed a randomized $((89 / 169) - \varepsilon)$-approximation algorithm [7] for the weighted case. An improvement of their work is given in [8], the approximation algorithm achieves an expected ratio of $0.5257(1 - \varepsilon)$.

With respect to the constraint programming approach, an empirical comparison of three constraint models for the MTP problem is provided [10]. Additionally in [3], a branch and bound for the k-MSP problem is proposed. Recently, Bertsimas, Iancu and Katz [11] developed a pseudo-polynomial time approximation algorithm for the k-MSP problem within an approximation ratio of $k^2 / (2k - 1)$.

Due to the APX-hardness of the MTP problem, work tackling the non-approximability within some approximation ratio can give us a better grasp of its approximability.

III. BASICS AND DEFINITIONS

In this section, we give some definitions which are necessary for the better understanding of the two proposed heuristics. The set of vertices $\{v_1, v_2,\ldots, v_n\} \in V$ has a size n, while the triangles set $\{k_1, k_2,\ldots, k_m\} \in K$ has a size m. The weight of a vertex v_i represents the number of triangles to which v_i belongs. These weights are called here α_i such that $i\in\{1,\ldots, n\}$. Figure 1 gives an example of a graph G and vertex weights. The weight of a triangle $k_j = \{v_i, v_g, v_h\} \in K$, is the sum of its vertices weights v_i, v_g, $v_h \in V$. We refer to the triangles weights as β_j such that $j\in\{1,\ldots, m\}$. We say that two triangles k_j and $k_{j'} \in K$ overlap, if they share a common vertex $k_j \cap k_{j'} = v_i \in V$. In Figure 1, the triangles k_1 and k_2 overlap in v_2. Moreover, the neighborhood degree λ_j of a triangle $k_j \in K$ represents the number of triangles that overlap k_j, called in [10] collision count. For example, the triangle k_1 has as a neighborhood degree value of 2 because it shares a vertex with k_2 and k_3. Table 1 gives the triangle weight and neighborhood degree for the same graph G.

IV. OUR HEURISTICS

Before describing the two proposed heuristics, we point out that each vertex of the input graph must belong to at least one triangle, without loss of generality. Moreover, in this study the outcomes of MTP instances are assessed regarding the number of obtained triangles, which means that the smallest gap between the optimal solution and the returned solution is better. We also pay attention to the computation time that it costs.

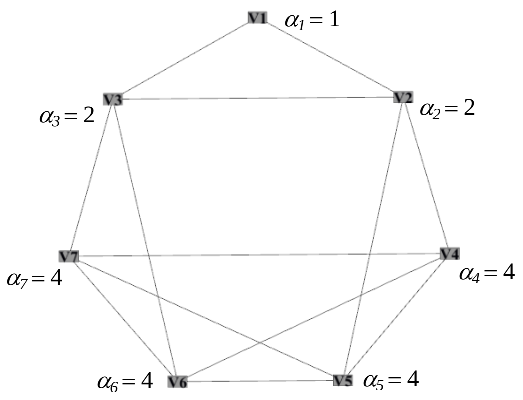

Figure 1. An example of a graph G and the vertex weights

TABLE I. TRIANGLE WEIGHT AND NEIGHBORHOOD DEGREE OF THE GRAPH G

Triangle	The triangles set K						
	$k_1=v_1$ v_2 v_3	$k_2=v_2$ v_4 v_5	$k_3=v_3$ v_6 v_7	$k_4=v_4$ v_5 v_7	$k_5=v_4$ v_5 v_6	$k_6=v_5$ v_6 v_7	$k_7=v_4$ v_6 v_7
β_j	5	10	10	12	12	12	12
λ_j	2	5	5	4	4	4	4

A. *Local subtitution-based heuristic*

In this section, we explain the behavior of the first heuristic as illustrated in Algorithm 1.

The algorithm starts by computing the neighborhood degree λ_j for each triangle. The next step is to find the triangle with the smallest λ_j value that is done with *TriMin* (). The motivation behind this is to select the triangles which outline the graph structure with the aim that the selection in the rest of the graph (i.e., the inner parts of the graph) will not generate couples and singletons. Algorithm 1 avoids selecting a triangle (i.e., *Tmin*) which overlaps with disjoint triangles neglecting a local optimality. For that *MaxDisjoint* () returns *Neigh* the maximum number of disjoint triangles which overlap with a certain triangle k_j. Notice that the number returned by *MaxDisjoint* () is upper bounded by 3, that is because we cannot have 4 disjoint triangles which overlap a triangle. The final step is to remove the selected triangle(s) and its (their) neighborhood triangles with *RemoveNeigh*(). The process is repeated until there remains no triangles in ξ.

