Loading Networks in Construct

Brian R. Hirshman, Kathleen M. Carley, Michael J. Kowalchuck

July 26, 2007
CMU-ISRI-07-116

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Construct is a multi-agent simulation tool that is commonly used to investigate dynamic behavior in complex socio-cultural systems. This technical report discusses the networks available in Construct, and how each of these networks may affect simulation behavior. It also describes the parameters necessary to define networks in the simulation, including networks deriving from an external source. This document is intended primarily as a reference guide, and assumes at least some familiarity with the Construct modeling environment.

This work was supported in part by the Office of Naval Research (ONR N00014-06-1-0921, N00014-06-1-0104 and N00014-06-1-0772), the National Science Foundation (SES-0452487), the Army Research Lab (DAAD19-01-2-0011 ARL CTA 6.1 and DAAD19-01-2-0009 ARL CTA), and the AirForce Office of Sponsored Research (MURI: Cultural Modeling of the Adversary, 600322) for research in the area of dynamic network analysis. Additional support was provided by CASOS - the center for Computational Analysis of Social and Organizational Systems at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the National Science Foundation, the Army Research Lab or the U.S. government.

Version 1.0

Keywords: construct, multi-agent simulation, dynamic network analysis, social network analysis, agents, agent modeling

Contents

1 Introduction & Motivation

In today's complex world, it is useful to understand how social networks will evolve and change. Construct, a dynamic network evolution tool, can help sociologists and policy makers model this evolution by predicting how social behavior changes in response to the spread of information among individuals [1, 2]. Past work using Construct has examined the formation of work groups, friendship networks, communication networks, and organizational social structure in response to information diffusion [1]. Future work using Construct will continue in these areas and expand into other areas as well, as simulation tools such as Construct become increasingly important tools for understanding society.

Though Construct is an *agent-based* model, meaning that it consists of a series of computational entities, its *agents* are highly interrelated [1, 3, 4]. Construct agents can represent the components of social network at various levels of fidelity; in some models, agents represents individuals in an interpersonal network, while in others agents represent teams, organizations, or corporations [1]. Networks play a key role in linking these agents together. Construct agents are embedded in agent-to-agent relationship networks, networks that help define the interaction among the various simulation actors. Additionally, agents are also members of meta-networks that relate agents to knowledge, tasks, and beliefs [5]. Together, these networks help to define how agents select interaction partners, communicate, and perform various functions during the simulation.

This technical report focuses on the creation and initialization of networks within Construct. Networks in Construct represent relationships among various aspects of the simulator, and most are user-customizable prior to simulation start. Networks have been studied by a broad range of academics, from sociologists to mathematicians, and numerous types of networks have been studied in detail. Construct attempts to leverage this past work by facilitating the creation or loading of networks with desired properties. While this report does not explicitly survey past work in network creation, Construct users may find it useful to review past literature on networks.

The primary goal of this technical report is to provide additional detail on the networks available and how to configure them in order to configure Construct virtual experiments. To this end, the remainder of this document is organized as follows. Section 2 provides a brief overview of the networks in Construct. Section 3 describes the types of network generators available in Construct. Section 4 provides additional detail about the use of these generators, and how to configure the generators when running Construct in batch mode. Section 5 concludes.

2 Networks in Construct

At its core, Construct consists of two important entities: agents and networks. Construct agents store information, perform tasks, hold beliefs, and are generally what are measured at the conclusion of the simulation; their behavior is explained in detail in other technical reports [4]. Networks hold information about the relationship between agents and various other resources in the system: for instance, the set of agents with which an agent can communicate is represented as a network. All of the networks specified in this section are set using the generator mechanisms described in section 3.

2.1 Agent-agent networks

Construct has a number of agent-to-agent networks that help define interaction patterns between agents. Agent-to-agent networks represent relationships between pairs of agents, and their values are set using an agent-by-agent matrix where the row agent is the sender and the column agent the recipient. While the below section introduces the interaction sphere, socio-demographic proximity, physical proximity, and social proximity networks, the function of these networks are more fully described in previous technical reports, especially CMU-ISRI-07-107 "Specifying Agents in Construct".

- *Interaction sphere*: The interaction sphere network specifies the potential interaction partners of another agent. In most simulations, it is not desirable (or even computationally practical) for an individual agent to potentially interact with all the other agents in the simulation. Instead, an agent's interaction sphere represents the pared-down group of agents from which an agent can select an interaction partner. All values in the interaction sphere should be binary; values of one represent a link between two agents and values of zero represent no link. Though the agent sphere is initially created with the generators specified in later sections, Construct contains a number of non-generator mechanisms (not covered in this technical report) to modify the interaction sphere as the simulation is running.

- *Socio-demographic proximity*: The socio-demographic proximity network specifies the socio-demographic similarity between two agents. Under the principle of homophily, the underlying driving force behind Construct [1], agents prefer to interact with agents that have higher socio-demographic similarity to themselves. All other factors equal, agents will have a higher propensity to interact with agents that have higher socio-demographic proximity scores. A number of special generators, including those specified in section 3.3, are designed to facilitate the generation of reasonable yet random socio-demographic similarity networks. Values in the socio-demographic proximity matrix can be and often are real numbers; values of zero represent no socio-demographic overlap and a low probability of interaction while values of one represent complete attribute overlap. In the current implementation of Construct, socio-demographic proximity does not change with time, and it cannot be easily modified once the simulation is running.

