Partitioning Sparse Biological Networks into the Maximum Edge-Weighted k-Plexes

Vladimir Filipović
vladaf@matf.bg.ac.rs

joint work with M. Grbić, A. Kartelj, S. Janković and D. Matić

About my Department

University of Belgrade
***
Faculty of Mathematics
***
Department of Computer Science
***

Problem Domain

- In recent years there is an increasing effort to provide algorithms for better understanding of biological structures and processes.
- Partitioning networks into **high density sub-networks**, has already been proven as a useful technique for obtaining new information in understanding complicated relations between biological elements:
  - The protein side chain packing problem is transformed into a problem of finding a maximum weight clique. [AHK+06]. The edge weighting function is defined in a way that reflects the frequency of contact pairs in a database of proteins [JKTA06].
  - Molecular modules that are densely connected within themselves, but sparsely connected with the rest of the network are discovered by analyzing the multibody structure of the network of protein–protein interactions [SM03], [GKBC04].

Clique Relaxation

- Large number of biological networks classes contain only **sparse** networks.
- It such situations:
  - Partitioning into cliques can be too restrictive method and many potentially useful information about the interference of biological objects can be neglected.
  - Clique **relaxation** approaches could be even more useful.
  - Objects in each partition are still highly connected in a particular way, but not so restrictively to form a clique.
- Clique relaxation is a partition with high degree of its internal cohesion.
- This allows detection of semantically or functionally logical groups which, called **k-plexes**.
- Total sum of weights within all partitions should be as large as possible.

Problem Definition

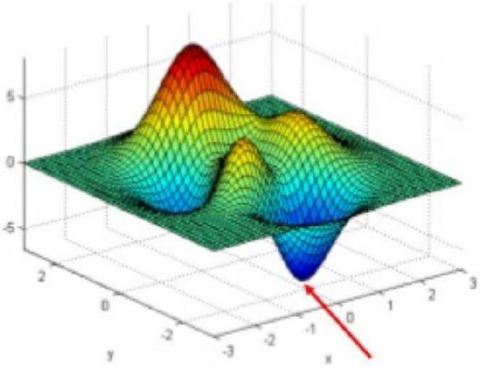
- Let a **network** (graph) be denoted as $G = (V,E)$, where $V=\{1,2,...,n\}$ is the set of vertices and E is the set of edges.
- Let graph G be unoriented, so edges are pairs of vertices, not the ordered pairs.
- Let uv be simply denote the **edge** $\{u,v\} \in E$. Vertices u and v are the **end-vertices** of the edge uv.
- With real numbers $w_{uv} > 0$, we denote the weight of the edge connecting nodes u and v.
- Let $k \geq 1$ be an integer. A set of nodes S is called **k-plex** if the degree of each node in the sub-network induced by set S is at least n-k.

Problem Definition (2)

- Weight of a k-plex is the sum of all its edge weights.
- Weight of the whole partition is the sum of the weights of all its k-plex components.
- Maximum k-plex partitioning problem (**Max-EkPP**) is defined as finding such a partition of G which is of the maximum total weight and each component is a k-plex.
  - If $k = 1$, the k-plex is a clique and the Max-EkPP is brought down to the maximum edge-weight clique partitioning problem (Max-ECP).
- Max-EkPP is an **optimization** problem.

Optimization problems

- Following elements are known:
  - search space S
  - solution space X, $X \subseteq S$
  - objective function f, $f{:}S \rightarrow R$, which maps elements of S to real numbers
- In **minimization** optimization problems, goal is to calculate $x^* \in X$, such that $f(x^*)=min\{f(x){:}x{\in}X\}$.



No Free Lunch Theorem for Optimization

- No Free Lunch Theorem for Optimization (**NFL**) is formulated by Wolpert and Macready [WM97].
- It states that there is no universal generalized strategy adequate for solving any opimization problem.
- Consequently, the only way one method can outperform another is if it is specialized to the specific problem under consideration.

Computational intelligence

- Computational intelligence is sub-discipline of Artificial intelligence.
- It includes wide spectrum of methods for solving **hard problems**, in situations when traditional approaches can not produce satisfiable solution.
- Computational intelligence methods often tolerate various kinds of inprecisement and approximation.
- Examples of computational intelligence methods:
  - neural networks
  - evolutionary algorithms
  - fuzzy sets
  - tabu search
  - etc.

