Research Article

Open Access

CrossMark

Global network management under spatial grasp paradigm

Abstract

The paper describes basics of high-level management model and technology for dealing with large distributed human or technical systems which can be represented as dynamic physical-virtual networks covering any terrestrial or celestial environments. The main technology component, Spatial Grasp Language (SGL), allows us to obtain powerful and compact spatial solutions of different problems by directly expressing their top semantics while hiding traditional system organization and management routines inside efficient networked implementation. Different network creation, evolution, matching, and transformation approaches are investigated and shown in SGL on general networks, which may be practically useful in a variety of areas influencing the dangerously growing world dynamics and caused, for example, by climate change, military, religious and ethnic conflicts, terrorism, refugee flows, weapons proliferation, political and industrial restructuring, growing inequality, economic instability, global insecurity, and very recently, due to the world-wide pandemic horror. The demonstrated networking approach, with its unlimited parallel and fully distributed capabilities to operate without vulnerable central information and management resources, can also cover much greater spheres, up to the creation and evolution of the very universe, by offering real mechanisms for its simulation on arbitrary large distributed computer networks with millions to billions of communication nodes. The developed networking technology can be readily implemented even in traditional university environments, as was done in the past for its previous versions in different countries under the author's supervision.

Keywords: world dynamics, high-level network management, spatial grasp technology, spatial grasp language, network evolution, network dynamics, graph pattern matching, distributed simulation, parallel and distributed programming

Volume 6 Issue 3 - 2020

Peter Simon Sapaty
Institute of Mathematical Machines and Systems, National Academy of Sciences, Ukraine

Correspondence: Peter Simon Sapaty, Institute of Mathematical Machines and Systems, National Academy of Sciences, Glushkova Ave 42, 03187 Kiev Ukraine, Email petersaaty@gmail.com

Received: September 29, 2020 | **Published:** October 21, 2020

Introduction

We are witnessing rapidly growing world dynamics caused by climate change, military, religious and ethnic conflicts, terrorism, refugee flows, weapons proliferation, political and industrial restructuring, inequality, economic instability, global insecurity, and very recently, due to the world-wide pandemic horror.[1–10] Dealing with frequently emerging crises may need rapid integration of scattered heterogeneous resources into capable operational forces pursuing goals which may not be known in advance. Proper understanding and managing of unpredictable and crisis situations urgently need their detailed simulation at runtime and even ahead of it.[11–30] This may also require deep integration of advanced simulation with live control and management within united and enriching each other concepts of virtual, physical, and executive worlds, which should be effectively organized in both local and global scale.[31–42]

The developed Spatial Grasp formalism and Technology (SGT), which was patented and revealed in numerous previous publications (Wiley, Springer, and Emerald books including)[43–48] provides basics for deep integration, actually symbiosis, of different worlds allowing us to unite advanced distributed simulation with spatial parallel and fully distributed control. The investigated applications included classical graph and network theory problems, missile defense, massive collective robotics, evolution of space systems, flexible command and control, industrial, social and international security problems, also effectively expressing main gestalt theory laws allowing them to cover any distributed systems rather than just human mind and brain. The developed formalism allows us to directly exist, operate, and move in different worlds and their combinations, while shifting traditional numerous and boring system management and simulation routines (DIS and HLA[19–27] including) completely to automatic networked interpretation of the basic Spatial Grasp Language (SGL), with resulting solutions often hundreds of times shorter and simpler.

Many problems in the mentioned areas can be formulated on distributed dynamic physical and virtual networks, from their initial creation, growth and evolution to possible decline and death. The current paper analyzes and shows SGT capabilities for parallel and often holistic expression of some basic operations on general networks of arbitrary size and physical distribution, which may be practically useful in all listed above areas for solving various problems. The demonstrated networking approach can also cover much greater spheres, up to creation and evolution of the very universe, by offering practical mechanisms for its simulation on arbitrary large distributed computer networks with millions to billions of communication nodes.

The rest of the paper is organized as follows. Section 2 provides basic details of the developed spatial paradigm that resulted in Spatial Grasp Technology (SGT) with its basic Spatial Grasp Language (SGL) suitable for creation and management of large dynamic systems in distributed and parallel mode. Section 3 describes examples in SGL of simulation of hypothetical business networks covering certain physical spaces, highlighting top level network creation, its hierarchical growth, appearance of new inter-node relations, and further unlimited evolution. Section 4 gives an example of how arbitrary large network can be created in SGL in a randomized and parallel mode, in a single breath, symbolically mimicking "Big Bang" hypothesis. Sections 5 and 6 are investigating different kinds of pattern matching techniques on the created network example. In Section 5, only constant patterns are used with known names of all nodes and links, also ranging from simple to arbitrary topologies. In Section 6, different patterns with variables are considered, first with variables in nodes only, then with variables in both nodes and links, and additionally, with variable graph structures.

Examples of possible global network dynamics are considered in Section 7, from their gradual shrinking to unlimited expansion. Regarding the shrinking process, it is shown how to substitute arbitrary sub-network with a single node having same links to the remaining nodes the removed nodes had. This shrinking also continued in a repeated swallowing by such node of new neighbors in a "Black Hole" mode, until the whole net degenerates into a single node. Another possible network self-destruction is shown where nodes self-discovering fewer neighbors than a threshold given are ceasing to exist, thus weakening in such a way their direct neighbors, and so on. A technique is also shown in SGL for the opposite process – unlimited network growth by the number of nodes and links, and also expansion in physical space up to the whole universe (imitating "Dark Matter" hypothesis too). Section 8 concludes the paper showing possibility of SGT implementation in traditional environments and the ongoing researched of its applicability in other areas.

Spatial grasp technology basics

General SGT idea

Within Spatial Grasp Technology (SGT), a high-level scenario for any task to be performed in a distributed world is represented as an active self-evolving pattern rather than traditional program, sequential or parallel. This pattern, written in a high-level Spatial Grasp Language (SGL) and expressing top semantics of the problem to be solved, can start from any world point. It then spatially propagates, replicates, modifies, covers and matches the distributed world in parallel wavelike mode, while echoing the reached control states and data found or obtained for making decisions at higher levels and further space navigation. This inherently parallel and fully distributed spatial process is very symbolically shown in Figure 1, where reached physical or virtual world points, whatever remote they happen to be, can launch new spatial wave processes remaining under control from the previous points or becoming independent from them.

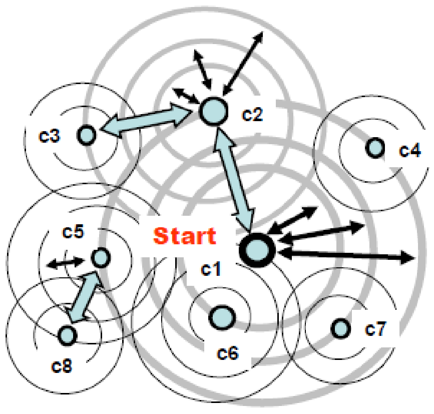

Figure 1 Controlled forward-backward navigation & matching & grasping of distributed spaces.

Many spatial processes in SGL can start any time and in any places, cooperating or competing with each other, depending on applications. The self-spreading & self-matching SGL patterns-scenarios can create *knowledge infrastructures* arbitrarily distributed between system components which may cover any regions (Figure 2), from terrestrial to even celestial in the future (Figure 2).

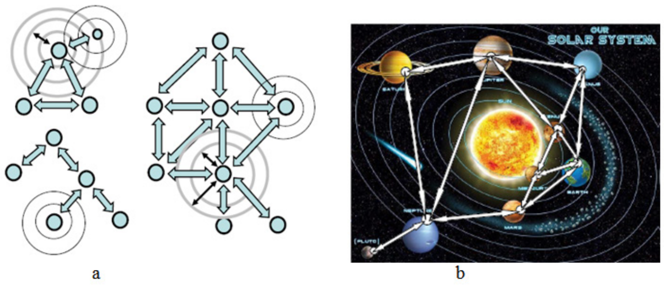

Figure 2 Spreading spatial patterns and creation of distributed infrastructures.

These infrastructures, which may remain active and launch new spatial waves anytime, can effectively support or express distributed databases, advanced command and control, situation awareness, autonomous and collective decisions, as well as any existing or hypothetical computational and or control models.

Spatial grasp language

General SGL organization is as follows, where syntactic categories are shown in italics, vertical bar separates alternatives, parts in braces indicate zero or more repetitions with a delimiter at the right if multiple, and constructs in brackets may be optional:

grasp → *constant* | *variable* | **[** *rule* **] [({** *grasp*,**})]**

From this definition, an SGL scenario called grasp, supposedly applied in some point of the distributed space, can just be a constant directly providing the result to be associated with this point. It can be a variable whose content, assigned to it previously when staying in this or (remotely) in other space point (as variables may have non-local meaning and coverage), provides the result in the application point too. It can also be a rule (expressing certain action, control, description or context) optionally accompanied with operands separated by comma (if multiple) and embraced in parentheses. These operands can be of any nature and complexity (including arbitrary scenarios themselves) and defined recursively as grasp too, i.e. can be constants, variables or any rules with operands (i.e. as grasps again), and so on.

