Chapter 6

Offer networks: a model of decentralized exchange

Offer networks is a work-in-progress concept of an alternative economy where heterogeneous and independent agents (humans, AIs and more or less simple programs and intelligences) find, negotiate and execute locally and globally beneficial series of hybrid exchanges of tangible and intangible goods. The initial concept was proposed by Goertzel (2015a) in the context of a post-money economy and further explored by Heylighen (2017) in the broader context of decentralized coordination, distributed intelligence and global brain research. Heylighen (ibid.) also proposed an alternative name for such general system: the *synergy web*. Our purpose of referring to offer networks in this thesis is mainly motivated by using it as a domain model for trying out the open-ended decentralized computing architecture described in Chapter 5 in terms of an actual working code and integration of various existing software frameworks. Nevertheless, we first introduce the economic context of offer networks, which is important to grasp for appreciating a case of a complex adaptive and necessarily decentralized self-organizing system.

6.1 Economic context

Established economic theories, classical and neo-classical alike, largely understate or ignore subjectivities and complexities inherent in how individual humans determine the value of goods and services and achieve their exchanges. Social structures are seldom discussed behind the notion of the market as an ideal resource-allocating mechanism in which dynamics is abstracted from the individual behaviours and preferences of its participants (Gode and Sunder, 1993; Jackson, 2006). The ideal market image is perfectly competitive, where all sellers and buyers are homogeneous or, at best, categorized into large homogeneous groups. Such a market (or exchange) is utilitarian and driven by utility maximization principles, which are objectively definable for each group. A canonical example is the financial market, which is carefully stripped of any social interaction by strict regulations. Yet it is a special case which does not apply to economic exchanges in general, nor is it a desirable model of them.

Economic sociology, a term coined in late 19th century, is "the application of the frames of reference, variables, and explanatory models of sociology to that complex

of activities which is concerned with the production, distribution, exchange, and consumption of scarce goods and services" (Smelser and Swedberg, 2005). A central concept in contemporary economic sociology is the embeddedness of economic actions in concrete and ongoing systems of social relations (Granovetter, 1985) – i.e. social networks and sub-networks. Exchange, as the mechanism of coordination of actors and owners of resources, is the outcome of their local interactions via dynamic social networks which evolve in time as a product of the totality of these interactions (Johanson and Mattsson, 1994). Note that this is clearly a case of progressive determination (see Section 3.2.4). Obviously, network science plays an important role in the research methodology of economic sociology. Markets are highly connected complex adaptive systems and understanding them requires integration of ideas for reasoning about network structures, strategic behaviours, feedback effects, reflexivity, and more. From the perspective of open-ended distributed computing, markets are seen as assemblages of economic agents and as platforms for the emergence of further sub-assemblages and local social networks. In economic sociology, markets and strategic interactions in networks can be explored by combining game theory, which focuses on interactions among independent agents, and graph theory, which focuses on their social relations (Easley and Kleinberg, 2010).

Another important assumption of "mainstream" economic theories is the rationality of market actors, including the completeness and transitivity of their preferences. In simple terms, completeness means that market actors always have clear preferences with respect to the choices that they are faced with. Transitivity then basically enables all preferences to be ordered with respect to each other. Despite plenty of evidence against the theory of rational choice, assumptions of rationality seem to be holding their ground (Mandler, 2005) at least in part because it is difficult to build a clearly interpretable model of economy and society without them.

Practically, however, people often do not know their preferences *a priori*, and these preferences are not comparable to each other or variate over time and depending on the context. Consider, for example, a flea market where sellers offer unique items that can have aesthetic, emotional and practical value. More often than not, buyers come to the market with only a vague, if any, idea of an item they are interested in. They mostly make a decision only after seeing a number of items, interacting and negotiating with sellers. Such negotiations are not only technical exercises of matching supply and demand in order to come up with the fair price (as in perfectly competitive marketplaces), but also a social interaction during which preferences themselves individuate. The agreed price of an item may even not be the main aspect of a transaction, and surely may be different depending on individual preferences of buyer, seller or simply a circumstantial result of their interaction. Such dynamics cannot be captured by traditional models of faceless buyers and sellers asking and bidding on quantities and prices of standardized goods. Summarizing, the value of an item of exchange is at least in part subjective, unique and dependent on singular expressions of immediate social interaction, not fully reflected in its monetary value. The idea of offer networks is to reclaim the subjective value of economic exchange, neglected by utilitarian efficiency-directed theories and standardised markets. In the context of large-scale digitalized markets, where the face-to-face conversations of the flea market are no longer an option, there is a need for a computational solution.

In a nutshell, the concrete computational problem addressed by experiments described here is *search and matching*. It is a well established and researched problem in labour economics (Dao, 2011), an aspect of which is matching unemployed persons'

skills with free jobs in economy. The problem of matching job openings with potential employees obviously involves more variables than the salary level. Likewise, offer networks is a call to conceive a market where exchanges are driven by matching the subjective, diverse and multidimensional values of each participant engaged in an exchange rather than reducing them to one all-permeating dimension mediated via single currency (Goertzel, 2015a). Open-ended decentralized computing allows the practical implementation of search and matching in the light of a *radical decentralization* perspective as a process of bottom-up local self-organization, not unlike how real world markets operate. Furthermore, it exposes numerous aspects of fundamental distinction between *centralized* and *decentralized* approaches to the governance of complex, fluid and self-organizing systems – including economics, society, IT systems and artificial intelligence.

With the help of OfferNets we continue to weave the thread which intertwines the computational, conceptual and philosophical aspects of this work. The offer networks model provides a concrete case for a computer simulation based on the computational model that represents a much broader class of complex self-organizing cognitive systems. Note that conceptual, computational and software models of offer networks are strongly intertwined and the boundary between them is fuzzy. For the sake of the discussion's clarity we will refer to the conceptual aspect as **offer networks** and computational / software architecture aspect as **OfferNets**, while interchangeably using both.

6.2 Software architecture

OfferNets is a simulation modelling framework of radically decentralized economic interaction, powered by a diverse network of independently operating and interacting agents. It combines two research and development paths which are tightly related, yet embrace different levels of abstraction:

- *Decentralized computing*: a scalable computing model and a software framework supporting asynchronous execution of heterogeneous processes. These processes concurrently use a shared distributed data structure that allows any mixture of emergent and controlled coordination to be modelled (see Chapter 4).
- *Offer networks* economy: a decentralized economy providing an alternative to purely currency-based exchanges. This economy features a complex network of interactions and optimizes reciprocal exchanges of goods and services by finding agents with compatible or complementary preferences and coordinating their interactions. However, we do not attempt, conceptually or computationally, to model the whole economy here, but restrict ourselves to the demonstration of a specific search and matching problem, as introduced earlier.

Recall that, architecturally, software framework is composed of two subsystems: simulation engine and analysis engine. The concrete implementation of both is explained below. See Figure 5.12 for an explanation of interaction between these subsystems.

6.2.1 Simulation engine

OfferNets simulation engine architecture, software framework and relations to the implementation of the offer networks domain model are illustrated by Figure 6.1:

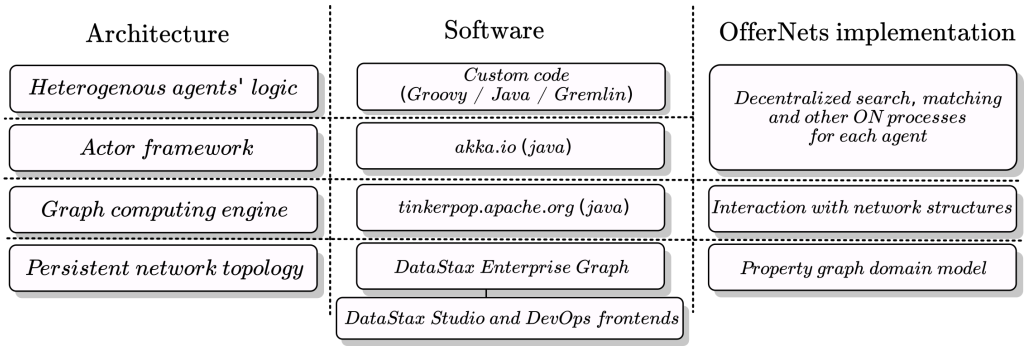


FIGURE 6.1: Relation between architecture, software and the offer network domain model implementation of the simulation engine.

