A Tool for Calculating GDVs over a PPI Network

Project in Bioinformatics

Technion – Israel Institute of Technology
Computer Science Faculty

Samer Mattar & Reuven Shirazi
5/1/2010

Contents

Introduction

Approach

As part of the proteomics research, *PPI networks* are used to model the interactions between proteins. In order to understand the function of a protein, a method which has been suggested is to analyze its connectivity properties in the PPI network.

Finding an appropriate and agreeable method for characterizing topologies in a network is required for evaluating the connectivity properties. For that purpose, the *graphlet* concept has been introduced to define a specific topology, or pattern, inside a network.

One common direction for setting a hypothesis on the functionality of a specific protein is representing the count of occurrences of the protein at a specific position in a graphlet.

Concepts

- **Graphlet** - Small connected graph patterns
  - Non-isomorphic Graphlets are identified separately
  - Large networks induce Graphlets as sub-graphs
  - Each node in a graphlet is identified by a specific role which is defined by its position in the graphlet (See the Q section in figure 1)

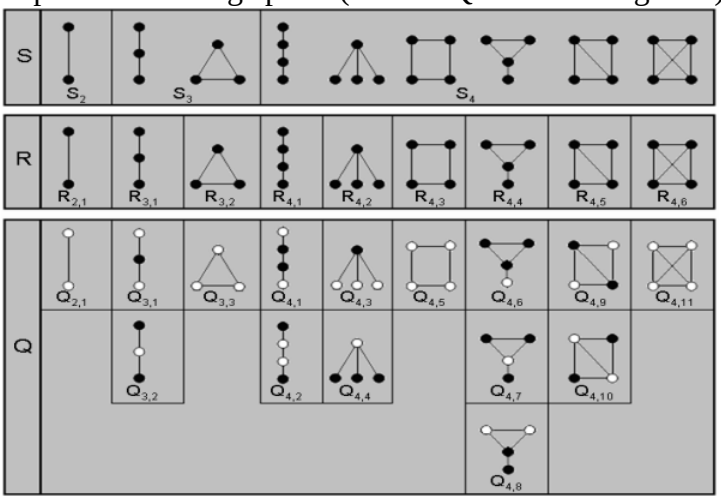


Figure 1: graphlets of max size 4

- **Graphlet Degree Vector** - GDV (with graphlet size=n):
  - Is a vector, indexed by all the roles in all the graphlets up to size n (e.g. for n=3 then the GDV size is 15, figure 1 above).
  - Given a graph, then a GDV of each node describes the number of occurrences of that node at each specific role.
  - The GDV can be reduced to shorter and more general vectors:
    - Q-type: number of occurrences in each role of each graphlet of each size up to n
    - R-Type: number of occurrences in each graphlet of each size up to n.
    - S-type: number of occurrences in each graphlet size up to n

- **PPI Network** - Protein-Protein Interactions in a specific organism (or a tissue..) represented as a graph
    - Each node represents protein
    - An edge usually represents physical interaction

- **Graphlet Satisfaction over a Set of Nodes** - given graphlet Gt and graph G=(E,N) and Node set $S \subseteq G$. Then it is said that Gt is satisfied over S iff there $\exists E' \subseteq E$ and $E' = \{(v, w) | v, w \in S\}$ such that $Gt = (E', S)$.

Project Objectives

1. The general objective of the project is developing a tool which calculates the GDV for each protein (node) given a PPI network.
2. Supporting common PPI networks formats, parsing them to a common data structure representation (Graph)
3. Providing the option for a user to apply different parameters for executing the calculation and getting the output data (Max graphlet size, GDV type, required graphlets)

Challenges

Efficiency

Calculating GDV requires traversing the whole network, thus there are two main aspects to optimize the tool:

- An appropriate Data structure needed to be selected to represent the network in a way that optimizes access to the nodes of the network.
- In order to traverse the network in an optimized way and to minimize reoccurrences, graphlet identification must be in a recursive way thus starting from a small graphlet and branching out of it to the bigger graphlets. In this way we use the information is already known for the smaller graphlet.

