ISSN: 3066-8131
Research Article

Science Set Journal of Physics

Basic Network Operations under Spatial Grasp Model

Peter Simon Sapaty

Institute of Mathematical Machines and Systems National Academy of Sciences of Ukraine

*Corresponding author: Peter Simon Sapaty, Institute of Mathematical Machines and Systems National Academy of Sciences of Ukraine.

Submitted: 31 January 2024 **Accepted:** 07 February 2024 **Published:** 14 February 2024

https://doi.org/10.63620/MKSSJP.2024.1035

Citation: *Sapaty, P. S. (2024). Basic Network Operations under Spatial Grasp Model. Sci Set J of Physics, 3(1), 01-16.*

Abstract

The paper analyzes the rapidly growing popularity and importance of the use of graph and network models and tools in many areas, like transport, goods distribution, communications, sociology, economy, defense, security, psychology, and many others. It briefs the developed and patented Spatial Grasp Model and Technology (SGT) which allows for fully distributed and parallel operations on large networked structures, including general technology issues, basic Spatial Grasp Language (SGL), and the networked language implementation. Also investigates and classifies different works on network and graph operations and presents detailed solutions for the most basic network creation, modification, path finding, spanning and shortest path trees, strong and weak components, finding graph structures with spatial pattern matching, also flows in networks. The latter includes the well-known Ford Fulkerson method with an effective solution in SGL, which is much simpler and shorter than traditional implementations in Java and C. The investigated networks can be physical, virtual or combined, having both addresses and physical coordinates. The shown simple and compact solutions in SGL can operate on arbitrary large and complex networks in a highly parallel and fully distributed mode. They were obtained in a direct spatial thinking, spatial pattern recognition and pattern matching mode cultivated by the Spatial Grasp model and its psychology, rather than under traditional logic-based and algorithmic philosophy and culture. This may challenge existing opinions that parallel and distributed algorithms are usually more complex than traditional sequential ones, just proving the opposite, and especially for the network-related applications.

Keywords: Basic Network and Graph Operations, Spatial Grasp Technology, Spatial Grasp Language, Spatial Pattern Matching, Parallel and Distributed Computing, Strong and Weak Components, Spanning Trees, Maximum Flow

Introduction

Networks as models and operations are widely used in practically every area of human activity. We have analyzed in detail and reviewed the following types and areas of networking, with cited main ideas and methods highlighted in the related numerous publications, as follows.

Transport Networks: Transport networks Transportation networks Multiservice Transport Networks Packet Transport Networks Air Transport Networks Geographical routing protocol for vehicular networks [1-6].

Communication Networks: Communication network, Computing in communication networks Optimization and control of communication networks [9]; Introduction to wireless communications and networks Underwater communications and networks Network security 6G mobile wireless networks [7-13].

Social Networks: Definition of social network, Social networking, Social network analytics, Most popular social networking sites, Social and economic networks [14-18].

Battle and Military Networks: Battle Networks A 21st Century battle network the outlined new battle network Challenges of military networks and technological developments the problems of building a safety net [19-23].

Economic Networks: Economic networks an economic network Networks for innovative and resilient economies Understanding economy by networks Cluster networks in economy Economy in the 21st century [24-29].

Distribution Networks: A distribution network A distribution network as the flow of goods Types of distribution networks in supply chain management Benefits of a distribution network Optimal operation of active distribution networks [30-34].

Virtual Networks: A virtual network Virtual networking Network virtualization Pluralistic approach to virtual networks Fundamentals of virtual networking [35-39].

Neural Networks: A neural network Neural network in machine

learning Neural network types and applications Artificial neural network Graph neural networks [40-44].

Psychological Networks: Network analysis in psychological science Visualization of psychological networks Network analysis in psychology Person-specific networks in psychopathology Network analysis and its applications in psychology [45-49].

Criminal Networks: Explanation of criminal networks Understanding of criminal networks in criminology and justice studies New forms of organized crime using networks Criminal networks and law enforcement Detecting and disrupting criminal organizations [50-55].

The main goal of this paper is to investigate and analyze potential and practical suitability and applicability of the developed and patented Spatial Grasp Technology (SGT) and its basic Spatial Grasp Language (SGL), already tested on numerous applications, networking including, for expressing and solving the basic networking problems, especially in large distributed environments [56-71]. The rest of the paper is organized as follows.

Section 2 provides a review of publications on main network and graph operations. Section 3 briefs the developed Spatial Grasp model and technology, including its general issues, Spatial Grasp Language, and its distributed networked interpretation. Section 4 describes elementary networking operations in SGL, including creating individual nodes and links, network topology formation, and nodes and links removal. Section 5 is devoted to paths in networks and describes finding and collecting any and all paths between certain nodes. Section 6 describes creation of different types of spanning trees from a node, like breadth-first and depth-first, also shortest path tree to all other nodes. Section 7 describes finding different components in networks, like strongest sub- networks or cliques and weakest or articulation points. Section 8 provides finding matching solutions for arbitrary patterns based on a path through their all nodes or all links. Section 9 is devoted to managing flows in networks including package delivery to the given destinations which may be via arbitrary routes or by using shortest path tree; it also describes effective and extremely compact implementation in SGL of the well-known Ford Fulkerson method for finding maximum flow in a graph and its comparison with existing Java solution. Section 10 concludes the paper confirming effectiveness of SGT for basic network operations, and References cite many publication sources analyzed in the paper. The Appendix provides updated Summary on SGL syntax and its main constructs used in all presented solutions of different tasks.

A Review of Publications on Main Network and Graph Operations

The following sources were analyzed and investigated in detail for drafting the repertoire of main networking operations which may be useful, even critical, for their usage within areas described in the previous chapter.

An Introduction to Networks is in [72]. A network is simply a collection of connected objects. We refer to the objects as nodes or vertices, and usually draw them as points. We refer to the connections between the nodes as edges, and usually draw them as lines between points. Networks can represent all sorts of systems in the real world. For example, one could describe the Internet as a network where the nodes are computers or other devices and the edges are physical (or wireless, even) connections between the devices. The World Wide Web is a huge network where the pages are nodes and links is the edges. Other examples include social networks of acquaintances or other types of interactions, networks of publications linked by citations, transportation networks, metabolic networks, and communication networks. You can click on the following images for more information about their respective networks.

The Degree Distribution of a Network is in [73]. A network can be an exceedingly complex structure, as the connections among the nodes can exhibit complicated patterns. One challenge in studying complex networks is to develop simplified measures that capture some elements of the structure in an understandable way. One such simplification is to ignore any patterns among different nodes, but just look at each node separately. If one zooms in onto a node and ignores all other nodes, the only thing one can see is how many connections the node has, i.e., the degree of the node. Node.

Random Networks are in [74]. When analyzing a network, one approach is look at the network as a single fixed entity. But sometimes, it is useful to think of the edges as random variables. With this random network perspective, a given network is more than a single object. Instead, we can view the random network as a sample from a probability distribution. We can then study the whole probability distribution to gain insight into the network. We can view the probability distribution of random networks as defining a whole ensemble of many different networks, where the probability of any particular network is determined by the probability distribution. Rather than looking at the particular properties of a single network, we can study the properties of the whole ensemble of networks.

Network Algorithms are in [75]. Network structures characterize how networks "look" – large or small diameter, number of edges, sparse or dense, degree distributions, clustering, etc. The distance between two vertices is the length of the shortest path connecting them. This assumes the network has only a single connected component. If two vertices are in different components, their distance is infinite. The diameter of a network is the maximum distance between a pair of vertices in the network. Eccentricity as the length of the longest shortest path.

10 Fundamental Network Algorithms are discussed in [76]. This looks at the algorithms used to perform network calculations. It starts with some simple algorithms for calculating quantities such as degrees, degree distributions, and clustering before moving on to more sophisticated algorithms for shortest paths, betweenness, maximum flows, and other non-local quantities. Exercises are provided at the end of the chapter.

Graph and Network Algorithms are in [77]. Graphs model the connections in a network and are widely applicable to a variety of physical, biological, and information systems. You can use graphs to model the neurons in a brain, the flight patterns of an airline, and much more. The structure of a graph is comprised of "nodes" and "edges". Each node represents an entity, and each edge represents a connection between two nodes.

Distributed Network Algorithms are in [78]. Distributed network algorithms play a major role in many networked systems, ranging from computer networks (such as sensor networks, peer-to-peer networks, software- defined networks, datacenter networks, networks on chip) to social or even biological networks. The goal of this course is to introduce the fundamental formal models and methods needed to reason about the correctness and performance of distributed network algorithms.

Data Structures and Network Algorithms are in [79]. There has been an explosive growth in the field of combinatorial algorithms. These algorithms depend not only on results in combinatorics and especially in graph theory, but also on the development of new data structures and new techniques for analyzing algorithms. Data Structures and Network Algorithms attempts to provide the reader with both a practical understanding of the algorithms, described to facilitate their easy implementation, and an appreciation of the depth and beauty of the field of graph algorithms.

What is Graph in Data Structure and Types of Graph are in [80]. Graphs in data structures are non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them. Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks, and social networks. For example, it can represent a single user as nodes or vertices in a telephone network, while the link between them via telephone represents edges.