The four layers of architecture are implemented as follows:

i. *Heterogeneous agents' logic* is written in custom code using Groovy[1], Java[2] and Gremlin[3] programming languages. The choice of languages is mostly constrained by the choice of actor framework and graph computing framework which are both JVM based;

ii. The chosen *actor framework* is Akka[4], which is an implementation of the actor model on Java Virtual Machine and is a toolkit for building highly concurrent, distributed and resilient message-driven applications on Java and Scala[5]. Depending on the complexity of actor code, it may support up to 50 million messages per second on a single machine, 2.5 million actors per 1 GB of memory, actor systems spanning multiple machines and a persistence layer.

iii. *Graph computing framework* is the Apache TinkerPop[6], which is also the home for Gremlin graph traversal language. The framework is open source and vendor-agnostic, meaning that it can be integrated with many data sources, database systems and programming languages of the open source and commercial world, including massively scalable cloud-based technologies[7]. Apache TinkerPop is a matured, but actively developed, de-facto graph computing standard with the third major stable version at the time of writing.

iv. *Persistent network topology* (equivalent to the "data source" in TinkerPop's vocabulary) is implemented using DataStax Enterprise (DSE) Graph[8], which is a distributed and scalable across multiple machines graph database. While the DSE Graph is built with and natively supports TinkerPop technology, it is the

[1]https://en.wikipedia.org/wiki/Apache_Groovy
[2]https://en.wikipedia.org/wiki/Java_(programming_language)
[3]https://en.wikipedia.org/wiki/Gremlin_(programming_language)
[4]https://akka.io/
[5]https://www.scala-lang.org/
[6]http://tinkerpop.apache.org/
[7]For the current list of TinkerPop-enabled providers and integration introduction, see http://tinkerpop.apache.org/providers.html
[8]https://www.datastax.com/products/datastax-enterprise-graph

most easily replaceable layer of the stack. Furthermore, from the perspective of the computational model, a graph database is not a necessity, since graph computing can in principle be implemented in a completely decentralized manner without a central database (see Figure 5.11) by using other means for ensuring persistence (e.g. Akka persistence layer). The usage of DSE Graph in the current version of OfferNets is a matter of convenience, since no graph computing engines decoupled from graph databases currently exist.

As illustrated in Figure 6.1, the actor model (via Akka) and agents' logic supports the implementation of OfferNets decentralized search and matching, as well as any offer networks related algorithms which can be implemented independently on each agent. The graph computing engine (via Tinkerpop) enables agents to define a domain model and interact with the network structure, the persistence of which is supported by the DSE Graph. Note that the computational model and software architecture, described in Chapter 4 and Chapter 5, does not constrain the choice of actual software for any layer of the stack, given they support the actor model and graph computing.

Further, the OfferNets domain model is specified as a property graph schema in terms of types of node, their properties, types of edge, their properties and processes defining graph traversal and mutation constraints. Beyond that, every agent operating in the network is allowed to implement any process. Processes which require interaction with the OfferNets graph are implemented as graph traversals; other processes – as regular algorithms using general purpose programming languages.

This decentralized computing model and architecture allows us to implement, test, deploy and observe the evolution of a very large number[9] of computational processes interacting and coordinating directly or indirectly within the same ecosystem. The challenge is then to define concrete processes, design their interaction and fine-tune the system to the preferred dynamics of the offer networks economy.

6.2.2 Monitoring and analysis engine

Simulation modelling requires the collection and analysis of information about events happening in the system during runtime. Since a decentralized computing framework by definition does not contain a single point of access to the system, we have built a specialized engine for collecting and handling large amounts of streaming data coming from many sources (i.e. single actors potentially scattered across multiple machines). The basic principle of the engine is based on issuing monitoring messages on behalf of each agent and then catching and indexing them into a single (but possibly distributed) database. The technical basis of the engine is ElasticStack[10] – an integrated streaming data management and analysis solution (see Figure 6.2).

The monitoring pipeline is fully distributed, with the possibility to be scaled to multiple machines and is tolerant to failures and restarts of each component separately. Likewise, the simulation engine is readily scalable to multiple machines depending on the required load for simulation or production environments. Both provide real time monitoring capabilities across all machines via web front-ends. Additionally, real time network, agent activity monitoring and event capturing is

[9] The scalability of the framework is of course dependent on the amount of dedicated hardware resources.

[10] https://www.elastic.co/

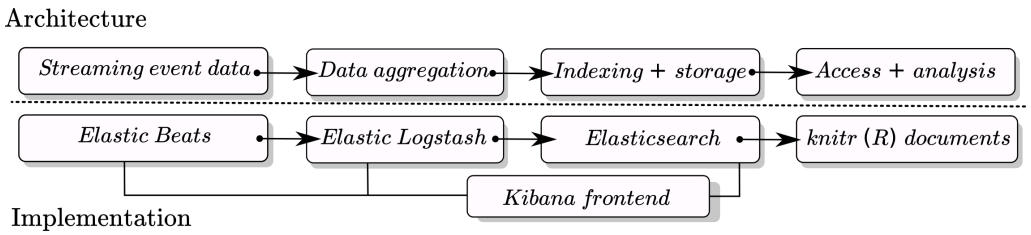


FIGURE 6.2: Architecture, software and implementation of monitoring and analysis engine.

available via custom-based web front-ends accepting data streams from other parts of the infrastructure.

6.2.3 Simulation modelling

Designing the dynamics of OfferNets is a simulation modelling research programme. It amounts to conceiving, implementing and running computational experiments on the simulation engine and then analysing data collected via the monitoring engine. Due to the large parameter space and many simulations needed for exploring it properly, this is a computationally intensive process. Furthermore, it is an open-ended process in the sense that every simulation raises questions and informs the set-up of the next one, thus iteratively perfecting both computational infrastructure and the domain model.

In the following sections we provide details about simulations aimed at comparing centralized and decentralized search algorithms on the same graph structures. Analysis, interpretation, and design for new experiments are compiled into the electronic laboratory notebook[11] using *R markdown*[12] living documents. The raw data is publicly available at the project's GitHub repository[13].

6.3 OfferNets: informal specification

Our goal is to design a software framework for the system pictured in Figure 6.3 and explore stigmergic cooperation between agents by running computational experiments. The system should allow matches of *offers* and *demands* (i.e. data *items*) to be found in an economic exchange network of *agents* in a way that enables each agent to extract maximum value from its exchanges – with respect to an individual measure of value – and by that contribute to the total increase of value created in the system. Already here we can start to appreciate the distinction between the decentralized aspect – the measure of value defined individually for each participant – and the centralized one – some sort of measure of "universal" value in the system[14].

[11] Electronic laboratory notebook is accessible at https://singnet.github.io/offernet/public/elabnotebook/

[12] https://rmarkdown.rstudio.com/

[13] Project's GitHub repository is accessable at https://github.com/singnet/offernet

[14] This distinction also presents a methodological difficulty: if every agent in the system has a different notion of "value", which, moreover, may change in time, then the notion of "universal value" is difficult to define. This means that it is not trivial to measure the overall "progress" of a decentralized system. These are open issues in relation to the *theory of value*, further discussion and development of which is outside the scope of this work.

Figuring out how these aspects relate and influence each other is the holy grail of research in the domain of collective intelligence and distributed cognition (Heylighen, 2011, 2013).

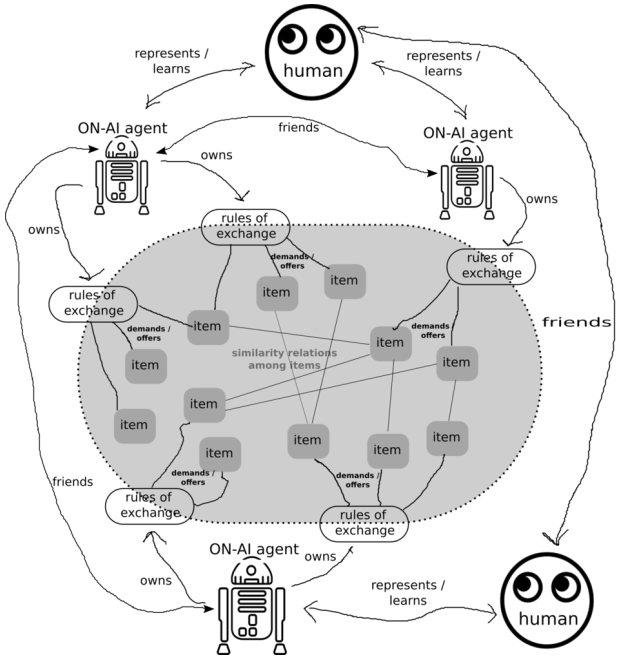


FIGURE 6.3: Conceptual architecture of OfferNets.

In OfferNets, humans can be represented in the network by one or more *ON-AI agent*. These computational agents, with more or less complex behaviours, can learn certain preferences of a represented human. Since there is no central authority in the network, humans can connect to it only via other humans, i.e. via a recommendation. Every human can create as many *ON-AI agents* as wished. The sole goal of an ON-AI agent is to announce preferences for the network and find ways to satisfy them. Preferences are satisfied when an agent finds other agents in the network (representing other humans) with opposite preferences. In OfferNets, preferences are expressed in terms of preferred exchanges of *items* – which in the real world could be goods, services, data or procedural knowledge. To that end, each ON-AI agent can own a set of *works*, which encode preference relations among the set of items and define items that are demanded and items that are offered (see Figure 6.4). The search and match of an ON-AI agent is successful when, given it *offers* an item, a similar enough item *demanded* by another agent on the network is found (this of course also works the other way).