- *Physical proximity*: The physical proximity network specifies the physical similarity between any two agents. All other factors equal, agents will have a higher propensity to interact with agents that have higher physical proximity scores. At this time Construct does not have a generator that will link geo-spacial data to physical proximity; however, future versions of the tool may be able to convert geo-spacial coordinates into physical proximities directly. Values in the physical proximity matrix can be and often are real numbers; values of zero represent no physical proximity (infinite distance) between two agents, while values of one represent agents who are located on top of each other. In the current implementation of Construct, physical proximity does not change with time, and it cannot be easily modified once the simulation is running.

- *Social proximity*: The social proximity network is a catch-all network that captures any other similarity factors. Often, social proximity is used as a way of representing un-modeled simulation knowledge facts; often, it can serve as a convenient method for decreasing the likelihood that social network agents will interact with an intervention when they are otherwise drawn to the interventions via knowledge homophily. All other factors equal, agents will prefer to interact with those agents with which they are more similar. Great care should be taken using the social proximity network, and it should only be used for well-justifiable and clear-cut reasons. Values in the social proximity matrix can be and often are real numbers; values of zero represent no overlap while values of one represent maximal proximity. In the current implementation of Construct, social proximity does not change with time, and it cannot be easily modified once the simulation is running.

2.2 Fact-related networks

Construct agent-to-fact networks help determine how agents know, learn, and forget facts. Agent-to-fact networks represent relationships between agents and facts, and their values are set using an agent-by-fact matrix where the row agent is associated with the column fact. While the below section introduces the initial knowledge matrix and the knowledge priority, knowledge interaction weight, and knowledge transmission weight networks, the function of many of these networks are more fully described in previous technical reports, especially CMU-ISRI-07-107 “Specifying Agents in Construct”.

- *Initial knowledge*: The initial knowledge matrix specifies the initial distribution of knowledge in the social network. If an agent knows (or partially knows) a fact, it is specified in this network. Once the simulation begins, this network will change as agents learn facts, forget facts, and modify their beliefs. Knowledge is used in the calculation of knowledge similarity and expertise, and helps to drive the simulation as agents modify their behavior as they learn new knowledge. Knowledge values can be real values if the appropriate simulation parameter is set; values close to zero represent no knowledge while values close to one represent complete knowledge. Though the initial knowledge matrix is initially created with the generators specified in later sections, Construct contains a number of non-generator mechanisms (not covered in this technical report) to modify agent knowledge as the simulation is running.

- *Knowledge priority*: The knowledge priority matrix specifies the importance of particular facts for particular agents. When communicating, agents will select only among facts of the highest priority level, and will only select facts from the next-highest level if they have communicated all facts at the one above. Often, knowledge priority is used to guarantee that an intervention has a specific effect; while general facts can be given to the intervention to increase knowledge homophily with a particular subset of agents, the specific intervention facts can be given a high priority to ensure that those facts (and not the homophily facts) are communicated. Knowledge priority does not affect interaction partner selection in any way, and only becomes a factor once one has been selected. Great care should be taken using the knowledge priority network, and it should only be used for well-justifiable and clear-cut reasons. Knowledge priority must be integer-valued, and the priority values are unbounded; by convention, the lowest priority level is set as one and facts with increasingly higher priority are set accordingly higher. In the current implementation of Construct, knowledge priority does not change with time, and it cannot be easily modified once the simulation is running.

- *Knowledge interaction weight*: The knowledge interaction weight network specifies the weight that an agent will place on a fact when calculating knowledge similarity and knowledge expertise, two critical factors for choosing an interaction partner. When selecting an interaction partner, an agent may value certain facts more than others and may even not value some facts at all. In simulations involving a complex phenomena, the phenomena facts may be oversampled relative to the other facts in society; to compensate, the knowledge interaction weight can be adjusted to ensure that agents are not overly predisposed to interact with other agents due to overlap on the phenomena alone. Knowledge interaction weight does not affect fact communication in any way, and ceases to be a factor once an interaction partner is selected. Knowledge interaction weight must be integer-valued, and the priority values are unbounded; by convention, the lowest interaction weight level is set as one and increasingly higher weights are set using larger integers. Note that the interaction weight is the same when used in similarity and expertise calculations; at this time, it is not possible to change the weight to be larger in one and smaller in the other. In the current implementation of Construct, knowledge interaction weight does not change with time, and it cannot be easily modified once the simulation is running.

- *Knowledge transmission weight*: The knowledge transmission weight network specifies the weight that an agent will place on a fact when selecting a fact for transmission. When selecting a fact to send to another agent, agents may value certain facts more than others and may even not value some facts at all. In simulations involving complex phenomena, communication of phenomena facts may be under-sampled unless agents are made more predisposed to communicate phenomena facts; to compensate, the knowledge transmission weight can be adjusted to ensure that agents are more likely to transmit phenomena facts. Knowledge priority does not affect interaction partner selection in any way, and only becomes a factor once one has been selected. Great care should be taken using the knowledge priority network, and it should only be used for well-justifiable and clear-cut reasons. Knowledge transmission weight must be integer-valued, and the priority values are unbounded; by convention, the lowest transmission weight is set as one and increasingly higher weights are set using larger integers. In the current implementation of Construct, knowledge transmission weight does not change with time, and it cannot be easily modified once the simulation is running.

2.3 Task-related networks

Construct task networks help determine properties about tasks and how they are performed by agents. Fact-to-task networks represent relationships between tasks and facts, where the relationship is set by associating the row fact and the column task. Agent-to-task networks represent relationships between agents and tasks, and their values are set using an agent-by-task matrix associating the row agent with the column task. The below section introduces the task requirement matrix and the task truth matrix, two networks that are only briefly summarized in CMU-ISRI-07-107 “Specifying Agents in Construct”; a third network, the agent assignment network, is also covered here but is expanded upon in the agent tech report.