Graph Operations are discussed in [81]. Graphs are often a good representation for problems involving objects and their relationships because there are standard graph operations that can be used to answer useful questions about those relationships. Here we discuss two such operations: depth-first search and breadth-first search, and some of their applications. Both depth-first and breadth-first search are "orderly" ways to traverse the nodes and edges of a graph that are reachable from some starting node. The main difference between depth-first and breadth-first search is the order in which nodes are visited.

Basic Graph Theory is discussed in [82]. This considers such features as: undirected and directed graphs, neighborhood and degree, density and average degree, paths, loops, paths in directed graphs, length, distance, network structure, connected graphs and connected components, hubs, clusters, network dynamics, random graph algorithm, clusters and homophily, triadic closure algorithm, hubs and cumulative advantage, preferential attachment algorithm, and many others.

Graph Data Structure is explained in [83]. Graphs are data structures that consist of nodes or vertices linked by edges. Graphs are used to depict relationships and links between diverse parts, making it possible to simulate and evaluate complicated systems more efficiently. Data structures are specialized formats that we use to store, process, and retrieve data according to our needs. There are several data structures that serve different purposes depending on the requirements. Some basic operations in a graph are considered like: add/remove vertex or edge, breadth-first search, and depth-first search. Different types of graphs also discussed, like: finite, infinite, trivial, simple, multi, null, complete, pseudo, regular, bipartite, labelled, direct, connected, disconnected, cyclic, and acyclic. Also, representation of graphs by adjacency matrix or list.

Ford-Fulkerson Algorithm for Maximum flow Problem is provided in [84]. In Graph Theory, maximum flow is the maximum amount of flow that can flow from source node to sink node in a given flow network. The Ford-Fulkerson algorithm is used to detect maximum flow from start vertex to sink vertex in a given graph. In this graph, every edge has the capacity. Flow can be defined as any physical/virtual thing that we can transmit between two vertices through an edge in a graph, for example - water, current, data and in this case "trains". Assuming steady conditions (Number of trains on a railway track at any instance of time must not exceeds the

track's capacity), if we need to find what is the maximum number of trains (maximum flow) that we can send from City A to city B then the problem is known as the "Maximum Flow Problem".

As the summary on the analyzed various networking sources, we collected the number of networking features particularly suitable for the use of SGT and SGL for their expression, analysis and processing, which include the following: A directed network; An undirected network; Different types of nodes and edges; Different weights of edges and nodes; Degree of the node; Diameter of a network; Eccentricity as the length of the longest shortest path; Clustering; Betweenness; Breadth-first graph search; Depth-first graph search; Shortest path between two single nodes; Shortest path tree from node; All paths between two graph nodes; Maximum flow in graph; All cycles in graph; Small-world networks; Network partition; Heaps; Linking and cutting trees; Minimum spanning trees; Network flows; Matchings; Connected graphs; Connected components; Hubs [1-84].

A number of these and other issues have been already investigated and implemented using SGT, which can be found in the existing publications [56-71]. But some very basic and fundamental features, including new networking solutions, are provided and explained in detail in the following chapters.

The Spatial Grasp Model and Technology

Only most general features of the developed paradigm are included, with availability of existing extended publications on its philosophy, features, organization, and numerous applications, some in [56-71].

- **General issues**

Within Spatial Grasp Model and Technology, a high-level operational scenario expressed in recursive Spatial Grasp Language (SGL), starting in any world point, as in Fig.1, propagates, covers, and matches the distributed environment in parallel wave-like mode. Such propagation can result in echoing of the reached states and data, which may be arbitrarily remote, to be represented as the final result, or used for higher level decisions and launching other waves. These capabilities altogether provide holistic spatial solutions unachievable by any other models and

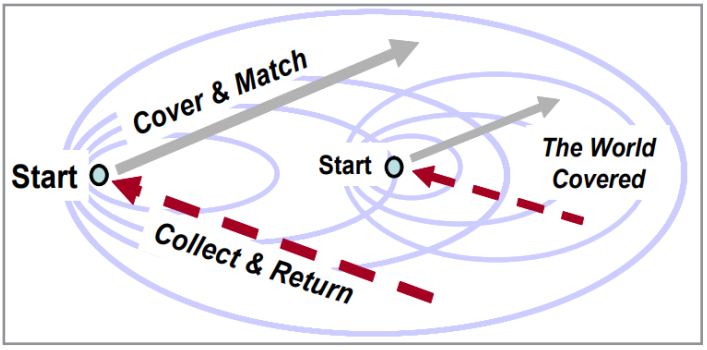


Figure 1: Parallel Recursive Covering, Collecting, Returning, and Analyzing Knowledge with Spatial Grasp Model

This concept is based on quite different philosophy of dealing with large distributed systems. Instead of representing systems and solutions in them in the form of communicating parts or agents, the developed Spatial Grasp paradigm is organizing everything by the integral, holistic and parallel substance covering and conquering distributed worlds which may be as follows:

Physical World (PW), considered as continuous and infinite where each point can be identified and accessed by physical coordinates,

Virtual World (VW), which is discrete and consists of nodes and semantic links between them,

Executive World (EW), consisting of active "doers", which may be humans or robots, with communication possibilities between them.

Different kinds of ***combinations of these worlds*** can also be possible within the same formalism.

- **Spatial Grasp Language (SGL)**

The SGL (with many details in) allows for organizing direct space presence and operations with unlimited powers and parallelism. Its universal recursive organization with operational scenarios called grasp can be expressed just by a single formula [56-71].

Grasp → Constant | Variable | Rule ({grasp,})

Where SGL rule expresses certain action, control, description or context accompanied with operands, which can themselves be any grasps too. Top SGL details can be expressed as:

constant → information | matter | custom | special
variable → global | heritable | frontal | nodal | environmental
rule → type | usage | movement | creation | echoing | verification assignment | advancement | branching | transference | exchange timing | qualifying

The rules, starting in some world points, can organize navigation of the world sequentially, in parallel, or any combinations thereof. They can result in staying in the same application point or can cause movement to other world points with obtained results to be left there, as in the rule's final points. Such results can also be collected, processed, and returned to the rule's starting point, the latter serving as the final one on this rule. The final world points reached after the rule invocation can themselves become starting ones for other rules. The rules, due to recursive language organization, can form arbitrary operational and control infrastructures expressing any sequential, parallel, hierarchical, centralized, localized, mixed and up to fully decentralized and distributed algorithms. Details of the latest SGL version are summarized in Appendix.

- **SGL Interpreter Organization**

The interpreter consists of a number of specialized functional modules working with specific data structures, both serving SGL scenarios or their parts which happen to be inside this interpreter at this moment of time, also organizing exchanges with other interpreters in case of distributed SGL scenarios. Each SGL interpreter copy can handle and process multiple active SGL scenario code propagating in space and between the interpreters. Communicating interpreters of SGL can be in arbitrary number of copies, up to millions and billions, which can be effectively integrated with any existing systems and communications, and their dynamic networks can represent powerful spatial engines capable of solving any problems in terrestrial and celestial environments. Such collective engines can simultaneously execute different cooperative or competitive scenarios without any central resources or control. Hardware or software SGL interpreters, shown in Fig. 2 as universal computational, control and management units may be stationary or mobile. They can be dynamically installed, and if needed created, in proper physical or virtual world points on the request of self-evolving SGL scenarios.

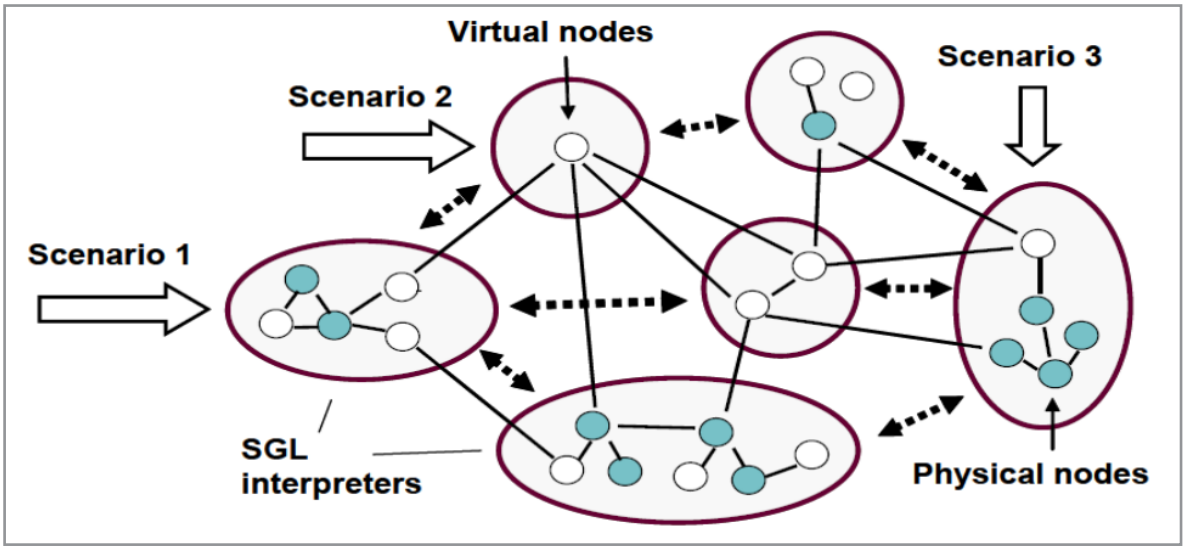


Figure 2: SGL Distributed Networked Interpretation

As both backbone and nerve system of the distributed interpreter, its self-optimizing **Spatial Track System** provides hierarchical command and control, as well as remote data and code access, it also supports spatial

variables and merges distributed control states for decisions at higher organizational levels. The track infrastructure is automatically distributed between active components (humans, robots, computers, smart-phones, satellites, etc.) during scenario self-spreading in distributed environments.