ON-AI agents relate to each other via *knows* links which define the $agent \xrightarrow{\text{knows}} agent$ social sub-graph of OfferNets. Likewise, *items* can link to each other via *similarity* relations that define $item \xleftrightarrow{\text{similarity}} item$ order and a sub-graph. Note that the whole graph and its sub-graphs are dynamic and change in the course of interaction that results from collective attempts by agents to find locally and globally beneficial exchanges (see Figures 6.7 and 6.9). The ability to represent different objects and types of link is supported by the property graph model.

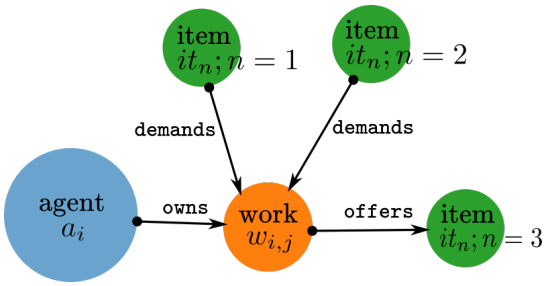


FIGURE 6.4: Graphical representation of preferences in OfferNets in terms of relations between agents, works and items.

From the mathematical perspective, the described OfferNets structure and dynamics can be unambiguously and equally well formalized as an optimization problem (Goertzel, 2015b), a reaction network in artificial chemistry (Dittrich, Ziegler, and Banzhaf, 2001; Heylighen, 2017) or a graph match and search problem (Goertzel, 2017b). However, the mathematical perspective is in a sense "logically centralized" and requires assumptions to be made and global modelling constraints imposed that detach the resulting model from observable dynamics in real-world decentralized markets (not unlike "mainstream" economic thinking). These constraints are best formulated in terms of preferences (Hansson and Grüne-Yanoff, 2018):

i. *Individual (psychological) preferences are known and complete.* This constraint reflects the assumption that agents (humans or AIs): (1) can name their preferences – i.e. make a list – and (2) can unambiguously order this list, at least potentially. We will leave aside the phenomenological issue of whether any real world item is in principle describable by a list of properties that can furthermore be ordered by their "essentiality".

ii. *Revealed preferences are complete.* Revealed preferences are those which an agent discloses to other agents engaged in interaction or a market place. In a sense, revealed preferences are more important than individual psychological preferences in the context of search and match – since preferences have to be externalized in order to facilitate any interaction in the first place. We assume that normally not all psychological preferences are revealed. The correspondence between revealed and psychological preferences is a separate, while not unimportant, question which we again will not ponder here.

iii. *Preferences are consistent.* In the context of OfferNets, consistency of preferences means that they are stable in time. This applies both to individual psychological and revealed preferences, but the latter are more important.

With these constraints in mind, recall the example of a flea market. Obviously, none of the constraints hold in this context: buyers do not know exactly what their preferences are before seeing items, while sellers may change their prices depending on the interaction with a potential buyer. Preferences of either get individuated and only then revealed, and revealed preferences are not complete[15]. Furthermore, no preferences are stable – one can have a completely different "set" of preferences after walking 10 minutes or half an hour in the market. Actually, most of these constraints do not hold even in regular physical or on-line shopping: it is common to

[15]There is an inherent asymmetry among revealed preferences of buyers and sellers, where the latter normally reveal more. This asymmetry has been addressed by *web Of Needs* project (Kleedorfer and Busch, 2013).

emerge from a shop with completely different purchases in the basket than were imagined before entering it. On the other hand, there are different markets where these constraints may hold with different strengths: e.g. standardized commodity, stock or transportation markets, where both supply (offers) and demand can be specified a priori and interaction kept to a minimum. Yet these can be considered rather exceptional, where supply and demand prices carry most of the information needed for decision making. The conceptual question that offer networks poses is therefore whether and how complex exchanges can be enriched and possibly their micro-economic dynamics affected by introducing ways for multi-dimensional information exchange (Goertzel, 2015a). Enabling dynamic interactivity of buyers and sellers is essential for large scale practical applicability of this idea and this is precisely why the offer networks model is well suited for implementing open-ended decentralized computing architecture.

From the computational point of view, OfferNets can be described in terms of a combination of a *data structure* and *processes*. It is customary (and remarkably useful in the practice of designing computational systems) to think of processes as functions operating on data structures by mutating them. Yet it is entirely reasonable (but not equally intuitive) to think of processes without reference to *a priori* data structures which they change – which relates to progressive determination and stigmergic computing (see Section 4.5).

6.3.1 Data structure

OfferNets is defined according to the *property graph model* which: (1) contains nodes and relationships; (2) nodes contain properties and can be labelled; (3) relationships are named, directed, can contain properties and always have start and end nodes (see page 42). This model is more expressive than a theoretical mathematical graph and can represent arbitrarily complex real world structures[16].

OfferNets property graph schema includes vertices of type [*agent, work, item*] and edges of type [*knows, owns, demands, offers, similarity*]:

Agents represent actual participants of the economic exchange – *ON-AI Agents* (see Figure 6.3) – which together form a social network. Every agent in the network is connected to at least one another agent. This gives rise to an important property of the graph: it is necessarily *connected*, i.e. there exists a path via *knows* relation between every pair of agents. This property is crucial for decentralized computing, as will be explained later. Agents also relate to one or more *works* via *owns* links – representing situations in which an agent publishes that it "wishes" to exchange something in the network.

OfferNets is not only a data structure, but also a computational medium and as such has to account for computing resources needed to run decentralized processes. Agents participating in the network (people or AIs) are those entities which own computing resources. Note a direct relation of OfferNets to the actor model

[16] A *hyper-graph*, which may better correspond to many biological and neural structures found in nature (Goertzel, 2017a) is a generalization of the property graph in that it allows one edge to connect more than two vertices. Hypergraph data structures are preferred by some AGI (namely OpenCog) and reasoning (namely GRAKN.AI) systems. Note, however, that a property graph can be transformed to a hypergraph without data loss, but not the other way round in a general case (Blockeel, Witsenburg, and Kok, 2007; Robinson, Weber, and Eifrem, 2015).

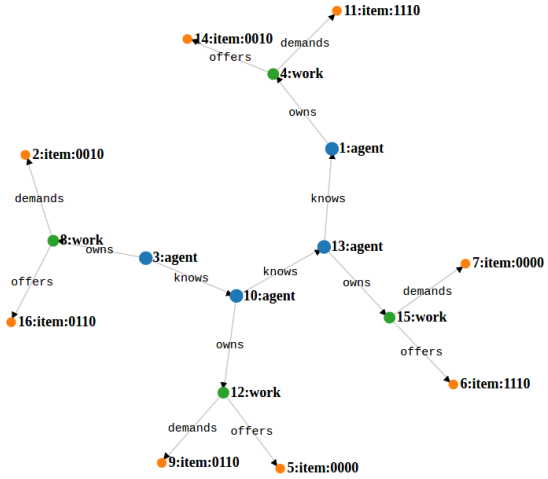


FIGURE 6.5: OfferNets graph structure. Note that this is an initial structure which does not contain *similarity* links, which appear in the graph only after running processes, as explained in Section 6.3.2 on page 160.

of computation (see Section 5.1) where every actor controls its own computing resources. Moreover, note the similarity to the notion of the computation graph as a self-organizing interaction of elementary processes, which makes OfferNets a special case of open-ended decentralized computing (see Figure 5.10). In this sense, *work* is a special *process* (see Section 4.5.1) with *demand* as an input and *offer* as an output. This process is neither immediate nor resource-free: any actual exchange involves costs (time, transportation or other arrangements) which have to be represented in any practical application. These, and similar, issues are left out from the current simplified model of OfferNets, but can be readily implemented by including additional features as defined the by general model of open-ended decentralized computing.

Work therefore represents the "process of exchange" in which an agent is willing to engage with other agents in OfferNets. By "owning" a *work*, an agent pledges that the work will be executed if a match is found[17]. In the current simplified OfferNets model *work* connects only one *demand* with one *offer*, yet in principle arbitrary complex works can be represented featuring more than one input (energy, computational resources or a monetary payment) or output. Moreover, *work* can represent the exchange of data output (e.g. textual description of an image) for data input (e.g. an image to be described), a monetary payment (in fiat or crypto- currency), computational resources needed for the task, or location of the process in the network's topology. In this case, a *work* would not be a "process of exchange" but rather a "process of text summarization" which nevertheless can be perfectly well represented within the same framework.