- *Requirement*: The task requirement matrix specifies the facts that are required to perform a task. When an agent performs a task, it only needs to use a subset of its knowledge to perform the task; facts that are not required are irrelevant and are ignored in the calculation mechanism. The facts that are required are then used to compute whether or not the agent knows enough information to complete the task. In the current implementation of Construct, the task requirement matrix does not change with time, and it cannot be easily modified once the simulation is running.

- *Truth*: The task truth matrix specifies the ground truth for the required facts in a binary task. When an agent performs a task, it compares its task knowledge to the knowledge needed when performing the task. The truth network specifies the facts that are necessary in order to complete the task. When performing the task, if an agent knows a particular fact then is able to use that fact. If an agent does not know (or partially knows) a fact, then the agent guesses. If an agent knows a sufficient number of task-related facts, or guesses correctly on the facts that it does not know, then the agent is able to successfully complete the task. In the current implementation of Construct, the task truth matrix does not change with time, and it cannot be easily modified once the simulation is running.

- *Agent assignment*: The agent task assignment network specifies which agents are associated with which tasks. Agents can perform one or multiple tasks at once, and the task assignment matrix indicates the tasks with which an agent is associated. Tasks can be organized in hierarchies such that agents must perform a small subtask before beginning work on a general task. Additionally, agents can be automatically assigned to tasks by an internal assignment algorithm that focuses primarily on agent knowledge expertise for a particular task. In the current implementation of Construct, task assignment can (and often does) change with time, though agents can be permanently assigned to specific tasks if that is the purpose of the simulation.

2.4 Belief-related networks

Construct belief networks help determine how agents come to their beliefs. Fact-to-belief networks represent the relationships between facts and beliefs, the facts that influence particular beliefs by associating the row belief with the column fact. Agent-to-belief networks represent relationships between agents and beliefs, and are inferred from the agent's initial knowledge. While the below section introduces the fact belief weight network and touches on the initial belief network, beliefs are more fully described in previous technical reports, especially CMU-ISRI-07-107 "Specifying Agents in Construct".

- *Fact belief weight*: The belief weight network specifies the belief weights associated with particular facts. Some facts will influence particular beliefs, and some facts can influence more than one belief. Other facts may not influence beliefs at all. Belief weights can be set to real numbers between -1.0 and 1.0, where the former represents complete disagreement and the latter complete agreement. While it is often common to set belief facts to the two extremes, belief weights are real numbers so some facts can have a minor (but by no means insubstantial) effect on agent beliefs. In the current implementation of Construct, the fact belief weight does not change with time, and it cannot be easily modified once the simulation is running.

- *Initial beliefs*: The initial beliefs network is not a network in the sense of the other networks described in this technical report, but is included here for completeness. The initial beliefs network is computed by examining an agent's knowledge and computing an agent's beliefs based upon an agent's knowledge and the associated fact belief weight. The initial beliefs network is not defined exogenously – it cannot be set directly using the generators below – so for agents to have particular beliefs it is necessary to assign them the necessary belief facts and allow the simulator to determine the beliefs based on the facts. During the simulation, it is very common for agents to learn new facts and modify their initial beliefs, or to be affected by the beliefs of other agents. Such patterns, however, are by no means guaranteed.

2.5 Time-dependent networks

While most Construct networks are constant over the course of the virtual experiment, some networks will change over time. Time-dependent networks represent relationships between various parameters and time, and are highly dependent upon the simulation period. Most of these networks are configured with the parameter on the row axis and time on the column one. While the below section introduces the agent active time periods, turnover, proximity weights, knowledge similarity weight, and knowledge expertise weight network, some of these networks are more fully described in previous technical reports, especially CMU-ISRI-07-107 "Specifying Agents in Construct".

- *Agent active time periods*: The agent active time periods matrix specifies the time periods in which agents will interact. When agents are inactive, they will neither initiate nor receive communication, neither update beliefs nor influence the beliefs of others, and neither influence resources nor perform tasks. When an agent is inactive, it is as if the agent did not exist during the particular time period. When an agents become active again, they suffer no penalty for being inactive, though their transactive memory may be incorrect because the network changed in their absence. It is extremely common to make interventions active for only a few simulation periods, especially if their effects are meant to influence the social network and not to influence all agents directly. The agent active time periods matrix consists of binary values, where the value is one if the agent is active and zero if it is not. The agent active time periods matrix must be defined and fixed before the experiment starts; at this time, it is not possibly to isolate agents dynamically based on characteristics and knowledge accrued during the simulation.

- *Turnover*: The agent turnover matrix specifies the time periods at which agents leave the system and are replaced by new agents. When an agent is replaced, its knowledge is cleared and its beliefs are reset; however, its other networks are left unchanged. The syntax of this matrix is somewhat confusing, so it should be initialized with care. The row value is the time period at which the turnover event will occur, and the column value is the agent that is replaced; this is different than most other networks. If a value is present at (row, column), then a turnover event is guaranteed to occur; the value at (row, column) specifies the fact density that should be used when re-initializing the knowledge of the agent. This may appear confusing if the agent turnover matrix is printed, as the floating point values in the matrix do not indicate probabilities. When reinitializing agent knowledge, it is not possible to specify that an agent knows a particular fact or to specify different fact densities for specific regions. Future versions of Construct may modify this, or to completely overhaul the way the turnover matrix is implemented in order to make it possible to fully re-generate a replacement agents.