Elementary Networking Operations in SGL

- **Creating Individual Nodes, Links**

Create isolated virtual node with some name a (see Fig, 3, a):
create_node(a)

The physical location, size, and shape of such node can be added as follows, after hoping into the already created node and staying in it:
hop_node(a); WHERE = X_Y, SIZE = big; SHAPE = star

These full details can also be set up during the node creation:
create_node(NAME:a, WHERE:X_Y; SIZE:big, SHAPE:star)

Adding to the single node a link named r into another node b, to be created too, will be as (see Fig, 3, b):hop(a); create(link(r), node(b));

Explicitly adding the created link details, when staying the node, it leads to, after passing the link, will be as follows:
create(link(r), node(b)); ORIENT = +, WEIGHT = 10; LENTH = 50

Or providing the full link details during its creation from the starting node:
hop(a); create (link (LINK:r, ORIENT:+, WEIGHT:10, LENGTH:50), node(b))

Full details of both nodes and link connecting them can be set up as follows (see Fig. 3, c): create_node (NAME:a, WHERE:X1_Y1, SIZE:big, SHAPE:star); create(link(LINK:r, ORIENT:+, WEIGHT:10, LENGTH:50),
node (NAME:b, WHERE:X2_Y2, SIZE:small, SHAPE:triangle))

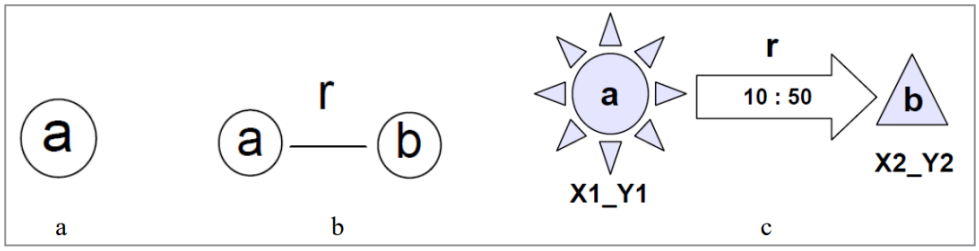


Figure 3: Representation of network nodes and links

- **Network Creation**

Crating tree-like network, as in Fig. 4, a:
create(node(a);
((link(r), node(b)); (link(r), nodes (d, e))),
((link(r), node(c)); (link(r), nodes (f, g))))

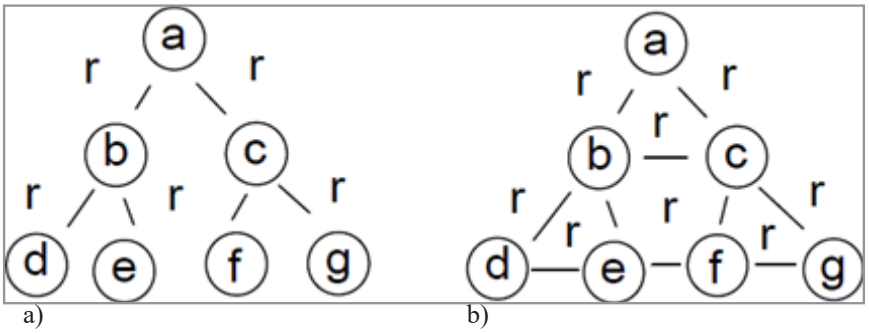


Figure 4: Tree-Like and Arbitrary Network Topologies

Creating Arbitrary Network, as in Fig, 4, b:
Top = (a:(b,c), b:(a,c,d,e), c:(a,b,f,g), d:(b,e),
e:(b, d,f), f:(e,c,g), g:(c,f));
align (split (Top); frontal (Next) = VAL [2]; create (VAL [1])); hop (Next); NAME > BEFORE; linkup (r, BEFORE)

- **Nodes and Links Removal**

Remove node b of Fig. 4, b, see the result in Fig. 5, a:
remove(hop_node(c))

Remove all links associated with nodes c and d of Fig. 4, b, with the result in Fig. 5, b.
hop_nodes(c, d); remove(all_links)

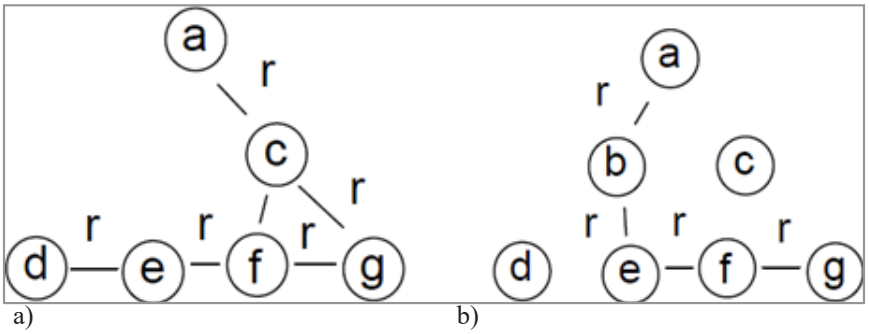


Figure 5: Removing nodes and links from the network

Finding Paths in Networks

- **Finding and Collecting any Path Between Certain Nodes**

Having created a network infrastructure, like the one as in Fig. 4, b, we may, starting in some node liked, reach any other node like c, by the following SGL scenario navigating the network in wavelike mode, i.e. stepwise, in parallel, and without cycles (see Fig. 6 with different solutions for this task).
hopfirst(d); repeat(hopfirst(all_links);
if (NAME == c, done))

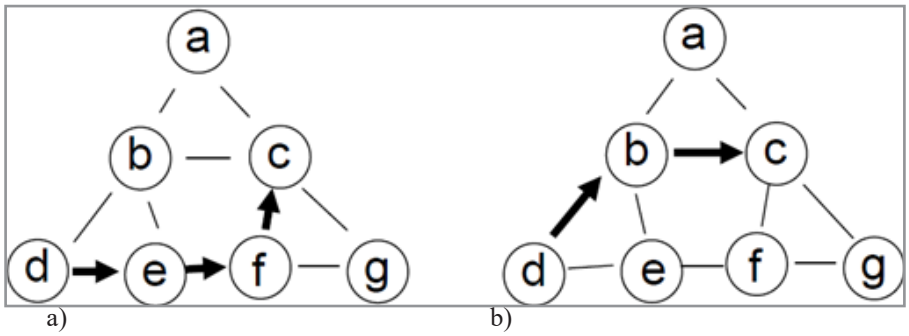


Figure 6: Reaching a node from Another Node

We may additionally decide to collect the past path and organize its output at the destination node c, as follows.
hopfirst(d); frontal (Path = NAME); repeat(hopfirst(all_links); Path &&= NAME;
if (NAME == c, done_output(Path)))

This will give (d,e,f,c) for Fig, 6,a, and (d,b,c) for Fig. 6,b. Such paths can also be issued in the starting node d by the modified scenario:
hopfirst(d); frontal (Path = NAME); output_repeat(hopfirst(all_links); Path &&= NAME;
if(NAME == c, blind(Path)))

- **Finding and Collecting all Paths Between Certain Nodes in the Network**

It is easy to find any paths in networks in SGL. For example, all simple paths from node d to node c (i.e. with non-repeating nodes) can be easily found from Fig. 4, b by:
frontal (Path); hop(d);
repeat (append (Path, NAME); if (NAME == c, done_output(Path)); hop(all_links); notbelong(NAME, Path))

If we apply this scenario to the network of Fig. 4, b, will receive the following paths:
(d,b,c), (d,b,a,c), (d,e,b,a,c), (d,e,b,c), (d,e,f,c), (d,e,f,g,c)

Spanning Trees

- **Breadth-First Spanning tree (BST) from a Node to all other Nodes**

In one of the previous examples, we have found a possible path from a node to some particular node, which was then issued outside the network within the network asynchronous wavelike navigation. In a similar network navigation, we may create a Breadth-first Spanning Tree (BST) starting in the same start node and covering the whole network, thus explicitly showing possible paths to all other nodes. This can be easily done with registering this BST directly in the distributed network structure, by remembering predecessor node names in a special variable Up associated with each node, as follows, see also Fig. 7, a,b for possible BST solutions:

nodal (Up); hopfirst(d);
repeat(hopfirst(links_all)); Up = BACK)

