Carnegie Mellon

OrgAhead: A Computational Model of Organizational Learning and Decision Making [Version 2.1.5]

CASOS Technical Report
Ju-Sung Lee[1] and Kathleen M. Carley[2]
December 2004
CMU-ISRI-04-117

[1]Social and Decision Sciences,
Carnegie Mellon University
jusung@andrew.cmu.edu

[2]International Software and Research Institute,
Carnegie Mellon University
kathleen.carley@cmu.edu

Abstract. OrgAhead is a computational model of organizational learning and decision-making. The simulated organization consists of agents whose communication structure resembles hierarchies and whose primary goals are to **learn** the correct decision or answer to one or more tasks, or objective functions (e.g. typically the majority classification task); we refer to these task functions as the task environment. The organization also seeks to **adapt** to an optimal structure under the specified, and possibly changing, task environment, by admitting changes in the form of turnover and re-assignment of personnel and tasks. OrgAhead can be used to test various aspects of real life organizations, such as complexity in the task environment and constraints on structure and adaptability, under the intellective paradigm of simulation models. An intellective model contains analogous entities, constructs, and complexities of the modeled organizations rather than mimicking each specific behavior.

Keywords: simulated annealing, organizational learning, adaptation, dynamic, decision-making, organizational model, computational organization theory**.**

1 Overview

OrgAhead is a computational model of organizational learning and decision-making. The simulated organization consists of agents whose communication structure resembles hierarchies and whose primary goals are to **learn** the correct decision or answer to one or more tasks, each in the form of an objective function, which can be as simple as a majority classification task; we refer to the task function(s) as the task environment. The organization also seeks to **adapt** to an optimal structure under the specified, and possibly changing, task environment, and has the ability to admit changes in the form of turnover and re-assigning personnel and tasks. As an adaptation feature, the organization has a **look-ahead** ability that allows it to assess the short-term impact of a change before taking action; this is where the "Ahead" part of the OrgAhead comes from. The look-ahead ability can be used in conjunction with one of two optimization heuristics, hill-climbing or simulated annealing [Carley and Svoboda, 1996]. With hill-climbing, the organization selects only beneficial moves or changes at every opportunity for change. Under simulated annealing, the selection of moves depends on an annealing schedule that would allow the organization to select some bad moves, so that it wouldn't get caught in local optima.

The OrgAhead is used to test various aspects of real life organizations, such as complexity in the task environment and constraints on structure and adaptability, under the intellective paradigm of simulating models. An intellective model contains analogous entities, constructs, and complexities of what it is modeling rather than mimic each specific behavior. Hence, the sizes of organizations will tend to be small, less than 100 agents. Models that capture a higher level of organizational detail are called 'emulative' models.

OrgAhead can be (and has been) used for both validation and hypothesis testing. Validation exercises include comparison of performance results with those of real-life A2C2 teams [Lee et al, 2003] and nursing units across multiple hospitals [Lee et al, 2003]. We have also validated OrgAhead with an emulative model, VDT or Virtual Design Team, through a process called 'docking' [Louie et al, 2003] in which two models are tested and compared using equivalent, or close to equivalent, inputs and parameters.

We can use and test a specified organization or allow OrgAhead to randomly generate them, testing more theoretical hypotheses. We often test hypotheses through virtual experiments, meaning we run Monte Carlo simulations for each of the experimental conditions and statistically compare results. Questions we have asked and answered include:

- How do adaptive organizations differ structurally from maladaptive ones? [Carley and Lee, 1998]
- What are the adaptation patterns that lead to higher levels of performance? [Lee and Carley, 1997]
- How does a hierarchy differ from teams and under what circumstances does one perform better? [Carley, 1992]
- Does initial learning, or training, have long-term effects?

OrgAhead is the successor and aggregate of a series of past organizational simulation models as the following figure shows.

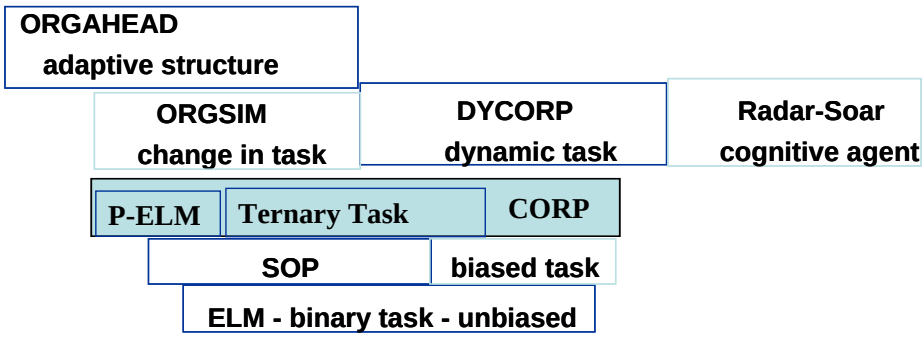

Fig. 1. Lineage of OrgAhead consists of half-a-dozen or so distinct models.

2 Details of OrgAhead

2.1 Structure of the Organization and Task (Static Representation)

The two primary components of OrgAhead are the organization and the task that the organization works on and solves. The following figure depicts a static snapshot of a typical organization that OrgAhead might model:

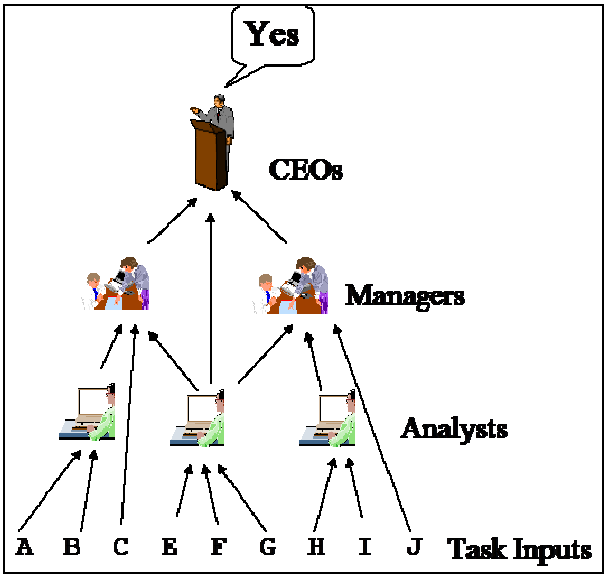

Fig. 2. Typical hierarchical organization having three levels. Task inputs appear as a feature vector. Top level agents give the final decision or answer, upon which performance and adaptation is based.

At the bottom of the figure is the representation of the task or inputs level. Different actors may see different parts of the input/task. The organization here is a three-level hierarchy; OrgAhead can model an arbitrary number of levels each containing up to an arbitrary number of agents. Traditionally, we label these levels similarly to those of corporate organization: analysts, managers, and CEOs. Decisions are communicated upwards the hierarchy; that is, analysts can only report to managers or CEOs, and

managers to only CEOs. Currently there is no intra-level communication. Finally, the upper level agents provide a final decision or answer as its response to the task vector; here the response can be a 0 or 1 and it is 1. In the figure, the organization works on a single task and receives feedback from the environment as to whether its answer was correct or not.

The authors have used the radar task as a metaphor for the kind of task that OrgAhead tries to solve. An incoming aircraft is detected and the organization, or radar-tower, must assign it the appropriate danger level: friendly, neutral, or hostile.