- *Proximity weights*: The agent proximity weights specify the weights for the various proximity measures specified above in section 2.2. The three types of proximity – socio-demographic proximity, physical proximity, and social proximity – all affect how agents select interaction partners. Agents with high weights in a particular category will place more emphasis on agents which share that characteristic. Some experimenters may wish to have these values fluctuate during the experiment, for instance to represent the advent of new technologies like the Internet which can mitigate the effect of physical distance. Alternatively, experimenters may use the generators described in later sections to set the weights to have a standard deviation around an average value. Proximity weights are set using real values between zero and one; low values indicate decreased importance and high values indicate increased importance for that proximity measure. The weight values must be specified at the beginning of the experiment and cannot be updated dynamically as the experiment is running. Note that the sum of these three proximity values, the knowledge similarity weight, and the knowledge expertise weight must sum to one over each time period, so setting these values to high numbers necessarily decreases the values to which the others can be set.

- *Knowledge similarity weight*: The knowledge similarity weight parameter specifies the value that agents will place on knowledge similarity. An agent with a high knowledge similarity weight will prefer to interact with agents who have the same knowledge that it does, and this may lead it to interact more frequently with agents with whom it has already interacted. Because the knowledge similarity weight can change over time, experimenters can allow this value to fluctuate around a mean or to change over time. Knowledge similarity weights are set using real values between zero and one; low values indicate decreased importance and high values indicate increased importance for that measure. The weight values must be specified at the beginning of the experiment and cannot be updated dynamically as the experiment is running. Note that the sum of the three proximity weights, this weight, and the knowledge expertise weight must sum to one.

- *Knowledge expertise weight*: The knowledge expertise weight parameter specifies the value that agents will place on knowledge expertise. An agent with high knowledge expertise weight will prefer to interact with agents that have knowledge that it does not have, and this may lead it to interact more frequently with new agents. Knowledge similarity weights are set using real values between zero and one; low values indicate decreased importance and high values indicate increased importance for that measure. Because the knowledge expertise weight can change over time, experimenters can allow this value to fluctuate around a mean or to change over time. The weight values must be specified at the beginning of the experiment and cannot be updated dynamically as the experiment is running. Note that the sum of the three proximity weights, this weight, and the knowledge expertise weight must sum to one.

2.6 Agent-only networks

Construct has a number of networks that are agent-specific. The dimensions of these networks are agent x 1, meaning that these networks have different values for each agent but are constant across the other dimension. While these may be thought of as agent properties, they are networks inasmuch as they are set via the generator mechanism specified in section 3. While the below section introduces the influentialness, influenceability, learning rate, forgetting rate, learning-by-doing rate, misrepresentation rate, and selective attention networks, the function of these networks are more fully described in previous technical reports, especially CMU-ISRI-07-107 "Specifying Agents in Construct".

- *Influentialness*: The influentialness network specifies how strongly the belief of one agent will affect the belief of another. Agents with high influentialness values will have a much stronger effect on other agents; such agents can be thought of, or deliberately modeled to be, opinion leaders and trend setters. In the current version of Construct, influentialness is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same influentialness value with all other agents. Influentialness is specified as a real number between zero and one, where values of zero indicate low influentialness and values of one indicate high influentialness. Future versions of Construct, however, may allow the influentialness value to vary by agent or to change over time.

- *Influenceability*: The influenceability network specifies how much weight an agent will put on the beliefs of other agents. Agents with high influentialness values will be much more affected by other agents; such agents can be thought of, or deliberately modeled to be, agents that take belief cues from their surrounding social network. In the current version of Construct, influenceability is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same influenceability value with all other agents. Influenceability is specified as a real number between zero and one, where zero indicates low influenceability (more independence) and one indicates high influenceability (more likely to be swayed by the opinions of others). Future versions of Construct may allow the influenceability to vary by agent or to change over time.

- *Learning rate*: The learning rate network specifies how quickly an agents learn facts from other agents. When an agent communicates with another agent, it has the potential to learn new knowledge if the partner presents it with a new fact. Since facts are represented using real values, it is possible for an agent to only partially learn a fact via communication if the recipient's learning rate is low. If the learning rate is particularly low, then the recipient may not learn anything from the communication whatsoever. In the current version of Construct, the learning rate is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same learning rate for all facts. The learning rate is set to a real number between zero and one, where zero indicates no ability to learn and one indicates complete learning. Future versions of Construct may allow the learning rate to vary by fact or to change over time.

- *Forgetting rate*: The forgetting rate specifies how quickly an agent forgets facts that it has learned but has not communicated recently. When an agents fails to communicate a fact, it has the potential to lose a piece of knowledge. Since knowledge is represented using real values, it is possible for an agent to partially forget a fact in this fashion; however, if the fact is repeatedly ignored and the forgetting rate is set particularly high, it is possible for agents to forget a fact entirely. In the current version of Construct, the forgetting rate is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same forgetting rate for all facts. The forgetting rate is set to a real number between zero and one, where zero indicates no forgetting and a one indicates complete forgetting each period. Future versions of Construct may allow the forgetting rate to vary by fact or to change over time.

- *Learning-by-doing rate*: The learning-by-doing rate specifies how quickly an agent learns knowledge while performing tasks. A task can have facts associated with it, and agents can learn these facts as they perform the task. Agents with high learning-by-doing rates will learn the facts more rapidly, and more completely, than agents who have lower learning-by-doing rates. In the current version of Construct, the learning-by-doing rate is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same learning-by-doing rate for all facts and for all tasks. The learning-by-doing rate is set to a real number between zero and one, where zero represents an inability to learn by doing and a one indicates complete success. Future versions of Construct may allow the learning-by-doing rate to vary by fact, by task, or by time period.