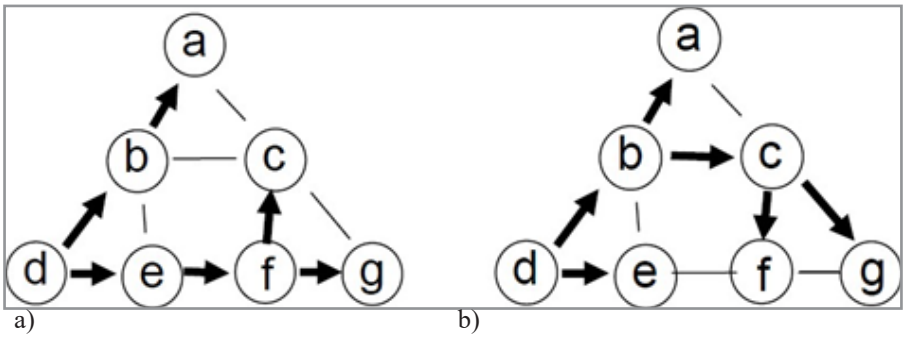


Figure 7: Creating any Spanning tree from Start d to all other nodes

Such a tree Can Directly Guide Movement from any Node, like c, to the Starting Node d:
hop(c); repeat (hop (Up))

We may also collect this passed path as follows.
hop(c); Path = NAME;
repeat (hop (Up); Path = NAME && Path); output (Path)

Results: (d,e,f,c) for Fig. 7,a and(d,b,c) for Fig. 7,b.

- **Depth-first Spanning tree (DST) from a Node to all Other Nodes**

See Fig. 8, a,b for possible DST solutions, with starting from node d:

DST = {sequence(hopfirst(links_all); Up = BACK; run (DST))}; hopfirst(d); run (DST)

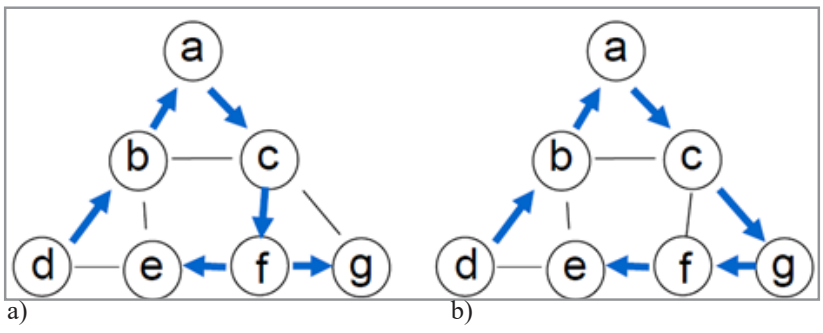


Figure 8: Possible Depth-first Spanning trees from Start d to all Other Nodes

Collecting path from Start d to any Other Node, Say, e:
hop(e); Path = NAME;
repeat (hop (Up); Path = NAME && Path); output (Path)

Result: (d,b,a,c,f,e) for Fig. 8,a, or(d,b,a,c,g,f,e) for Fig. 8,b.

- **Shortest Path tree (SPT) from a Node to all Other Nodes**

With some modification, we may create and register instead of any spanning tree, a Shortest Path Tree (SPT) from a node to all other nodes (which, in general, may have more than a single solution, as shown in Fig. 9, a,b)

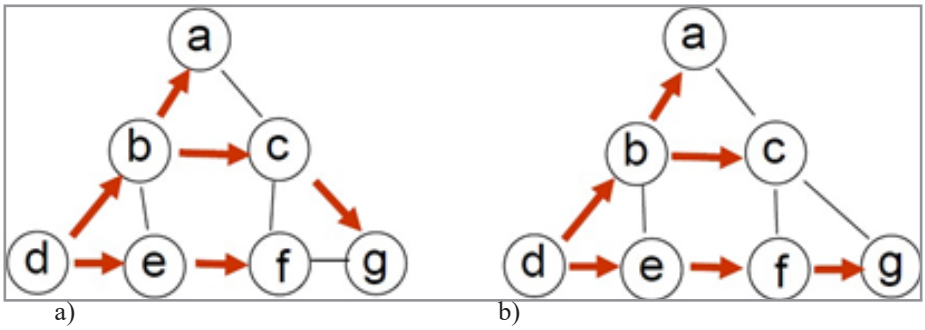


Figure 9: Creating Shortest Path Tree from Start d to all Other Nodes

The following scenario can accomplish this with registering the resultant SPT in the network structure similarly to the spanning tree scenario before. The main difference will be with re-registering the SPT predecessor nodes in variables Up if shorter solutions from the start to these nodes appear available.
nodal (Dist, Up); hop(d); Dist = 0; frontal (Far); repeat (
hop(links_all); Far += 1;

or (Dist == nil, Dist > Far); Dist = Far; Up = BACK)

SPT will be embedded into the network structure with Dist and Up in all nodes. Starting in any node, say c, and following Up you can easily receive SP to it from the head of the SPT, similar to BST above:
hop(c); Spath = NAME;
repeat (hop (Up); Path = NAME && Path); output (Spath)

Result of SP from d to c: (d,b,c) for both SPT in Fig. 9.

Finding Different Components in Networks

- **Finding Strongest Sub-Networks, or Cliques**

We present here a universal solution in SGL for finding all cliques in a network (i.e. maximum possible full subgraphs), assuming their number of nodes should correspond to some threshold. The following scenario is finding all cliques in parallel in the network of Fig. 10 with the number of nodes not less than three (only one clique, with four nodes, is highlighted in the figure).

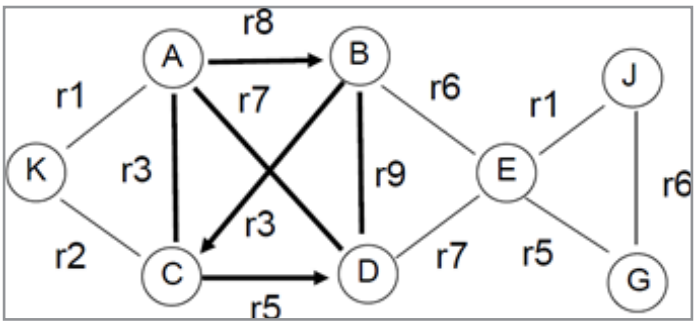


Figure 10: Finding Strongest Parts, or Cliques, in the Network

hop_nodes(all); frontal (Clique) = NAME; repeat (
hop_links(all); not_belong(NAME, Clique); yes(and_parallel(hop(links_any, nodes(Clique)))); if(PREDECESSOR > NAME, append(Clique, NAME), blind));
count (Clique) >= 3; output (Clique)

This scenario, starting in all nodes and following their links to other nodes in parallel, is collecting node names in new individual hops which have links with all previously collected nodes unless such nodes cannot be found, declaring the collected set of node names in the frontal variable Clique as a new clique. As the same clique can be collected in such a way when started from all nodes of the same clique, the duplicates are blocked by allowing inclusion of any new node into the clique's list only if the value of its name is lower than of the previous one. The obtained full results for Fig. 10 will be as:
(A, B, C, D), (A, C, K), (B, D, E), (E, G, J)

If names of links in cliques are of interest too, the modified scenario will look like: hop_nodes(all); frontal (Clique = NAME, Links); repeat (
hop_links(all); not_belong(NAME, Clique); yes(and_parallel(hop(links(any), nodes(Clique)))); if(PREDECESSOR > NAME, (append (Clique, NAME);
append (Links, (hop (links_any, nodes (Clique)); LINK)), blind));
count (Clique) >= 3; output (unit (Clique, Links))

The obtained results will be as follows:
(A, B, C, D, r8, r3, r3, r5, r7, r9), (A, C, K, r3, r2, r1),
(B, D, E, r9, r7, r6), (E, G, J, r5, r6, r1)

- **Discovering Weakest, or Articulation Points**

Weakest or articulation points when removed split the network into disjoint parts, like node E in Fig. 11.

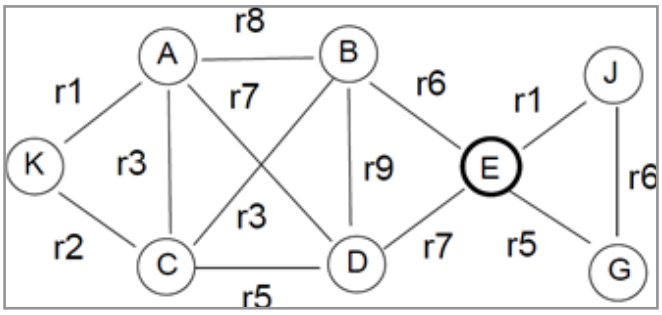


Figure 11: Weakest or Articulation Points of the Network

Next is parallel and fully distributed solution for finding all articulation points in the network, by which each network node, first selecting one neighbor randomly, tries to navigate and mark the whole network from it while excluding itself from this process. After termination, if the node discovers still unmarked neighbors, it declares itself articulation point and outputs its name, as follows:
hop_nodes(all); IDENTITY = NAME; hopfirst_node(current); stay(hopfirst_random(links_all);
repeat(hopfirst(links_all))); if(hopfirst(links_all), output (NAME))

The answer will be E, as the only point of such type in the network of Fig. 11.