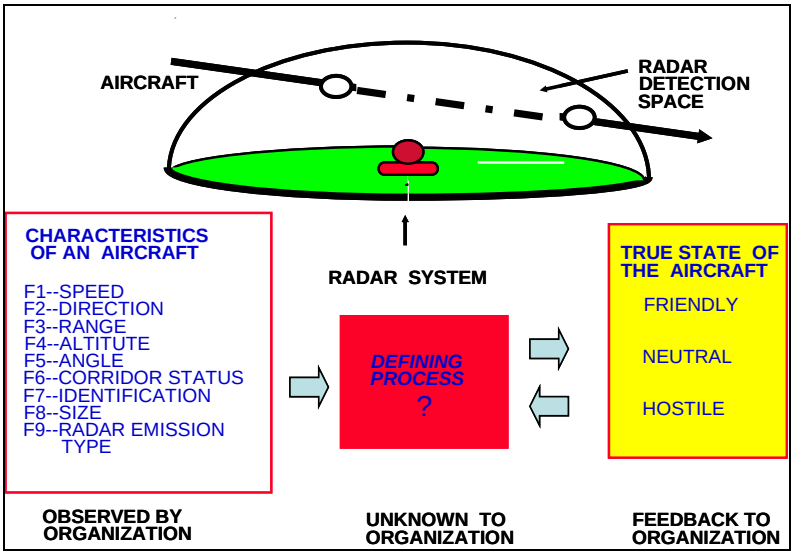

Fig. 3. A sample empirical task. The radar task requires the organization to properly categorize a detected, incoming aircraft as friendly, neutral, or hostile using a finite set of feature values about aircraft. Similar classification tasks map well onto the OrgAhead task schema.

2.2 The Dynamic Organization

However, we cannot infer general behavior from the solving of a single task instance. The power of computational modeling lies in the ability to generate multiple instances of our problem allowing us to make statistical inferences from a population of samples. So, the modeled organization works on a series of tasks; the length of this series is defined by the user as a parameter (-task_limit) and constitutes the life cycle of the organization. Furthermore, since OrgAhead does not have an explicit representation of human time, we **think** of the series of tasks or task cycles as the proxy for time. The following figure depicts OrgAhead operating across time (i.e. tasks):

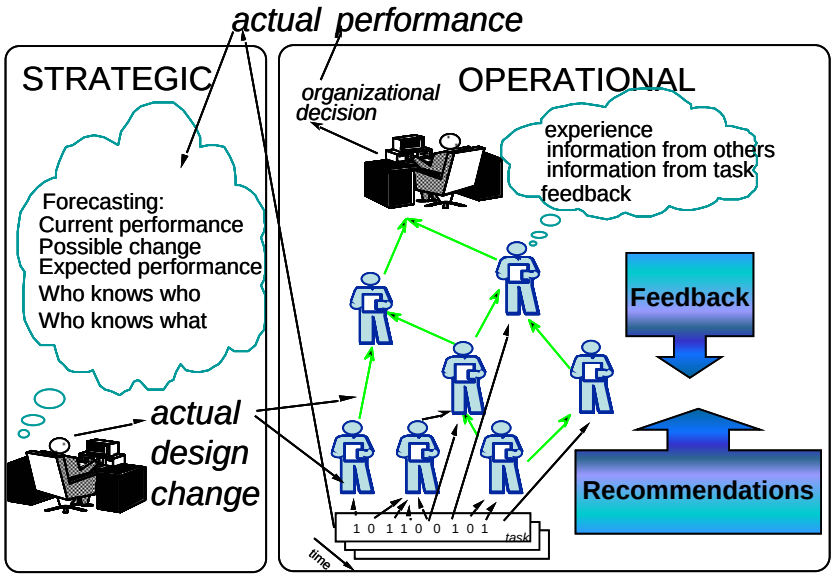

Fig. 4. Organization working on tasks over time. This figure also characterizes the feedback of information constituting learning and the strategic behavior including forecasting of performance and effecting change.

As the organization solves the series of tasks, it keeps a count of correctly performed tasks (i.e. correct answers). The proportion of correctly answered tasks out of total tasks worked on is the organizational performance; in the model output, we use the term "efficiency".

We briefly introduce OrgAhead's two primary performance output measures, absolute efficiency and relative efficiency. Absolute efficiency measures the percentage of correct answers for all tasks worked on, as defined by the `-task_limit` parameter. However, say we are interested in how the organization is doing every *x* many task cycles. The `-efficiency_cycle`, or `-ec`, parameter specifies this periodic check.

`OrgAhead -tl 20000 -ec 500` means we simulate an organization over 20,000 tasks. However, we take efficiency/performance checks every 500 task cycles or windows.

2.3 Detailed Organization

The next figure shows the same organization with more operational detail:

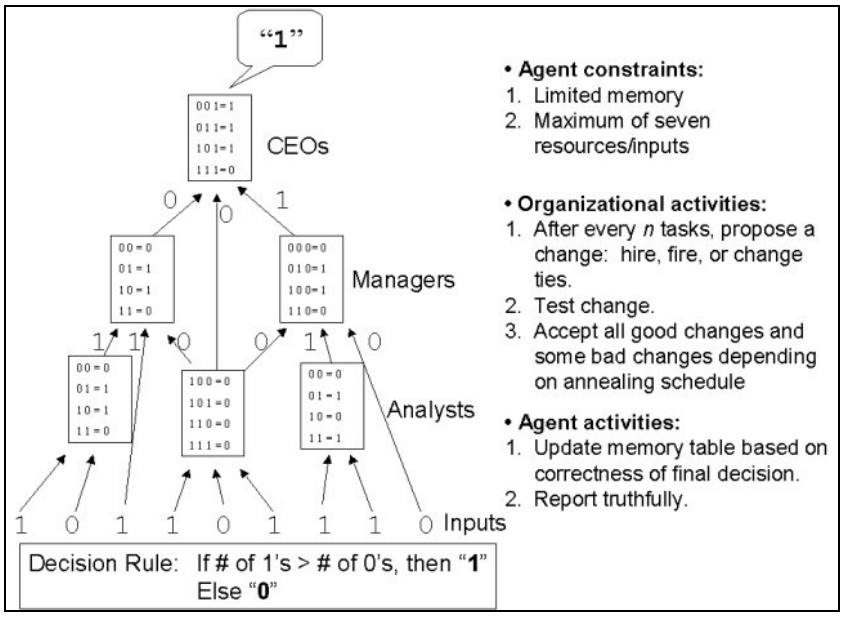

Fig. 5. Detailed organization: agents appear as memory tables and the task is represented as a binary vector. A list of common dynamics are listed on the right

We see now the numerical task representation: a series of binary bits; OrgAhead can handle up to trinary bits for the task. Through the course of the simulation, the organization sees many of such task instances, typically generated randomly from a Bernouilli distribution, meaning that each bit has a 50/50 chance of being a 0 or 1.[1]

The objective function, or decision rule, typically used is a majority classifier. This means, for a given task bit vector, if there are more 1s than 0s, the correct answer is 1, and 0 otherwise. In this example, we show only one decision rule meaning there is only one kind of task to solve. The job of the organization is to learn and adapt to produce correct answers as often as possible.

The organization employs reinforcement learning to improve accuracy. Each agent in the figure is assigned resources from one or more sources, either task information or reports from a subordinate. These interconnections form the organizational "structure". How these are initially assigned is left up to the user: the assignments can be specified or randomly made. The agent makes a single decision (for each task) based its inputs and reports its decision to its superiors.

For a binary task, each communication takes on a value of 0 or 1. For each combination of values an agent sees, the agent keeps track of which combinations produced the correct answer for the entire decision rule. For instance, say an agent has two resources; it does not matter who the sources are, task or another agent. There are four combinations of values this set (of two resources) can take: 00, 01, 10, or 11. For each task instance, the organization sees, this agent will receive information from its resources which will take on one of the combinations. The agent refers to its memory to see which decision (0 or 1) in that past has been most often matched the true answer and passes its decision its superiors. If the agent has no experience (i.e. the organization is just starting out), its decision is randomly chosen.

[1] The Bernoulli probability can be user-altered; e.g. instead of 50/50, it could be 30/70.

The following figures provide a glimpse of the organizational performance pattern over time as the organization works on a series of tasks as well as how performance can be dependent on the most basic of parameters such as organizational size and density of communication links.