Rules, starting in some world point, 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. These algorithms, called spatial, can effectively operate in, with, under, in between, over, and instead of (as for simulation) large, dynamic, and heterogeneous spaces, which can be physical, virtual, management, command and control, or combined.

SGL full syntax description, as of its latest version, is as follows, with the words in Courier New font being direct language symbols (boldfaced braces including).

grasp → *constant* | *variable* | [*rule*] [({ *grasp*,})]

constant	→	*information* | *matter* | *custom* | *special* | *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 | usual | real | simulate
variable	→	*global* | *heritable* | *frontal* | *nodal* | *environmental*
global	→	G{*alphameric*}
heritable	→	H{*alphameric*}
frontal	→	F{*alphameric*}
nodal	→	N{*alphameric*}
environmental	→	TYPE | IDENTITY | NAME | CONTENT | ADDRESS | POINT | QUALITIES | WHERE | BACK | PREVIOUS | PREDECESSOR | DOER | RESOURCES | LINK | DIRECTION | WHEN | TIME | STATE | VALUE | IDENTITY | IN | OUT | STATUS | MODE | COLOR
rule	→	*type* | *usage* | *movement* | *creation* | *echoing* | *verification* | *assignment* | *advancement* | *branching* | *transference* | *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 | pass | return | follow
creation	→	create | form | 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
verification	→	equal | nonequal | less | lessorequal | more | moreorequal | bigger | smaller | heavier | lighter | longer | shorter | empty | nonempty | belong | notbelong | intersect | notintersect | yes | no
assignment	→	assign | assignpeers | associate
advancement	→	advance | slide | repeat | align | fringe
branching	→	branch | sequence | parallel | if | or | and | choose | quickest | cycle | loop | sling | whirl | split
transference	→	run | call
exchange	→	input | output | send | receive | emit | get
timing	→	sleep | allowed
qualification	→	contain | release | free | blind | quit | abort | stay | lift | seize

SGL interpreter

The SGL interpreter main components and its general organization are shown in Figure 3. The interpreter consists of a number of specialized functional processors (shown by rectangles) working with and sharing specific data structures. These include: Communication Processor, Control Processor, Navigation Processor, Parser, different Operation Processors, and special (external & internal) World Access Unit directly manageable from SGL. Main data structures (also referred to as stores) with which these processors operate (shown by ovals) comprise: Grasps Queue, Suspended Grasps, Track Forest, Activated Rules, Knowledge Network, Grasps Identities, Heritable Variables, Fontal Variables, Nodal Variables, Environmental Variables, Global Variables, Incoming Queue, and Outgoing Queue. SGL interpretation network generally serves multiple scenarios or their parallel branches simultaneously navigating the distributed world, which can cooperate or compete with each other.

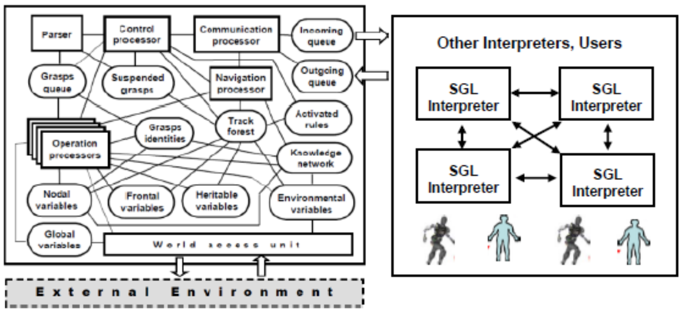

Figure 3 SGL interpreter main components and their interactions.

As both backbone and nerve system of the distributed interpreter, its hierarchical spatial track system dynamically spans the worlds in which SGL scenarios evolve, providing automatic control of multiple distributed processes. Its part related to the current interpreter is kept in the Track Forest store which is interlinked with similar parts in other interpreters, forming altogether global control coverage. Self-optimizing in parallel echo processes, this (generally forest-like) distributed track structure provides hierarchical command and control as well as remote data and code access. It also supports spatial variables and merges distributed control states for making decisions at different organizational levels. The track infrastructure can be automatically distributed between different world points during scenario spreading in distributed environments.

Each interpreter can support and process multiple SGL scenario code which happens to be in its responsibility at different moments of time. More details on SGT, SGL, its implementation and investigated and tested applications can be found elsewhere, including in.[44–48] Implanted into any distributed systems and integrated with them, the interpretation network (having potentially millions to billions of communicating interpreter copies) allows us to form spatial world computer with practically unlimited power for simulation and management of the whole mankind.

Creation and growth of business networks

We will show here how the birth and growth of hypothetical business centers with subordinate units and evolution of different kinds of channels and relations between them can be expressed in the spatial grasp mode provided by SGL. All network nodes will be considered as having all three (i.e. physical, virtual, and executive) dimensions discussed in,[49–63] and the randomized development of business network will be taking place in a physical region with certain boundaries.

Top level network creation

Creation and activation of initial top level business nodes (having names for simplicity in digits) with their random physical distribution, as in Figure 4, may be done in SGL as follows (where these initial business loci can be created in parallel, thus simulating possible concurrent appearance of different businesses in a distributed area).

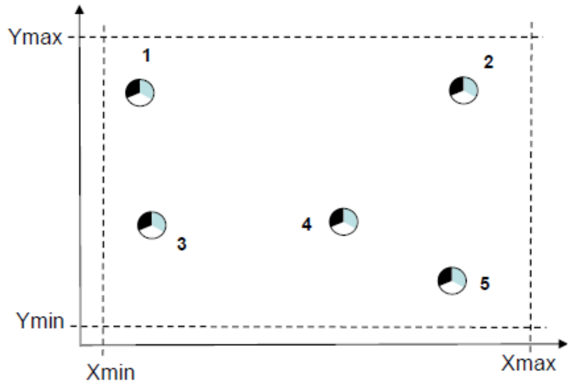

Figure 4 Creation of initial business centers.

parallel(1,2,3,4,5); Name = VALUE;

create_node(

IDENTITY(Name), CONTENT("top"),

coordinate(random(*Xmin*, *Xmax*), random(*Ymin*, *Ymax*)));

activate(current)

Explicit mentioning of the combined type of these nodes (i.e. by using TYPE = P_V_E) is optional, because such features as IDENTITY and linkage to physical (i.e. X-Y defined) space are just speaking for themselves. In a three-dimensional environment (like, for example, in outer space) coordinate Z may be needed too.

Linking the created top level nodes by a sort of global channels, as shown in Figure 5 by hard lines, may be done as follows.

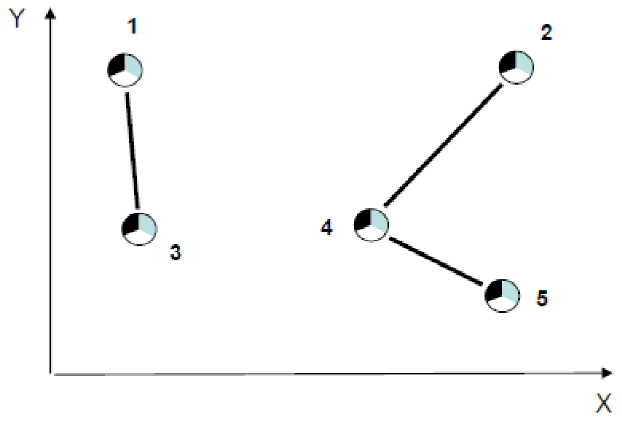

Figure 5 Linking business centers by global channels.

parallel(

(hop_node(1); linkup("global", node(3)),

(hop_node(4); parallel_linkup("global", nodes(2, 5)));

Introducing additional top level nodes randomly distributed in space too, which could be done in parallel, with random and parallel linking them by global channels to the already created nodes, as in Figure 6, may be achieved by the following SGL scenario.

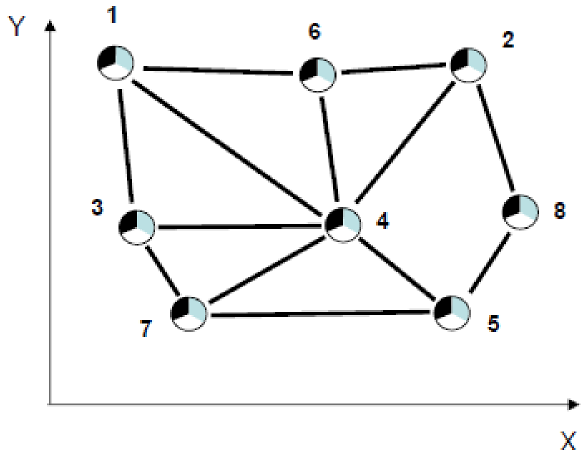

Figure 6 Creating new centers and connecting with previous ones.

parallel_branch(6,7,8); Name = VALUE;

create_node(

IDENTITY(Name), CONTENT("top"),

coordinate(random(Xmin, Xmax), random(Ymin, Ymax)));

stay_parallel_linkup("global", randon_nodes(1, 2, 3, 4, 5)),

activate(current))

Hierarchical network evolution and growth

Let us consider a possible further hierarchical extension and growth of the created network, by introducing additional subordinate nodes to the already created top nodes with establishing directed management links from them, as in Figure 7, with possible SGL scenario following. Three subordinate nodes (with digital sub-names from 1 to 3) for each top node are planned, with a randomly defined distance to them within certain threshold (expressed in italics).