Pattern Matching

- **Arbitrary Network Matching Patterns**

In distributed networks, like shown in Fig. 12, a, any graph structures can be found with the use of SGT, for example, as the one shown in Fig. 12, b. Applied in parallel from all network nodes, it can then self-evolve from them in parallel matching mode with finding meanings of variables N1 to N4. There may be different possibilities of how to represent arbitrary search patterns as matching templates in SGL.

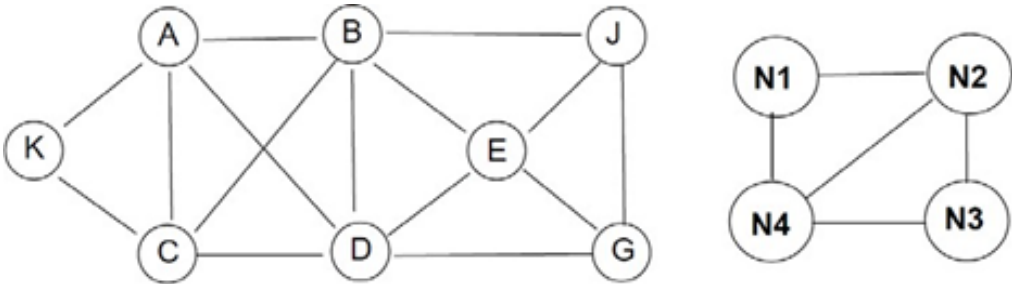
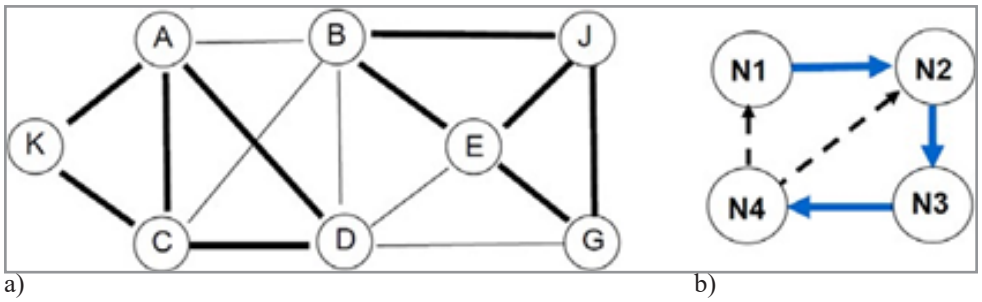


Figure 12: A Network and its Search Pattern

- **Template Representation Based on a Path Through all Nodes**

Among simplest matching template representations may be the one based on a path through all pattern's nodes, as shown in Fig. 13, b (for more complex cases such a path may have to include some nodes more than once). The SGL self-matching scenario based on the oriented path of Fig. 13, b will be as follows.

hop_nodes(all); frontal (Nodes) = NAME;
three_repeat(hop_link(all); notbelong(NAME, Nodes); Nodes && = NAME); true(andparallel(link(any), Nodes[1,2]));
output (Nodes)



a) b)

Figure 13: Matching with Template Based on a path Through all Nodes

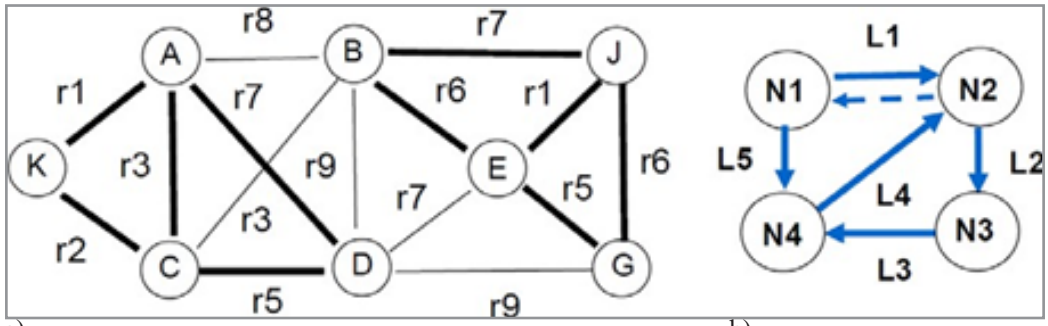
The nodes of found substructures corresponding to variables N1 to N4 will be as follows (also considering that the pattern may start from different nodes of the same network structures, with only two solutions highlighted in Fig. 13, a):

(N1, N2, N3, N4) →(K, A, D, C), (B, J, G, E), (A, B, E, D), (C, B, E, D), ...

The output of contents of variables found will be accomplished in nodes corresponding to variable N4, which will be C, E, D, D and others, in another matching.

- **Template Representation Based on a Path Through all Links**

Another simple template representation in SGL, fully universal too, can be based on a path through all template links, as in Fig. 14, b (with numbers of variables on links corresponding to the order of their passing), where in general some links may have to be passed more than once. Such template representation is especially convenient for the joint collection of the matched node and link names, as shown in Fig. 14, a,b, with variables Ni for nodes, and Li for links. The corresponding SGL scenario for finding and issuing values of all these variables in the successful matches will be as follows.



a) b)

Figure 14: Matching with Template Based on a Path Through all Links

hop_nodes(all); frontal (Nodes = NAME, Links); three_repeat(hop_link(all); notbelong(NAME, Nodes);
Links &&= LINK; Nodes &&= NAME); hop(link(all), Nodes [2]); Links &&= LINK; hop (Links [1], Nodes [1]);
hop(link(any), Nodes [4]); Links &&= LINK; output (Nodes, ":", Links)

Some of the solutions with variables at both nodes and links will be as follows (issued in nodes corresponding to variable N4, same as in previous example of Fig. 13):
(N1, N2, N3, N4) :(L1, L2, L3, L4, L5) →
(K, A, D, C) :(r1, r7, r5, r3, r2), (B, J, G, E) :(r7, r6, r5, r1, r6), ...

Deliveries and Flows in Networks

- **Package Delivery to the Given Destinations**

Reaching from Start node the set of particular Fin nodes (using any simple casual paths to them, i.e. without loops) and delivering a sort of Pack to them (like attaching it to their CONTENT), may be expressed as follows, see also Fig. 15.
frontal (Pack = …, Fin = (E, G, N, O, P)); nodal (Start = K);
hopfirst(Start);
repeat (if (belong (NAME, Fin), CONTENT &&= Pack); hopfirst(links_any))

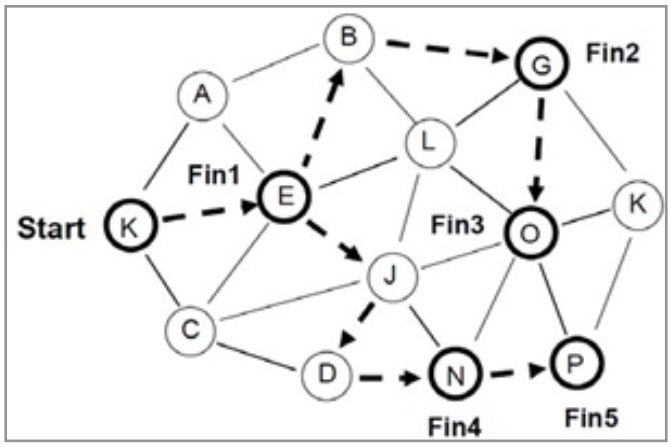


Figure 15: Possible Delivery Paths from Start to Certain Destinations

With not optimal casual delivery paths of this case, there may be inefficiency and redundancy of the network usage, also delays with deliveries.

- **Optimal Package Delivery Using SPT**

We may first find shortest path tree of this network, as in Fig 16, a, originating from Start, and then use only paths in it leading to the Fin nodes (as in Fig. 16, b), altogether forming the best delivery infrastructure from Start to Fin (some of its nodes may happen to be on the same shortest paths from Start). The related delivery scenario may be as follows.

```
nodal (Dist, Up, Up2, Fin = (E, G, N, O, P)); frontal (Far);
nodal (Start) = K; hop (Start);
sequence (repeat(hop(links(all)); Far += 1;
        or (Dist == nil, Dist > Far); Dist = Far; Up = BACK),
 (hop (Fin); repeat (Up2 = Up; hop (Up))),
(frontal (Pack = delivery);
repeat(hop(all_links); Up2 == BACK;
        if (belong (NAME, Fin), append (Pack, CONTENT)))))
```