Scalability

Developing the algorithm for max graphlet size of 4 was rather a big change for graphlet size of 3. Thus, when trying to develop the algorithm for max size of 5 – we approached the problem in a more general way.
As a final result, we developed a quite generic algorithm which is based on data from an external description file in XML format, which "describes" any supported graphlet to the application in an agreeable format and sets the maximal supported graphlets size.
In order to calculate larger graphlets than the application currently supports, the only change needs to be done to that description file according to its existing format, and basically any size of graphlet can then be supported.

Flexibility

Any users may want the information in a different way containing different details. We developed a mechanism to get users requirements and provide him the result table according to his requests.

Duplication

A general problem when traversing graphs is determining when a certain topology is encountered more than once. For example a symmetric graphlet can be "encountered" twice or more as isomorphic shapes. Since we want to calculate each graphlet roles only once, we took into consideration a factor that is responsible to correct such duplication. See the solution under Project Description section below.

Project Description

Input Data

The main input for the tool is a PPI file. That is an XML file which represents the PPI Network as a graph to be analyzed and can be submitted in two well known formats MIF and XIN

Algorithm Configuration:

The algorithm of the tool can be configured using an XML file: *GraphletDesc.xml*. That file contains data on all the supported Graphlets which can be found and calculated by the tool over a PPI network. This file actually defines the way the main algorithm traverse a given input PPI network, thus the editing of that file should be done wisely, and is not required to be edited for each calculation. The format for describing a graphlet is consisted of 2 main parts: *Description* and *Roles*.

The *Description* part defines the details for the algorithm on how to find the graphlet and satisfy it over a given set of nodes. This definition is recursive, meaning the description of graphlet of size *n*, is either as an addition of a **node** to a **sub-graphlet** of size *n-1*, or as an addition of an **edge** to a **sub-graphlet** of size *n*.

*In the following example, the nodes are marked according to the order of finding them by the traversal algorithm (not graphlet roles!).Graphlet G4 is created by an addition of a new **node** (#3) to **sub-graphlet** G1, connected by an edge from an existing node (#1):*

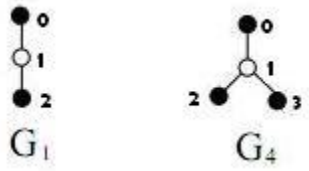


Figure 2

The *Roles* part defines the details for the tool on how to modify the GDV of each node within the satisfied graphlet. The set of nodes which were found to satisfy the graphlet by the algorithm is stored in an array. Typically, a mapping between the node indices in that array to their role in the satisfied graphlet is defined in this part, such that modifying the node's GDV is directly derived from this mapping.

In the following example, the nodes are marked according to their role position in the GDV. The GDV of the central node will be modified in position (role) #7 and the external nodes will be modified in position (role) #6 according to Figure 3

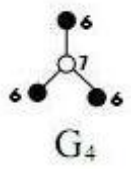


Figure 3

Another detail defined for each graphlet in the file is *Duplication* value, which describes a certain symmetry property of that graphlet. A graphlet can be found and satisfied a few times during the algorithm execution, and this number is derived from its symmetry. Thus, the tool needs to take this value into consideration when preparing the GDVs output. Typically, the values in a GDV at positions that correspond to roles which are part of that graphlet need to be divided by that right *Duplication* value.

In the previous example, graphlet G4 has Duplication value of 6. It has 3 nodes with the same role, thus we identify the same graphlet 6 times, multiplying the following:
3 times for the symmetry of node #0 (according to Figure 2*),*
2 times for the symmetry of node #2 (after node #0 was already set).
There's no need to consider the symmetry of node #3 after its 2 isomorphic nodes were already set.

See also Appendix A for an example.

Software Architecture:

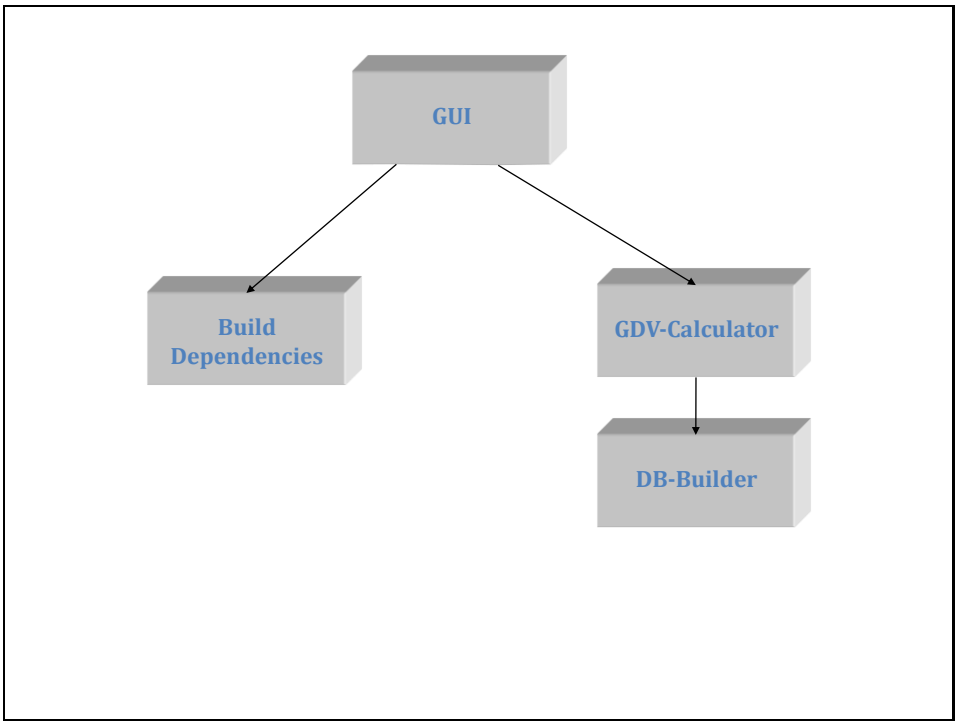


Figure 4 – Project S W Architecture

Build Dependencies (BuildDependencies.exe):
A stand-alone application which constructs 2 files with parsed information based on the *GraphletDesc.xml* file:

1. *Dependencies.xml*
   The file describes the graphlets dependency tree represented in XML file syntax. This file describes the order of search for graphlets, given a starting node extending it up to graphlets of max size.
   Each *branching* XML-node in that tree contains a *graphlet_group* XML-node. Within this group, graphlets are based on the same set of nodes (Infrastructure

nodes) and differ between them only by one edge. A new *branching* XML-node represents the extension of the infrastructure nodes by one more node which connects to a specific node of the infrastructure nodes. See also Appendix BAppendix B for an example.

2. *RoleMap.txt*
   The file defines a double mapping of Roles → Graphlets → Size. This data is used by the calculation algorithm for filtering the full results according the user defined parameters.

DB-Builder (in GDVCalc.exe):
A module which is responsible of parsing the PPI Network files (XIN/MIF) into a unified data structure which represents the network. The data structure in use is an adjacencies table implemented as a nested string hash-table (Hash keys are nodes IDs).

GDV-Calculator (in GDVCalc.exe):
A module which actually executes the GDV calculation algorithm for each node in a given PPI Network and outputs the results to the file *GDVout.txt*. The following 5 data items are required as input to the algorithm (See also):

1. PPI adjacency table
2. *Dependencies.xml*
3. *GraphletDesc.xml*
4. *RoleMap.txt*
5. User input:
   Max graphlet-size (3, 4, 5), GDV type (Q \ R \ S) and specific graphlet selection.