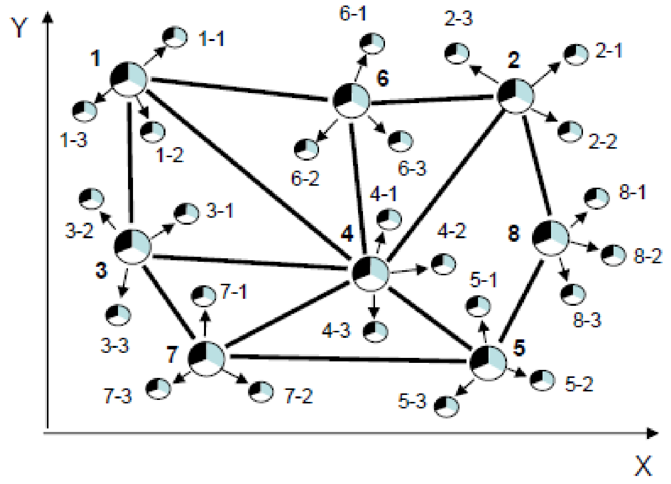

Figure 7 Hierarchical evolution and growth of business network.

hop_nodes(all);

parallel(1,2,3); Below = VALUE;

create(link(+"manage"),

node(IDENTITY(NAME & '-' & Below), CONTENT("subordinate"),

coordinate_random(*maxdistance*)));

activate(current)

Appearance of additional inter-node relations

Imagine now that these new subordinate nodes (already having direct control, management and business links with their top level nodes) want to establish additional direct local business or even joint production relations with other subordinate nodes existing in some vicinity, as shown by dashed lines in Figure 8 and by the SGL scenario following.

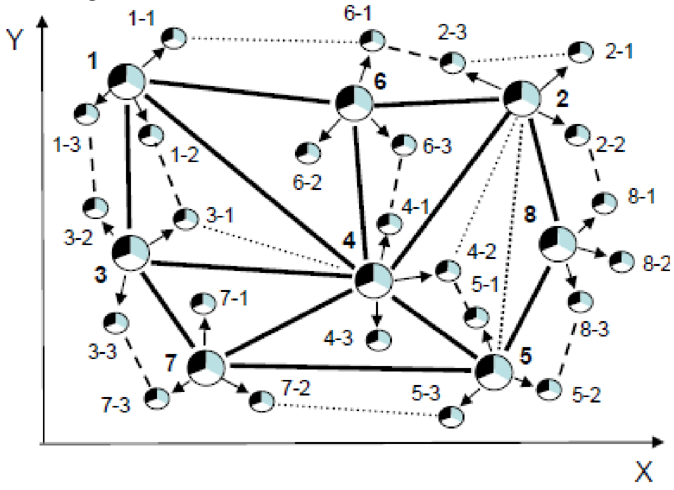

Figure 8 Establishing new business and information relations between nodes.

hop_random_nodes(CONTENT("subordinate"));

linkup("business",

hop_random_nodes(distance(*maxdistance*), CONTENT("subordinate")))

We may also suppose that any nodes of this network already operating for some time, may establish different kinds of information exchange or shared knowledge links regardless of distance between them, as shown in Figure 8 in dotted lines and by the following scenario.

hop_random_nodes(all);

linkup("information", random_nodes(others))

Further network growth

Using similar SGL scenarios as above, we can continue growing the network of Figure 8, both hierarchically by adding more levels of nodal subordination (names of lower level nodes may be extended from the names of the previous level, similar to Figure 4), and also introducing additional direct links between different types of nodes, as shown in Figure 9.

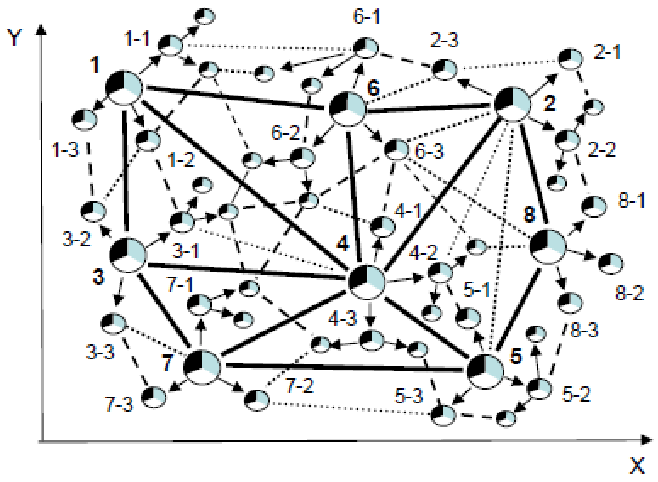

Figure 9 Further growth of the business network.

In further developments, new top level nodes may appear with new global links between themselves and already existing top nodes, which, in their turn, may create subordinate nodes within any levels of hierarchy. Various new links with other nodes can be established too, and so on, thus effectively imitating industrial growth in both terrestrial and celestial environments, including its inevitable extension to Moon, Cislunar Space, even Mars and beyond, and all this can be clearly and concisely described and simulated in SGL.

Parallel creation of arbitrary network

In the previous section we have described an example of creation and growth of industrial-like networks in distributed environments which, despite generality, had certain specifics like general hierarchical organization and particular semantics-oriented types of relations and connections between nodes.

For investigation of various operations on general networks in the subsequent sections we will consider here the creation of arbitrarily large exemplary virtual network in a single breath mode, symbolically imitating the "Big Bang" hypothesis.[49] It will be using node names expressed for simplicity by digits and links randomly connecting such nodes with random number of other nodes, with link names as lower case alphabetic letters. This is shown in Figure 10 and by the following parallel SGL scenario (using for compactness of the picture only a limited number of nodes named from 1 to 20, with the number of possible connections to other nodes just between 2 and 6).

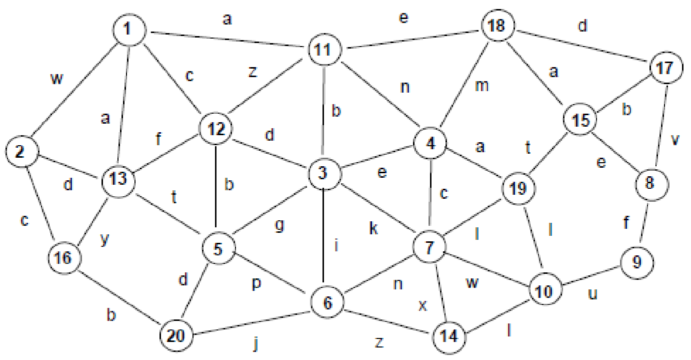

Figure 10 Parallel creation of exemplary network in a single breath.

create_parallel_nodes(fromto(1, 20));

linkup_parallel(

random(lower_case_letters),

nodes(number_random_fromto(2, 6), names_random(all_others)))

If to consider distribution of the created nodes in physical space, the scenario may look like follows, with nodes supposedly allowed to be randomly linked with each other only within certain threshold distance between them. And the nodes' physical positions should also be within certain boundaries defined by: Xmin, Xmax, Ymin, and Ymax.

create_parallel_nodes(

fromto(1, 20), coordinates(random(*Xmin*, *Xmax*), random(*Ymin*, *Ymax*)));

linkup_parallel(

random(lower_case_letters),

nodes(number_random_fromto(2, 6), names_random(all_others),

distance(*maxdistance*)))

As the network and its distribution in physical space were performed randomly by this scenario, its real visual planar picture may not be as nice as in Figure 10, which we have drawn here only for conveniently showing and explaining various solutions on general networks, to be discussed in the subsequent sections. It to consider a 3-D network creation, distribution and growth, say, like both on Earth and in outer space, we should engage the third dimension too, with Zmin and Zmax as its expected limits.

Network pattern matching with constant patterns

Describing and finding different structures in distributed networks has numerous applications in different areas of system management. We are starting here with discovering various structures in arbitrary networks that have known topology and names of all their nodes and links, which can be found by applying corresponding constant graph patterns to the whole network.

Examples of particular patterns and their matches

Three simple traditional patterns are shown in Figure 11, with Pattern 1 just reflecting two linked nodes, Pattern 2 as a star, and Pattern 3 as a tree.

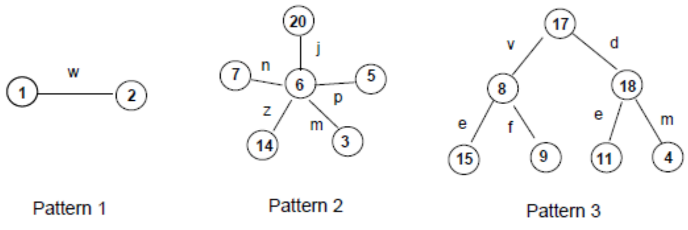

Figure 11 Simple graph patterns with constant nodes and links.

The following is there expression and network matching in SGL, with output of the matching success possible in different locations.