- *Misrepresentation rate*: The misrepresentation rate specifies how often an agent intentionally or unintentionally does not tell the truth while performing a task. When an agent performs a binary task, it uses its knowledge to determine whether it returns true or false on a task. Due to misrepresentation, however, an agent may submit the opposite value compared to what would be expected given the agent's knowledge. Agents with high misrepresentation rates will submit the opposite vote more than agents who have lower rates; agents with a misrepresentation rate of zero will always return the value that correspond to their knowledge. Note that misrepresentation is independent of knowledge and only comes into play once agents have used their knowledge to compute a possible response to the task; the facts themselves are never misrepresented. In the current version of Construct, the misrepresentation rate is specified as a network, although it is perhaps more accurately classified as an agent property since an agent will have the same misrepresentation rate for all all tasks. The misrepresentation rate is a real value between zero and one, where zero represents no misrepresentation (tasks always performed based on knowledge) and one represents complete misrepresentation (always chosen based on the opposite of knowledge). Future versions of Construct may allow the learning-by-doing rate to vary by task or by time period.

- *Selective attention*: The selective attention parameter specifies the percentage of facts that an agent will examine as it looks for facts to communicate to another agent. When an agent examines its memory when determining facts to transmit, it will ignore a certain percentage of its known facts; the number of facts ignored is specified by the selective attention rate. The selective attention rate is technically a network, inasmuch as it varies by agent and can be set using the generators specified in section 3. However, the selective attention rate does not vary over time, and it must be the same value for all facts. Selective attention is a real number between zero and one, where zero indicates that the agents will not consider any facts at all and one indicates that agents will consider all possible knowledge. Future extensions to construct may lift these restrictions, allowing the rate to change over time or to be different for different components of agent knowledge.

2.7 Other networks

Several Construct parameters have a network-like character, but are currently implemented as constant values. Future versions of Construct are likely to update these parameters so that they can vary over time, and possibly over other attributes as well.

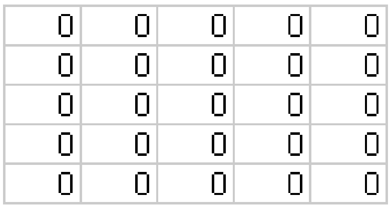
- *Communication weight*: Construct has four communication weights: communication weight for fact transmission, weight for fact transactive memory transmission, weight for belief transmission, and weight for belief transactive memory transmission. These parameters specify the broad outlines of an agent message: for instance, when the fact transmission weight is set to a particular value, agents will send facts in that percentage of all messages, regardless of how many other pieces of information they know. Thus, this parameter allows the simulation designer to specify the broad outlines of the kinds of information being transmitted in the simulation and to select appropriate figures that mirror human communication patterns in the domain being simulated. In the current implementation of Construct, communication weight values are all real-valued, user-specified constants and must be the same for all agents and for all time periods. Future versions of Construct may may allow these parameters to vary by agent or by time.

3 Generators in Construct

Construct has a number of mechanisms available for setting parameters. These mechanisms, called *generators*, allow a user to specify the values of a particular network region. Some generators must set all the values in a network at once, while other generators are capable of iteratively setting small sections of a network piece by piece. Most generators write to rectangular regions of the network, and are specified by indicating the minimum and maximum row as well as the minimum and maximum columns. If the region being generated is square, it is possible to generate a symmetric network using any of these generators.

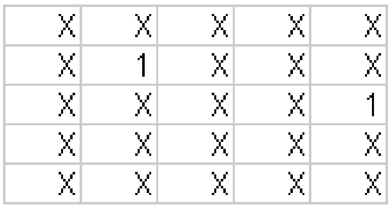
3.1 Constant generators

Construct has two generators for creating constant data. The values created by these generators will be the same over repeated Construct runs.



0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Constant value, setting all zeros



X	X	X	X	X
X	1	X	X	X
X	X	X	X	1
X	X	X	X	X
X	X	X	X	X

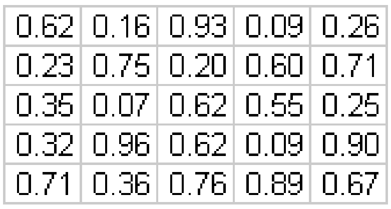
Data, setting the appropriate ones

Figure 1: Generators for Setting Constant Values

- *Constant*: The constant generator will fill a region with a specific constant value. The value can either be binary or real, depending upon the type of network required. This generator is best when rectangular regions need to be filled, or when a large number of values must be set to the specified constant. The results from a constant value generator can be seen on the left half of figure 1.
- *Data*: The data generator will also fill a region with specific constant values. However, unlike other generators, this generator can be used to set particular (row, column) values individually. This can allow the experiment designers to specify the values of a few coordinates very quickly, though it is probably unwieldy to use for specifying large regions. The results from a constant value generator can be seen on the right half of figure 1.

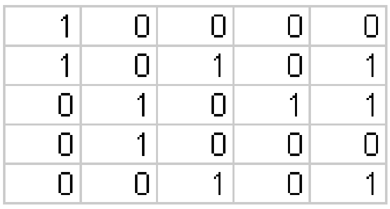
3.2 Random generators

Construct has two generators in order to create random regions. The random generators allow experiment designers to introduce additional variation into the experiment and to ensure that runs are not biased by the structure of a particular network. The regions created in this fashion are not random networks; in order to create more specific forms of random networks, it is necessary to use a more complex network generator such as those available in other CASOS tools such as ORA [6].