An **item** is an actual "thing" that is put forward by an agent for a potential exchange. In OfferNets this is limited to actual physical or non-physical things, but in general an item could be a representation of any input or output of a generalized process (i.e. data, token, energy units, etc.).

In order for the processes operating within OfferNets (Section 6.3.2) to be able

[17] Any such pledges in a decentralized system have to be supported by the system of "distributed trust", which is another important and interesting issue which is outside the scope of our present treatment of offer networks.

to find matching offer and demand pairs which could be exchanged, a *similarity* relation between any pair of items should be defined. That is, given an arbitrary pair of items registered in the network, a process or algorithm should be able to calculate how similar (or dissimilar) they are. If and when the similarity between two items is calculated, a symmetric *similarity* relationship between them is created in the OfferNets graph[18].

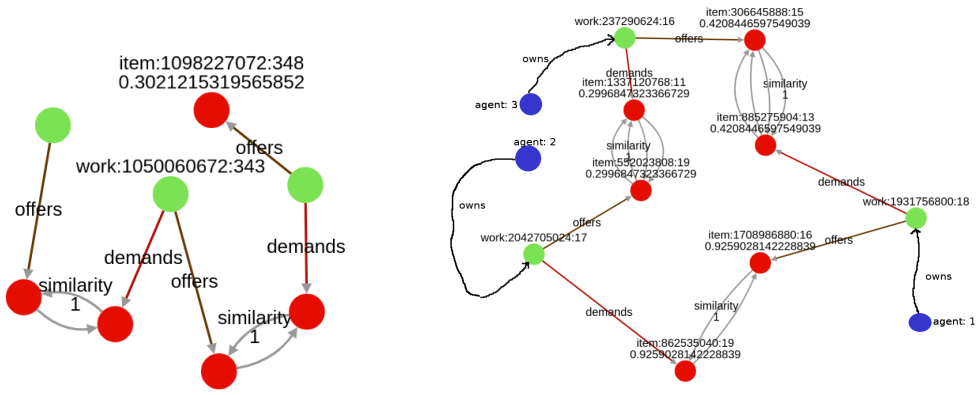
FIGURE 6.6: OfferNets graph schema.

This means that every *item* has to contain a description according to which its similarity with other items in the network can be potentially estimated or calculated. Here we touch again the issue of properties and preferences (see page 156). For the purposes of this OfferNets simulation, the description of an item and similarity relation are defined as a real number between zero and one. While these choices of definitions are not arbitrary, they are of minor importance in the context of experimenting with centralized and decentralized search and match in OfferNets. It should nevertheless be noted that representation of and similarity measuring between real-world items is a large and interesting domain of inquiry in itself. Similarity can be an arbitrary complex relationship depending on the practical application of the framework. A truly decentralized Offer Network type system should be able to accommodate different and possibly competing variants of measures and algorithms. The architecture of OfferNets allows for plugging in any chosen similarity representations.

Relations of similarity between items enables chains and cycles of works to be found. A **chain** forms itself when there is an item i_1 offered by work w_1 and item i_2 demanded by work w_2 and when items i_1 and i_2 are sufficiently similar (see Figure 6.7a). "Sufficiently similar" in this context means that the calculated similarity measure between items is greater than or equal to an arbitrary threshold set by an agent engaging in the exchange of items (different agents may have different thresholds). If this parameter is equal 1, then the requirement is to exchange only strictly similar items, which results in a standardized exchange. A **cycle** is simply a chain forming a loop (see Figure 6.7b). When a chain or a cycle is found in OfferNets, it indicates the possibility of an actual exchange between agents.

Apart from the chains and cycles illustrated in Figure 6.7, works can be connected in arbitrary complex workflows in cases where a domain model allows them to have multiple demands and offers (see Figure 6.8). The problem of search and matching is precisely a generalized process of finding workflows in such a network – the complexity of which as well as a comparison of algorithms from the mathematical perspective are investigated by Goertzel (2017b).

[18] In the OfferNets graph schema, a symmetric *similarity* relation is represented by two directed edges $a \xrightarrow{\text{similarity}} b$, $b \xrightarrow{\text{similarity}} a$, where a, b are items and $a \neq b$



(A) Chain (B) Cycle

FIGURE 6.7: Examples of a chain and a cycle in OfferNets. Red spheres are *items*, green – *works*, blue – *agents*. Figures are visualizations of the results of an OfferNets simulation.

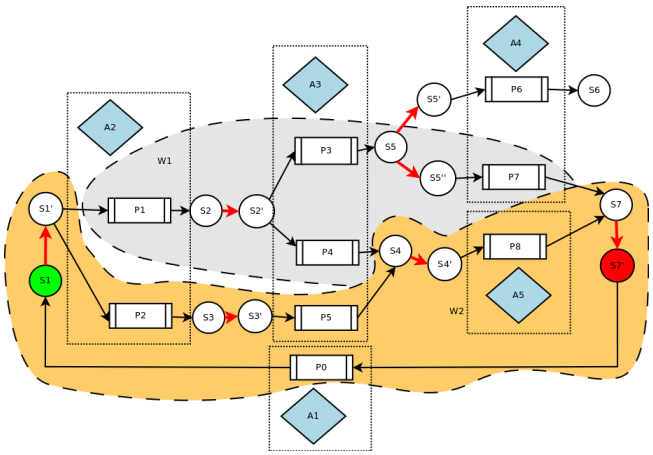


FIGURE 6.8: Data centric approach, describing workflow as a sequence of transitions between data states (inspired by Cushing, 2015; Cushing et al., 2015). In OfferNets terms, *work* here is represented as a process P_i, *items* as data states S_i and *agent*s as blue rectangles A_i.

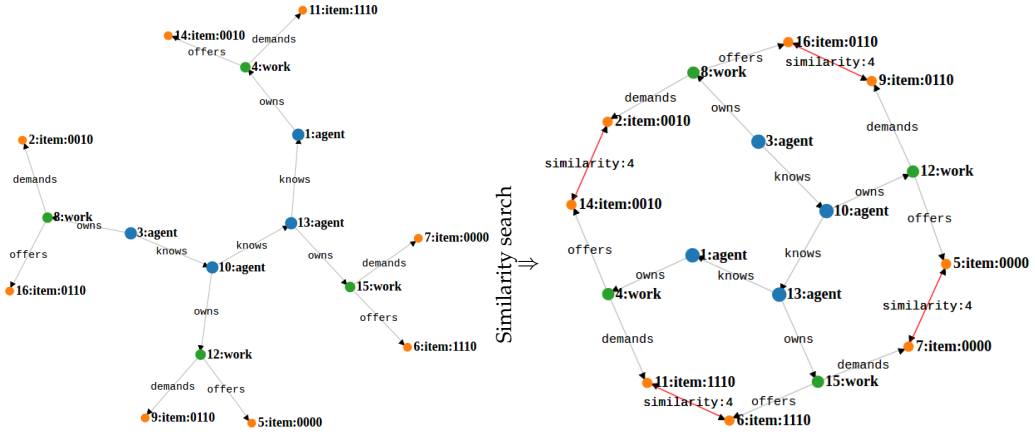
6.3.2 Processes

Following the basic principles of open-ended decentralized computing, OfferNets is implemented as an ecosystem of decentralized processes interacting via a stigmergic medium. The list of decentralized processes is open-ended in the sense that any process can be executed by an agent participating in the network. Processes needed for basic functionality of OfferNets are: (1) similarity search, (2) find cycles of changeable items, (3) execute exchange cycles and (4) find and connect items of exchange via similarity links. These processes are described in detail below.

Process #1: Similarity search

This process searches similar *items* in the network according to given criteria and connects them with explicit *similarity* links. The ability to measure similarity of items

is based on an agreed representation and description of each item. Similarity measures can also take different forms depending on item representation. If the representation is the real number, then the similarity measure could be a difference. If items are described by vectors, similarity can be measured by a Hamming distance[19] or as cosine similarity[20]. In the case of natural language descriptions or images, NLP techniques or image summarization could be used. Note that these measures are global parameters of simulation framework and different forms of them can be "plugged" and "unplugged" dynamically. Furthermore, in a decentralized system, nothing prevents agents from agreeing themselves on the usage of different similarity measures within the same framework. In terms of the overall dynamics of OfferNets, the goal of the similarity search process is to change the topology of the graph so that similar items become topologically closer to each other (see Figure 6.9 and Figure 6.10).



(A) Before similarity search processes: 12 nodes (of type *agent*, *work*, *item*); 12 edges (of type *knows*, *owns* and *demands* | *offers*); zero *similarity* links.

(B) After similarity search processes, 4 *similarity* links forming a cycle.

FIGURE 6.9: Visualization of graph mutations and topology change due to similarity search process and *similarity* links creation on a small OfferNets sub-graph.