- *Pattern 1*

Starting in the first node and output in the second node:

hop_node(1); hop(link(w), node(2)); output(OK)

Starting in the first node and output in it too:

hop_node(1); if(hop(link(w), node(2)), output(OK))

Output in the outside location from which the scenario was issued and then started in the first node:

if((hop_node(1); hop(link(w), node(2))), output(OK))

Similar to all above will be if to start matching from the second node of the pattern.

The only match of this pattern is shown in Figure 12.

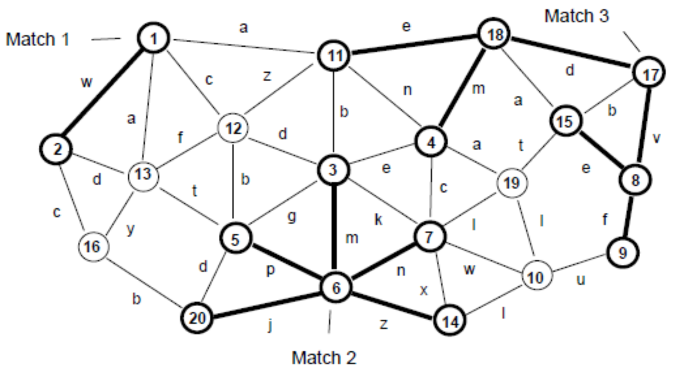

Figure 12 Matching solutions for simple patterns.

- *Pattern 2*

The output in case of matching success can be issued in the central pattern's node (i.e. 6) or in the scenario starting location, with SGL code for the second case following and matching result in Figure 12.

```
if((hop_node(6);
  and_parallel(
    hop(link(j), node(20)),
    hop(link(p), node(5)),
    hop(link(m), node(3)),
    hop(link(z), node(14)),
    hop(link(n), node(7)))),
 output(OK))
```

- *Pattern 3*

The output for this tree-structured pattern can be issued in the top tree node (i.e. 17) or in the scenario starting location as for the previous pattern, with SGL code for the second option following and successful match shown in Figure 12.

```
if((hop_node(17);
  and_parallel(
    (hop(link(d), node(18));
     and_parallel(hop(link(m), node(4)),
                  hop(link(e), node(11)))),
    (hop(link(v), node(8));
     and_parallel(hop(link(f), node(9)),
                  hop(link(e), node(15)))))),
  output(OK))
```

Dealing with arbitrary patterns

Any constant graph pattern can be easily represented as a tree too, as in the previous case, which should cover all pattern's nodes and all links, and for this, some nodes may be repeated more than once, as for the pattern in Figure 13 and one of its possible tree representation shown in Figure 13. The repeated nodes in this tree will be as: 6, 7, 10, and 14.

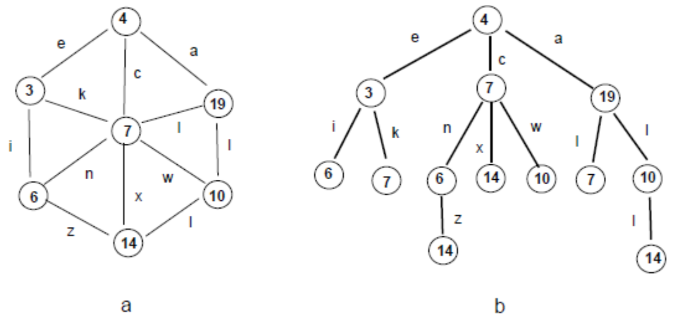

Figure 13 Representation of arbitrary graph pattern by a tree with repeating nodes.

The SGL matching scenario will be as follows, with matching result to be issued in the outside position issuing the scenario (the output can also be organized in the top tree node, here 4).

```
if((hop_node(4);
  and_parallel(
    (hop(link(a), node(19));
     and_parallel((hop(link(l), node(10)); hop(link(l), node(14))),
                   hop(link(l), node(7))),
    (hop(link(c), node(7));
     and_parallel(hop(link(w), node(10)),
                  hop(link(x), node(14)),
                  (hop(link(n), node(6)); hop(lik(z), node(14)))),
    (hop(link(e), node(3));
     and_parallel(hop(link(k), node(7)),
                  hop(link(i), node(6)))))),
  output(OK))
```

Successful matching result for this tree-converted pattern is shown in Figure 14.

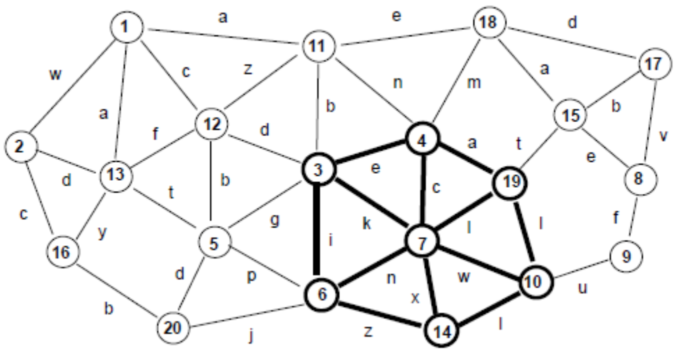

Figure 14 Matching result for arbitrary graph pattern.

Using graph patterns with variables

In the previous section we considered finding parts of the network with exact structures, exact number of nodes and links, and all link and node names as known constants, with all this expressed in detail in the search patterns. In the current section, we will be considering matching patterns having variables associated with their different elements: nodes, links, as well as total graph structures with not known in advance numbers of nodes and links.

Patterns with variables in nodes only

Such patterns will be having variables in all nodes only with their meanings to be found after successful matches with the network, by using different constant graph structures, from simplest to most general.

Particular patterns with nodal variables

We will be using simple patterns with variables in nodes as in Figure 15, which are similar to the patterns with all constants of Figure 11.

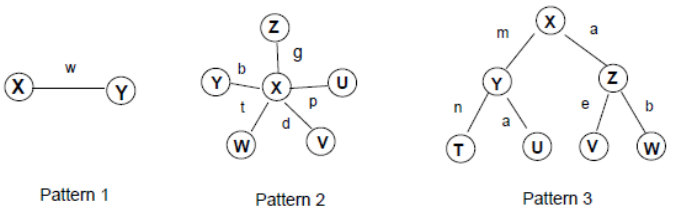

Figure 15 Simple graph patterns with variables in nodes.

- *Pattern 1*

Collecting X and Y meanings and printing the successful match in the second (or Y) node, as follows.

```
frontal(X);
hop_nodes(all); X = NAME; hop_link(w); Y = NAME; output(X && Y)
```

Printing the match in the second node without explicit using variables X, Y:

```
hop_nodes(all); hop_link(w); output(PREDECESSOR && NAME)
```

Similar, but printing the matching result in the first (or X) node:

```
hop_nodes(all); output(hop_link(w); PREDECESSOR && NAME)
```

Similar, but printing all possible matches in the external location from which the scenario was issued (showing particular results as parenthesized units to distinguish between different solutions):

```
output(hop_nodes(all); hop_link(w); unit(PREDECESSOR && NAME))
```

Output of all solutions (Figure 16) will be: (2, 1), (1, 2) , (7, 10), (10, 7).

- *Pattern 2*

Printing the match found in the central, X- related node can be achieved by:

```
hop_nodes(all);
output(
 and_parallel(
  'X:' & NAME,
  (hop_link(b); 'Y:' & NAME),
  (hop_link(g); 'Z:' & NAME),
  (hop_link(p); 'U:' & NAME),
  (hop_link(d); 'V:' & NAME),
  (hop_link(t); 'W:' & NAME)))
```

Output (Figure 14) will be as: X:5, Y:12, Z:3, U:6, V:20, W:13.

Printing in the scenario starting location of all possible matches where different matches as separate units should be enclosed in parentheses to be distinguishable from each other, if more than one (we only have a single match for this pattern, as in Figure 16).

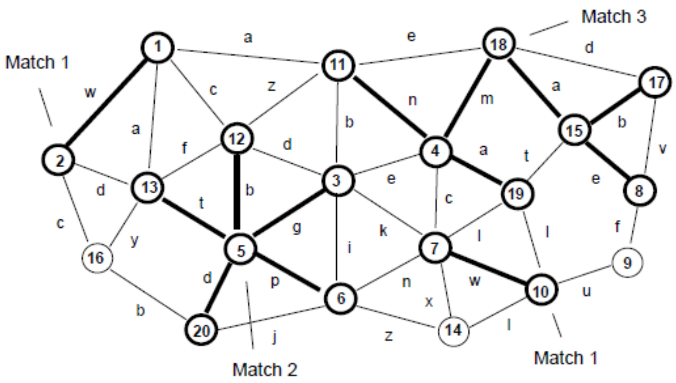

Figure 16 Matching results for simple patterns with variables.

```
output(
 hop_nodes(all);
 unit(and_parallel(
     'X:' & NAME,
     (hop_link(b); 'Y:' & NAME),
     (hop_link(g); 'Z:' & NAME),
     (hop_link(p); 'U:' & NAME),
     (hop_link(d); 'V:' & NAME),
     (hop_link(t); 'W:' & NAME))))
```

The printed solution will be as: (X:5, Y:12, Z:3, U:6, V:20, W:13).