Metaheuristics

- Metaheuristics are **generalized** computational intelligence methods that can be successfully **adopted** to various problem domains.
- Metaheuristics are trying to obtain the optimal solution, or the solution that is close to optimal one.
- Basic metaheuristics components vary from fast and simple local searches up to complex learning processes.
- Metaheuristic algorithms are characterized with **approximation** and **non-determinism**.
- Basic metaheuristics concepts are abstractly represented.
- Metaheuristics should be adapted to problem domain, otherwise they should won't obtain satisfiable solution.

Metaheuristic vs. exact methods

- Issues with **exact** methods:
  - time resources
  - memory resources
  - sometimes, solution can not be obtained at all
- Issues with **metaheuristic** methods:
  - obtain approximate solution
  - obtain good solution
  - optimality of the obtained solution is not guarantied
  - limited computational resources

Classification of metaheuristic methods

- Population-based
  - Evolutionary algorithms
  - Particle Swarm Optimization
  - Electromagnetism-based Metaheuristics
  - etc.
- Single-solution
  - Tabu Search
  - Simulated Annealing
  - Variable Neighborhood Search
  - etc.

Variable Neighborhood Search

- Variable Neighborhood Search (**VNS**) algorithm is a robust metaheuristic introduced by Hansen and Mladenović[MH97].
- It is single-solution metaheuristic method based on neighborhood search.
- Neighborhood structure, or neighborhood, N of the search space X is function $N{:}X \rightarrow P(X)$, where $P(X)$ is partitive set of X. In other words, neighborhood is a function that maps solution to set of its neighbors.
- Determining neighborhood is based on distance function. Metrics function $d{:}X \times X \rightarrow R$ induces distance among points in solution space (solutions).
- ϵ-neighborhood of the solution $x \in X$ is set $N_\epsilon(x) = \{y \in X : d(x,y) < \epsilon\}$

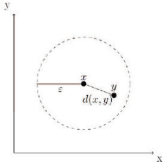
VNS - Metrics and neighborhoods

- Examples of metric functions and relevant neighborhoods:

  - $X = \mathbb{R}^2$

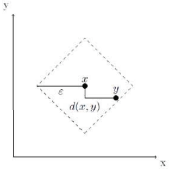
    Euklid

    $d(\mathrm{x},\mathrm{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$

    Menhetn

    $d(\mathrm{x},\mathrm{y}) = |x_1 - y_1| + |x_2 - y_2|$

  - $X = \{0,1\}^n$

    Haming $\quad d(\mathrm{x},\mathrm{y}) = \sum_{i=1}^{n} |x_i - y_i|$

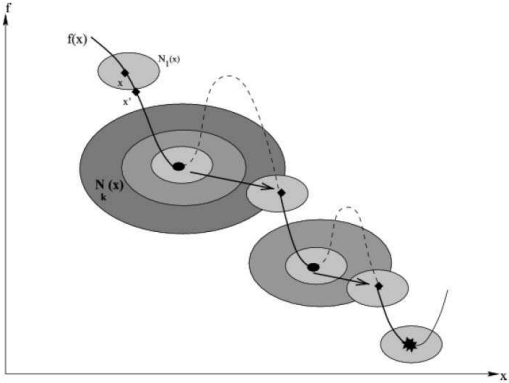
d(1101101001,1101001101)=2

VNS - Basic principles

- The main searching principle of a VNS is based on the following evidences:
  1. a local optimum found in one neighborhood structure is not necessarily a local optimum for some other neighborhood structure.
  2. global optimum is local optima for all neighborhood structures.
  3. (empirical) multiple local optima are correlated in some sense - usually, they are close to each other according to selected metrics.

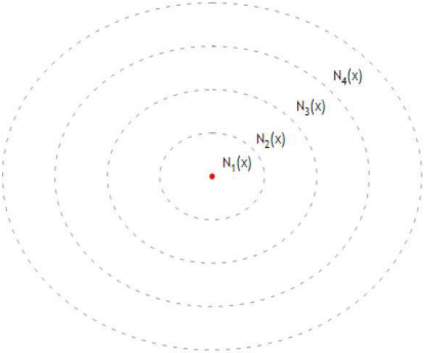
Phases in VNS

- Execution of VNS method is basically repetitive execution of two operations:
  - Shaking - extend the search space of the current solution
  - Local search - improvement of the obtained solution within



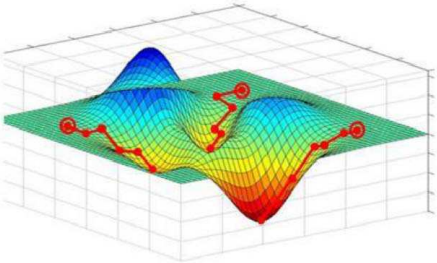
VNS - Shaking

- Main purpose of the **shaking** procedure is to extend the search space of the current solution in order to reduce the possibility that the algorithm falls into suboptimal solutions.
- Within shaking procedure, algorithm creates a system of neighborhoods used for deriving new solutions based on the current best solution x.