0.62	0.16	0.93	0.09	0.26
0.23	0.75	0.20	0.60	0.71
0.35	0.07	0.62	0.55	0.25
0.32	0.96	0.62	0.09	0.90
0.71	0.36	0.76	0.89	0.67

Random uniform, min of 0.0 and max of 1.0



1	0	0	0	0
1	0	1	0	1
0	1	0	1	1
0	1	0	0	0
0	0	1	0	1

Random binary, mean of 0.4

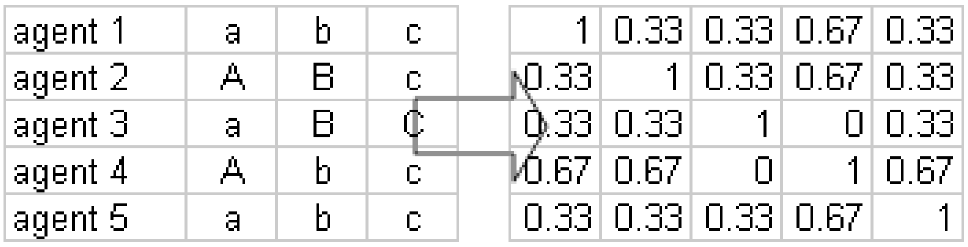
Figure 2: Generators for Setting Random Values

- *Random uniform*: The random uniform generator creates a region of real-numbered values. To specify such a network, a min and a max must be supplied; each value in the region will be set to a random value between the min and max. Though the min and max values can be any value, almost all Construct networks must have a minimum greater than or equal to zero and maximum less than or equal to one. The random numbers chosen by the random generator are chosen independently from a uniform distribution. To select random numbers from a different type of distribution, it will be necessary to generate the network outside of Construct and load it via an external generator. An example of a random uniform network is specified in the left half of figure 2.

- *Random binary*: The random binary network creates a region of binary values. To specify such a network, a cutoff value (called a mean) between zero and one must be specified; this mean specifies the mean fraction of ones in the region. To fill each value in the region, a random value is chosen from a uniform distribution and its value is compared to the mean; if the value is less than the mean a one is entered, if it is greater the value is left at zero. It is important to note that a mean percentage of ones is specified, and not a guaranteed number of ones. Thus, for small regions especially, there is no guarantee that the mean number of ones will occur in the region. Future versions of Construct may contain generators that guarantee a specific number of ones in a particular region. An example of a random binary network is specified in the left half of figure 2.

3.3 Task-specific generators

Construct has a number of specialized generators that can be used to generate networks. These generators often require additional Construct parameters or additional input matrices in order to function, so their behavior may be based on information not always specified in the generator. Often, these generators are geared toward a specific network, though they are flexible enough such that experiment designers may find other uses for them.

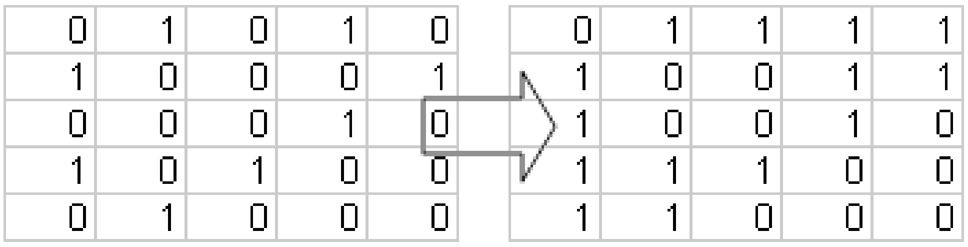


agent 1	a	b	c
agent 2	A	B	c
agent 3	a	B	C
agent 4	A	b	c
agent 5	a	b	c

1	0.33	0.33	0.67	0.33
0.33	1	0.33	0.67	0.33
0.33	0.33	1	0	0.33
0.67	0.67	0	1	0.67
0.33	0.33	0.33	0.67	1

Figure 3: Attribute Generator Operating on Binary Attributes A, B, and C

- *Attribute*: The attribute generator will generate an agent-agent network based upon the attribute similarity between two agents. Initially, all dyadic similarities between agents are initialized to zero. Then, for each agent attribute, if two agents have the same value for a particular attribute, then similarity is increased. If two agents differ on that attribute, then there will be no increase in similarity. Because agents can have multiple attributes, values created by this generator may be larger than one and often are not integer values. Most often, it will be necessary to normalize the values using the scaled generator described below. The attribute generator is most often used for generating the socio-demographic proximity matrix. An example of an attribute-specified network is specified in figure 3; the overlap over three binary attributes is calculated, and the results are normalized into the zero-one range.

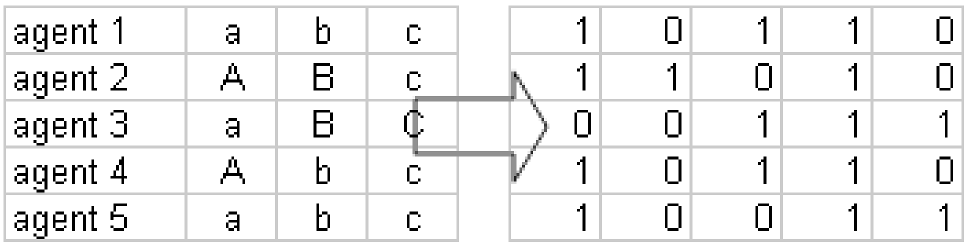


0	1	0	1	0
1	0	0	0	1
0	0	0	1	0
1	0	1	0	0
0	1	0	0	0

0	1	1	1	1
1	0	0	1	1
1	0	0	1	0
1	1	1	0	0
1	1	0	0	0

Figure 4: One-Away Generator Operating on Leftmost Network

- *One Away*: The one-away generator will generate an agent-agent network based on another agent-agent template. This generator is most often used for creating the agent interaction sphere network based on a template; for instance, if a series of historical interaction patterns is presented to Construct, the one-away generator can be used to create a hypothetical but plausible social network for evolving the social structure. For each agent in the network, the one-away network will include all agents immediately reachable from that agent via edges in the template graph. However, the network will also include all agents reachable from that first set of agents, thus including the "friends of friends" of the initial agents. In social network terms, this generator captures the ego net of size two and returns it as the one-away network. This generator is most often used for generating the interaction sphere network. An example of a one-away-specified network is specified in figure 4; the leftmost interaction matrix is used to calculate the rightmost matrix of agents that are one-away reachable.