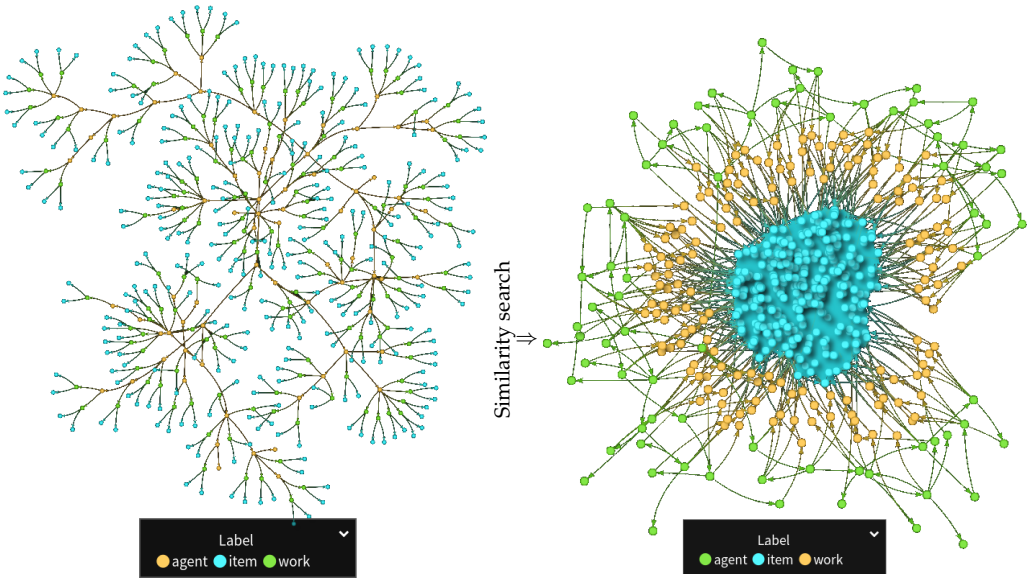
In this example *item* values are represented as real numbers in the range $[0, 1]$. Similarity between two items is then calculated using the formula $Sim = 1 - abs(value_{i1} - value_{i2})$ which also results in the real number in the range $[0, 1]$. The closer this number to one, the more similar the items are.

Algorithmically, the similarity search process is implemented in two ways – centralized and decentralized. The comparison of their performance forms the basis of the experiment discussed in Section 6.4.

Centralized similarity search

A centralized similarity search simply fetches all *items* in the network, compares each of them to every other and creates a *similarity* link among those that have a similarity value exceeding a parameter of *similarityConnectThreshold*. This parameter regulates the density of connectivity between *items* on the one hand and the ability for agents to exchange "fuzzy" similar items on the other.

[19] https://en.wikipedia.org/wiki/Hamming_distance
[20] https://en.wikipedia.org/wiki/Cosine_similarity



(A) Before similarity search processes: 534 nodes (of type *agent*, *work*, *item*); 534 edges (of type *knows*, *owns* and *demands* | *offers*); zero *similarity* links.

(B) After similarity search processes, 46.574 *similarity* links

Nodes		Links	
Agent	75	'knows'	74
Work	153	'owns'	153
Item	306	'demands' / 'offers'	306

FIGURE 6.10: Visualization of graph mutations and topology change – same as Figure 6.9 but on a larger graph.

A centralized similarity search requires a full scan of the graph in order to collect data on all items demanded or offered by agents at a certain moment in time, combining this data into a single data structure and then processing it in a centralized (but possibly distributed) manner.

Decentralized similarity search

A decentralized similarity search, unlike the centralized one, works only on behalf of an agent that initiates the search and does not require the fetching of all *item* values from the network. On the other hand, a decentralized process requires concurrent and asynchronous execution on behalf of each agent. It operates as a spreading activation which starts with the initiating agent's items and checks their similarity to those of the agent's neighbours. A decentralized similarity search takes *similarityConnectThreshold* and *maxDistance* parameters. The former serves in the same way as in a centralized search, while the latter determines how far into an agent's neighbourhood the spreading activation process traverses.

The process follows this logic (see also Listing 6.1 for the pseudo-code):

i. traverse *knows* relations in a *agent* $\xrightarrow{\text{knows}}$ *agent* sub-graph starting with agent a_1 and reach all connected items i_i of a "friend of friend" within distance *maxDistance* from a_1;

ii. calculate a similarity measure $s = similarity(i_1, i_i)$ for all found items and create a similarity relation between i_1 and i_i if and only if the global parameter *similarityConnectThreshold* is larger or equal to s.

LISTING 6.1: Pseudo-code of similarity search process via $agent \xrightarrow{\text{knows}} agent$ sub-graph.

```
# parameters:
# -- me: the agent that initiates traversal
# -- maxDistance: number of hops in traversal;
# -- similarityConnectThreshold: only items with this and higher similarity
    are connected;

    myItems <- me.getAllDemands() + me.getAllOffers();
    discoveredItems <- emptyList();
    distance = 0;

    function getItemsOfNeighbours(agents):
            for each agent in agents do:
                discoveredItems <- discoveredItems + agent.getAllDemands() +
                    agent.getAllOffers();
                neighbours <- agent.knowsAgents();
                getItemsOfNeighbours(neighbours);
                distance = distance +1;
                if distance = maxDistance do:
                    break from cycle;

    getItemsOfNeighbours(me)

    for each discoveredItem in discoveredItems do:
            for each myItem in myItems do:
                similarityValue = calculateSimilarity(discoveredItem, myItem)
                if similarityValue >= similarityConnectThershold do:
                    createLink(from: myItem, to: disoveredItem, type:
                        similarity, value: similarityValue)
```

Running this process a sufficient number of times results in items with similar values forming densely connected clusters in the graph – i.e. it alters network topology as illustrated in Figure 6.10 in a way that enables decentralized search of paths and cycles as well as making further similarity searches more efficient.

Process #2: Find cycles of changeable items

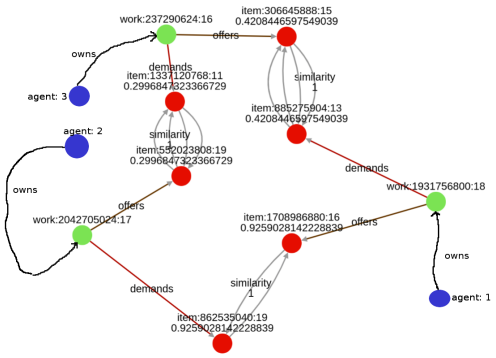
In graph theory, a cycle is a collection of vertices and edges between them where each vertex is reachable from itself via edges present in the collection (see Figure 6.7b). Cycle search is the process that finds such data structures in a messy and unstructured initial graph. Concretely, in OfferNets, a decentralized cycle search process is a *vertex centric graph traversal* which, for a chosen work w_1 owned by an agent a_1, and given *similaritySearchThreshold* and *maxDistance* parameters, initiates the following process:

i. traverse *offers* and *demands* relations starting with work w_1 and find sub-graphs of the pattern $work_1 \xrightarrow{demands|offers} item_1 \rightarrow similarity_{12} \rightarrow item_2 \rightarrow work_2$ (see Figure 6.7a), where $similarity_{12} >= similaritySearchThreshold$; repeat the same traversal until encountering the start of the traversal $work_1$, but not more than *maxDistance* times;

ii. return information on paths and cycles found to $agent_1$, which initiated the traversal.

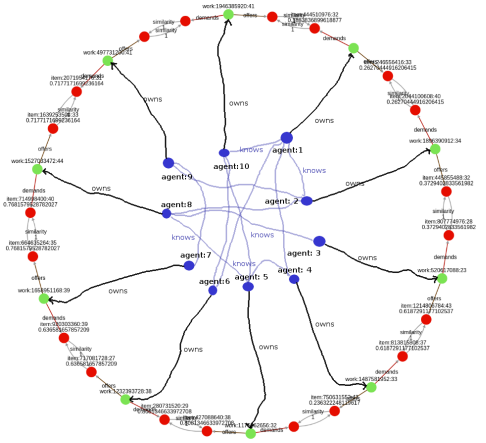
A discovered cycle (see Figure 6.11 below) represents a match of demands and offers of at least two agents participating in the marketplace and the possibility of actual exchange between them. More formally, it is a sub-graph of the OfferNets, where agents "know" other agents' preferences with respect to the discovered match

and can agree to execute the exchange cycle (see process #3). Importantly, the cycle in this case is discovered solely via the decentralized vertex-centric graph traversal without any central processing or data storage – as required by the model of open-ended decentralized computing.



(A) Small cycle

Agents	3
Works	3
Items	6



(B) Larger cycle

Agents	10
Works	10
Items	20

FIGURE 6.11: Cycles discovered in the OfferNets graph by cycle search processes; note how it relates to the conceptual architecture of offer networks (Figure 6.3 on page 155).