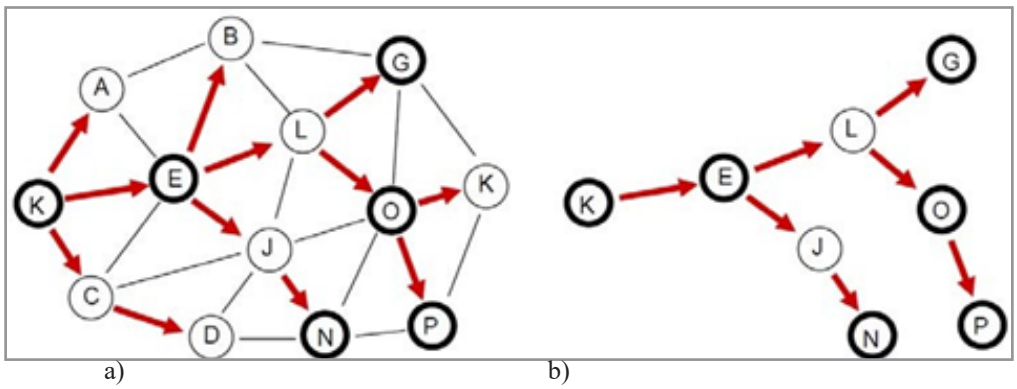


Figure 16: Forming Shortest Path Delivery Infrastructure to Certain Destinations

- **Maximum Flow in a Graph**

In graph theory, maximum flow is the maximum amount that can flow from source node to sink node in a given flow network. The Ford-Fulkerson algorithm [13]. is used to detect maximum flow from start vertex to sink vertex in a given graph, where every edge has certain capacity, as in Fig. 17 (with initial flow shown as zeros). In simple terms, Ford-Fulkerson Algorithm is: As long as there is a path from source A to sink F with available capacity on all its edges, we may send the possible flow from that path and find another path, and so on. Flow in the network has the following restrictions: Input flow must match to output flow for each node in the graph, except the source and sink node; Flow out from source node must match with the flow in to sink node. Flow from each edge should not exceed the capacity of that node. The following SGL solution is based on the algorithm described in [13].

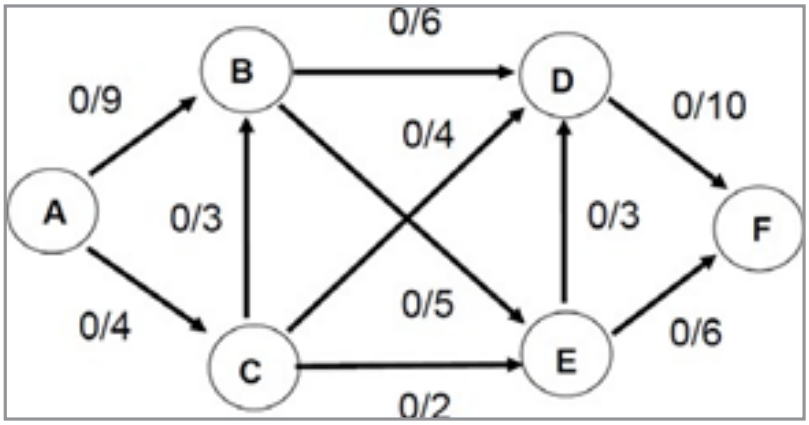


Figure 17: Initial Network State for Finding Maximum Flow

We start with creation of the network of Fig. 17:
Top = (A:(9: B,4:C), B:(6:D,5: E), C:(3: B,4:D,2: E), D:(10: F), E:(3:D,6: F), F);

align (split (Top); frontal (Next) = VAL [2]; create (VAL [1])); split (Next); linkup (+VAL [1], VAL [2]); THRU = 0

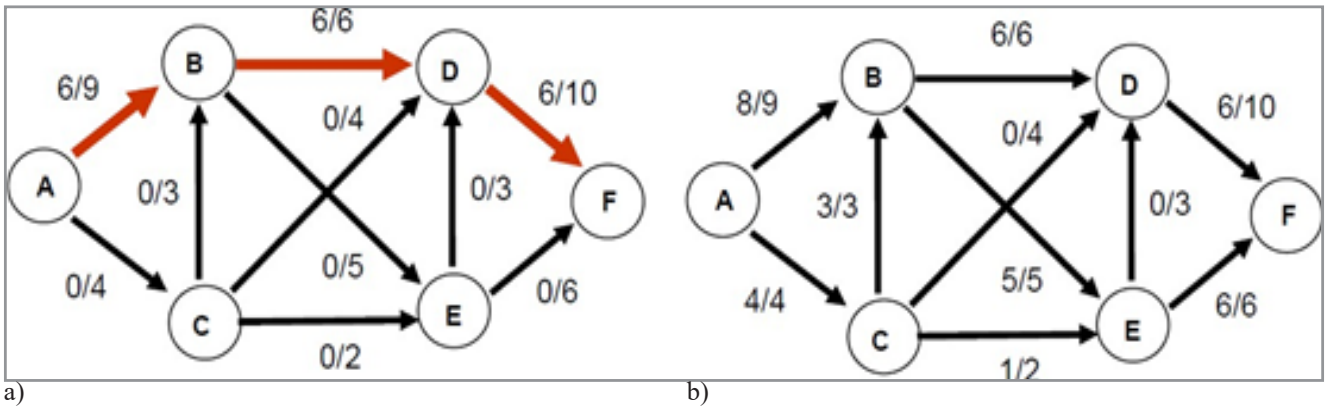
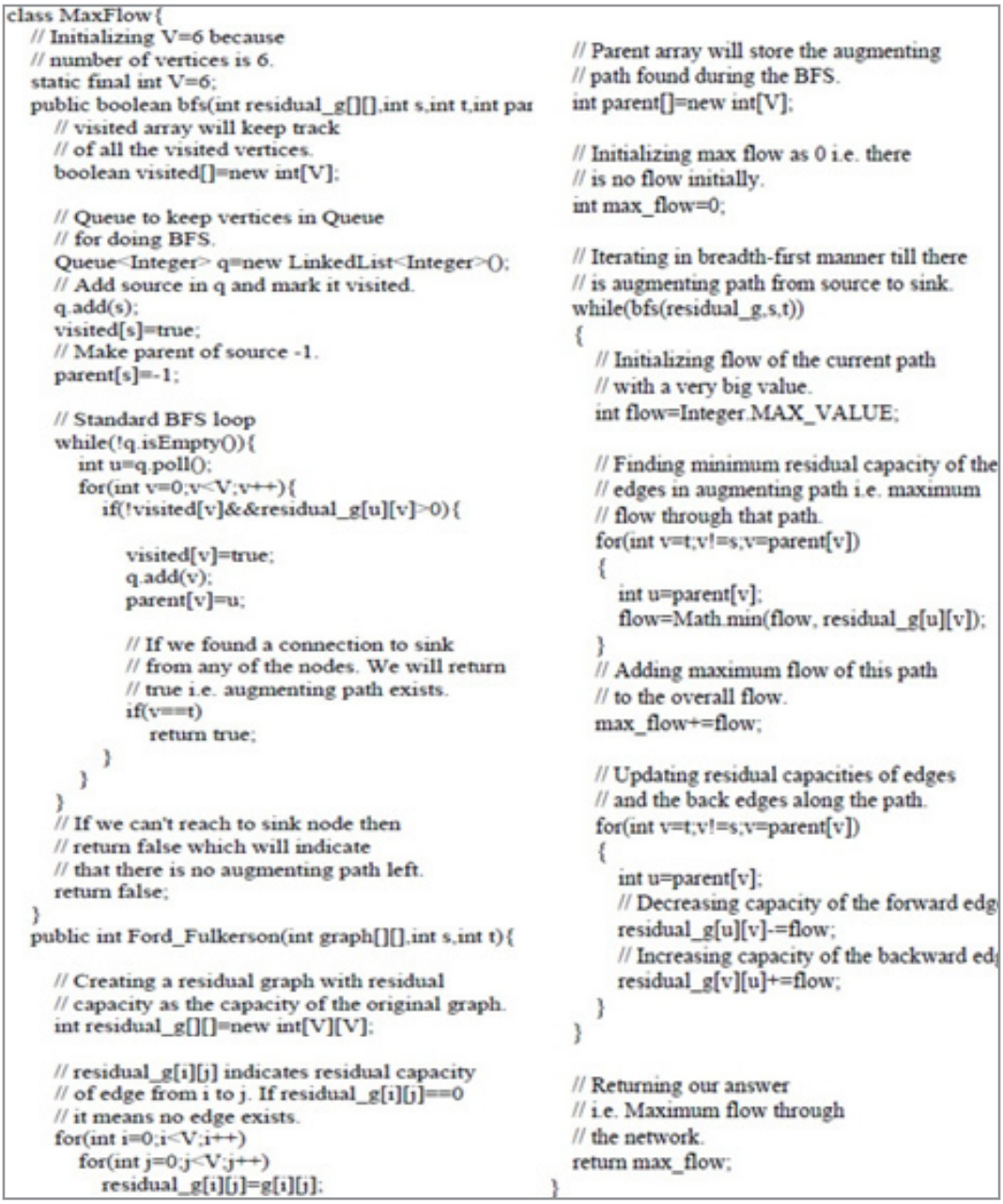