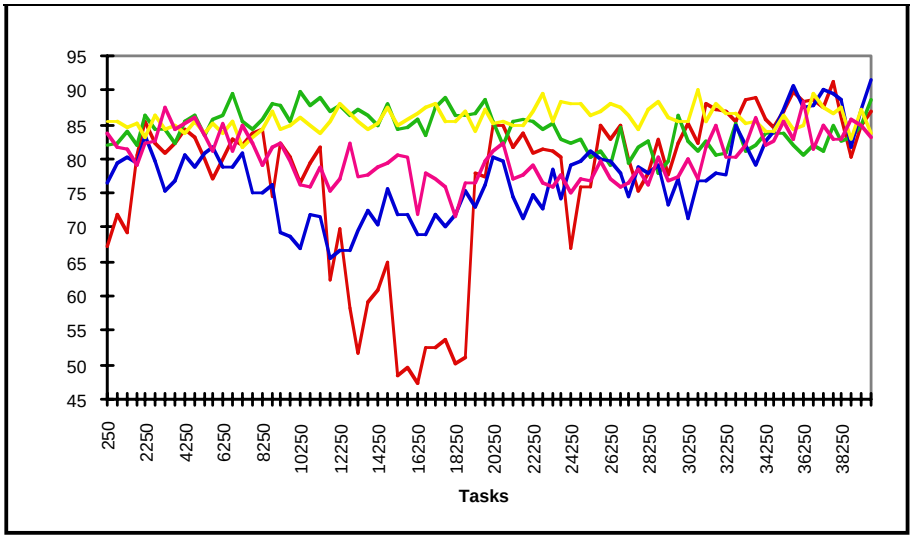

Fig. 6. Performance time series. Y-axis shows performance as a percentage (45% to 95%) while the X-axis shows the cumulative number of tasks worked on across time (i.e. sequentially). Each line depicts the performance measures of a separate run of OrgAhead. The behavior of OrgAhead is rarely perfectly stable and often large variations in performance occur as a result of other externalities such as specific parameters or maladaptations.

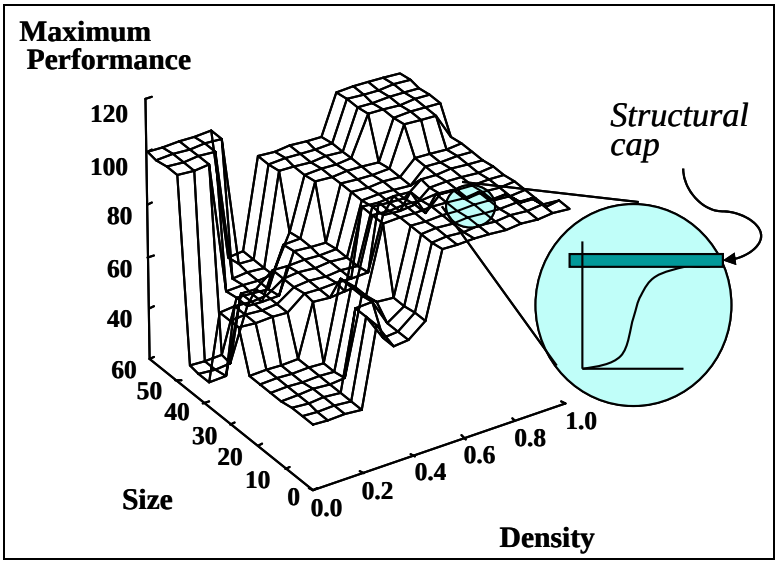

Fig. 7. Performance surface response. OrgAhead's performance is highly contingent on the interconnectivities (i.e. density) and the raw number of agents (i.e. size) in the organization. However, the relationships can be neither monotonic nor linear as depicted in the figure.

2.4 Organizational Adaptation

While we can test an unchanging organizational structure using OrgAhead, the power of this tool comes from the adaptation component. After every *x* tasks the organization works on, the organization proposes a change in its structure. The user defines how often this can occur and also the details of the change. Currently, possible changes include:

1. Turnover (i.e. hire a person, fire a person, or replace a person)
2. Task re-assignment (i.e. change, add, or remove a link between a task bit and a person)
3. Personnel re-assignment (i.e. change, add, or remove a link between people)

The `-change_cycle` (`-cc`) parameter determines how often the organization proposes a change. For example `-cc 500` means, at every 500 tasks worked on, the organization randomly generates a change.

The organization does not automatically accept and implement the change. OrgAhead implements a look-ahead feature that allows the organization to test the change for a short time horizon, which can be defined by the user. If the change produces a higher level of performance/efficiency, the organization will accept the change. If the change is not necessarily performance-enhancing, the organization *might* accept the change anyways depending on the simulated annealing schedule. The solution landscape for organizations is often complex and reaching an optimal solution or organizational form requires a more sophisticated strategy than merely picking the better move at every opportunity; this is also known as *hill-climbing*. OrgAhead overcomes this limitation by allowing the organization to sometimes take bad moves; this algorithm is known as *simulated annealing*. Refer to [Carley and Svoboda, 1996] for further details. Typically, nascent organizations suffer from what organization theorists call a 'liability of newness', meaning they are uncertain at their outset and should take risky moves in order to grow and adapt. However as organizations mature, they need not take so many risky moves. The probability of accepting risky or bad moves is determined by an exponential function taking several user-defined parameters:

Probability of accepting move (Metropolis criterion):

[1] $\boldsymbol{p}_t = e^{-\boldsymbol{cost}_t * k / \boldsymbol{T}_t}$

Cooling of "temperature" $\boldsymbol{T}$:

[2] $\boldsymbol{T}_{t+1} = \alpha \cdot \boldsymbol{T}_t$ where $0.0 < \alpha < 1.0$ (cooling ratio)

Cost of next move:

[3] $\boldsymbol{cost}_t = \boldsymbol{current\_performance}_t - \boldsymbol{lookahead\_performance}_t$

The following figure shows what risk probability curve looks like, as measured from a set of Monte Carlo runs; the theoretical curve would be perfectly smooth:

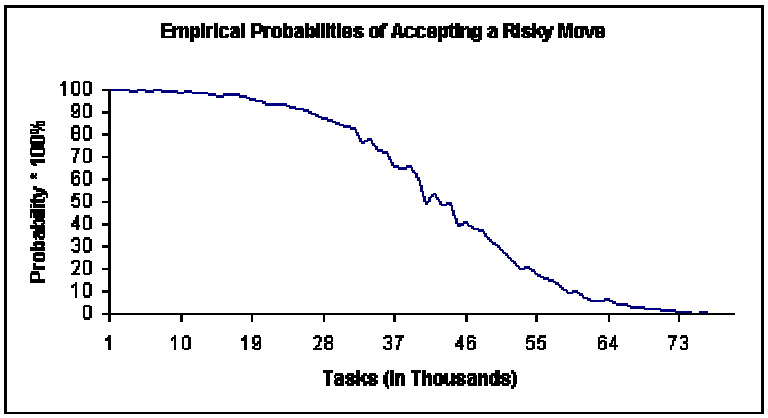

Fig. 8. Annealing curve denoted by percentage of risky moves accepted at various points of the organizations life cycle

The shape of this curve is somewhat user configurable. How fast the curve drops and whether not it spikes up again, restarting at 100% are configurable.[2]

2.5 Other Primary Parameters

2.5.1 The memory cycle and the relative efficiency cycle.

All agents have the same amount of memory, or a tally of each combination of resources seen and which of the agents' decisions were correct. This memory capacity is set with the –memory_cycle (-mc) parameters. This means the memory is zeroed after every *x* tasks, where *x* is the number set with –memory_cycle. The memory cycle parameter is often coupled with the efficiency cycle parameters, -ec, that we discussed earlier.

```
OrgAhead –ec 500 –mc 500
```

After every 500 tasks the organization works on, each agent's memory is reset and the performance for the past 500 tasks is recorded, and output if desired.