VNS - Local search

- **Local search** consumes most time during VNS execution.
- There are two main local search strategies:
  - first improvement strategy
  - best improvement strategy



VNS Pseudo-code for Max-EkPP

input: n_{min}, n_{max}, it_{max}, $itrep_{max}$, t_{max}, $prob$, k
output: $\mathbf{x}$

```
x ← initializeSolution();
n ← n_min;
it ← 1;
while it < it_max ∧ (it − it_lastimpr) < itrep_max ∧ t_run < t_max do
    x′ ← shaking(x,n);
    x″ ← localSearch(x′, k);
    move ← shouldMove(x, x″, prob);
    if move then
        x ← x″ ;
    else if n < n_max then
        n ← n + 1;
    else
        n ← n_min;
    it ← it + 1;
end
return x;
```

Solution representation for Max-EkPP

- The solution of the proposed VNS algorithm is represented by an **integer array** x of the length $\|V\|$.
- Each element of the array corresponds to one vertex of the graph, denoting to which component (partition) the corresponding vertex is assigned.
  - More precisely, the vertex i is assigned to the component V_j if x_i =j.
- The initial solution is created by assigning each element of the array x random integer number from interval $[1, 2, ..., \sqrt{\|V\|}]$.
  - The upper bound for the initial number of partitions $\sqrt{\|V\|}$ is empirically determined.
- Unfeasible solutions are not implicitly disallowed by the representation.

Solution representation for Max-EkPP (2)

- Let $(V_1, V_2, ..., V_l)$ be a (not necessarily feasible) solution of the Max-EkPP.
- Let w_{total} be the total sum of the weights of all edges in the network G, e.g. $w_{total} = \sum_{uv \in E} w_{uv}$.
- We introduce the term **correct** vertex in a solution. A vertex $v \in V_j$, $j \in \{1, 2, ..., \}$ is **correct** if the degree of v in the network induced by V_j at least $|V_j| - k$.
  - **Corollary:** If every vertex in a partition is correct, then that partition is a **k-plex**.
- Let $correct_{total}$ be the total number of correct vertices in the solution e.g. in the partition $(V_1, V_2, ..., V_l)$.
- Additionally, in the partition $(V_1, V_2, ..., V_l)$, with w_{sol} we denote the total sum of the weights of all edges with correct end vertices.

Objective function for Max-EkPP

- Objective function of the solution is:
  $f(V_1, V_2, ..., V_l) = correct_{total} + \frac{w_{sol}}{w_{total}}$
- Since the value w_{sol}/w_{total} is always less or equal to 1, the objective function mainly depends on the first term, i.e. the total number of correct vertices in the solution.
- Consequently:
  - Objective function of any feasible solution will be greater than the objective function of any infeasible solution.
  - If two solutions have the same number of correct vertices, then the solution with greater total sum of the weights has also greater objective value.
  - Maximization process discards solutions with many incorrect vertices and directs the search into the feasible regions.
- At the same time, the proposed objective function properly orders infeasible solutions.

Shaking procedure for Max-EkPP

For defining the k-th neighborhood we use the following procedure:

1. k vertices from V are randomly chosen.
2. For each chosen vertex v, the algorithm changes its component as follows:

   2.1 If l is the total number of partitions, then an integer q is randomly chosen from the set $\{1, 2, ..., l + 1\}$.

   2.1.1 If $q < l + 1$, then the vertex v is moved to the existing partition V_q.

   2.1.2 If $q = l + 1$, then a new singleton partition is established ($V_{l+1} = \{v\}$) and the total number of partitions is increased by one.

   2.1.3 If the old partition, from which the vertex v was chosen, becomes empty, then the total number of partitions is decreased by one.