Figure 18: Intermediate and Final Maximum Flow Processing

The following scenario is incrementally finding maximum flow in the network of Fig. 17, each time selecting a single path from starting node A to final one F (from all possible paths traversed in parallel) through which the flow can still be increased, like the one in Fig. 18, a, with amendment the flow through this path, until this remains possible. One of possible final solutions is shown in Fig. 18, b.

```
frontal (Flow = infin, Path); Proc =
{repeat (
hop(+all_links); notbelong (NAME, Path);
Flow = min (LINK - THRU, Flow); Flow! = 0; append (NAME, Path); if (NAME == T,
done_global (Result += Flow;
repeat (move (-link, withdraw (Path)); THRU += Flow))))};
hop(A);
repeat (if (run (Proc), stay, done(hop(T); output (Result))))
```

This maximum flow implementation in SGL is much clearer and more compact than in Java (its solution copied in Fig. 19) or different versions of C [13], as fundamentally based on direct spatial presence, thinking, and pattern matching provided by Spatial Grasp Model and Technology, see for example [56, 59, 60, 62, 71].



```
class MaxFlow{
   // Initializing V=6 because
   // number of vertices is 6.
   static final int V=6;
   public boolean bfs(int residual_g[][],int s,int t,int par
      // visited array will keep track
      // of all the visited vertices.
      boolean visited[]=new int[V];

      // Queue to keep vertices in Queue
      // for doing BFS.
      Queue<Integer> q=new LinkedList<Integer>();
      // Add source in q and mark it visited.
      q.add(s);
      visited[s]=true;
      // Make parent of source -1.
      parent[s]=-1;

      // Standard BFS loop
      while(!q.isEmpty()){
         int u=q.poll();
         for(int v=0;v<V;v++){
            if(!visited[v]&&residual_g[u][v]>0){

               visited[v]=true;
               q.add(v);
               parent[v]=u;

               // If we found a connection to sink
               // from any of the nodes. We will return
               // true i.e. augmenting path exists.
               if(v==t)
                  return true;
            }
         }
      }
      // If we can't reach to sink node then
      // return false which will indicate
      // that there is no augmenting path left.
      return false;
   }
   public int Ford_Fulkerson(int graph[][],int s,int t){

      // Creating a residual graph with residual
      // capacity as the capacity of the original graph.
      int residual_g[][]=new int[V][V];

      // residual_g[i][j] indicates residual capacity
      // of edge from i to j. If residual_g[i][j]==0
      // it means no edge exists.
      for(int i=0;i<V;i++)
         for(int j=0;j<V;j++)
            residual_g[i][j]=g[i][j];

      // Parent array will store the augmenting
      // path found during the BFS.
      int parent[]=new int[V];

      // Initializing max flow as 0 i.e. there
      // is no flow initially.
      int max_flow=0;

      // Iterating in breadth-first manner till there
      // is augmenting path from source to sink.
      while(bfs(residual_g,s,t))
      {
         // Initializing flow of the current path
         // with a very big value.
         int flow=Integer.MAX_VALUE;

         // Finding minimum residual capacity of the
         // edges in augmenting path i.e. maximum
         // flow through that path.
         for(int v=t;v!=s;v=parent[v])
         {
            int u=parent[v];
            flow=Math.min(flow, residual_g[u][v]);
         }
         // Adding maximum flow of this path
         // to the overall flow.
         max_flow+=flow;

         // Updating residual capacities of edges
         // and the back edges along the path.
         for(int v=t;v!=s;v=parent[v])
         {
            int u=parent[v];
            // Decreasing capacity of the forward edg
            residual_g[u][v]-=flow;
            // Increasing capacity of the backward ed
            residual_g[v][u]+=flow;
         }
      }

      // Returning our answer
      // i.e. Maximum flow through
      // the network.
      return max_flow;
   }
```

Figure 19: Java Version of Ford-Fulkerson Method

Conclusions

The results of this work confirm availability and efficiency of using SGT and SGL for investigating, expressing, and implementing most basic network and graph creation and management issues. SGL allows us to provide detailed, very clear and extremely compact solutions for the network problems which can operate on arbitrary large and complex networks, and in highly parallel and fully distributed mode. The solutions provided and discussed in this paper were obtained in a direct spatial thinking, spatial pattern recognition, and spatial pattern matching mode, rather than under traditional logic-based and algorithmic philosophy and culture. Being highly parallel and fully distributed, they may challenge existing opinions that parallel and distributed algorithms are usually much more complex than traditional sequential ones for solving same problems, just proving the opposite, and especially for the network-related problems. The latest version of SGL can be quickly implemented by a group of system programmers in standard environments, as was done for the previous versions in different countries with the author's supervision. Forthcoming plans of this work include preparation of the new book on "Spatial Networking in the United Physical, Virtual, and Mental Worlds", which can use the material and results of this paper.

References

1. Transport networks, Inspire Registry. (2015). Retrieved from https://inspire.ec.europa.eu/theme/tn
2. Transportation networks, Travel Forecasting Resource. Retrieved from https://tfresource.org/topics/Transportation_networks.html
3. Durkin, J., Goodman, J., Frank Posse, Rezek, M., Wallace, M., et al. (2012). Building Multiservice Transport Networks

(Networking Technology) (1st ed.). Cisco Press.
4. Murakami, M., Li, H., & Ryoo, J. D. (2014). Packet Transport Networks: Overview and Future Direction (2nd APT/ITU Conformance and Interoperability Workshop (C&I-2)).
5. Air Transport Network. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Air_Transport_Network
6. Qureshi, K. N., Ul Islam, F., Kaiwartya, O., Kumar, A., & Lloret, J. (2020). Improved Road Segment-Based Geographical Routing Protocol for Vehicular Ad-hoc Networks. Electronics, 9(1248).
7. Sujan. (2023). What is Communication Network? Definition, Importance, and 4 Types (Explained). Retrieved from https://www.google.com/search?q=https://tyonote.com/communication_network/
8. Fitzek, F. H. P., & Granelli, F. (2020). Computing in Communication Networks: From Theory to Practice (1st ed.). Academic Press.
9. Srikant, R., & Ying, L. (2013). Communication Networks: An Optimization, Control, and Stochastic Networks Perspective (1st ed.). Cambridge University Press.
10. Raghunandan, K. (2023). Introduction to Wireless Communications and Networks: A Practical Perspective (Textbooks in Telecommunication Engineering) (1st ed.). Springer.
11. Lou, Y., & Ahmed, N. (2021). Underwater Communications and Networks (Textbooks in Telecommunication Engineering). Springer.
12. Kaufman, C., & Perlman, R. (2022). Network Security: Private Communication in a Public World (Prentice Hall Series in Computer Networking and Distributed Systems) (3rd ed.). Addison-Wesley Professional.
13. Wu, Y., & Singh, S. (2022). 6G Mobile Wireless Networks (Computer Communications and Networks) (1st ed.). Springer.
14. Social network. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Social_network
15. Kenton, W. (2023). What Is Social Networking? Investopedia. Retrieved from https://www.google.com/search?q=https://www.investopedia.com/terms/s/social-networking.asp%23:~:text%3DSocial%2520networking%2520refers%2520to%2520using,)%2C%20Instagram%2C%20and%20Pinterest.%2C%20Instagram%2C%20and%20Pinterest.)
16. Nanawati, S. (2019). Social Network Analytics. Analytics Vidhya. Retrieved from https://medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16
17. Singh, C. (2023). 20+ Most Popular Social Networking Sites in 2024. Retrieved from https://www.socialpilot.co/social-networking-sites
18. Jackson, M. O. (2010). Social and Economic Networks (Illustrated ed.). Princeton University Press.
19. Harrison, T. (2021). Battle Networks and the Future Force. Center for Strategic and International Studies. Retrieved from https://www.csis.org/analysis/battle-networks-and-future-force
20. Stillion, J., & Clark, B. (2015). What It Takes To Win: Succeeding In 21st Century Battle Network Competitions. Center for Strategic and Budgetary Assessments (CSBA). Retrieved from https://csbaonline.org/uploads/documents/What-it-Takes-to-Win.pdf
21. Hadley, G. (2023). DAF Outlines a New 'Battle Network' as Its Contribution to JADC2. Air & Space Forces Magazine. Retrieved from https://www.airandspaceforces.com/daf-battle-network-contribution-jadc2/
22. The Challenges of Military Networks and Technological Developments. Elbit Systems. Retrieved from https://www.google.com/search?q=https://elbitsystems.com/elynx_multi_channel/
23. Themnér, A., & Karlén, N. (2020). Building a Safety Net: Explaining the Strength of Ex-Military Networks. Security Studies, 29, 268-300.
24. Vietnambiz. (2019). What is an Economic Network? Advantages and disadvantages of economic networks. Retrieved from https://vietnambiz.vn/mang-luoi-kinh-te-economic-network-la-gi-uu-nhuoc-diem-cua-mang-luoi-kinh-te-20191011113428212.htm
25. Halton, C. (2022). Economic Network: What it is, Pros and Cons, Examples. Investopedia. Retrieved from https://www.google.com/search?q=https://www.investopedia.com/terms/e/economic-network.asp%23:~:text%3DKey%2520Takeaways-,An%2520economic%2520network%2520is%2520a%2520combination%2520of%2520individuals%252C%2520groups%252C%2520or,companies%2520or%2520partnerships%2520between%2520corporations.
26. Todo, Y., & Matous, P. (2015). Economic networks for more innovative and resilient economies. Retrieved from https://cepr.org/voxeu/columns/economic-networks-more-innovative-and-resilient-economies
27. Emmert-Streib, F., Tripathi, S., Yli-Harja, O., & Dehmer, M. (2018). Understanding the World Economy in Terms of Networks: A Survey of Data-Based Network Science Approaches on Economic Networks. Frontiers in Applied Mathematics and Statistics, 4.
28. Wixted, B. (2009). Innovation System Frontiers: Cluster Networks and Global Value (Advances in Spatial Science). Springer.
29. Manyika, J. (2020). What the economy looks like so far for individuals in the 21st century. LinkedIn. Retrieved from https://www.google.com/search?q=https://www.linkedin.com/pulse/what-economy-looks-like-so-far-individuals-21stcentury-james-manyika
30. Hayes, A. (2021). Distribution Network: Definition, How It Works, and Examples. Investopedia. Retrieved from https://www.google.com/search?q=https://www.investopedia.com/terms/d/distribution-network.asp%23:~:text%3DIn%2520a%2520supply%2520chain%252C%2520a,or%2520through%2520a%2520retail%2520network.
31. CFI Team. Distribution Network. The flow of goods from a producer or supplier to an end consumer. Corporate Finance Institute. Retrieved from https://corporatefinanceinstitute.com/resources/valuation/distribution-network/
32. oboloo FAQ's. What Are Types Of Distribution Networks In Supply Chain Management? Retrieved from https://www.google.com/search?q=https://oboloo.com/blog/what-are-types-of-distribution-networks-in-supply-chain-management/
33. Distribution Network – What Is It and What Are the Benefits? Retrieved from https://www.salesbabu.com/blog/what-is-a-distribution-network-and-its-benefits/
34. Wu, Q., & Shen, F. (2023). Optimal Operation of Active