2.5.2 Standard Operating Procedures (SOP)

OrgAhead allows agents to make decisions in a routine fashion, using organizational SOP, rather than their experiences. The `-s` or `-stupid` switch parameter forces each agent to simply pick the most common input as its output/decision. SOP can be set probabilistically for each level by first setting `-s` and the using `-sop <level> <probability>` where `<level>` starts from 0 for the lowest and `<probability>` is a real number between 0.0 and 1.0.

[2] OrgAhead's Medeiro parameters define an annealing curve that spikes back up to, or near to, 100% temperature; refer to publications by F. Medeiro for further information on the advantages of this kind of cooling curve.

2.5.3 Personnel Restrictions

Maximum Resources (`-max_resources` or `–mr`) restricts the maximum number of resources/inputs an agent may have whether these resources are task bits or decisions from subordinate agents. Default is 7 resources.

Maximum People (`-max_people` or `-mp`) restricts the maximum number of people that can exists at each hierarchy level. Default is 15 people.

2.5.4 Specifying the Organization

By default, OrgAhead randomly generates an organization of up to three tiers and with up to fifteen people in each level. The user can specify the exact structure of the initial organization using meta-matrices (see Version 2 section below) or through the command-line options.

2.6 Task Specifications

Thus far, we have only touched upon one way the task environment can vary (i.e. binary vs. trinary type bits). In this section, we go into all the ways the user may configure the task environment.

2.6.1 Task Complexity

Task complexity is the size of the task, or the number of bits that represent each task vector. It is crucial for the user to understand the implications of this parameter since it can drive much of the performance/efficiency outputs.

The task complexity is defined by the `–task_complexity`, or `–tc`, parameters and defaults to 9 bits; several other task parameters default to values that assume the 9 bit default.

i.e. at some command-prompt: `OrgAhead –tc 9` or `OrgAhead –task_complexity 9`

2.6.2 Task Generation

Internally, a task bit can take on three values, 1, 2, or 3; why these aren't 0, 1, and 2 will become clear in section.

2.6.3 Task Bit Generation

By default, these are randomly generated with almost equal probabilities, .33, .34, and .33. These probabilities are set with the `–task_friendly`, `-task_neutral`, `-task_hostile` parameters, or `–tf`, `-tn`, `-th`, respectively. The previous figures used a

binary task, which requires an alternate setting of these probabilities: `OrgAhead -tf .5 -tn .0 -th .5`

2.6.4 Decision Rule

The decision rule, that determines the correct answer, has two components: the objective function and a partitioning, or set of cut-offs.

The default objective function multiplies each of the bits. Say at task cycle *t* the task vector $\boldsymbol{x}$ of size 9 is generated. The correct answer $\boldsymbol{y}_t$ is:

Default Objective Function:

[4] $$\boldsymbol{y}_t = \prod \boldsymbol{x}_{it}$$

The range of values for $\boldsymbol{y}_t$ is partitioned into 2 or 3 whole regions; 2 for binary and 3 for trinary task. Currently, OrgAhead does not allow un-segmented partitioning.

The task cut-off parameters determine which partitions represent an answer of 1, 2, or 3. The parameters are `-primary_cutoff_friendly` and `-primary_cutoff_hostile` or `-pcf` and `-pch`. The partitioning formula is as follows:

[5] friendly region < friendly cutoff <= neutral region
[6] neutral region <= hostile cutoff < hostile region

Any objective value less than `-pcf` has an answer of 1 or friendly. Any value that is greater than `-pch` has an answer of 3 or hostile. Any other value is neutral or 2. The defaults for these parameters assume a trinary task. If we wanted a binary majority classifier under 9 task bits, we would use parameters like `-pcf 82` and `-pch 82`. Setting `-pcf` equal to `-pch` automatically implies a binary task; none of the answers will be neutral or 2. Any product of nine digits of 1 or 3 that contains five or more (majority) 1s will result in values of 81 or less. Any product having five or more 3s will result in values of greater than 81; actually 243 and above.

2.6.5 Changing the Task Environment

Users have the option of making the task environment even more uncertain, by allowing the decision rule to shift. Secondary parameters, `-secondary_cutoff_friendly` and `-secondary_cutoff_hostile` (`-scf` and `-sch`), control a secondary decision. How it takes effect is determined by the `-primary_duration_mean` and `-secondary_duration_mean`, which denote how long the primary decision rule is in effect and then how long the secondary rule remains in effect. If these durations sum less than the `-task_limit`, we go back to the `-primary_duration_mean` and the `-pcf` and `-pch` decision rules.

2.7 Inter-Relation of Parameters

How the organization performs, that is the range of performance values it exhibits, is strongly constrained by how the user sets the various cycles, the task complexity, and the cut-off/decision rules. Let's take an extreme example. Let's say the task complexity is 2 (i.e. two bits of binary information). There are only four values that the task can take on: 00, 01, 10, and 11. The default memory cycle is 500, meaning the agents can remember feedback for the past 500 tasks. However, it doesn't take many examples of 2 bits to learn the majority classification pattern. An organization with just a single agent can learn this task perfectly within a few dozen tasks. A memory cycle of 50 would allow the agent to perform perfectly, and 500 is certainly over-kill.

The user should analogize the relationship between the cycles and the task complexity as the relative difference in complexities between the real life learning and adaptive capabilities of the organization and the complexity of the task it is working on. One can structure the parameters such that the organization will consistently perform at a miserable level (e.g. 30%) or an excellent level (e.g. 80%) of performance. Usually the goal is to structure the parameters such that there is sufficient variance in the results, as one would expect to see in the real life organization. Alternatively, one can imagine an organization with an environment that is heavily constrained, such that the model would also give results with low variance. The user in this case might use OrgAhead to find an optimal structure whose performance excels despite the constraints.

3 Version 2 Features of OrgaAhead

3.1 Meta-matrix Linking OrgAhead with CASOS Tools

The second generation of OrgAhead (Version 2) includes modeling of more complex organizational dynamics, appearing the form of a meta-matrix. The meta-matrix captures networks of dependencies important to organizational dynamics. Most CASOS tools, including OrgAhead, allow for meta-matrix inputs.

	Personnel	Resources	Tasks
Personnel	Networks (e.g., authority and communication) Size Span of control	Capabilities Coverage	Assignments Workload
Resources		Substitutes Uniqueness	Needs Usage
Tasks			Precedence Complexity

Fig. 9. Meta-matrix comprises key dependencies in organizational dynamics. OrgAhead does not implement all of them

The meta-matrix for OrgAhead includes people (or personnel), resources, and tasks. Another meta-matrix component which OrgAhead does not currently implement is knowledge. The personnel-to-personnel matrix refers to the communication network already discussed. Personnel-to-resources matrix refers to agent links to task bits, which are now called "resources" in version 2. Earlier, we implied that the organization solves one kind of task, for which there is only one set of primary cutoffs or one primary objective function. In Version 2, the organization may be required to solve multiple tasks, each of which is dependent on all or a subset of the resources, or task bits. Remember, "resources" originally referred to any kind of input into an agent. In version 2, we make the distinction between personnel links and task-bit information links. Hence, we assign multiple tasks to various personnel (personnel-to-tasks matrix), and have those tasks depend on various resources (personnel-to-resources matrix). Furthermore, we can specify an ordering or precedence for the tasks (tasks-to-tasks matrix). OrgAhead does not currently employ resource substitution (resources-to-resources matrix).

The meta-matrices for OrgAhead need to be contained in a text file.

4 Running OrgAhead

OrgAhead runs under Windows and two Unix platforms, Solaris (Sun Microsystems) and Linux. Under Windows, users have the option of running OrgAhead from the command-line or use the GUI (graphical user interface) implemented in Java. The command-line version allows for batch/scripted runs of OrgAhead.

4.1 Inputs

The primary OrgAhead inputs and associated command-line parameters appear below. Omitting a parameter engages its default value. An exhaustive list of parameters appears in the Appendix.

4.1.1 Organizational Structure

Organizational structure may be specified through a file or on the command line. The command-line options accommodate up to three hierarchy levels and are:
`-ceos`, `-managers`, and `-analysts` or `–c`, `-m`, and `–a`.