Local search procedure for Max-EkPP

```
input: x', k
output: x''
x'' ← x';
n ← |x''|;
impr ← true;
while impr do
    impr ← false;
    i ← ir ← random(1,n);
    do
        l ←countDistinctValues(x'')+1;
        p ← pr ← random(1,l);
        do
            newObj ← repositionObjectiveValue(x'',i,p,k);
            if newObj > x''.obj then
                x'' ← reposition(x'',i,p,k);
                impr ← true;
                break;
            p ← (p mod l) + 1;
        while p ≠ pr;
        if impr then
            break;
        i ← (i mod n) + 1;
    while i ≠ ir;
end
return x'';
```

- Notes:
  - LS is based on "1-swap first improvement" strategy
  - LS iteratively examines new solutions formed by moving a single vertex from its belonging component to some other component
  - Let v be a vertex which is the subject of movement. If l is the total number of components, then a random integer p is chosen from $\{1,2,\ldots,l+1\}$. If p is less than $l+1$, the vertex v is moved to the existing partition Vp, else a new partition $Vp=\{\ v\ \}$ is established.

Experimental data

- In order to make the comparison to the other method from the literature as fair as possible, we used the same benchmark data sets as in [Mar16].
- We tested them for three values of k, namely $k \in \{1, 2, 3\}$.
- The first two sets contain biological instances created on metabolic reactions from [FFF+03], [Mar16].

Tabela: View of considered biological metabolite networks

inst.	$\|V\|$	$\|E\|$	*density*	*inst.*	$\|V\|$	$\|E\|$	*density*
m-t1	991	4161	0.0085	r-t1	1393	56276	0.0580
m-t2	602	1520	0.0084	r-t2	1183	17776	0.0254
m-t3	177	269	0.0173	r-t3	663	1782	0.0081
m-t4	129	166	0.0201	r-t4	377	321	0.0045
m-t5	75	84	0.0303	r-t5	45	27	0.0272

Experimental data (2)

- The third set of instances was taken from the well known DIMACS database, available at http://www.dcs.gla.ac.uk/∼pat/maxClique.
- Testing the VNS on DIMACS instances consists of two phases:
  1. In the first phase, we followed the approach from [Mar16] and took DIMACS instances with less than 100 vertices and larger sparse instances with less than 200 vertices and density at most 0.25.
  2. In the second phase,we tested our VNS on the rest of 73 DIMACS instances.
- Since the original DIMACS instances are not weighted, like in other papers [Mar16, GM15], we also followed the weighting strategy proposed in [Pul08], by setting $w_{i,j} = ((i + j) \mod 200) + 1$.

Experimental environment

- For each instance, we performed 10 independent executions of the VNS algorithm.
- Termination criterion is based on the combination of three criteria:
  1. Maximum total number of iterations reached, where $it_{max} = 20000$;
  2. Maximum number of iterations without improvement, which is set to 10000, is reached;
  3. Maximum total execution time (set to 1 hour) is reached.

Experimental environment (2)

- Other control parameters are set as follows: n_{min}=1 i $n_{max} = 80$, $prob = 0.1$.
- All experiments are performed using the following configuration:
  - Intel Xeon E5410 Computer, CPU @2.33 GHz with 16 GB RAM and Windows Server 2012 2R 64Bit operating system.
  - For each execution only one thread/processor is used.
  - The VNS is implemented in C programming language and compiled with Visual Studio 2015 compiler.

Experimental results obtained on SC-NIP-m-tr instances

k	*inst.*	*opt*	*best*	VNS_{best}	VNS_{avg}	VNS_{gap}	$VNS_{t\text{-}tot}$	*ILP*	ILP_t
1	m-t1	1866	1866	*opt*	1864	0.11	3600.22	*opt*	2296.94
1	m-t2	1538	1538	*opt*	1538	0	1072.51	*opt*	1.25
1	m-t3	910	910	*opt*	910	0	92.96	*opt*	0.02
1	m-t4	831	831	*opt*	831	0	45.5	*opt*	0
1	m-t5	723	723	*opt*	723	0	15.73	*opt*	0
2	m-t1	-	2151	*new*	2147.3	0.17	3600.14	-	-
2	m-t2	-	1773	*new*	1771.8	0.07	1495.49	-	-
2	m-t3	1021	1021	*opt*	1021	0	100.74	*opt*	50.43
2	m-t4	907	907	*opt*	907	0	54.75	*opt*	3.03
2	m-t5	801	801	*opt*	801	0	16.42	*opt*	0.2
3	m-t1	-	2353	*new*	2337.1	0.68	3600.18	-	-
3	m-t2	-	1943	*new*	1939.4	0.19	1988.38	-	-
3	m-t3	-	1141	*new*	1141	0	121.08	-	-
3	m-t4	-	1022	*new*	1022	0	69.79	-	-
3	m-t5	887	887	*opt*	887	0	17.62	*opt*	34.2