Algorithm 1 (Local substitution)
Input: the graph G
Output: a packing of triangles PT

```
a)  $\xi \leftarrow K$, $PT \leftarrow \emptyset$, $Neigh \leftarrow \emptyset$; $Tmin \leftarrow \emptyset$;
b)  While ($\xi \neq \emptyset$)
c)      Calculate the neighborhood degree $\lambda_j$ for each
d)      $j\in\{1,\ldots, m\}$;
e)      $Tmin \leftarrow TriMin(\xi)$;
f)      $Neigh \leftarrow MaxDisjoint(Tmin)$;
g)          if ($Neigh > 1$) $PT \leftarrow PT + Neigh$, $\xi \leftarrow \xi \setminus Neigh$,
h)                           $RemoveNeigh(Neigh)$;
i)                      else $PT \leftarrow PT + Tmin$, $\xi \leftarrow \xi \setminus Tmin$,
j)                           $RemoveNeigh(Tmin)$;
k)    end while;
l)  return $PT$;
m)  end.
```

To better understand the aforementioned heuristic we take the graph in Figure 1 as an example and we briefly outline its steps. First of all, the set of solution PT is initialized to $\emptyset$ and ξ is initialized to K. After that k_1 is assigned to *Tmin*, because it has the smallest λ_j value and $\{k_2, k_3\}$ to *Neigh*, this is due to the fact that k_2 and k_3 are disjoint and overlap k_1. Finally, $\{k_1, k_2, k_3, k_4, k_5, k_6, k_7\}$ are removed from ξ and the algorithm returns $PT = \{k_2, k_3\}$, which is one of the optimal solutions.

B. *The second heuristic*

The MTP problem for a graph $G = (V, E)$ can be stated as an Integer Linear Program (ILP), like in (1) where $B \in \{0, 1\}^{n * m}$ is the vertex-triangle **B**elonging matrix. The columns of the matrix B contains exactly 3 ones, which means that the vertices representing the 3 non-zero values belong to the triangle (column). Let $x_k = 1$ if the kth triangle is selected in the returned solution, 0 otherwise. A surrogate relaxation of (1) is presented in (2).

$$\begin{aligned}&\text{Maximize} && \textstyle\sum_{k=1}^{m} x_k \\ &\text{Subject to:} && \textstyle\sum_{k=1}^{m} B_{ik}\, x_k \leq 1 \qquad \forall\, i = \{1, \dots, n\} \\ & && x_k \in \{0,\ 1\}^m\end{aligned} \tag{1}$$

$$\begin{aligned}&\text{Maximize} && \textstyle\sum_{k=1}^{m} x_k \\ &\text{Subject to:} && \textstyle\sum_{k=1}^{m} \beta_k\, x_k \leq \alpha_0 \\ & && \beta_k\ = \textstyle\sum_{i=1}^{n} B_{ik}\, \alpha_i \qquad \forall\, i = \{1, \dots, m\} \\ & && \alpha_0\ = \textstyle\sum_{i=1}^{n} \alpha_i \\ & && x_k \in \{0,\ 1\}^m\end{aligned} \tag{2}$$

Detailed work on the multiple varieties of relaxations for the k-MSP problem is presented in [12]. In (1) a constraint is devoted to each vertex, whereas in (2) we attribute the vertex weight α_i to each constraint. Hence, a reformulation of the constraint-weighted ILP leads to (2) where the ILP is variable-weighted. The relaxed problem is reduced to the one-dimensional knapsack problem (1-KP) [1] where the items represent the triangles and the knapsack capacity is α_0. In this case, the optimal solution for the 1-KP is to select the items in the increasing weight order, because all the items have 1 as profit value. The solution of the relaxed problem yields an upper-bound on the optimal solution of MTP problem. The following Algorithm 2 allows us to derive a feasible solution based on the relaxed solution.

Before outlining the different steps of Algorithm 2, we define what a packing solution is. A packing solution exists if no triangle of a solution overlaps. Algorithm 2 takes as an input the graph G and the vector of weights α and β. It starts by calling SolveILP (), which solves (2) in a linear time. After that, Algorithm 2 verifies whether the returned solution is a packing solution or not. In the negative case, it chooses the first encountered triangles, which overlaps and selects the one having the smallest triangle weight β_j value, maximizing the $\alpha_0 - \beta_j$ for future iterations, this is done with *Intersect* (). To avoid that the selected triangle in Step e intersects again, we attribute an exclusive penalty-weight to the neighborhood of the selected triangle via *Penalty* (). The penalty is a large integer value which is assigned to a triangle weight β_j without affecting its vertices weights α_i (i.e., not uniformly distributed to its vertices weights). This process is repeated until the returned solution is a packing solution and terminates.