Refer to the Appendix X for details on the format of the structures. The `–sfn`, `-structure_filename` option allows the user to specify a file containing the metamatrices. Meta-matrices may be specified using

```
-sfn <filename>
-structure_filename <filename>
```

`<filename>` includes specifications for the number of levels and the number of people in each level as well as the following meta-matrices: People X People, People X Resources.

People X Tasks, Resources X Tasks, and Tasks X Tasks. Refer to Appendix X for details on the meta-matrix.

4.1.2 Task Environment

The partitioning of the primary solution space is specified with the `-pcf` and `-pch` options (`-primary_cutoff_friendly` and `-primary_cutoff_hostile`). If a secondary task is specified, then a set of secondary cutoffs is appropriate using `-scf` and `-sch`.

4.1.3 Operation

`-ec` / `-efficiency_cycle <integer>` A performance review/check every <number> tasks.
`-cc` / `-change_cycle <integer>` Org attempts a change every <number> tasks.
`-mc` / `-memory_cycle <integer>` Agents remember feedback of <number> tasks.
`-tp` / `-training_period <integer>` Agents initially train on <number> tasks which does not count towards performance.

4.1.4 Annealing

`-ip` / `-initial_partition <real>` Setting for initializing the annealing curve.
`-fp` / `-freezing_partition <real>` Setting for determining when the problem is "frozen".
`-cr` / `-cooling_ratio <real>` How fast the temperature drops each change cycle.

4.1 Outputs

`-po` / `-print_organization` Organizational structure at each change cycle.
`-pc` / `-print_change` The change attempted.
`-pe` / `-print_efficiency` The efficiency/performance at each check.

The following produces much output:
`-pt` / `-print_task` Print each task the org solves and decisions made by each person and the org.
`-pp` / `-print_persons` Print the memory tables of each person.

4.1.1 Illustrative Input and Output

The following parameters have been used for various OrgAhead publications and worksheets; these may be set through the GUI as well (see below).

```
OrgAhead –ec 500 –cc 500 –mc 500 –tp 500 –ip .95 –fp 1e-900 –cr .7 –pcf
82 –pch 82 –tc 9 –tl 20000 –tf .5 –tn .0 –th .5 –po –pe –pc
```

Explanation of parameters:

```
-ec 500, - cc 500 , -mc 500, -tp 500 , -tc 9 , -pcf 82
```

Since the various cycles and task complexity are interrelated (see 2.7), these parameters can give the organization a range of performance from about 40% to 90%. The cycles set to 500 are primarily aligned the task complexity of 9, which has a solution space of 512 (i.e. 2^9). `-pcf 82 -pch 82` gives us the majority classification decision rule. `-th .5 -tn .0 -th .5` produce only the friendly and hostile bits, our definition of the binary task.

`-ip .95 -fp 1e-900 -cr .7 -tl 20000` gives us a smooth annealing curve for 20,000 tasks whereby the probability of taking a costly move starts from 1.0 and shrinks to .0 without ever freezing. The `-fp 1e-900` guarantees non-freezing.

Notice we don't include a meta-matrix file nor specify the organization structure with the `-ceo`, `-manager`, and `-analyst` commands because for most of our OrgAhead experiments we deal with randomly generated, Monte Carlo organizations.

4.2 Using the OrgAhead GUI (Windows version)

The user may use the graphical user interface (GUI) for OrgAhead which works currently only for the Windows platform. Java JRE 1.4.1 must be preinstalled or will install as part of the main OrgAhead installation process. The interface may be downloaded from the main CASOS website. Currently, the GUI supports mainly version 1.0 of OrgAhead and the meta-matrix feature of 2.0.

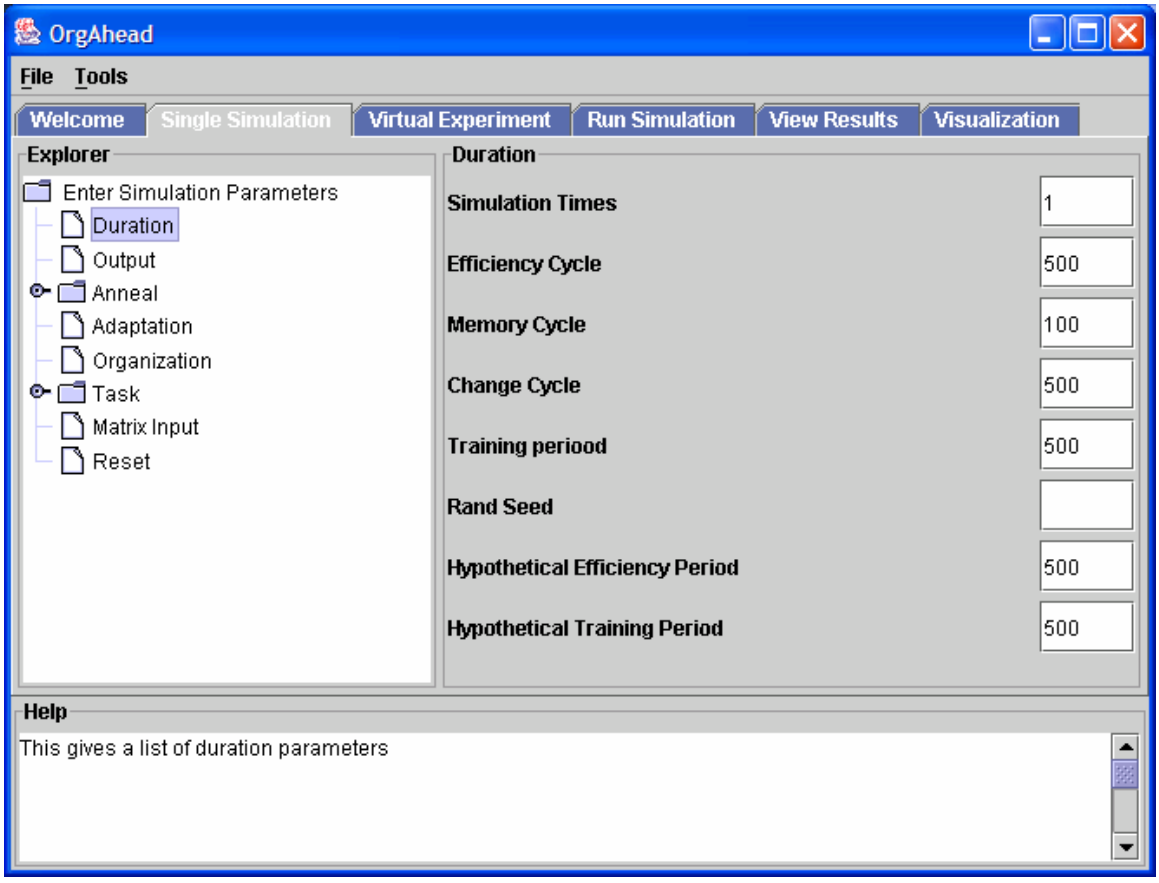

Fig. 10. A sample screenshot of the OrgAhead GUI. In this page, we see cycle/period settings

4.4 Benchmarks

The following measurements were taken using a Pentium 1.5 GHz machine using v2.1.6 of OrgAhead. There were 20 Monte Carlo runs, meaning randomized organizations, each of which ran for 20000 tasks. The organizations had a maximum hierarchy of 3 levels. We vary base the benchmarks on parameters that are likely to affect the running times: the maximum number of people per level (`-mp`) and maximum resources per agent (`-mr`).

Table 1. Performance times for OrgAhead varying size per level. Maximum resources is set to 7. Typical experiments with OrgAhead involve between three and twenty people per level.

Max people per level	Time
5	8.902s
10	14.521s
15	21.041s
20	34.114s
30	47.554s

Table 2. Performance times for OrgAhead varying resources or inputs per person. Maximum people per level set to 15. In typical experiments, agents will have between zero and twenty resources.