Cycles are temporary structures – they get dissolved when executed. Actually, the OfferNets system can be considered more successful and beneficial when more cycles are discovered, executed and dissolved per unit of time, which is a measure of fluidity of the network. Another important aspect, emphasized in the figure, is that while cycles are of dynamic nature – emerging and dissolving during operation of the OfferNets graph – they are basic elements of the conceptual architecture of the self-organizing network of interacting agents (see Figure 6.3).

Process #3: Execute exchange cycles

Finding cycles only increases the probability that certain items will be exchanged, but is far from guaranteeing it. While in a centralized system an execution of the discovered cycle is straightforward, it is not so in the decentralized case. First, since many search and match processes are executed asynchronously, more than one cycle can emerge involving overlaps of agents and items. In this case, a consensus has to be reached, via negotiations, on which cycle will be executed and which will be neglected. This already involves a game-theoretic aspect. Second, when a cycle involves fuzzy matches, the willingness of agents to exchange items that are not strictly similar has to be confirmed. Third, the presence of insufficient information in the network due to incomplete preferences (Mandler, 2005) should be considered – implying the necessity of a round of negotiations between agents involved in the cycle before its execution. Finally, exchanges in distributed systems cannot rely on a single provider of trust and therefore must use a distributed trust model (Abdul-Rahman and Hailes, 1997) including, but not necessarily limited to, the blockchain technology (Swan, 2015) and a reputation system or systems (Kolonin et al., 2018). When an agent receives a list of paths and cycles from the cycle search process, it

decides, according to an built-in logic (which may be different for heterogeneous agents), whether to initiate a transaction in case a valid cycle is found. The resulting dynamics of an OfferNets depends on competition and collaboration between agents with different logic and goals.

Process #4: Search and connect similar items of exchange via similarity links

Computational complexity and success in finding matches in the graph are mostly due to the similarity search process rather than the cycle search. Likewise, search via the *agent* $\xrightarrow{\text{knows}}$ *agent* sub-graph is relatively costly because it requires many agents in the neighbourhood graph to be checked in order to find a similar item. Furthermore, in a decentralized scenario and depending on the topology of the graph, the search may not succeed even in cases where a cycle exists. Therefore, it may be much more efficient to traverse the *item* $\xleftrightarrow{\text{similarity}}$ *item* sub-graph directly where similar items are clustered together due to traces of process #1. This type of search would utilize graph topologies on the right rather than left of Figures 6.9b and 6.10b – where similar items are topologically closer, which implies shorter average traversal paths. Listing 6.2 provides a pseudo-code of the logic of this process.

LISTING 6.2: Pseudo-code of similarity search process via *item* $\xleftrightarrow{\text{similarity}}$ *item* sub-graph.

```
# parameters:
# -- me: the agent that initiates traversal
# -- maxDistance: number of hops in traversal;
# -- similarityConnectThreshold: only items with this and higher similarity
    are connected;
# function getItemsOfNeighbours(agent) is defined in pseudocode for Process
    #1;

    myItems <- me.getAllDemands() + me.getAllOffers();
    distance = 0;

    function searchAndConnectViaSimilarityLinks(myItem, otherItem):
        similarityValue <- calculateSimilarity(item, otherItem)
        if similarityValue >= similarityConnectThreshold do:
            createLink(from: myItem, to: iterItem, type: similarity, value:
                similarityValue)
        if distance > maxDistance do:
            break from the loop;
        else:
            nextMostSimilarItem <- item.getSimilarItems(max(similarityValue))
            searchAndConnectViaSimilarityLinks(myItem, nextMostSimilarItem)


    for each myItem in myItems do:
        distance = distance + 1;
        if distance = maxDistance do:
            break from cycle;
        if item.hasSimilarityLinks() do:
            mostSimilarItem <- item.getSimilarItems(max(similarityValue))
            searchAndConnectViaSimilarityLinks(myItem, mostSimilarItem)
        else:
            itemsOfNeighbours <- getItemsOfNeighbours(me)
            for each discoveredItem in itemsOfNeighbours do:
                searchAndConnectViaSimilarityLinks(myItem, discoveredItem)
```

As specified, the above process traverses *item* $\xleftrightarrow{\text{similarity}}$ *item* sub-graph directly if it exists; otherwise, it reverts to traversing the *agent* $\xrightarrow{\text{knows}}$ *agent* sub-graph as defined by process #1. When many such processes operate concurrently, initiated and sustained by different agents, each process changes the local graph structure around

itself, which is then traversed by many "neighbouring" and possibly some "distant" processes. Together, the combination of asynchronous processes and the graph as a stigmergic medium leads to the fluid dynamics of progressive determination (see Section 3.2.4) and open-ended decentralized computing (see Section 4.4.1).

6.3.3 Research questions

The ambition of the OfferNets simulation modelling project is to *conceive, implement and test mechanisms of search, matching and execution of exchange of goods and services in a radically decentralized system*. This ambition is pursued by asking (and answering) concrete research questions which allow the clear formulation and testing of scientific hypotheses – a process leading to incremental building of the system. The current research horizon encompasses the following research questions:

i. *What are the parameters that determine the advantages and disadvantages of decentralized and centralized search algorithms in different contexts*? In principle, OfferNets logic and goals can be achieved by either centralized (global) or decentralized (local) processes running on the same data structure as defined earlier. In practice, however, the feasibility of any of the approaches is largely determined by concrete circumstances and context-specific aspects. For example, centralized algorithms can optimize results at the cost of larger computational complexity needed to carry them, together with the need for central storage of data structure and privileged access to it (see the fundamental trade-off of decentralized computing on page 31). Decentralized algorithms on the other hand may achieve sub-optimal, but still "good enough" results faster at the cost of giving up control of the whole data structure. Note that the very term "good enough" implies context dependency. In order to provide at least some insights to this question we set up centralized and decentralized search processes in OfferNets and tested them with different parameters (see Section 6.4 for design and results of a computational experiment on comparing decentralized versus centralized search).

ii. *Can we conceive processes which make themselves more efficient by utilizing results of "traces" left by preceding processes*? Such processes, interacting with each other in a progressively determining way, would implement the learning ability of the network. Similar to the first research question, this can be implemented in a centralized or decentralized manner.

6.4 Centralized versus decentralized processes of OfferNets

This section presents and discusses results of the computational experiment designed to answer the research question concerning the advantages and disadvantages of decentralized versus centralized similarity search process in OfferNets. It demonstrates how principles, including, but not limited to stigmergic cooperation between software agents, can be utilized in a software architecture based on the open-ended decentralized computing principles.

6.4.1 Setup

In order to compare process #1 and process #2 in their both centralized and decentralized implementations we follow these steps:

1. first, we create an OfferNets graph of predefined size. We experiment with the random graph (where agents are randomly connected with *knows* links) and a small-world graph, where we know the diameter of the network in advance;
2. then we artificially create a list of items which, if correctly searched and connected in the OfferNets graph, will form a chain (see Figure 6.7a). The length of the predefined chain is set by the simulation parameter *chainLength*;
3. items from the list are assigned to random agents in the network;
4. then, we create a special *taskAgent* owning a *work* which, when correctly connected to the potential chain inserted into the network by step 2, closes it into a loop forming a cycle (see Figure 6.11);
5. finally, we run the decentralized and centralized processes on the same graph and record the running times of each method.
    - The similarity search process connects all similar items with *similarity* links, as explained in Section 6.3.2;
    - The cycle search process is run on behalf of *taskAgent* and discovers the cycle inserted by step 2 only if the similarity search process connects a large enough number of items – as explained in Section 6.3.2.

What is hereinafter called an "experiment run" is a series of simulations, each of which takes a different combination of the following parameters for the purpose of exploring the parameter space (see Figure 6.12):

- *agentNumber*: the number of agents in the network;
- *similarityConnectThreshold*: the minimum similarity value between items connected with explicit link by similarity search process;
- *chainLength*: the length of an artificially created chain inserted into the network by step 2;
- *similaritySearchThreshold*: minimal similarity of items to be considered as eligible for exchange;
- *maxDistance*: radius of agent's neighbour network when searching for similar items;
- *randomWorksNumberMultiplier*: the number of random works and items which are assigned to the agents in the network to make cycle search more realistic;

6.4.2 Observed dynamics

Here we present the observed dynamics of simulations, mostly in terms of the relation of time to the topology and size of the OfferNets graph. We briefly describe the technical aspects behind these observations. Interpretation of and conceptual insights from the observations are discussed further in Section 6.5.