To illustrate Algorithm 2, let us consider the example in Figure 1. Algorithm 2 starts with X = {0, 0, 0, 0, 0, 0, 0} and α_0 = 1 + 2 + 2 + 4 + 4 + 4 + 4 = 21. Step c returns X = {1, 1, 0, 0, 0, 0, 0}. Step d checks if the returned solution is a packing solution. A negative response is returned because k_1 overlaps k_2 in v_2. *Intersect* () is called and returns k_1 as the chosen triangle, due to $\beta_1 < \beta_2$. A penalty is attributed to each of k_2 and k_3 excluding them from the returned solution, because k_2 and k_3 are neighborhood triangles of k_1. Algorithm 2 starts a second iteration and has as a result of Step c X = {1, 0, 0, 1, 0, 0, 0}. A positive response is returned because k_1 does not overlap k_4. Algorithm 2 terminates and returns {k_1, k_4} which is another optimal solution for this example.

Algorithm 2 (relaxed ILP)
Input: the graph G, the weight vectors α and β
Output: a packing of triangles $X \in \{0, 1\}^m$

```
a)   $\xi \leftarrow K$, $X \leftarrow \{0\}^m$;
b)   Do
c)     $X$ = SolveILP ($X$, (2), $\alpha$, $\beta$);
d)         if ($X$ is a packing)  goto step h;
e)             else Intersect ($X$ , $\xi$);
f)                 Penalty ($X$ , $\beta$);
g)   While ($X$ is not a packing solution);
h)   return $X$;
i)   end.
```

V. COMPUTANTIONAL COMPARISON

We now give some details of the experimental tests. We have implemented and tested the two aforementioned algorithms using different MTP instances and have compared their results to the solution returned by (1) via CPLEX 12.5 [14]. Due to the lack of benchmarks, the MTP instances are randomly generated in the following way: we start with a stable graph of size n, we fix a vertex $v_i \in V$ and chose randomly two other vertices v_g and $v_h \in V$, creating the triangle $k_j = \{v_i, v_g, v_h\} \in K$. We repeat this process until all vertices participate in at least one triangle. The size of the MTP instances are up to m = 4182 and n = 3000.

All the tests are done on Intel® Core™ i7-2600 CPU (3.40 GHz, 3.70 GHz), with 16GB of RAM. Algorithm 1 and Algorithm 2 are coded in the C Language. Table 2 summarizes the number of returned solutions along with the computation times. In the columns tuples (2, 3), (4, 5) and (6, 7) the number of returned triangles and the computation times of Algorithm 1, Algorithm 2 and CPLEX respectively are represented.

From Table 2, we observe that Algorithm 1 outperforms Algorithm 2 regarding the returned packing solutions for all generated MTP instances. However, both are dominated by CPLEX outcomes. We highlight that we interrupted CPLEX resolution after 48 hours for the last two MTP instances. Regarding the computational times, Algorithm 1 needs little CPU time compared to Algorithm 2 but both are still polynomial. These two heuristics provide results relatively quickly and are more suitable for user interaction system applications. In this context, the systems must return a solution with a certain quality of the approximation in a small amount of time.

We remark that the gap between the returned number of triangles by the two proposed heuristics and CPLEX solutions strongly depends on the order of the triangle set ξ. Indeed, in the case when two triangles have the same weight value, the first in the triangle set ξ is chosen. This choice allows to have deterministic solutions, if it happens to run the two heuristics several times in a row. An improvement of the two heuristics will be to sort the triangle set ξ in such a way that the most promising triangle will be placed first in ξ.