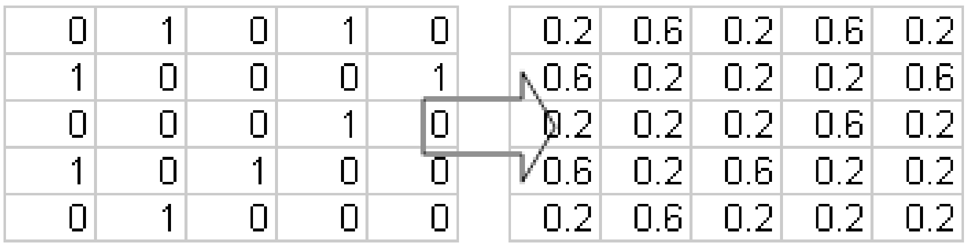


agent 1	a	b	c
agent 2	A	B	c
agent 3	a	B	C
agent 4	A	b	c
agent 5	a	b	c

1	0	1	1	0
1	1	0	1	0
0	0	1	1	1
1	0	1	1	0
1	0	0	1	1

Figure 5: Socio-Demographic Generator Operating on Binary Attributes A, B, and C

- *Socio-demographic*: The socio-demographic generator will generate an agent-agent network based on attribute similarity between two agents. Unlike the attribute generator, which specifies real values, the socio-demographic generator specifies binary values. When using the generator, the experimenter must specify a fraction of agents that have a specific number of overlapping attributes; the sum of these fractions must add up to less than one, with the potential remainder being assigned randomly. When creating the network, the generator first evaluates the number of overlapping attributes between all pairs of agents, then tries to define an interaction sphere which has the proper fraction of attribute overlaps. The generator begins by trying to select agents with the highest number of overlapping attributes, selecting agents at random from the set of all agents who meet that criteria. If there are not enough agents with a specific number of overlapping attributes (say, three), the remaining fraction is added to the number to be selected from the next lower level (say, two), and the process continues until the network is filled to the required density. It is important to note that this generator does not generate symmetric matrices by default, and the symmetrizing of generator results – while technically correct – may create a matrix of a different density. The socio-demographic generator is most often used for creating the interaction sphere matrix. An example of a socio-demographic-specified network is specified in figure 5; a density of .6 is specified and agents try and select 50% of agents with 2 overlapping attributes and 50% of agents with 1 overlapping attribute.



0	1	0	1	0
1	0	0	0	1
0	0	0	1	0
1	0	1	0	0
0	1	0	0	0

0.2	0.6	0.2	0.6	0.2
0.6	0.2	0.2	0.2	0.6
0.2	0.2	0.2	0.6	0.2
0.6	0.2	0.6	0.2	0.2
0.2	0.6	0.2	0.2	0.2

Figure 6: Scale Generator Operating on Leftmost Network

- *Scaled*: The scaled attribute generator can be used to scale another network. Some of the networks created here, as well as some inputted networks, may not be normalized between zero and one; the scaled network generator can help resolve this problem. Other experimenters may wish to scale a network to have a different minimum or maximum value in order to test different experiment assumptions; the scaled network generator can be invaluable here. The scaled generator must scale an entire network and cannot scale only a particular region. Future versions of Construct may expand upon this behavior, and allow scaling by regions and scaling by an average but non-constant amount. An example of a scale-specific network is specified in figure 6; the leftmost network is rescaled to have all values between 0.2 and 0.6

3.4 External generators

Construct has two methods of loading matrices from external sources. If a matrix exists in a different format, it is best to load that matrix into ORA and to convert it into one of these formats. While these generators are most often used to load in entire Construct networks all at once, these generators can also be used to load in sub-regions of a network and leave other regions to be specified by other networks.

- *CSV*: The CSV generator loads a region from a CSV file. CSV is a flexible, comma-delimited format which can be created by a third-party matrix generator. This generator is best used for the entry of complex data, especially data with complex patterns. In order to vary the values loaded by this generator it is necessary to manually modify the CSV file externally; Construct does not have an automatic reminder mechanism. Note that the CSV file must specify a rectangular region whose dimensions are equivalent to the size of the network or region specified in the Construct input file. Additionally, the ordering of values in the CSV file must correspond to the ordering of values in Construct: the first row of the file must correspond to the first agents, fact, or other network element, while the first column of the file must correspond to the first agent, fact, or other network element. Headers should not be included in the CSV file, as Construct is not equipped to process them and will likely treat them as data values.

- *DynetML*: The DynetML generator loads a region from a DynetML network. DynetML format may be superior to CSV for sparse matrices in terms of data storage. DynetML is the default output from other CASOS tools such as ORA and AutoMap. Note that the DynetML file must specify a rectangular region whose dimensions are equivalent to the size of the network or region specified in the Construct input file. The values specified in the DynetML file do not have to be in any particular order because the layout of the DynetML file is much less particular than the layout of the CSV file.