- *Pattern 3*

A particular match for this pattern, if found, can be issued in the X-related node, and all matches can also be printed in the scenario starting position (similar to the previous case). We are showing here the second option, with only a single match available for the network of Figure 16.

```
output(
 hop_nodes(all);
 unit(and_parallel(
     'X:' & NAME,
     (hop_link(a);
      and_parallel('Z:' & NAME,
            (hop_link(b); 'W:' & NAME),
            (hop_link(e); 'V:' & NAME))),
     (hop_link(m);
      and_parallel('Y:' & NAME,
            (hop_link(a); 'U:' & NAME),
            (hop_link(n); 'T:' & NAME)))))
```

Output of the only match (Figure 14) will be: (X:18, Y:4, Z:15, U:19, V:8, W:17, T:11).

Using arbitrary graph patterns

One of possible matching techniques for arbitrary patterns with variables in nodes, actually the simplest one, can be based on a path through all pattern's nodes, simplifying collection of all found values of variables for a particular match at the end of the path (Figure 17). Some nodes and links may have to be represented more than once in such a pass (not for the case of Figure 17).

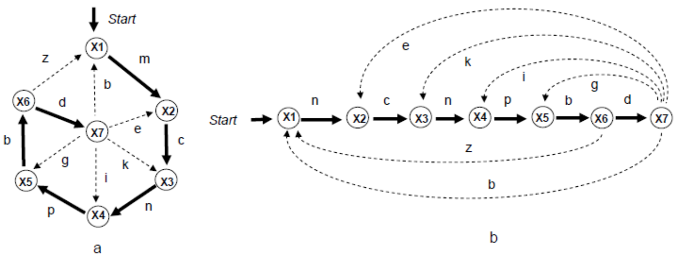

Figure 17 Universal linear representation of arbitrary pattern with variables in nodes.

With the resultant node matches represented in the order reflecting indexing of variables Xi in the path, and the remaining links always leading to the previous nodes of the path, the SGL solution for Figure 17 will be as follows, with the output in final node of the path corresponding to variable X7.

```
hop_nodes(all); frontal(X) = NAME;
hop_link(n); X &&= NAME;
hop_link(c); X &&= NAME;
hop_link(n); X &&= NAME;
hop_link(p); X &&= NAME;
hop_link(b);
true_hop(link(z), node(X[1]));
X &&= NAME; hop_link(d);
true_hop(link(e), node(X[2]));
true_hop(link(k), node(X[3]));
true_hop(link(i), node(X[4]));
true_hop(link(g), node(X[5]));
X &&= NAME; output(X)
```

Resultant match for variables X1 to X7, issued in the final node match 3, will be as: 11, 4, 7, 6, 5, 12, 3 (Figure 18).

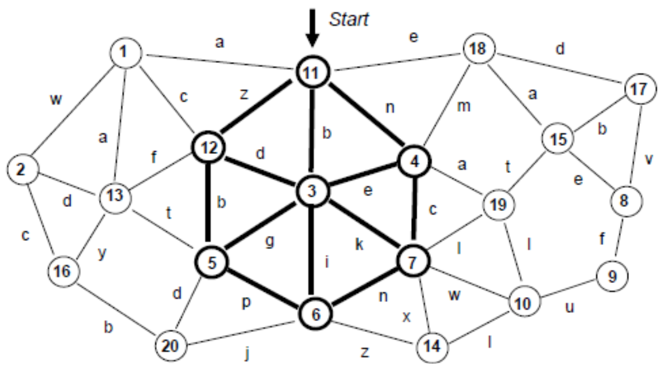

Figure 18 Matching result for the arbitrary pattern with variables.

By introducing parallel branches and issuing all possible matches in the external position from which the matching scenario was issued, the SGL solution will be as follows:

```
output(
 hop_nodes(all); frontal(X) = NAME;
 hop_link(n); X &&= NAME;
 hop_link(c); X &&= NAME;
 hop_link(n); X &&= NAME;
 hop_link(p); X &&= NAME;
 hop_link(b);
 true(hop(link(z), node(X[1])));
 X &&= NAME; hop_link(d);
 true_and_parallel(
   hop(link(e), node(X[2])),
   hop(link(k), node(X[3])),
   hop(link(i), node(X[4])),
   hop(link(g), node(X[5])));
 unit(X && NAME))
```

The only available match (same as before) will be issued in the outside position as a parenthesized unit: (11, 4, 7, 6, 5, 12, 3). We could also issue each match after its full completion in the starting node of the pattern (corresponding to variable X1, i.e. found node 11), as was shown for the previous patterns.

Arbitrary graph patterns with variables in both nodes and links

An example of such pattern, similar to the previous one of Figure 17 but with variables on links too, is shown in Figure 19, where the indexing of variables for nodes and links is chosen to be arbitrary and possibly more convenient, not necessarily following the path though all nodes as before. (This is accomplished by organizing sets of variables as indexed lists allowing for their growth and access to elements in any order, by explicit indices, during storing of different matches of nodes and links).

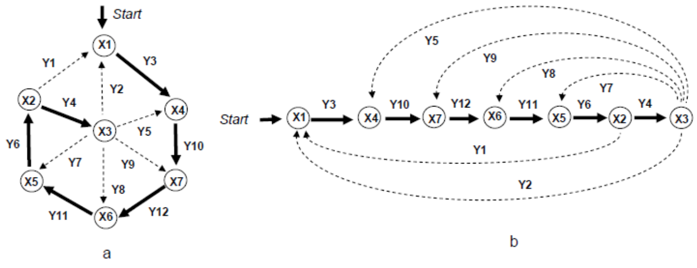

Figure 19 Arbitrary graph pattern with variables in nodes and links and its linear matching template.

Each matching solution can be issued in the path final node related to X3, in its starting node related to X1, and we can also collect all possible matches in the scenario starting outside position, similarly to the previous patterns, with the solution for the third option just following.

```
output(
 hop_nodes(all); frontal(X, Y);
 X[1] = NAME
 hop_link(any); X[4] = NAME; Y[3] = LINK;
 hop_noback_link(any); X[7] = NAME; Y[10] = LINK;
 hop_noback_link(any); X[6] = NAME; Y[12] = LINK;
 hop_noback_link(any); X[5] = NAME; Y[11] = LINK;
 hop_noback_link(any); X[2] = NAME; Y[6] = LINK;
 nonempty(Y[1] = (hop(link(any), node(X[1])); LINK));
 hop_noback_link(any); X[3] = NAME; Y[4] = LINK;
 true_and_parallel(
   nonempty(Y[5] = (hop(link(any), node(X[4])); LINK)),
   nonempty(Y[9] = (hop(link(any), node(X[7])); LINK)),
   nonempty(Y[8] = (hop(link(any), node(X[6])); LINK)),
   nonempty(Y[7] = (hop(link(any), node(X[5])); LINK)));
 unit(unit('X: ', X), unit('Y: ', Y)))
```

The output of all matches for the network of Figure 10 (possibly, in a different order) will be as follows, with the order of printed names of nodes and links in each match corresponding to the indices of related X and Y variables of the pattern in Figure 19.

14 matches found for the graph of Fig. 18 shown in bold:

((X: 11, 12, 3, 4, 5, 6, 7), (Y: z, b, n, d, e, b, g, i, k, c, p, n)), … ,

And also there are 14 matches for the graph of Fig. 14, in bold too:

((X: 4, 3, 7, 19, 6, 14, 10), (Y: e, c, a, k, l, i, n, x, w, z , l)), … .

Multiple matches for same graphs of Figures 14 & 18 appeared because each node of these graphs can match the starting pattern's node, i.e. X1, and also the circular path from it via remaining nodes can develop in two opposite directions (like clockwise and counterclockwise), but all the mentioned above matches are formally different and legitimate.

Patterns with variable structures

All previous cases with constants or variables in nodes and links considered exact, fixed structures of the patterns which should be matched with the network. But the pattern's structure can also be a sort of a variable too, say, by fitting solutions with different number of nodes and links, as well as their interconnections. We will consider here a simple example of finding structures representing a cyclic chain (ring) of interconnected nodes which may not have known in advance number of nodes (say, with its maximum limited by some threshold), with all nodes and links as variables too, as in Figure 20. Special constraints are also be added to this pattern -- that all nodes of these rings do not have other connections with each other than those forming the ring, and also there is no at least a single outside node that has links with all nodes of the ring (like "external authority", say, for special applications).

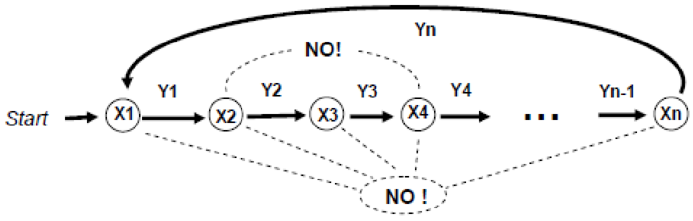

Figure 20 Example of a pattern with variable number of nodes and links.