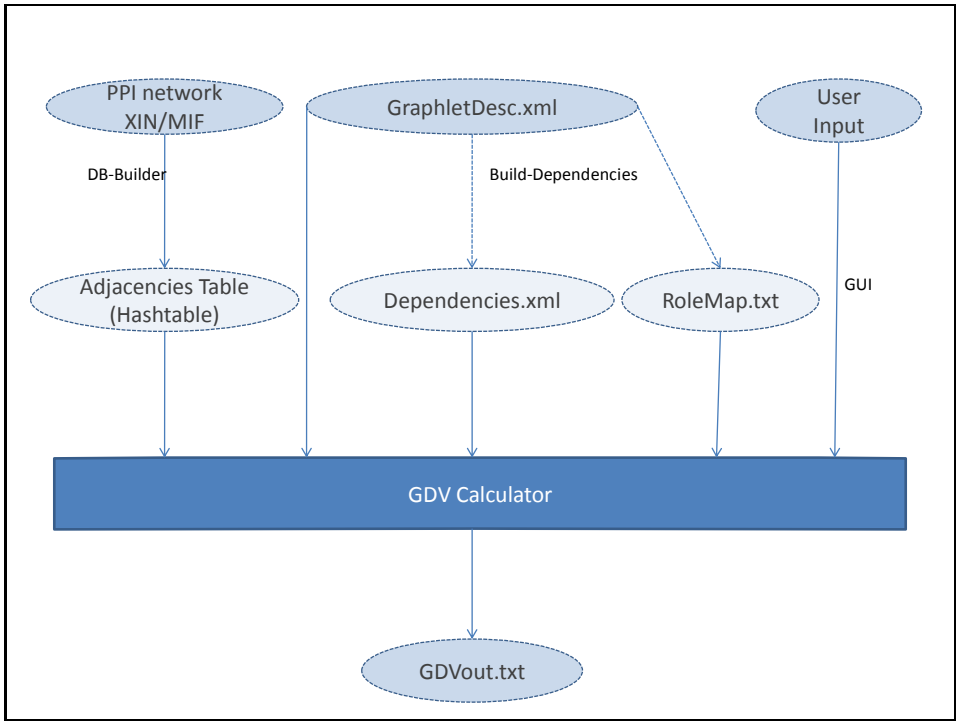


Figure 5 – Calculation Flow

GUI (GDV.UI.exe):
A .NET Form Application , which is used for getting input and user preferences for the calculated results. This module also manages the framework workflow, as described below, and after verifying all the application components exist, including the different XML files, it starts Build Dependencies app if needed, then executes and GDVCalc app.
At any stage, the GUI displays corresponding messages for letting the user know the application status.

Workflow:

- Data Creation:
  - Build Dependencies is called to create the files *Dependencies.xml and RoleMap.txt* if they don't already exist, as described in Software Architecture section above.
  - The DB-Builder is called to convert the given PPI Network to the unified Data Base, as described in Software Architecture section above.

- Algorithm:

The backbone of the algorithm traverses each of the nodes in the graph, and starting from that node – tries to seek graphlets according to branching of the Dependencies XML file.
For each of the graphlets in a branching group, all the neighbors of branching node are taken into consideration.
When a graphlet is found to be satisfied, the GDV of each of its nodes is updated in the corresponding role.

The algorithm is based on the following recursive call:

GDV_Calc_Rec(***S***, ***B***):
1. Let ***S*** be base graphlet set of nodes, and ***B*** be dependencies branch. Do:
   1.1. For each of the graphlets belongs to ***B***, try to satisfy ***S*** and then update GDVs of each of the nodes in ***S***.
   1.2. If still (size-of-***S*** < max_graphlet_size) - Do:
      1.2.1. FOREACH branch ***B'*** of branch ***B*** - Do:
         1.2.1.1. FOREACH neighbor ***N*** of the branching node (Taken from the head graphlet of branch ***B*** group) – make the recursive call *GDV_Calc_Rec*(***S***=***S***∪{***N***}, ***B***=***B'***)

The starting point of the algorithm:

Main:
1. For each node ***N*** in the network do:
   1.1. Start the recursive call *GDV_Calc_Rec*(***S***= {***N***}, ***B***= Dependencies Root)
   1.2. Correct GDVs duplications
   1.3. Export the calculated GDVs to the 2 output files.

Output files

The application products are 2 output files:

1. *GDVout.txt* - A file containing the data according to the user parameters:
   - GDV type Q, R or S
   - The specific graphlets that were selected
2. *GDVoutFull.txt* - A file containing the full calculated data with no reductions

The files format is tab-delimited, so they can be opened and manipulated by an application for editing\viewing data tables such as MS Excel.

Complexity

When analyzing the complexity of an algorithm which performs nested traversal over a graph with set of vertices *V*, set of edges *E* and limitation on recursion depth *M* – the intuitive upper bound is $O(|V|\cdot|E|^M)$.