Max resources per person	Time
3	15.39s
5	22.64s
7	26.85s
9	51.94s

5 Validation

Validation on OrgAhead has included docking approaches and direct comparisons with empirical data across several different contexts. Docking refers to aligning the parameters of two similar models and comparing their results; empirical data may be also compared, if appropriate.

5.1 Validation by Docking with SimVision

OrgAhead has been docked with the simulation model SimVision developed at Stanford by Raymond Levitt et al. Alignment or docking involves assessing which of the important parameters of each model has analogues in the other.

Table. 3. This table shows results from docking OrgAhead with SimVision and compares both model outputs to the empirical data [Marcus et al, 2004]. The rank ordering of OrgAhead results matches the human performance scores; however the specific results of SimVision naturally differs from OrgAhead's.

TABLE I
COMPARISON OF MODEL PREDICTIONS

	ORGAHEAD 2002 (% Accuracy)	SimVision (Duration in days)	Human Performance (Outcome Scores)
A06	60.2	73.7 ± 2.5	79.7
A14	59.4	87.6 ± 3.0	76.2
A16	65.1	71.2 ± 2.5	85.1

For ORGAHEAD 2002 and SimVision, the mean performance over all simulation runs is reported. An analysis of variance demonstrates a significant difference between each of the organizational forms. The performance reported for the Real Data are the actual scores reported during the experiment.

5.2 Validation with Military Command and Control Structures

Validation of empirical data involves aligning OrgAhead parameters with those of the real situation or experiment and comparing OrgAhead outputs to measures obtained from the real situation. The Navy conducts A2C2 (adaptive architecture and command-control) war game experiments to assess the efficacy of organizational structure and information flow on tactical situations. We have used OrgAhead to reproduce some the results from these experiments.

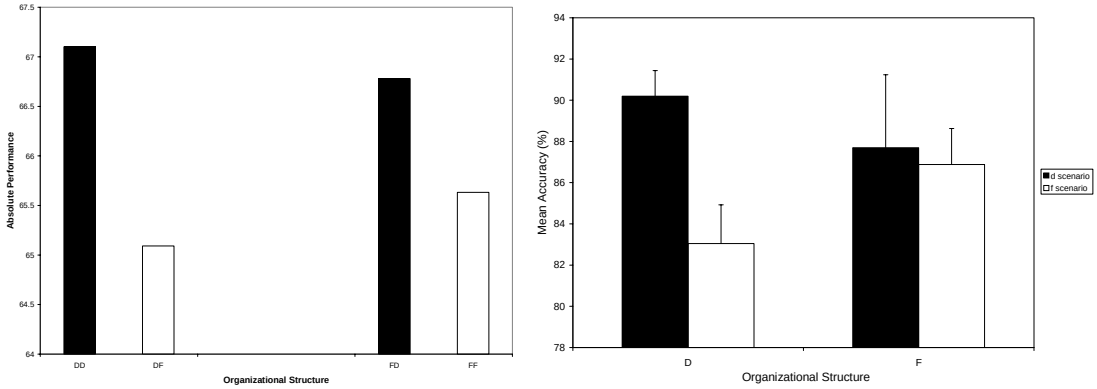

Fig. 11. Comparison of A2C2 performance results (left) with OrgAhead (right). The results above show the ranking of OrgAhead results matching the empirical results.

5.3 Validation with Hospital Unit Performance

Faculty and administrators at the Nursing College of the University of Arizona desired to use OrgAhead to help improve the efficacy of various hospital units in the city of Tucson, Arizona. By using expert informed parameters, we obtained the following comparison of OrgAhead results and the unit performance measures.

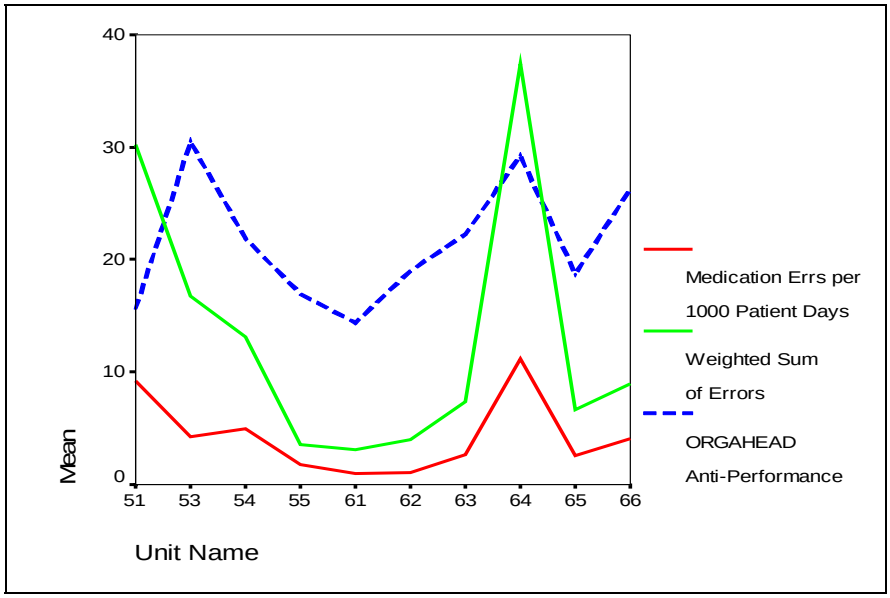

Fig. 12. OrgAhead results roughly match empirical performance of hospital units. The Spearman rank correlation $\rho = 0.60$ with significance/p-value = 0.067.

The correlation result shows strong predictive abilities of OrgAhead on the performance of these hierarchical groups; these results are especially surprising considering the small sample size.

6 Web Sources

CASOS Web Page: http://www.casos.cs.cmu.edu
OrgAhead Page: http://www.casos.cs.cmu.edu/projects/OrgAhead/index.html

7 Acknowledgements

The original version of OrgAhead was developed under NSF IST-8607303. More recently, research to extend, validate and develop an interface for OrgAhead has been supported in part by the National Science Foundation NSF IRI9633 662, 662, NSF KDI IIS-9980109, the NSF IGERT program in CASOS, and by the Office of Naval Research (ONR), United States Navy Grant No. N00014-97-1-0037, the Army Research Lab Contract No. DASW01-00-K-0018 and NASA. Additional support was provided by CASOS and ISRI 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 Army Research Labs, NASA, the National Science Foundation, or the U.S. government.

References

Carley, K.M. 1992, "Organizational Learning and Personnel Turnover." *Organization Science*, 3(1): 20-46.

Carley, K.M. and A. Newell. 1994. "The Nature of the Social Agent." *Journal of Mathematical Sociology* 19(4): 221-262.

Carley, K.M. and M. Prietula. 1994. "ACTS Theory: Extending the Model of Bounded Rationality" in *Computational Organization Theory*. Edited by Kathleen Carley and Michael Prietula. Hillsdale, NJ: Lawrence Earlbaum Associates.

Carley, K.M. and D.M. Svoboda. 1996. "Modeling Organizational Adaptation as a Simulated Annealing Process." *Sociological Methods and Research*. 25(1):138-168.

Carley, K.M. and J. Lee. 1997. "C2 Adaptation in a Changing Environment." pp. 287-297 in *Proceedings of the 1997 International Symposium on Command and Control Research and Technology*. June, Washington, DC.

Carley, K.M. 1998. "Organizational Adaptation." *Annals of Operations Research.* 75: 25-47.

Carley, K.M. and J. Lee. 1998. "Dynamic Organizations: Organizational Adaptation in a Changing Environment." Ch. 15 (pp. 269-297) in Joel Baum (Ed.) *Advances in Strategic Management Disciplinary Roots of Strategic Management Research*. Vol. 15. JAI Press, Inc.

Carley, K.M. 1999. "Learning Within and Among Organizations." Ch. 1 (pp 33-56) in Philip Anderson, Joel Baum and Anne Miner (Eds.) *Advances in Strategic Management: Population-Level Learning and Industry Change*, Vol. 16. Elsevier Science Ltd.