Experimental results obtained on SC-NIP-m-tr instances (2)

- Results from previous table shows that:
  - ILP method from was more successful for $k = 1$ comparing to the two other values of k. It succeeded to find all optimal solutions for $k = 1$, three optima for $k = 2$ and one optimum for $k = 3$.
  - Proposed VNS succeeds to find all 9 known optimal solutions.
  - In addition, for each of these instances, the VNS reaches the optimal value in each of 10 runs.
  - For the set of 6 instances, where ILP model could not find any solution, the VNS succeeds to find solution in a reasonable time up to 1 hour.
  - Average gap for VNS is rather small (less than 1 for all instances), so we can conclude that the VNS is rather stable while solving this class of instances.

Experimental results obtained on SC-NIP-r-tr instances

k	*inst.*	*opt*	*best*	VNS_{best}	VNS_{avg}	VNS_{gap}	$VNS_{t\text{-}tot}$	*ILP*	ILP_t
1	r-t1	-	57681	*new*	57544.6	0.24	3607.77	-	
1	r-t2	34576	34576	*opt*	34561.6	0.04	3601.2	*opt*	4.26
1	r-t3	5411	5411	*opt*	5411	0	1550.95	*opt*	0.08
1	r-t4	1232	1232	*opt*	1232	0	327.82	*opt*	0
1	r-t5	140	140	*opt*	140	0	3.71	*opt*	0.02
2	r-t1	-	57729	*new*	57496	0.4	3602.58	-	
2	r-t2	-	34592	*new*	34563.6	0.08	3601.65	-	
2	r-t3	-	5423	*new*	5423	0	1569.11	3183	>10800
2	r-t4	1245	1245	*opt*	1245	0	331.75	*opt*	6.4
2	r-t5	140	140	*opt*	140	0	3.82	*opt*	0.01
3	r-t1	-	57775	*new*	57587.4	0.33	3602.19	-	
3	r-t2	-	34641	*new*	34572.5	0.2	3601.26	-	
3	r-t3	-	5465	*new*	5465	0	1496.84	-	
3	r-t4	-	1245	*new*	1245	0	327.45	-	
3	r-t5	140	140	*opt*	140	0	3.84	*opt*	0.14

Experimental results obtained on SC-NIP-r-tr instances (2)

- Results from previous table shows that:
  - VNS achieves all 7 known optimal solutions.
  - For the rest of 8 instances VNS achieves the new best results.
  - ILP method succeeds to find 7 optimal solutions: four optima for k=1, two optima for k=2 and one optimum for k=3.
  - Proposed VNS succeeds to find all these optima, also providing high quality solutions for other cases.
  - For five SC-NIP-r-tr instances, the algorithm stopped after the time limit is reached (1 hour), while for other instances, the termination happened after maximum number of iteration was reached.
  - The average gap for this class is again rather small (less than 1) for all instances.

Experimental results obtained on biological instances

- Results obtained on biological instances (previous two tables) indicate that:
  - SCNIP-r-tr instances are more challenging than SCNIP-m-tr because of their dimensions, so necessary run-time is proportionally greater comparing to the execution times for SC-NIP-m-tr instances.
  - In both classes of biological instances, execution time depends on the graph density, i.e. smaller density induces smaller execution time.
  - Natural explanation for such behavior is that smaller number of edges causes the lowering of the total number of executions of the local search procedure, which further leads to the shorter overall execution time.
  - Value of objective functions increases with increasing the value of k - because the total number of edges included in clusters increases with the relaxation of the adjacency conditions in each cluster.