Finding such a match with output in its finally found (i.e. Xn related) node can be achieved by the following scenario, with the threshold number of nodes (or count) in such matches taken as 5.

```
hop_nodes(all); frontal(X = NAME, Y);
repeat(
 hop_noback_links(all); append(LINK, Y);
 if(NAME = X[1],
   done(no(hop_nodes(X); hop(links(all), nodes(not(X));
           and_parallel_hop(links(any), nodes(X)));
        output(unit('X:', X), unit('Y:', Y)));
 notbelong(NAME, X); no_hop_noback(links(any), nodes(X));
 append(NAME, X); count(X) <= 5)
```

Output in the scenario starting position of all possible matches can be achieved by:

```
output(
 hop_nodes(all); frontal(X = NAME, Y);
 repeat(
  hop_noback_links(all); append(LINK, Y);
  if(NAME = X[1],
    done(no(hop_nodes(X); hop(links(all), nodes(not(X));
            and_parallel_hop(links(any), nodes(X)));
         unit(unit('X:', X), unit('Y:', Y))));
  notbelong(NAME, X); no_hop_noback(links(any), nodes(X));
  append(NAME, X)); count(X) <= 5))
```

Output of only some matches in the scenario starting node will be:

((X: 1, 11, 12), (Y:a, z, c)), ((X: 3, 7, 6), (Y: k, n, i)), … -- all three nodes

((X: 16, 13, 5, 20), (Y: y, t, d, b)), ((X: 4, 18, 15, 19), (Y: m, a, t, a)) -- four nodes

((X: 19, 15, 8, 9, 10), (Y: t, e, f, u, l)) -- five nodes

Some of the found matches are shown in Figure 21 in bold or dashed lines if solutions intersect, where full result includes all triangles. Each match will actually have repetitions, as its search starts from all nodes of the same solution, also develops in both ways in the ring (like clockwise and counterclockwise), which will be reflected by indices of the resultant values of X and Y variables.

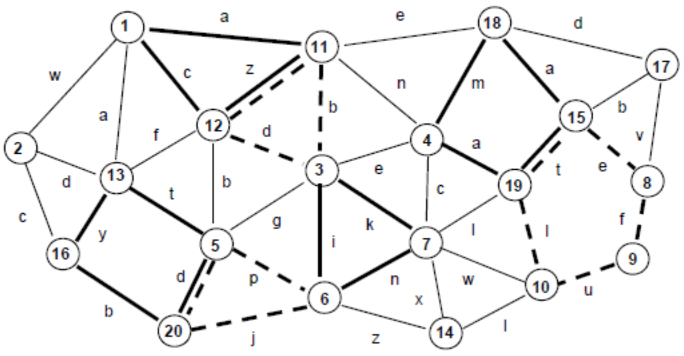

Figure 21 Some matching solutions for the pattern with variable structure.

Multiple repeated solutions, which may start from each node of the ring and develop in two directions, can be easily reduced to only two. This is achievable by allowing them to start only with the node having the strongest name (or address) among all nodes of the ring. In this respect, we can extend the string just before the last one in two previous scenarios as follows:

X[1]>NAME; notbelong(NAME, X); no(hop_noback(links(any), nodes(X)));

If to take other than 5 threshold number of nodes in the previous scenario, we will have additional ring solutions without "central authority", as below for count(X) <= 6, see also Figure 22.

((X: 2, 1, 12, 5, 20, 16), (Y: w, c, b, d, b, c))

((X: 13, 12, 3, 6, 20, 16), (Y: f, d, i, j, b, y))

((X: 11, 18, 15, 19, 7, 3), (Y: e, a, t, l, k, b))

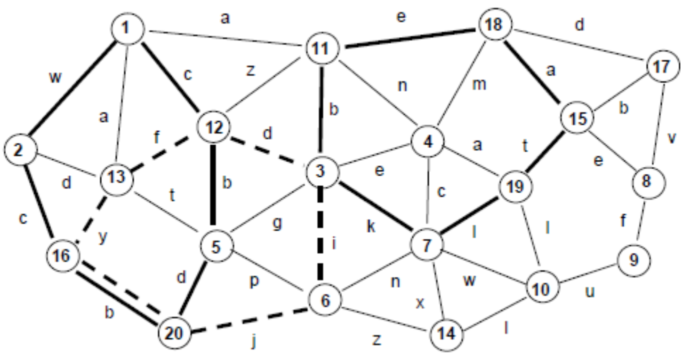

Figure 22 Additional matches for the variable pattern with increased threshold on the number of nodes.

Examples of global network dynamics

We will be considering here some massive operations and transformations on distributed networks in a global scale and their effective expression in SGL.

Shrinking networks

We will show here how to express massive gradual self-reduction of the network in its size, i.e. in a number of its nodes and links, from its full body to the ultimate naught, by using different kinds of parallel techniques.

Substituting of a group of nodes with a single node

By first considering the pattern of Figure 17 with variables in nodes, will be trying to substitute all nodes of its match found in Fig. 18 by a single node, say, with a symbolic name 100, which should have all links to the remaining nodes the substituted pattern had, as in Figure 23. Also assuming the CONTENT of this new node will reflect the number of substituted nodes by it.

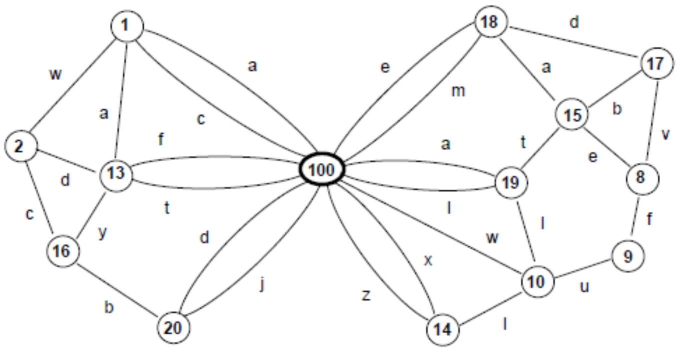

Figure 23 Substituting a group of nodes by a single node with saving links to other nodes.

A possible SGL solution for such substitution is shown below together with the initial finding the match of the pattern of Figure 17.

```
frontal(Group) = (
  hop_nodes(all); frontal(X);
  hop_link(m); X = NAME;
  hop_link(c); X &&= NAME;
  hop_link(n); X &&= NAME;
  hop_link(p); X &&= NAME;
  hop_link(b); X &&= NAME;
  true_hop(link(z), node(X[1]));
  hop_link(d); X &&= NAME;
  true_and_parallel(
    hop(link(e), node(X[2])),
    hop(link(k), node(X[3]));
    hop(link(i), node(X[4]));
    hop(link(g), node(X[5])));
  X && NAME);
sequence(
  (create_node(100); CONTENT = count(Group);
   frontal(New) = ADDRESS;
   hop_nodes(Group);
   hop(links(all), nodes(notbelong(Group));
   linkup(LINK, node(New))),
  remove_nodes(Group))
```

If the group's nodes to be substituted are known in advance, the SGL solution will be shorter, as follows. Also, we may place the new node into the averaged topological center of the deleted group if physical positions of its nodes are known, like for the case of virtual-physical world integration.

```
frontal(Group) = (11, 4, 7, 6, 5, 12, 3);
Center = average(hop_nodes(Group); WHERE);
sequence(
  (create_node(100, coordinate(Center));
   frontal(New) = ADDRESS; CONTENT = count(Group);
   hop_nodes(Group);
   hop(links(all), nodes(notbelong(Group));
   linkup(LINK, node(New))),
  remove_nodes(Group))
```

The CONTENT of node 100 will be 7, reflecting the number of nodes it substituted.

Black hole mode of further network shrinking

Having substituted part of the network by the new node named 100, as above, let us consider further shrinking of this network in the "Black Hole"[50] mode, where each time this new node absorbs all neighboring nodes and establishes all links with the nodes these neighbors had before their consumption. Let us also increase the CONTENT of this Black Hole node by the number of newly swallowed nodes by it. This spatial iterative process, shown in three stages in Figure 24 after obtaining the network of Figure 23, results in the only renaming node 100 as the ultimate Black Hole, with SGL solution of such gradual shrinking-consumption process being as follows.

```
frontal(Hole) = 100; hop_first_node(Hole);
repeat(
  nonempty(Around = (hop_links(all); NAME));
CONTENT += count(Around);
stay_sequence(
 (hop_first_links(all); hop_first_links(all));
  linkup(LINK, node(Hole))),
 remove(links(all), nodes(Around));
sleep(delay))
```

The final CONENT of the resulting Black Hole node will be: 7 + 5 + 5 + 1 = 18.

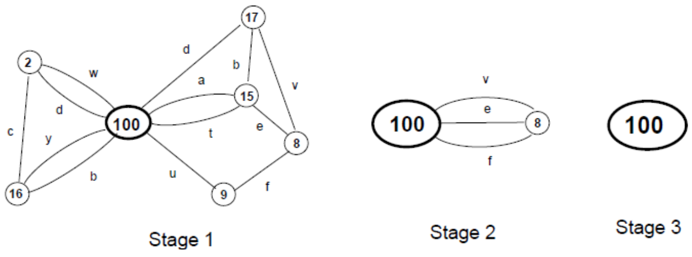

Figure 24 Repeated "black hole" modes for further network shrinking.