Carley, K.M. and D. Krackhardt. 1999. "A Typology for C^2 Measures." In *Proceedings of the 1999 International Symposium on Command and Control Research and Technology*. June, Newport, RI.

Carley, K.M., Y. Ren, and D. Krackhardt. 2000. "Measuring and Modeling Change in C_3I Architecture." In *Proceedings of the 2000 Command and Control Research and Technology Symposium*. June, Naval Postgraduate School, Monterey, CA.

Carley, K.M. forthcoming. "On the Evolution of Social and Organizational Networks." In David Knoke and Steve Andrews (Eds.) special issue of Research on the Sociology of Organizations on *Networks In and Around Organizations*. JAI Press, Inc.

Carley, K.M. and L.Gasser. forthcoming. "Computational Organization Theory." Ch. 7 in *Distributed Artificial Intelligence* edited by Gerhard Weiss. Cambridge, MA: MIT Press.

Lee, J. and K. M. Carley. 1998. "Adaptive Strategies for Improving C2 Performance." In *Proceedings of the 1998 International Symposium on Command and Control Research and Technology*. June, Monterey, CA.

Lee, J., K.M. Carley and J. Effken. 2003. "Validating and Justifying a Computational Model of Decision Making using Empirical Data." *NAACSOS 2003 Conference Proceedings*, Pittsburgh, PA.

Lin, Z. and K.M. Carley. 1993. "Proactive or Reactive: An Analysis of the Effect of Agent Style on Organizational Decision Making Performance." *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2(4): 271-288.

Lin, Z. 1994. "A Theoretical Evaluation Of Measures Of Organizational Design: Interrelationship And Performance Predictability", in K.M. Carley and M.J. Prietula (Eds.) *Computational Organization Theory*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Louie, M.A., K.M. Carley, L. Haghshenass, J.C. Kunz, and R.E. Levitt, 2003, "Model Comparisons: Docking ORGAHEAD and SimVision." *NAACSOS 2003 Conference Proceedings*, Pittsburgh, PA.

Appendix A – Details of Organizational Structure and Outputs

The information in this appendix also applies to OrgSim, a predecessor of OrgAhead, and OrgStat, an organizational statistical package which is the predecessor of ORA.[3]

Input Format for Initial Org Arguments (`-analysts`, `-managers` , `-ceos`)
Each of these arguments takes a quote-enclosed string. The string given should consist of words separated by spaces; each word indicates the resources to give to a person. For example "a 0b cd" means create 3 people, the 3rd gets two resources, and the others each get one.

Each word should consist of characters indicating the resources given to this new person. For example, the string "abc" means the new person gets the first 3 resources of the immediate lower level (i.e. CEO gets managers who get analysts who get tasks). Each character may have a modifying digit preceding it. The modifying digit can be '0' for task, '1' for analyst, or '2' for manager. Case is not important. This indicates that the level of the resource the person sees is not necessarily the default level for the person. For example "0b" indicates the person gets task B, even if the person is a manager or CEO. "a1bc" means the person gets resources A and C of the default level, and analyst B. (Of course, if the person is a manager, saying "abc" would have been the same as "a1bc".) A "-" indicates a person with no resources (who has to randomly guess at the answer), and a "." indicates no person (a silly thing to input, but nonetheless, it can be done). Passing an empty string indicates that there shall be no members on the level associated with the argument; however, if all three arguments get empty strings (`-analysts "" -managers "" -ceos ""`), then a random org is created for each simulation.

In addition, the `-president` switch indicates if the org should have a President, who oversees all CEOs and any unsupervised managers or analysts.

Output format for Organizations (seen if `-print_organization` is specified)
This format specifies the org's structural hierarchy, who supervises whom. If `-print_organization` is specified, it is printed once before simulation starts, once at the end, and every time a change to the org is made. Here is what a sample org might look like:

```
Organizational structure is: President: abc
a1a0a b1b0b 1c0c
abcgi defh
a b c d e f g h i
```

The bottom line indicates that the first analyst sees task A, the second analyst sees task B, and so on. The next line up shows 2 managers; the first one supervises the first, second, third, seventh, and ninth analysts, and the second manager oversees the other ones. The top line depicts three CEOs. The first one oversees the first manager (A), as well as the

[3] OrgStat and OrgSim are both available for public use.

first analyst (1A) and task (0A). The second one sees the second manager, analyst and task. The third one oversees the third analyst and task. Finally, the first line indicates that this org has a President, who supervises the three CEOs.

OrgStat reads orgs from a file or stdin in this same format. It only reads orgs whose preceding line reads "Final Organizational structure is". This way you could pipe output of OrgSim to OrgStat, and OrgStat will ignore initial and intermediate orgs, using only final orgs, one for each simulation performed.

Output format for People (seen if `-print_person` is specified)

This format specifies the experience of every person in the org. If `-print_person` is specified, this data is printed for every person after each efficiency check. Here is what a single person's experience might look like:

```
Manager #1 has resources: A1 T2  Eff:   43.00% (Recent:  43%) from 400 tasks.
For pattern: 1 1
   42 =  41    1    0  |   31    0    0     8    0    0    10    0    0 =    49
   13 =  12    1    0  |   12    1    0     1    0    0     6    0    0 =    20
    3 =   3    0    0  |    3    0    0     1    0    0     0    0    0 =     4
For pattern: 1 2
...
```

From the top line, we see that the first manager oversees the first analyst and the second task. He has an overall efficiency of 43.00, and a 43% relative efficiency from the last efficiency check. And he's seen 400 tasks. Then we will see a listing of each pattern he may see; only his output for pattern (1 1) is shown above. The data is collected into 4 3x3 matrices, and they indicate his absolute, initial, relative, and old relative experience, from right to left. The rows indicate what the correct answer was (top=1, middle=2, bottom=3), and the columns indicate what he guessed (left=1, middle=2, right=3). So, in the full 400 tasks, where the pattern (1 1) emerged, the answer was 1 42 times, and 41 of those times he guessed 1. However, when he started off (initial experience) there were 31 1's and he guessed correctly every time. Since the last efficiency check, there were 8 1's, and he guessed them all, and in the previous efficiency check, there were 10 1's and he guessed them all correctly again. The numbers on the left and right sides are sums, the right ones are useful in judging how he will react the next time he sees (1 1) Since the answer was 1 42 times (from the left side), we figure he'd be best to guess 1. That's what he'll do, because the right number (49) is greatest on the right side.

Output format for Tasks (seen if `-print_task` is specified)

This format specifies what happens for each task, what everyone decides, what the org decides, and what the correct answer was. If `-print_task` is specified, this is printed for each task. Here is what a single task might look like:

```
3
2     2     1
1     2     2     2     2     1     2     2
2     2     2     2     2     2     2     2  A: 3 D: 3
```

The bottom line shows that all the task bits were 2. The next line up indicates that the first (leftmost) and 6th analysts guesed 1; the others guessed 2. The next line shows that the first two managers guessed 2 while the third guessed 1. The top line shows that the CEO guessed 3. At the end of the bottom line, one can see that the answer was 3, and so was the decision.

Output format for Efficiency (seen if `-print_efficiency` is specified)

This format specifies what happens for each efficiency check, how everyone is doing, and how well the org is doing. If `-print_efficiency` is specified, this data is printed out during each efficiency check, before any orgal changes take place. A sample efficiency check might look like:

```
Organizational Efficiency:   53.00%  Overall:   54.03%
  54.00   54.00   37.00   54.00   54.00   54.00   54.00   45.00
  54.00   45.00   54.00   50.00
  37.00   53.00   48.00   45.00           47.00   51.00   56.00   47.00
```

The org's overall efficiency is 54.03%, while its relative efficiency (since the last efficiency check) is actually 53%. From the bottom line, we see the analysts all had efficiencies ranging from 37-56%. The managers, one line above, ranged from 45-54%, and the CEOs ranged from 37-54%.