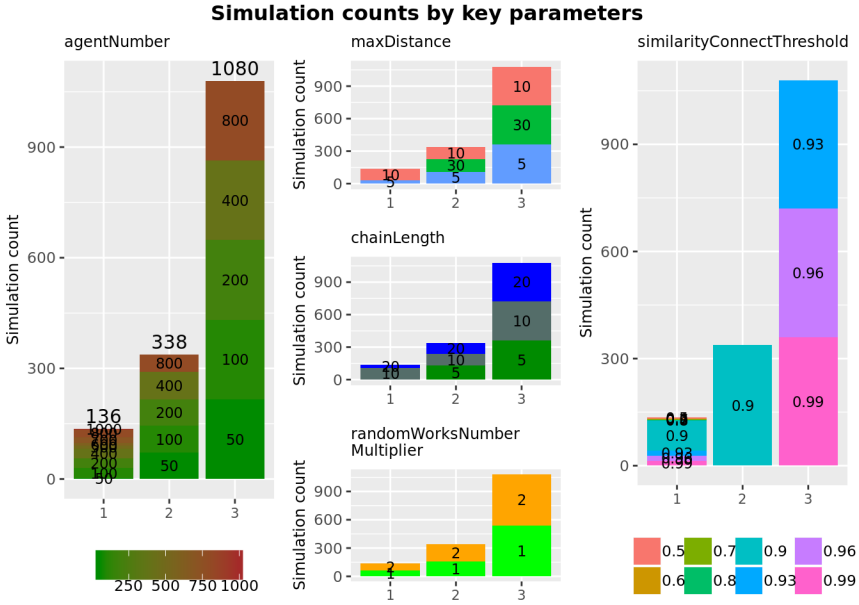


FIGURE 6.12: Distribution of simulation parameters of all analysed experiments (total number of simulations: 1,554; aggregate simulation time (user): 757 hours, events cached: about 1 billion).

Sensitivity to graph topology

First of all, data collected from simulations supports the intuition that decentralized and centralized searches have very different sensibility to the underlying graph topology. That is, the centralized search algorithm is more sensitive to how many agent nodes are in the graph rather than on how well they are connected. The decentralized search, on the other hand, is sensitive to the topology of the *agent* $\xrightarrow{\text{knows}}$ *agent* sub-graph. In part this is because decentralized graph traversals continue only as deeply as constrained by *maxDistance* parameters, which define the radius of exploration (see Section 6.4.1). Due to this, where the diameter of the graph is larger than *maxDistance*, the cycle may not be found, even though it exists in the network. It is of course possible to increase the *maxDistance* parameter to the arbitrarily large number, but this also may increase the time (see Figure 6.13). Another aspect is that the computational complexity of deep graph traversals depends on how many outgoing links every traversed node has (i.e. the branching factor of the tree constructed by a traversal) – which is a parameter of network topology. Obviously, the larger the branching factor the more computationally complex it is to traverse it.

For confirming the sensitivity of the decentralized search to graph topology we also ran simulations with the same parameters on two different graph topologies – (1) randomly connected (Erdös and Rényi, 1959) and (2) small world (Watts and Strogatz, 1998), with a diameter of less than 10. As shown in Figure 6.14, the success rate of finding a cycle in a random graph is often lower than 100%, except when the traversal depth is made very large – which takes more time, as shown in Figure 6.13. On the other hand, a small world graph structure with a known diameter and corresponding *maxDistance* parameter guarantees that the cycle will be found if it exists in the graph, even in a decentralized scenario.

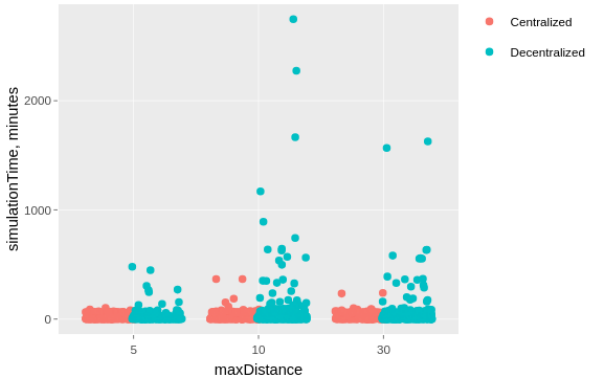


FIGURE 6.13: Dependence of decentralized search time on *maxDistance* parameter. Traversals with higher *maxDistance* have a higher probability of running longer. Runtimes are not deterministic because they depend to some extent on fine graph topology (i.e. $agent \xrightarrow{\text{knows}} agent$ sub-graph which is generated randomly).

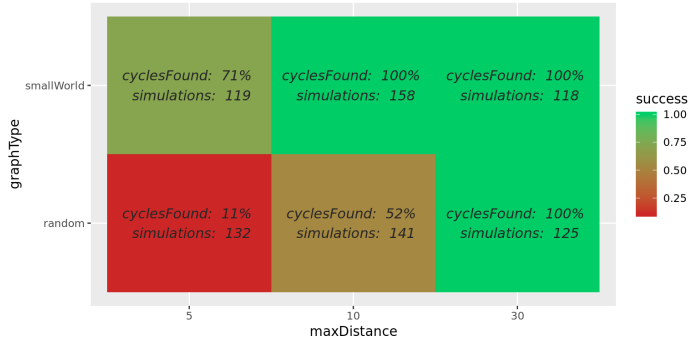


FIGURE 6.14: Success rate of finding a cycle in random and small world graphs with decentralized traversals with different *maxDistance* parameter. Small world graphs have a known diameter of *10*. Note that with random graphs, *maxDistance* of *5* and *10* is not enough to traverse the whole graph, while *30* often provides (but does not guarantee!) full coverage. Likewise, a decentralized search with $maxDistance = 5$ does not always find a cycle in the graph with diameter *10*.

Decentralized search time is sensitive to the number of edges

Figure 6.15 shows the dependency of simulation time on the number of *similarity* edges in OfferNet graphs with a different number of agents. The decentralized search seems to be marginally faster when there is relatively small number of links in the graph, yet its time complexity quickly exceeds the centralized search when the number of similarity edges increases.

This data show that the centralized search is almost always faster than the decentralized. Note also that the decentralized search may find the same cycle several times during the same simulation, which is not possible in the centralized case. This is because similarity search and cycle search processes run asynchronously and independently of each other. Therefore there is no easy way to stop all processes when one of them finds the cycle.

Centralized search time increases with the number of vertices

The dependency of time of simulation on the number of total vertices in the graph (i.e. of type *agent*, *work* and *item*) is visualized in Figure 6.16.

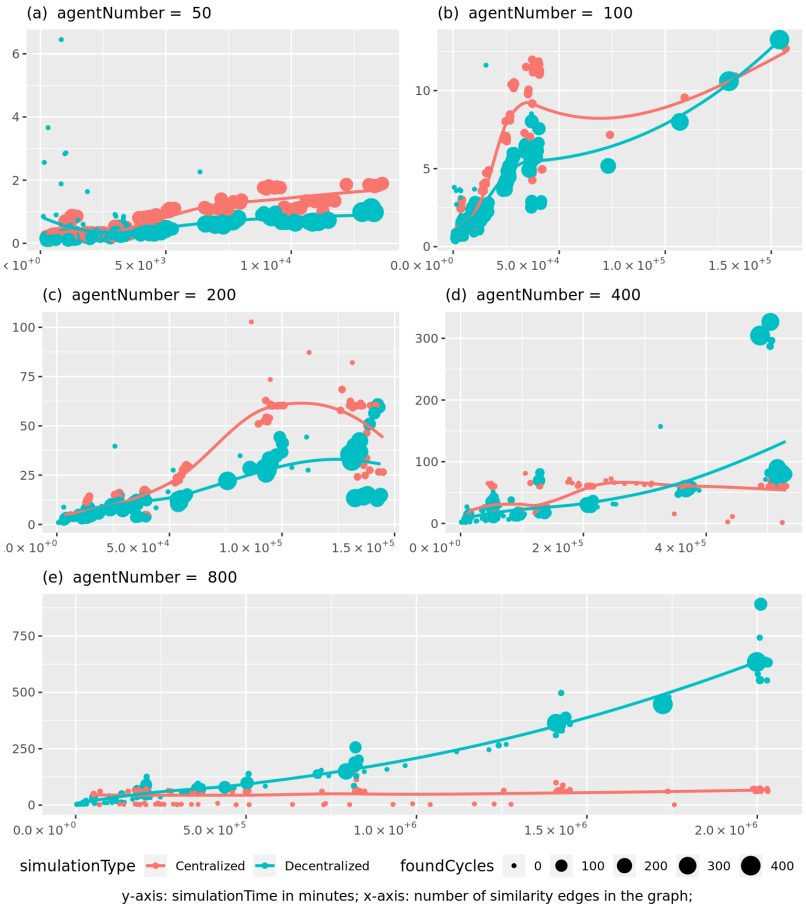


FIGURE 6.15: Dependence of simulation time on the number of *similarity* edges in the graph.