Gradual asynchronous self-destruction of the whole network

The main idea here is that nodes having fewer connections with other nodes than a certain threshold are considered weak and cannot exist any more, thus removing themselves from the network. The SGL solution below is hopping to all nodes only once and staying in them as long as possible until discovering the lower number of neighboring (i.e. directly connected) nodes than the established threshold, with subsequent self-destruction. In Figure 25, the three stages are shown of parallel self-shrinking of the network of Figure 10 (the fourth stage would just be the empty network), with the nodes initially or subsequently (after the neighboring nodes dying) having 3 or less neighbors ceasing to exist.

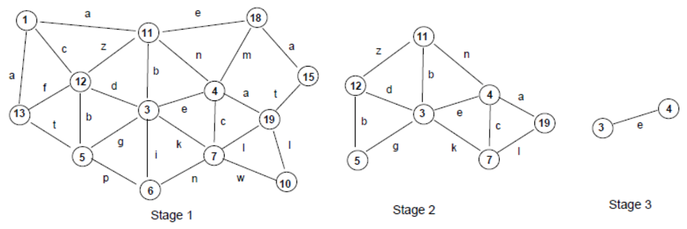

Figure 25 Gradual network self-destruction by the death of weakest nodes.

```
hop_nodes(all);
repeat(
 if(count(hop_links(all)) <= 3, remove(current));
 sleep(delay))
```

Under topologies other than of Figure 10, and also with more links between different nodes, a part or parts of the network can survive despite initially having nodes below the given threshold number of neighbors, and this threshold can also be made varying during the network dynamics.

Expanding networks

In the previous section we considered mechanisms that can provide global shrinking of networks, from their full body to ultimate naught. In the current section, we will be showing how the network can unlimitedly expand in size (i.e. the number of nodes and links) and in physical space, also imitating a sort of its symbolic explosion.

Growing by the numbers of nodes and links

We are providing here a very simple example of a possible massive expansion of the network of Figure 10 by introducing additional nodes instead of each link, which the new nodes connected with the two nodes of the substituted links by the same named links. The new nodes may also randomly establish additional links with other nodes in certain vicinity by obeying allowed distance (or radius) in the explored region, as shown in Figure 26 and expressed by the following parallel SGL scenario.

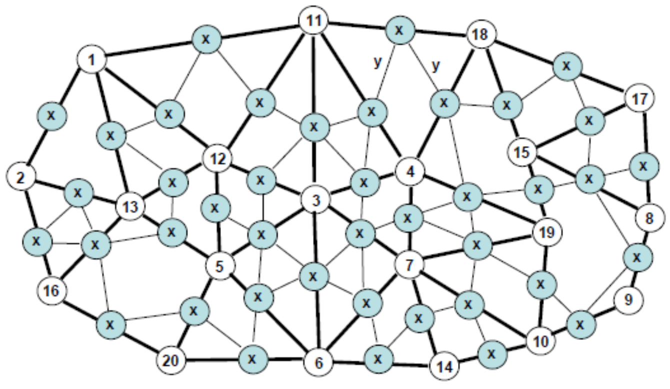

Figure 26 Massive network expansion with a symbolic "dark matter" effect.

```
hop_nodes(all); frontal(Start = NAME; Radius = maxdistance);
hop_links(all); Start > NAME; frontal(Link) = LINK;
remove(LINK); create(link(Link), node(x));
parallel(
 linkup(Link, node(Start)),
 linkup(y, random_nodes(Radius)))
```

In this elementary example of massive network expansion, we assume that all new nodes and all new links from them are same named, respectively as x and y. In Figure 26, links with previous names that directly connected nodes of the network of Figure 10 are shown in bold. After adding more semantics to this simplified network extension example the names and contents of new nodes and links may be quite different.

Using same ideas as above of the network extension, we may now substitute again all links between nodes by new nodes with establishing new links with other nodes, and so on, thus providing endless and unlimited extension, actually *explosion*, of the network of Figure 10. The following SGL scenario is based on the previous one by just repeating it certain number of times, here 50 (for prevention of unlimited explosion), with all new nodes and links, for simplicity, again named x and y. The scenario uses synchronized global repetition controlled from outside.

```
repeat_50(
   stay(hop_nodes(all); frontal(Start = NAME, Radius = maxdistance);
     hop_links(all); PREVIOUS > ADDRESS; frontal(Link) = LINK;
```

```
        remove(LINK); create(link(Link), node(x));
        parallel(
          linkup(Link, node(Start)),
          linkup(y, random_nodes(Radius)));
    sleep(delay))
```

Asynchronous internal unlimited self-growth, only first time contacting all nodes from outside, while further extending from the new nodes only, can be achieved by the following recursive scenario procedure named Blow.

```
frontal(Blow) =
 {frontal(Start = NAME, Radius = maxdistance);
  hop_links(all); PREVIOUS > ADDRESS; frontal(Link) = LINK;
  remove(LINK); create(link(Link), node(x));
  stay_parallel(
    linkup(Link, node(Start)),
    linkup(y, random_nodes(Radius)));
  sleep(delay); run(Blow)};
hop_nodes(all); run(Blow)
```

Network expansion in physical space

We can assume that the growing number of network nodes and links can be naturally linked with network's expansion in physical space too. The gradual expansion of the net in physical space can be organized as follows. If randomly found possible new location of a node (using the Radius-like threshold distance which can also grow during network's physical expansion) increases summary distance to other nodes directly connected with it, or at least minimal distance to them, the current node may change its physical coordinates in space to the new location found. Placing such rules into all nodes of the growing net, which can operate repeatedly all the time regardless of success or failure of current attempts to change physical position, may be done by the following scenario.

```
hop_nodes(all);
repeat(
  Radius = max(hop_links(all); distance(BACK, WHERE));
  Sum = add(hop_links(all); distance(BACK, WHERE));
  Min = min(hop_links(all); distance(BACK, WHERE));
  New = (move_random(Radius); WHERE);
  Sum1 = add(hop_links(all); distance(New, WHERE));
  Min1 = min(hop_links(all); distance(New, WHERE));
  if(or_seq(Sum1 > Sum, Min1 > Min), WHERE = New);
  sleep(delay))
```

Combining this physical extension with the previously considered network growth by the number of nodes and links, we can effectively simulate unlimited expansion, even explosion, of the network in both virtual and physical environments, actually covering the whole universe. This can be clearly expressed by the following SGL scenario using procedures Blow and Spread.

```
frontal(Blow) =
 {frontal(Start = NAME,
     Radius = average(hop_links(all); distance(BACK, WHERE)));
  hop_links(all); PREVIOUS > ADDRESS; frontal(Link) = LINK;
  remove(LINK); create(link(Link), node(x, random(Radius)));
  stay_parallel(
    linkup(Link, node(Start)),
    linkup(y, random_nodes(Radius)));
  sleep(delay); parallel_run(Blow, Spread)};
frontal(Spread) =
 {repeat(
    Radius = max(hop_links(all); distance(BACK, WHERE));
    Sum = add(hop_links(all); distance(BACK, WHERE));
    Min = min(hop_links(all); distance(BACK, WHERE));
    New = (move_random(Radius); WHERE);
    Sum1 = add(hop_links(all); distance(New, WHERE));
    Min1 = min(hop_links(all); distance(New, WHERE));
    if(or(Sum1 > Sum, Min1 > Min), WHERE = New);
    sleep(delay))};
hop_nodes(all); parallel_run(Blow, Spread)
```

In establishing new links of a node with other nodes, we are regularly updating the considered depth of their vicinity by recalculating the value of Radius, as the network itself is constantly expanding in physical space.

In our simple example, the shadowed new x nodes (Figure 26) may symbolically look like imitating a sort of "Dark Matter"[51] of the universe. This matter by the above scenario will, however, quickly dominate the whole network as the latter grows both virtually and physically only due to the increase of the number of shadowed nodes, with other, initial, nodes remaining in the same quantity.

Conclusions

In this paper we have shown how different operations on general networks can be described and implemented in fully distributed and highly parallel mode using the developed Spatial Grasp model and Technology and its basic spatial Grasp Language, SGL. The obtained experience of using SGT and SGL and shown exemplary solutions on networks may be useful for solving different problems in many important areas reviewed at the beginning of the paper, most of which can be conveniently formulated on distributed dynamic networks. These solutions in SGL proved to be simple and concise as the model and language allow us to directly exist and operate in distributed spaces by expressing top level problem semantics, with hiding numerous traditional system routines inside effective networked technology implementation.

At first sight, SGT and SGL may have some philosophical and conceptual resemblance to physical phenomenon like waves[52,53] (the ancestor versions of SGL were named as WAVE[43–45]), also to biological and computer viruses[54,55] and what is called "mobile agents".[56–58] Yes, SGL allows us to freely move in distributed spaces in a highly parallel mode, but it also readily provides, if needed, the return of any remote results directly to any previous space points, with their analysis and possible launching of new waves there, or the return to already gained remote space positions and further wavelike development from them, and so on. With such forward-backward recursive mode this is effectively covering and controlling any distributed systems with any power, to any depth, and by any hierarchy needed. Moreover, after and even during space coverage in recursive SGL mode, arbitrary complex and active infrastructure may be explicitly or implicitly embedded into the distributed world fabric (like openly, on agreements, or in a stealth mode for special applications). With these infrastructures effectively modeling any other concepts and models (like Petri nets or neural networks) and also capable of launching themselves new parallel waves, and so on, SGT may provide the most powerful paradigm and technology for conquering and ruling of the universe.