However our algorithm has 2 main aspects which are worth mentioning, as modification to the above analysis:

1. Let *D* be the maximal vertex degree of all the vertices in the graph. Since the algorithm advances only on existing edges, given a current vertex - the maximal number of options to extend it is *D*, rather than |*E*|.
2. As opposed to a straight forward algorithm - a given graphlet can be extended from each of its positions. Our algorithm introduced the concept of *branching*, which added to the complexity, the number of positions from which the branching can occur. For each recursive iteration *i* - the factor to multiply is *i-1*, so for each starting node we should multiply the complexity with *M!*.

In total we get an upper bound of $O(|V|\cdot D^M\cdot M!)$, where M equals, in our case, *Max_graphlet_size - 1*.

Anecdote

The ratio of raising complexity values as a function of M is D*M .

Numeric Example:
Assuming the time ratio between 2 runs of the application, on the following PPI network:
2500-nodes
4000-edges
Max-degree D of 20

Assuming the following 2 assumptions:
Increasing M from 3 to 4 changes runtime 80 times longer (e.g. 4 **minutes** for max graphlet size of 4, and 5 **hours** for size of 5)
Let's assume hypothetically the **actual runtime** is of the same order of the complexity **upper bound**
We can assume that running the application for the same graph with max graphlet size of 6, would take 100 times longer, meaning… more than 20 days.

Usage

Instructions for Building the Tool from Source

1. Extract the archive "GDV_Source.zip" to any location on host
2. Open solution GDV.sln using VS2008 and build all
3. Get the 3 executable files from the output directory
4. Get the 7 files from 'RuntimeFiles' directory

Instructions for Installation (zipped)

1. Extract the archive "GDV_Source.zip" to any location on host

Runtime Directory Content

Build output (3):

- GDV.UI.exe - User interface and work flow manager
- BuildDependencies.exe - Pre calculation app
- GDVCalculator.exe - Calculation app

Extra Runtime files (6):

- GraphletDesc.xml - Graphlet descriptions created manually

- xmlValid.exe - Validates XML files according to given schema
- MIF.xsd + MIF1.xsd - **MIF1.xsd** is used as a fix to a mistake in MIF.xsd
- Li2004a.mif - A sample PPI network
- Li2004a.xin - A sample PPI network

Technical Specifications

Software Technology:

- Unmanaged Native C++
  Specifically selected for implementing the "heavy" calculations, achieving better performance.
  Apps: GDVCalculator.exe
  BuildDependencies.exe

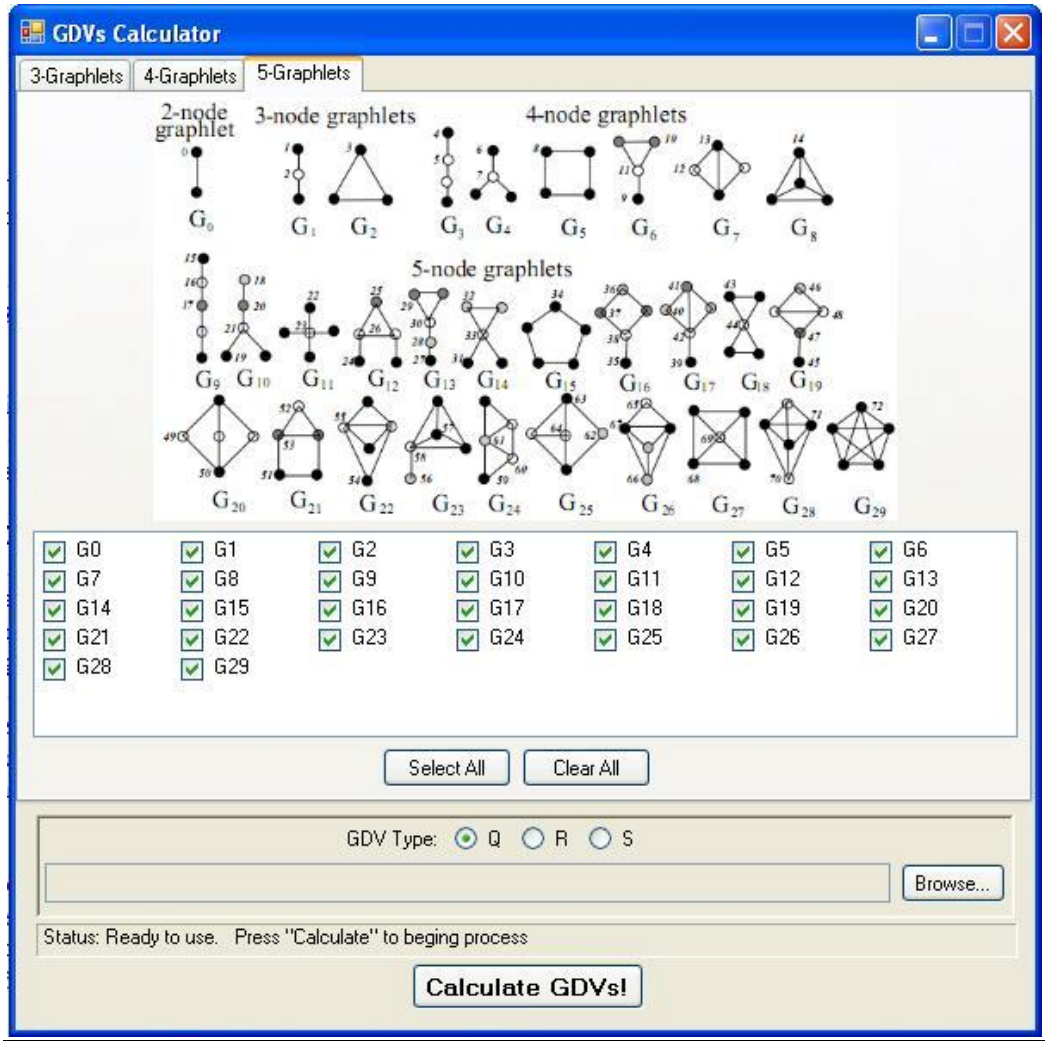
- Managed .NET C#
  Specifically selected for the UI application for providing an friendly intuitive interface which is also easy-to-maintain.
  Apps: GDV.UI.exe