Experimental results obtained on DIMACS instances

k	*inst.*	*opt*	*best*	VNS_{best}	VNS_{avg}	VNS_{gap}	$VNS_{t\text{-}tot}$	*ILP*	ILP_t
1	c200-1	98711	98711	*opt*	98711	0	234.43	*opt*	47.08
2	c200-1	98711	98711	*opt*	98543.2	0.17	202.87	*opt*	567.44
3	c200-1	-	98711	*new*	98571.8	0.14	193.7	-	-
1	c200-2	213248	213248	*opt*	213246.8	0	540.89	*opt*	0.22
2	c200-2	213248	213248	*opt*	212194.6	0.49	360.5	*opt*	47.28
3	c200-2	-	213248	*new*	211143.8	0.99	292.97	-	-
1	h6-2	65472	65472	*opt*	65472	0	114.53	*opt*	0.2
2	h6-2	-	65472	*best*	65472	0	61.91	*best*	>10800
3	h6-2	-	65472	*best*	65472	0	46.15	*best*	>10800
1	h6-4	6336	6336	*opt*	6336	0	53.29	*opt*	0.34
2	h6-4	-	8184	*new*	8184	0	74.81	6966	>10800
3	h6-4	-	10560	*new*	10560	0	77.57	4567	>10800
1	j8-2-4	1260	1260	*opt*	1260	0	7.63	*opt*	0.06
2	j8-2-4	-	1365	*new*	1363.5	0.11	10.41	1355	>10800
3	j8-2-4	-	1996	*best*	1996	0	7.34	*best*	>10800
1	j8-4-4	-	27874	*new*	27874	0	169.18	27864	>10800
2	j8-4-4	-	31320	*new*	31147.2	0.55	124.87	12770	>10800
3	j8-4-4	-	37096	*new*	35910.3	3.2	155.73	12948	>10800
1	M_a9	14868	14868	*opt*	14865	0.02	27.55	*opt*	1215.34
2	M_a9	-	23055	*new*	23053.8	0.01	25.96	23047	>10800
3	M_a9	33660	33660	*opt*	33660	0	14.23	*opt*	319.24

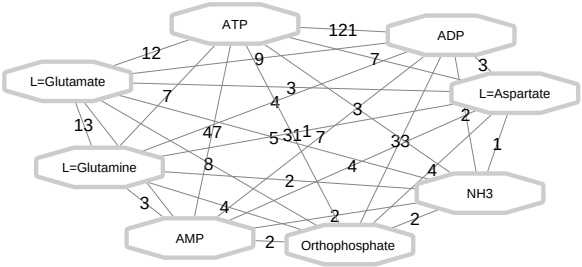
Experimental results obtained on DIMACS instances (2)

- All these instances belong to c-fat, MANN, hamming and johnson families.
- It can be concluded that the proposed VNS achieves 9/9 known optimal solutions. In the rest of 12 cases, VNS achieves 4 best known solutions, while in 8 cases finds new best results.
- In regards to the efficiency of the proposed VNS on the larger problem dimensions, the algorithm is also tested on the challenging set of the rest of 73 DIMACS instances.
- Although the Max-EkPP is mostly applied on sparse graphs, to achieve completeness of our approach, we decided to test the proposed VNS even on denser DIMACS instances.
- For these instances, up to now, no solution is presented in the literature.
- Although optimality cannot be proved, small gap values on these instances suggest that VNS obtained high quality solutions.
- Obtained results are available at the website: http://matinf.pmf.unibl.org/dimacs/

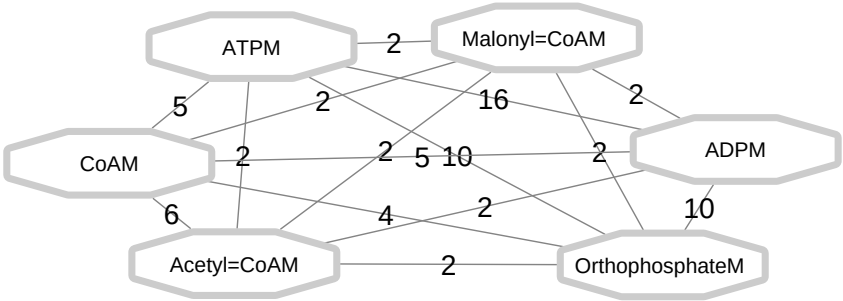
Biological explanation of obtained results

- Among many metabolic processes that appeared in various k-plexes obtained by the proposed VNS algorithm, we chose to discuss following processes:
  1. amino acid degradation process;
  2. fatty acids synthesis;
  3. vitamin B6 synthesis;
  4. oxidation of the succinate to the fumarate;
  5. formaldehyde oxidation.
- In order to confirm the reliability of the obtained results, particular information of the biochemical pathways of considered organism Saccharomyces cerevisiae are checked and confirmed with the data presented in Yeast Pathways Database [yea].