SGT continues its development in different areas, including advanced mosaic-type operations in distributed systems,[59] in trying to understand and simulate such extremely complex features as awareness and consciousness,[60] in providing philosophical and technological support of space conquest and advanced terrestrial and celestial missions,[61] and many others. Among the latest publications related to other SGT applications,[62–63] can be named. The latest SGL version can be implemented even within standard university environments, similar to its previous versions in different countries under the author's supervision. The technology can be installed in numerous copies worldwide and deeply integrated with any other systems, actually acquiring unlimited power for simulation and management of the whole world.

Funding

None.

Acknowledgments

None.

Conflicts of interest

The author declares that there was no conflict of interest.

References

1. Kardulias PN. World systems theory. *Encyclopedia of Archaeology*. 2008:2219–2221.
2. Qadri B. Understanding Dynamics of Modern World. *Research Gate*. 2018.
3. Burke, Anthony, Parker, et al. Global Insecurity, Futures of Global Chaos and Governance. *Pakgrave Macmillian*. 2017.
4. World Social Report 2020: Inequality in a Rapidly Changing World. 2020.
5. Malley R. 10 Conflicts to Watch in 2020. *International crisis group*. 2019.
6. Wade G. Environmental Threats Dominate 2020 Global Risks Report for the First Time in History. 2020.
7. Markovitz G. Top risks are environmental, but ignore economics and they'll be harder to fix. *World Economic Forum*. 2020.
8. Farand C. Climate change tops risks for world in 2020 – Davos report. *Climate Home News*. 2020.
9. Lai AY. Organizational Collaborative Capacity in Fighting Pandemic Crises: A Literature Review From the Public Management Perspective. *Asia-Pacific Journal of Public Health*. 2012;24(1):7–20.
10. Hamal PK, Dangal G, Gyawnli P, et al. Let Us Fight Together against COVID-19, Pandemic. *Journal of Nepal Health Research Council*. 2020;18(46).
11. Global Simulation System – a global scale, high granularity, stock and flow model. *Sympoetic*.
12. Hoffman R, McInnis B, Bunnell P. Simulation Models for Sustainability. *Sympoetic*.
13. Bostrom N. Are You Living in a Computer Simulation?. *Philosophical Quarterly*. 2003;53(211):243–255.
14. Hayatnagarkar HG. Simulated Reality - Do we live in a simulation?. *Research Gate*. 2016.
15. Wallentin G. Spatial Simulation. *Unigis*.
16. Page CL, Bousquet F. Multi-level spatial simulation. *The Seventh Conference of the European Social Simulation Association ESSA 2011*. 2011.
17. Sullivan DO, Perry GLW. Spatial Simulation: Exploring Pattern and Process. *Wiley online library*. 2013.
18. Darvishi M, Ahmadib G. Validation techniques of agent based modelling for geospatial simulations. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2014;XL-2/W3:91–95.
19. Fujimoto R. Parallel and Distributed Simulation. *Proceedings of the 2015 Winter Simulation Conference*. 2015.
20. Anagnostou A, Taylor SJE. A distributed simulation methodological framework for OR/MS applications. *Simulation Modelling Practice and Theory*. 2017;70:101–119.
21. Angelo GD. Parallel and Distributed Simulation from Many Cores to the Public Cloud (Extended Version). *Proceedings of the International Conference on High Performance Computing and Simulation*. 2011.
22. Frohn C, Ilov P, Kriebel S, et al. Distributed Simulation of Cooperatively Interacting Vehicles. *International Conference on Intelligent Transportation Systems*. 2018.
23. Sauerborn GC. The Distributed Interactive Simulation (DIS) Lethality Communication Server Volume I: Overview. *Army research laboratory*. 1999.
24. Topçu OH Oğuztüzün. Guide to Distributed Simulation with HLA. *Springer*. 2017.
25. Hakir P. Berthou TG. Addressing the Challenge of Distributed Interactive Simulation With Data Distribution Service. *Cornell University*. 2010.
26. Straßburger S. Overview about the High Level Architecture for Modelling and Simulation and Recent Developments. *Research Gate*. 2006.
27. Allen GW, Lutz R, Richbourg R. Live, Virtual, Constructive, Architecture Roadmap Implementation and Net-Centric Environment Implications. *ITEA Journal*. 2010;31:355–364.
28. Live, Virtual, Constructive Simulation Software. Scalable Network Technologies.
29. Teng TH, Tan AH, Teow LN. Adaptive computer-generated forces for simulator-based training. *Expert System Applications*. 2013;40(18):2013.
30. Defense Primer: Department of the Army and Army Command Structure. *Congressional Research Service*. 2020.
31. Burgess A, Fisher P. A Framework for the Study of Command and Control Structures. *DSTO Defence Science and Technology Organisation*. 2014.
32. Heartfield SM. Understanding the Chain of Command in Your Workplace. Human Resources Glossary. 2018.
33. Thoughts About C4I Systems; Blog on C4ISR systems development and maintenance. *CC BY 3*. 2013.

34. Girvan C. What is a virtual world? Definition and classification. *Springer*. 2018.

35. Bartle RA. Designing Virtual Worlds. *New Riders*. 2004.

36. Bell MW. Toward a Definition of "Virtual Worlds". *Journal of Virtual Worlds Research*. 2008;1(1).

37. https://en.wikipedia.org/wiki/Real_life

38. Thatje S. The Science of Nature. *Research Gate*. 2009.

39. Diamond A. Understanding executive functions: What helps or hinders them and how executive functions and language development mutually support one another. *Research Gate*. 2014.

40. https://en.wikipedia.org/w/index.hp?title=Special:Search&search=intitle%3A%22Executive%22&ns0=1

41. Sapaty PS. A Distributed Processing System. *European Patent Office*. 1993.

42. Sapaty PS. Mobile Processing in Distributed and Open Environments. *Wiley online library*. 1999.

43. Sapaty PS. Ruling Distributed Dynamic Worlds. *Wiley online library.* 2005.

44. Sapaty PS. Managing Distributed Dynamic Systems with Spatial Grasp Technology. *Springer*. 2017.

45. Sapaty PS. Holistic Analysis and Management of Distributed Social Systems. Springer. 2019.

46. Sapaty PS. Complexity in International Security: A Holistic Spatial Approach. *Emerald Publishing*. 2019

47. Uzan JP. The Big-Bang Theory: Construction, Evolution and Status. *L'Univers, S´eminaire Poincar´e*. 2015:1–69.

48. Penrose R. Genzel R, Ghez A. Theoretical foundation for black holes and the supermassive compact object at the Galactic centre, *Nobel Prize in Physics*. 2020.

49. Schaf J. The Nature of Dark Matter and of Dark Energy. *Journal of Modern Physics*. 2015;6(03):224–238.

50. Dervić K, Sinik V, Despotovic ZV. Basics of Electromagnetic Radiation. *IX International Conference Industrial Engineering and Environmental Protection*. 2019.

51. Caon M. Waves, Light Waves, Sound Waves and Ultrasound (The Physics of). *Examination Questions and Answers in Basic Anatomy and Physiology*. 2018;493–510.

52. Villarreal LP. Are Viruses Alive?. *Scientific American*. 2008.

53. https://us.norton.com/internetsecurity-malware-what-is-a-computer-virus.html

54. Baumann J, Hohl F, Rothermel K, et al. MOLE: A mobile agent system. *Journal of Software: Practice and Experience*. 2002;32(6):575–603.

55. Jezic G, Chen-Burger YHJ, Kusek M. Šperka R et al. Agents and Multi-agent Systems: Technologies and Applications 2019. *13th KES International Conference KES-AMSTA-2019 St. Julians, Malta, June 2019 proceedings*. 2019.

56. Braun P, Rossak WR. Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit. *Morgan Kaufmann*. 2005.

57. Sapaty PS, Mosaic warfare: from philosophy to model to solutions. *International Robotics & Automation Journal*. 2019;5(5):157–166.

58. Sapaty PS. Symbiosis of Real and Simulated Worlds Under Global Awareness and Consciousness. 2020.

59. Sapaty PS. Integral spatial intelligence for advanced terrestrial and celestial missions. *3rd International Conference and Exhibition on Mechanical & Aerospace Engineering*. 2015.

60. Sapaty PS. Advanced terrestrial and celestial missions under spatial grasp technology. *Aeronautics and Aerospace Open Access Journal*. 2020;4(3).

61. Sapaty PS. Spatial Management of Distributed Social Systems. *Journal of Computer Science Research*. 2020;2(3).

62. Sapaty PS. Towards Global Nanosystems Under High-level Networking Technology. *Acta Scientific Computer Sciences*. 2020;2(8).

63. Sapaty PS. Symbiosis of Distributed Simulation and Control under Spatial Grasp Technology. *SSRG International Journal of Mobile Computing and Application*. 2020;7(2).