Not unexpectedly, the time needed for the search increases with the number of vertices in the graph, which follows common sense intuition. This is correct for both centralized and decentralized versions. Note, however, that in decentralized traversals it is difficult to separate the role of number of vertices from the role of number of links, since they are highly correlated variables. Nevertheless, we can at least approximately estimate the effect of the number of links in OfferNets simulations with the help of the *similarityConnectThreshold* parameter. Recall that this parameter sets the minimum value for items' similarity in the network for them to be connected with a similarity link. Figure 6.17 shows how it modulates the topology of the graph, since when the parameter is lower, the probability of two items getting connected is higher, and therefore total number of edges in the graph is higher given the same number of vertices.

As can be seen from Figure 6.16, simulation times in the centralized search are almost non-dependent on the magnitude of the *similarityConnectThreshold* parameter. The opposite is true for the decentralized search, where the spread between lines representing different *similarityConnectThreshold* is obvious. Therefore, the centralized search is more sensitive to the number of vertices, and the decentralized to both number of edges and vertices.

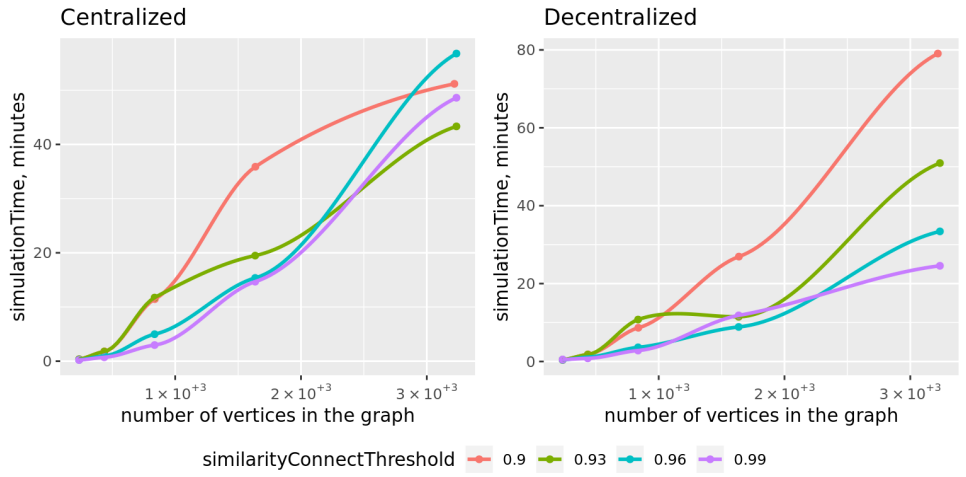
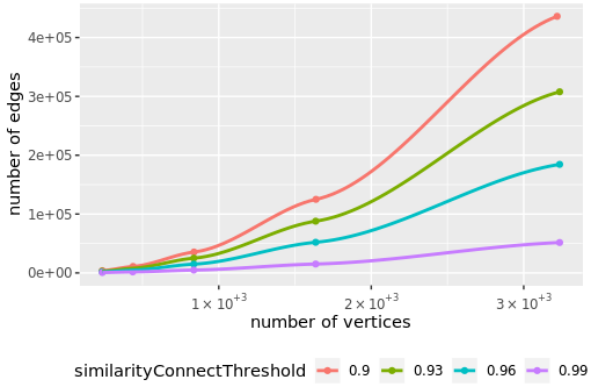


FIGURE 6.16: Time needed for centralized search roughly increases depending on number of vertices in the graph.

FIGURE 6.17: Graph topology as modulated by *similarityConnectThreshold* parameter. The lower value of this parameter on average means that more *items* get connected with *similarity* links, and therefore the total number of links in the graph increases.



6.5 Summary and discussion

This chapter presents and discusses the implementation of selected aspects of a decentralized exchange using the software architecture of open-ended decentralized computing. Concretely, we have implemented the logic of the offer networks domain model on the simulation engine, run a number of simulations with different parameters and presented their results with the help of analysis and the monitoring engine designed in Chapter 5. Within the scope of this work, the goal of OfferNets simulation modelling experiments was, first of all, to gain insights from actual implementation and operation of the architecture in the running computer code. Recall that the design-based research methodology of this work is motivated by the desire to connect ultimately abstract with the most specific. This connection is not to be understood as a "research funnel"[21], which describes stages of research in terms of how close they are to the problem that is being addressed by a research project, where the sole purpose of the funnel is to direct and restrict all stages of research towards the *a priori* formulated problem. In our case, philosophical, computational, software design and practical implementation are all aspects that inform each other and the whole research programme. The implementation and application of the framework to practical problems in a well defined domain adds a pragmatic layer

[21] Boulton, E. (2014, July 22). The Research Funnel [Blog]. Retrieved January 17, 2019.

to approaching the problematic of open-ended computing and dealing with decentralized complex systems as well as the centralization – decentralization dilemma in general.

Open-ended decentralized computing deals with *uncertainty*, rather than *efficiency*, which is a primary concern of closed computing (see Section 4.3). Simulation modelling experiments with OfferNets clearly expose this distinction. Observe, from the comparison of centralized and decentralized search results in Section 6.4.2, that the centralized search is more efficient in almost all cases. Leaving aside (without neglecting) the specifics of technical implementation of both types of algorithm, observe that the design of research questions for the computational experiments follows the hypothetico-deductive method of science[22]. In our case this method implies formulating a goal and then comparing how well centralized and decentralized algorithms fare in reaching it. This approach is perfectly legitimate in pragmatic terms, yet also misses the point – because the very existence of a common goal is set to measure efficiency, yet decentralized computing and algorithms deal with uncertainty instead. Therefore, the two are not easily comparable. Obviously, a computing model that deals with efficiency (i.e. centralized) will fare better when presented with a task requiring efficiency. However, this does not say that there is no point in comparing the two approaches – but that a just comparison has to involve both efficiency tasks and uncertainty tasks and the latter are more difficult to design, if at all possible within the framework of the hypotetico-deductive method.

Furthermore, observe that setting a goal for an algorithm in computational terms means setting a halting criterion for a computational process in the Turing (1937) sense. However, open-ended computing is conceived specifically for being able to define computation which is not dependent on explicit halting criteria. In OfferNets experiments it is revealed by the following: while a centralized search can be performed and then checked for correctness once, after the algorithm has been halted, a decentralized search has to check the cycle and path found by every asynchronous process internally and then halt when one is found – which clearly is more computationally costly in a goal-directed setting. Conceptually, this insight points to the observation that self-organization, open-endedness and open computation are more costly computationally and in terms of resources than goal-directed processes and closed computation when approached from the perspective of their final product. Consider, for example, biological evolution. The absolute majority of species that ever lived on Earth are extinct. Moreover, most of the evolutionary variations (a sort of decentralized exploration) are unsuccessful. If looking from the perspective of "final products" – i.e. currently living species – evolution is a huge waste of resources. Yet, if looking from the starting point, where nothing is determined, the explorations that supposedly lead nowhere are necessary for "organizing the unknown". This metaphor applies also to AI research perspectives (see Section 2.1) and above all to the distinction between AI and AGI. In the same manner, if the goal of an AI engine, machine learning implementation or a robot is known in advance, it will be always more computationally efficient to design a concrete algorithm which reaches that goal. Yet if one is concerned about intelligence expansion, radical novelty and the diversity of its forms, one is necessarily faced with the issue of dealing with uncertainty, which no combination of goal-directed AI engines or processes can account for. For one, this allows us to appreciate the power of the computational metaphor.

[22] See Section 1.3 for discussion of the hypothetico-deductive method of science.

Moreover, depending on the level of self-organization, surprise, generality and ability to deal with uncertainty that is desired from our systems – computational, social, economic or socio-technical– we need to balance efficiency criteria with the additional resources necessary for self-organization. The conclusion that determinism is a special case of non-determinism (see Section 4.2.3) is equivalent to the statement that goal-directed intelligence and behaviour is a special case of open-ended intelligence (see Section 2.2.3). In summary, self-organization itself is computationally costly and may be not the most efficient way of achieving results, when the goal of computation is known. On the other hand, computational self-organization is necessary when the problem domain is not clear.

The overall role of this chapter in terms of implementation of the offer networks domain model on the basis of open-ended decentralized computing architecture is to demonstrate the importance of connecting the deep philosophical and conceptual framework to concrete software development decisions. We believe that such connections play a crucial role in complex software development programmes, having long term and broad implications, certainly related to the artificial intelligence research programme, but also to complex engineering undertakings in general. We may call this approach *philosophical engineering* – not in the sense of engineering philosophy, but in the sense of emphasizing the role of philosophy *for* engineering complex systems. The next chapter presents and discusses how the open-ended decentralized computing model, and the conceptual paradigmatic shift that it carries, offers pragmatic perspectives to some of the most complex challenges of developing and governing contemporary socio-technological systems.