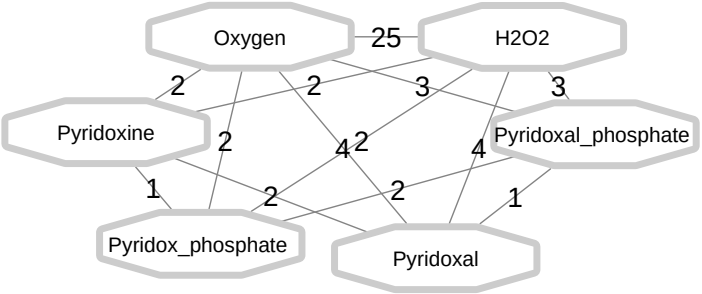
Amino acid degradation process



Fatty acids synthesis



Vitamin B6 synthesis



Conclusions

- Variable neighborhood search based heuristics is designed for solving the problem of partitioning sparse biological networks into edge-weighted structures called k-plexes.
- Max-EkPP problem has been solved for the first time by a metaheuristic approach.
- Proposed VNS implements a fast swap-based local search, as well as a specific objective function which favors feasible solutions over infeasible ones, taking into consideration the degree of every vertex in each partition.
- Extensive computational analysis is performed on existing sparse biological metabolic networks, as well as on the other artificial instances from literature.

Conclusions (2)

- From computational point of view:
  - It was shown that the proposed VNS succeeded to achieve all already known optimal or best solutions.
  - It was also shown that VNS is able to find new high quality solutions for the other, previously unsolved instances, in a reasonable time.
- From biological point of view:
  - In the deep analysis of the clusters identified by various values of k on a biological metabolic instance, we confirmed that the algorithm finds many clusters in which the intermediates, that figures in many important metabolic reactions, are highly connected.
  - The relaxation of the adjacency condition leads to obtainment of more useful clusters, which helps in discovering new biological relations or confirming the existing ones.

Direction for further research

- This research can be extended in several ways:
  - It would be interesting to apply the VNS on solving similar problems, including both biological and non-biological applications.
  - Another direction for the further investigation of this problem can include parallelization of the proposed VNS algorithms and running on some powerful multiprocessor system.

Literature I

- Tatsuya Akutsu, Morihiro Hayashida, Dukka Bahadur KC, Etsuji Tomita, Jun'ichi Suzuki, and Katsuhisa Horimoto, *Dynamic programming and clique based approaches for protein threading with profiles and constraints*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **89** (2006), no. 5, 1215–1222.
- Jochen Förster, Iman Famili, Patrick Fu, Bernhard Ø Palsson, and Jens Nielsen, *Genome-scale reconstruction of the saccharomyces cerevisiae metabolic network*, Genome research **13** (2003), no. 2, 244–253.
- Julien Gagneur, Roland Krause, Tewis Bouwmeester, and Georg Casari, *Modular decomposition of protein-protein interaction networks*, Genome biology **5** (2004), no. 8, R57.
- Luis Gouveia and Pedro Martins, *Solving the maximum edge-weight clique problem in sparse graphs with compact formulations*, EURO Journal on Computational Optimization **3** (2015), no. 1, 1–30.

Literature II

- Brown JB, Dukka Bahadur KC, Etsuji Tomita, and Tatsuya Akutsu, *Multiple methods for protein side chain packing using maximum weight cliques*, Genome Informatics **17** (2006), no. 1, 3–12.
- Pedro Martins, *Modeling the maximum edge-weight k-plex partitioning problem*, arXiv preprint arXiv:1612.06243 (2016).
- N. Mladenović and P. Hansen, *Variable neighbourhood search*, Computers & Operations Research **24** (1997), 1097–1100.
- Wayne Pullan, *Approximating the maximum vertex/edge weighted clique using local search*, Journal of Heuristics **14** (2008), no. 2, 117–134.
- Victor Spirin and Leonid A Mirny, *Protein complexes and functional modules in molecular networks*, Proceedings of the National Academy of Sciences **100** (2003), no. 21, 12123–12128.

Literature III

- David H. Wolpert and William G. Macready, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation **1** (1997), 67 – 82.
- *Yeast pathways database*, `http://https://pathway.yeastgenome.org/`, Accessed: 2017-11-11.

Thank you for your attention!

Contact info:

Vladimir Filipović
e-mail: vladaf@matf.bg.ac.rs
web: www.matf.bg.ac.rs/∼vladaf