TABLE II. THE OBTAINED RESULTS OF ALGORITHM 1, ALGORITHM 2 AND CPLEX USING RANDOMLY GENERATED MTP INSTACES

MTP Instances (vertices, triangles)	The results and their Computation Times (CT)					
	Algorithm 1		Algorithm 2		CPLEX	
	Sol	CT (sec)	Sol	CT (sec)	Sol	CT
(600, 671)	135	0.073	131	0.16	167	3.26 (sec)
(800, 834)	188	0.096	172	0.33	230	19.2 (sec)
(850, 869)	194	0.12	186	0.48	240	10.2 (sec)
(900, 978)	207	0.142	196	0.49	260	11 min
(1000, 1050)	228	0.188	217	0.64	291	9 min
(1000, 1132)	229	0.184	214	0.73	295	11 min
(1100, 1152)	251	0.206	246	0.9	319	10 min
(1150, 1225)	261	0.237	249	1	330	132 min
(1200, 1280)	270	0.270	259	1.1	346	101 min
(1400, 1495)	322	0.482	303	1.8	Interrupted	
(3000, 4182)	719	5.04	666	26.7	Interrupted	

Sol, sec and min represent solution, secondes and minutes respectivelly.

VI. CONCLUSION AND FUTURE WORK

In this work, we deal with the problem of finding the maximum number of unweighted vertex-disjoint triangles in an undirected graph *G*. The problem is well-known to be NP-hard and APX-hard. Two heuristics have been proposed for this problem. To investigate the heuristic's performance, we provide a computational comparison of both of them to the CPLEX solution with randomly generated MTP instances. We have shown that the first heuristic outperforms the second regarding the obtained packing solutions and the respective computation times.

As a perspective for future work, we plan to use other metrics to compute the triangles and vertices weight improving also the penalty process. We also project to compare the two proposed heuristics with other approximation algorithms and to consider the weighted case of the MTP problem. The main motivation behind this work is to use the obtained solution for the MTP problem to consider the disjoint 3-SC problem [13]. In the disjoint 3-SC problem the optimality is the minimum of sets within a collection *C* of size at most 3 which cover the population *P*. We expect to address the disjoint 3-MSC problem applied on a graph $G = (V, E)$ where the vertices set *V* is covered by the fewest number of 3-clique, 2-clique and 1-clique (a k-clique is a clique of size *k*) maximizing the 3-clique packing hopefully generating a cover with the fewest sets.

ACKNOWLEDGMENT

We would like to thank the anonymous referees for their pertinent remarks which improved the presentation of this paper.

REFERENCES

[1] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman and Co., New York, 1979.

[2] M. Ashley, T. B.-Wolf, P. Berman, W. Chaovalitwongse, B. DasGupta, and M.-Y. Kao, "On approximation for covering and packing problems," in Journal of Computer and System Sciences, vol. 75, 2009, pp. 287–302.

[3] D. I. Belov and R. D. Armstrong, "A constraint programming approach to extract the maximum number of non-overlapping test forms," in Computational Optimization and Applications, vol. 33, 2006, pp.319–332.

[4] A. Caprara and R. Rizzi, "Packing triangles in bounded degree graphs," In Information Processing Letters, vol. 84, 2002, pp. 175-180.

[5] G. Manić and Y. Wakabayashi, "Packing triangles in low degree graphs and indifference graphs," in Discrete Math., vol. 308, 2008, pp. 1455-1471.

[6] C. A. J. Hurkens and A. Schrijver, "On the size of systems of sets every *t* of which have an SDR, with an application to the worst-case ratio of heuristics for packing problem," in SIAM J. DISC. Math., vol. 2, No. 1, 1989, pp. 68-72.

[7] R. Hassin and S. Rubinstein, "An approximation algorithm for maximum triangle packing," Discrete Applied Math., vol. 154, 2006, pp. 971-979.

[8] Z.-Z. Chen, R. Tanahashi, and L. Wang, "An improved randomized approximation algorithm for maximum triangle packing," in Discrete Applied Math., vol. 157, 2009, pp. 1640-1646.

[9] J. Wang, Q. Feng, and J. Chen, "An $O^*(3.53^{3\kappa})$-time parameterized algorithm for the 3-set packing problem," in Theorical Computer Science, vol. 412, 2011, pp. 1745-1753.

[10] P. Prosser, "Triangle packing with constraint programming," In the 9th International Workshop on Constraint Modelling and Reformulation, 2010, pp. 1-15.

[11] D. Bertsimas, D. A. Iancu, and D. Katz, "A new local search algorithm for binary optimization," INFORMS Journal on Computing, Articles in Advance, 2012, pp. 1-14.

[12] R. Borndörfer and R. Weismantel, "Set packing relaxations of some interger programs," ZIB-Report SC 97-30, 1999, pp. 1-19.

[13] R. Duh and M. Fürer, "Approximation of k-set cover by semi-local optimization,", In Proceedings of the 29th annual ACM Sym. On Theory of Computing 1997, pp. 256-264.

[14] www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ [retrieved:10.05.2013].