System Requirements:

- Windows OS - Native C++ apps are built using VS C++ Compiler
- .NET Framework 2.0 - GDV.UI app is built in reference to its assemblies

Guidelines for using GUI

- Execute GDV.UI.exe in
- The tabs represent the max graphlet size the user wants to calculate.
- According to the size of the selected tab a list of graphlets is presented with checkboxes so that the user can select which graphlets to be included in the output.
- The user can select the GDV type of the output (Q, R or S).
- The user must browse for the PPI network file (MIF or XIN)
- To start the calculation the user should press the "Calculate GDVs" button.
- In order to abort the calculation before it ends the user can press the "Abort" button (Same button as "Calculate GDVs" when status is during calculation).
- The output file is *GDVout.txt*
- A log file exists, containing progress and error reports on the calculation process, named *GDVCalcLog.txt*



Enhancement Suggestions

- XSDs
  Currently the application checks validity of the PPI input files MIF and XIN according to given XSD. We found an XSD for the MIF format. However we didn't find such XSD for the XIN format thus the application validates the XIN files against an empty XSD.
  As an enhancement, XSD files can be created for the *GraphletDesc.xml* and validated by the application. In addition, if a formal XSD is found for the XIN – it can simply replace the existing empty one.

- Optimization
  We developed the algorithm with emphasis on finding the most efficient way to calculate the task. However, we didn't prove the optimality of our algorithm, and the question which remained open is whether this is really the optimal way or a more efficient algorithm can be developed.

- Threading
  Currently the application executes the calculation in a single thread. Since the complexity of the calculation is relatively "heavy", as an enhancement - calling to GDV_Calc_rec from the Main algorithm for each node can be in a new thread (see Main algorithm the Workflow: section).

- Dynamic Application webpage
  In case this tool is found useful – the main UI may be ported to a web application for common use.

- Potential implementation in other disciplines
  In case GDV concept is found useful in other disciplines such as Financial networks, Computer Networks, Graph Theory Research, Social Networks – the necessary changes are only in XML schemes for network formats and their parsers.

- Combining the applications
  As a technical constraint of the 2 developers – we built the tool in 3 different application. It is however possible to combine them into 1 application with different modules...

Third Party Resources Acknowledgement

We used the following 3rd party components, and here by acknowledge them:

XMLParser
A free, small, simple, cross-platform and fast C++ XML Parser by Dr. Ir. Frank Vanden Berghen.
Refer to: See 'Documents' directory for more details on that component.