Distribution Networks: Congestion Management, Voltage Control and Service Restoration. Academic Press.
35. Watts, S. (2020). what is a Virtual Network? BMC. Retrieved from https://www.bmc.com/blogs/virtual-network/
36. VMware by Broadcom. What is Virtual Networking? Retrieved from https://www.vmware.com/topics/glossary/content/virtual-networking.html
37. Network virtualization. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Network_virtualization
38. Carlos, O., Duarte, M. B., & Pujolle, G. (2013). Virtual Networks: Pluralistic Approach for the Next Generation of Internet (Networks and Telecommunications) (1st ed.). Wiley-ISTE.
39. Fundamentals of Virtual Networking. (n.d.). GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/fundamentals-of-virtual-networking/
40. Neural network. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Neural_network
41. Anand, V. (2023). Introduction to Neural Network in Machine Learning. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/
42. Ashtari, H. (2022). What Is a Neural Network? Definition, Working, Types, and Applications in 2022. Spiceworks. Retrieved from https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/
43. NVIDIA Developer. Artificial Neural Network. Retrieved from https://developer.nvidia.com/discover/artificial-neural-network
44. Wu, L., Cui, P., Jian Pei, & Zhao, L. (2022). Graph Neural Networks: Foundations, Frontiers, and Applications (1st ed.). Springer.
45. Borsboom, D., Deserno, M. K., Rhemtulla, M., Epskamp, S., Fried, E. I., et al. (2021). Network analysis of multivariate data in psychological science. Nature Reviews Methods Primers, 1(58).
46. Jones, P. J., Mair, P., & McNally, R. J. (2018). Visualizing Psycho logical Networks: A Tutorial in R. Frontiers in Psychology, Sec. Quantitative Psychology and Measurement, 9. https://doi.org/10.3389/fpsyg.2018.01742
47. Fonseca-Pedrero, E. (2018). Network Analysis in Psychology. Psychologist, 39, 1-12.
48. Bringmann, L. F. (2021). Person-specific networks in psycho pathology: Past, present, and future. Current Opinion in Psychology, 41, 59-64.
49. Yuqing, C., Shuyang, D., Shuai, Y., Chuan-Peng, H., et al. (2020). Network analysis and its applications in psychol ogy. Advances in Psychological Science, 28, 178-190.
50. Campana. (2016). Explaining criminal networks: Strategies and potential pitfalls. Methodological Innovations, 9, 1-10.
51. Bichler, G. (2019). Understanding Criminal Networks: A Research Guide (1st ed.). University of California Press.
52. Morselli, C. (2013). Crime and Networks (Criminology and Justice Studies). Routledge.
53. UNODS, Unite Nations Office on Drugs and Crime. New forms of organized crime: networked structure. Retrieved from https://www.unodc.org/e4j/en/organized-crime/module-7/key-issues/networked-structure.html
54. Hufnagel, S., & Moiseienko, A. (2021). Criminal Networks and Law Enforcement: Global Perspectives on Illegal Enterprise (Transnational Criminal Justice) (1st ed.). Routledge.
55. DHS Centers of Excellence. Detecting and Disrupting Transnational Criminal Organizations: Analytics for Interdependent Smuggling and Money-Laundering Networks, Criminal Investigation and Network Analysis. Retrieved from https://cina.gmu.edu/projects/detecting-and-disrupting-transnational-criminal-organizations-analytics-for-interdependent-smuggling-and-money-laundering-networks/
56. Sapaty, P. S. (1993). A distributed processing system (European Patent N 0389655). European Patent Office.
57. Sapaty, P. S. (1999). Mobile Processing in Distributed and Open Environments. John Wiley & Sons.
58. Sapaty, P. S. (2005). Ruling Distributed Dynamic Worlds. John Wiley & Sons.
59. Sapaty, P. S. (2017). Managing Distributed Dynamic Systems with Spatial Grasp Technology. Springer.
60. Sapaty, P. S. (2019). Holistic Analysis and Management of Distributed Social Systems. Springer.
61. Sapaty, P. S. (2019). Complexity in International Security: A Holistic Spatial Approach. Emerald Publishing.
62. Sapaty, P. S. (2021). Symbiosis of Real and Simulated Worlds under Spatial Grasp Technology. Springer.
63. Sapaty, P. S. (2022). Spatial Grasp as a Model for Space-based Control and Management Systems. CRC Press.
64. Sapaty, P. S. (2023). The Spatial Grasp Model: Applications and Investigations of Distributed Dynamic Worlds. Emerald Publishing.
65. Sapaty, P. S. (2024). Providing Integrity, Awareness, and Consciousness in Distributed Dynamic Systems. CRC Press.
66. Sapaty, P. S. (2023). Spatial management of air and missile defence operations. Mathematical Machines and Systems, 1, 30-49.

Appendix: Updated Summary on SGL Syntax and Main Constructs

Syntactic categories are shown below in italics, vertical bar separates alternatives, parts in braces indicate zero or more repetitions with a delimiter at the right, and constructs in brackets are optional. The remaining characters and words are the language symbols (including boldfaced braces).

- grasp → constant | variable | [rule] [({grasp,})]
- constant → information | matter | special | custom | grasp
- information → string | scenario | number
- string → '{character}'
- scenario → {{character}}
- number → [sign]{digit} [. {digit}[e[sign]{digit}]]
- matter → "{character}"
- special → thru | done | fail | fatal | infinite | nil | any | all | other | allother | current | passed | existing | neighbors | direct | forward | backward | synchronous | asynchronous | virtual | physical | executive | engaged | vacant | firstcome | unique
- variable → global | heritable | frontal | nodal | environmental global → G{alphameric}
- heritable → H{alphameric}
- frontal → F{alphameric}
- nodal → N{alphameric}
- environmental → TYPE | NAME | CONTENT | ADDRESS | QUALITIES | WHERE | BACK | PREVIOUS | PREDECESSOR | DOER | RESOURCES | LINK | DIRECTION | THRU | WHEN | TIME | STATE | VALUE | IDENTITY | IN

| OUT | STATUS

- rule → type | usage | movement | creation | echoing | verification | assignment | advancement | branching | ransference | exchange | timing | qualifying | grasp
- type → global | heritable | frontal | nodal | environmental | matter | number |
- string | scenario | constant | custom
- usage → address | coordinate | content | index | time | speed | name | place | center |
- range | doer | node | link | unit
- movement → hop | hopfirst | hopforth | move | shift | follow
- creation → create | linkup | delete | unlink
- echoing → state | rake | order | unit | unique | sum | count | first | last | min | max | random | average | sortup | sortdown | reverse | element | position | fromto | add | subtract | multiply | divide | degree | separate | unite | attach | append | common | withdraw | increment | decrement | access | invert | apply | location | distance
- verification → equal | nonequal | less | lessorequal | more | moreorequal | bigger | smaller | heavier | lighter | longer | shorter | empty | nonempty | belong | notbelong | intersect | notintersect | yes | no
- assignment →assign | assignpeers
- advancement → advance | slide | repeat | align | fringe
- branching → branch | sequence | parallel | if | or | and | orsequence | orparallel | andsequence | andparallel | choose | quickest | cycle | loop | sling | whirl | split | replicate
- transference → run | call
- exchange → input | output | send | receive | emit | get
- timing → sleep | allowed
- qualification → contain | release | trackless | free | blind | quit | abort | stay | lift | seize | exit