4 Use

The generators introduced in section 3 can be used to initialized the matrices in section 2. This can be done all at once using a full network generator as or piecemeal with a box generator. This section introduces both types of generators and highlights the difference between them.

4.1 Full-network generators

The full network generator is used when generating the entire network all at once. While this can be very powerful, it often may not be subtle enough to capture sophisticated input data. Full network generators are most often used in three places. First, full network generators are used if the full network has been defined in an external file; a full network generator can be used to load the entire file and use it. Second, full network generators are often used to initialize networks to constant values if the features defined by the network are not critical to the effects being studied. Lastly, full networks are often initialized using random values (either from a random binary or a random uniform generator) if the values are meant to be variable but should vary within the same range for all agents.

4.2 Box generators

In contrast to full network generators, box generators are used to generate a network in sections. Often, one generator is not sufficient to create a network; it may be desirable, for instance, to have a higher density in one area of a network than another. A box generator allows a network to be subdivided into sections and defined in terms of these subsection. Each subsection is rectangular, and is defined by a first row / last row pair and first column / last column pair. Each of the rectangular regions defined can be specified using a different internal generator: it is possible to define one subsection using a CSV file, and to define another section of the same network using a constant generator. Additionally, it is possible to overwrite the values of one generator with the values from another; since generators are read sequentially, it is possible to specify a region using a random binary generator and then supplement it with specific values set by a constant value generator.

In addition to the generators specified in section 2, there are several types of box generators that are not available for use as full network generators. These include the tied network generator, the diagonal network generator, and the periodic network generator.

- *tied*: The tied box generator ties the values in a specific region to the values specified in another. As with all box generators, the rows and columns of the target region are specified as parameters. To indicate the source region, the upper-left point must be specified; the dimensions of the entire region need not be. The generator then computes the dimension of the source region, given the length and width of the target region. While this approach is simple, it has two important caveats. First, changing the dimensions of the target region will implicitly change the dimensions of the source region, and Construct does not provide any warnings when this happens. Secondly, if the source region happens to overlap with the target region, the behavior of the tied generator is undefined. The tied generator can only work within the same network; it is not possible to tie the values in one matrix to values in another.
- *diagonal*: The diagonal box generator sets the values along the diagonals of a particular box. All other values in the square box are unmodified. This generator is often used to ensure that certain networks that must have specific values on the diagonals; for instance, some matrices require ones along the diagonal and this generator can ensure that this occurs. The values on the diagonal must all be set to a constant value, though future versions of the generator may allow the diagonal values to be set to a random binary or a random uniform value as well. Unless the values off of the diagonal are set with a different generator, the values in those locations will be undefined.
- *periodic*: The periodic box generator is similar to the constant value generator, but instead of setting every value in the network subregion it sets values periodically. This generator will set the first value of every row to the specified constant value, then will skip a number of columns equal to the value of the period parameter. Thus, unless the values between the periods have been set using a different generator, the values in those locations will be undefined. The period affects only columns; to skip rows, it would be necessary to specify multiple different periodic generators. Additionally, the period value must be constant, it is not possible to have a variable-length period or an ever-increasing period at this time. Future versions of Construct may modify these parameters.

With box generators, it is possible to specify multiple values for a single cell. For instance, it is often very common to use a box generator to set all values to zero, then to go through and specify a specific density for the matrix using a random binary matrix, then to set specific values using a data generator. Since data generators are read in the order that they appear in the input file, the last generator that references a particular value will be the one that sets the cell value. Note that the effects of generators are usually multiplicative, not additive: for instance, if a value is set by a binary generator with a mean of .67, and then re-set with a binary generator with a mean of .33, the cell will have a 78% chance (1 - .67 x .33) of being set to one – the generators operate sequentially, not simultaneously.

When using the box generator, some of the networks specified in section 2 have additional parameters that may be useful. When using the box generator, the symmetric attribute allows the creation of a symmetric matrix; his allows the creation of boxes that have the same values in (row, column) as in (column, row). This behavior is not yet available in the full network generator, though it may be implemented in future versions of Construct.

5 Closing Comments

This technical report has discussed the networks available in Construct and the various ways possible to define them. These networks are extremely important for the operation of the tool, and those who wish to use Construct should understand the purpose of these networks and how these values affect simulation results. While it is often difficult to perform sensitivity analysis on each network for every experiment, it is important that the experimenter justify (at least to himself or herself) the assumptions made when a specific network structure is used. Construct networks are powerful and their effects are complex, so these values should be chosen with care.

References

[1] Kathleen M. Carley. On the evolution of social and organizational networks. *Special Issue of Research in the Sociology of Organizations on Networks In and Around Organizations*, 16:3–30, 1999.

[2] Craig Schreiber, Siddhartha Singh, and Kathleen Carley. Construct - a multi-agent network model for the co-evolution of agents and socio-cultural environments. Technical report, Carnegie Mellon University School of Computer Science, May 2004.

[3] Kathleen Carley. Begining to end – construct. Unpublished Slide Presentation, Carnegie Mellon University School of Computer Science, 2005.

[4] Brian Hirshman and Kathleen Carley. Specifying agents in construct: Cmu-isri-07-107. Technical report, Carnegie Mellon University School of Computer Science, 2007.

[5] Kathleen Carley. *Smart Agents and Organizations of the Future*, volume 12, pages 206–220. Sage, Thousand Oaks, CA, 2002.

[6] Kathleen Carley and Jeff Reminga. Ora: Organizational risk analyzer. Technical report, Carnegie Mellon University School of Computer Science, 2004.