Thank you Frank!

Boost Unordered Map
We used Boost free c++ libraries, which is alternative STL, by linking unordered_map lib, which provides a hash table that is capable of efficient string hashing, for cataloguing protein names in a data structure.

That STL is downloaded as part of 'Boost' package, putting the 'Boost' directory directly in directory of the file GDVCalculator.vcproj. Download link: http://www.boost.org/users/download/

It then required adding the following path, for locating included header files from it. Instructions:
project properties -> config properties -> C/C++ -> General -> Additional Include Dirs = "./"

Thank you Boost.

XML Validator
A free tool that validates XML files according to a given XSD.
Refer to: See Project 'General' directory for more details on that component.

Thank you SellsBrothers.

Appendix A: GraphletDesc.xml

The following file should be **manually** created by an application administrator.
Refer to: See Project 'General' directory for more details on that component.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphlets max='5'>

  <!--Graphlets of size 2-->

  <!--For generalization of Graphlet descriptions - 'G0', which are actually every 2
neighbors, is defined as an extension of type 'node' without a specific name under
'sub_graphlet' tag, which typically describes a 'node' extension to an isolated node--
>
  <graphlet name="G0" size='2'>
    <description>
      <sub_graphlet/>
      <extension type="node" anode='0'/>
    </description>
    <roles>
      <role num='0'>
        <vertex node='0'/>
        <vertex node='1'/>
      </role>
    </roles>
    <duplication val='2'/>
  </graphlet>

  <!--Graphlets of size 3-->

  <graphlet name="G1" size='3'>
    <description>
      <sub_graphlet name="G0"/>
      <extension type="node" anode='1'/>
    </description>
    <roles>
      <role num='1'>
        <vertex node='0'/>
        <vertex node='2'/>
      </role>
      <role num='2'>
        <vertex node='1'/>
      </role>
    </roles>
    <duplication val='2'/>
  </graphlet>

  <graphlet name="G2" size='3'>
    <description>
      <sub_graphlet name="G1"/>
      <extension type="edge" anode='0' bnode='2'/>
    </description>
    <roles>
      <role num='3'>
        <vertex node='0'/>
        <vertex node='1'/>
        <vertex node='2'/>
      </role>
    </roles>
    <duplication val='6'/>
  </graphlet>
```

Appendix B: Dependencies.xml

The following file is **automatically** created by BuildDependencies.exe

```
<?xml version="1.0" encoding="utf-8"?>
<dependencies>
	<branching0 head="G0">
		<graphlet_group>
			<graphlet name="G0"/>
		</graphlet_group>
		<branching1 head="G1">
			<graphlet_group>
				<graphlet name="G1"/>
				<graphlet name="G2"/>
			</graphlet_group>
			<branching1 head="G3">
				<graphlet_group>
					<graphlet name="G3"/>
					<graphlet name="G5"/>
					<graphlet name="G6"/>
					<graphlet name="G7"/>
					<graphlet name="G8"/>
				</graphlet_group>
				<branching1 head="G9">
					<graphlet_group>
						<graphlet name="G9"/>
						<graphlet name="G12"/>
						<graphlet name="G13"/>
						<graphlet name="G15"/>
						<graphlet name="G16"/>
						<graphlet name="G17"/>
						<graphlet name="G18"/>
						<graphlet name="G19"/>
						<graphlet name="G20"/>
						<graphlet name="G21"/>
						<graphlet name="G22"/>
						<graphlet name="G23"/>
						<graphlet name="G24"/>
						<graphlet name="G25"/>
						<graphlet name="G26"/>
						<graphlet name="G27"/>
						<graphlet name="G28"/>
						<graphlet name="G29"/>
					</graphlet_group>
				</branching1>
				<branching2 head="G10">
					<graphlet_group>
						<graphlet name="G10"/>
					</graphlet_group>
				</branching2>
			</branching1>
			<branching2 head="G4">
				<graphlet_group>
					<graphlet name="G4"/>
				</graphlet_group>
				<branching1 head="G11">
					<graphlet_group>
						<graphlet name="G11"/>
						<graphlet name="G14"/>
					</graphlet_group>
				</branching1>
			</branching2>
		</branching1>
	</branching0>
</dependencies>
```