Experience: How Each Member Learns

Each person has a set of resources, which may be tasks, or the decisions of his inferiors (or both). From these resources, he sees a 'pattern', a single vector of numbers ranging from 1-3, and he must guess if the true answer is 1, 2, or 3 based on this pattern. He does this by storing experience matrices for every possible pattern. So when he next sees that pattern, he knows what the answer has been recently, and therefore can make a good educated guess on what the answer will be this time.

For each pattern, he stores 4 matrices. The first one, known as the initial matrix, records all of his experience while he is being trained. (He is in training as long as he has less than 500 tasks, this number is passed to the program under the parameter `-training_period`). When he is no longer in training this matrix ceases to be incremented.

He also stores relative experience and old relative experience. Assuming an efficiency check occurs every 100 tasks (the default), his relative experience will encompass all tasks he has seen since the last efficiency check (which will be less than 100 tasks). After the next efficiency check, his relative experience matrix gets copied to his old relative experience matrix, and then gets zeroed. So his old relative experience matrix records his guesses for the last 100 tasks before the last efficiency check.

The fourth matrix records his absolute experience, it gets updated for every task he views, but he does not use it in making decisions; it mainly exists to examine his efficiency.

Each matrix is a 3x3 matrix indicating how many times the member guessed 1, 2, or 3 and how many times the answer was 1, 2, or 3. Whenever a person receives feedback, he

increments a single index in his relative and absolute matrices, as well as his initial experience if he is still in training. As long as a person's resources don't change, his matrices give an accurate history of what tasks have transgressed.

When his resources change, then the numbers of his matrices will be altered to reflect the change. If he loses a resource, then the matrices that represent patterns only differing by that resource are summed together. When he adds a new resource, all his matrices are triplicated and divided by 3. Thus, adding and then deleting a resource should leave experience matrices close to their initial values (there will be precision errors from discarded remainders when dividing numbers by 3).

Making a Change in the Org

Org change is handled very formally, and has different meanings in OrgSim and OrgAhead. However, both treat change very similarly, so we will look at their similarities first, and then the difference in org change that distinguishes OrgSim from OrgAhead.

OrgSim and OrgAhead conduct org change differently, but they may only change the org at specific points in the simulation. Periodically they stop simulation and attempt a change...if a change succeeds, the simulation continues with the changed org, otherwise, the original org continues until the next time for changing comes up.

There are five ways to change an org: hire someone new, fire someone, add a connection between two members, or a member and a task, change a connection, and delete a connection. Each of these may be done several times at once, but all the times must apply to the same level (or levels). For example the org can hire 3 new analysts, or change 2 connections between managers and tasks, but it may not hire an analyst and a manager at the same time.

Once a specific type of change is requested, OrgSim and OrgAhead both proceed with the type of change in the same way. First they decide how many such changes can occur. This can be specified as either a constant (hire 2 people), or the program can be instructed to randomly decide how many changes to do based on a Poisson distribution.

Next, the programs pick a level of the org (for hiring and firing), or they pick a superior and inferior level (for add/change/delete connections).

Then they must decide which people on the levels picked are influenced. Usually there are parameters to decide this, and the options will be adjectives like "best", "worst", "busiest", "laziest", and "random". Specifying "best" means use the person on that level with the highest relative efficiency, and "worst" means the person on that level with the lowest relative efficiency. Similarly "busiest" means the person on that level with the most tasks and "laziest" means the person with the least tasks. Finally, "random" means pick anyone on that level without regard to their efficiency or resources.

At this point the idiosyncrasies of each change come into play. The behavior of each change is described below:

For hiring, the level chosen indicates at what level the new member will exist in the org. If that level is full of members (the number of members meets a limit variable set by the program), no one may be hired on that level. The new person can receive no resource, or a random task or inferior person to supervise. Or he can receive resources from a 'mentor', a colleague on that same level. Like all members chosen, you can elect to use the busiest, laziest, best, worst, or random person to be a mentor. The mentor gives half his tasks to the new person. You may elect to have the mentor continue to supervise the resources he gave away, or to drop them. Finally a superior on the next level is assigned to supervise the new person (unless the new person is a CEO).

The above scenario is typical for when a person is hired on a level that already has people on it (who could serve as mentors). When a person is hired on a level with no current people on it, the program behaves slightly differently, depending on the level. When a CEO is added to an org with no CEOs, he supervises all the managers, or all the analysts if there are no managers. When a manager is added to an org with no managers, he supervises all the analysts, and the CEOs that were (presumably) supervising the analysts drop them. Or the manager supervises all the tasks if there are no analysts. When an analyst is added to an org with no analysts, he gets all the tasks his supervisor is currently overseeing. One final note: instead of a person supervising all the resources on a level, the program can be directed to give him one resource at random on that level instead.

For firing, the level chosen indicates what level the 'victims' occupy. A victim must be chosen from the level, and all his resources are given to one of his colleagues. It is possible to fire the last member on a level, but only if other members exist on other levels... you cannot fire the last person in an org. As with hiring, the programs behave slightly differently when the last person on a level gets fired. When the last CEO gets fired, his resources go to anyone that can supervise them (keeping mind that a supervisor must be on a higher level than a resource he must supervise). When the last manager gets fired, his task resources go to the analysts, and his analyst resources go to the CEOs if the CEOs were supervising the manger. When the last analyst gets fired, his tasks get distributed amongst his supervisors, or the existing managers, if no one is supervising the analyst.

When adding a connection, two levels must be chosen, the level of the superior and the level of the inferior, and someone on the superior level winds up supervising someone (or something) on the inferior level. The program first decides if the inferior resource must be an 'orphan' resource, that is, the resource is currently not supervised by anyone on the superior level. Then it picks an appropriate inferior resource, and superior person, and directs the superior to supervise the resource. (It ensures that the superior is not already supervising the resource.)

Changing connections also requires a superior and inferior level, and it is not possible to change a connection from level A to level B if there is no connection there in the first place. For example, you cannot change a manager-task connection if no managers are supervising tasks. First it picks a person on level A and resource on level B on which a

connection exists. Then, it picks, either a second person on level A, or a second resource on level B, depending on whether it has been instructed to change the connection on the superior side or the inferior side, and makes the change.

Compared to the above, deleting connections is quite simple. A member on the superior level is found, who contains a resource on the inferior level chosen, and he loses that resource. Easy, isn't it?

OrgSim and OrgAhead differ in how they decide what kind of change to make. OrgSim assigns for each change, a threshold. As long as the relative efficiency does not change by more than that change's threshold, the org will not undergo that change. Also, if a change is successful, the org cannot be changed for a period of time depending on the type of change.

OrgAhead ignores thresholds. It uses simulated annealing to determine what kind of change would be profitable; if a particular change improves the org, or at least, does not worsen the org significantly, that change is implemented, and the new org continues simulating, otherwise the old org continues until the next opportunity for change.

How to Handle Murphies

In organizational jargon, a 'murphy' is an event that embodies Murphy's Law; i.e. it is some catastrophe that afflicts orgs. A murphy can be represented as a person being lost, due to the person leaving the org, or disappearing. Or it can be represented as a line of communication that breaks, where one person cannot contact another, although both people are functioning perfectly fine in all other aspects.

OrgSim and the other programs provide flexible enough parameters to handle murphies, although they aren't implicitly aware of what a 'murphy' is. Here is how you would specify murphies to OrgSim:

A murphy that destroys a person can be simulated in OrgSim by firing a random number of people at a random level at a random time. Unlike normal firing, the org doesn't plan the action, so it does nothing (immediately) to reallocate resources. So the victim's resources are not immediately allocated to other colleagues. Of course, OrgSim cannot destroy the last person in an org. Murphies should be used with hiring enabled, which simulates an org losing people at random intervals and having to patch itself up by hiring new people.

There is another type of murphy; it destroys a communication line between two people, leaving them otherwise intact. This is identical to deleting a random connection between two random people at two levels. This can be done with adding and changing connections enabled, which simulates an org trying to compensate for losing communication lines at random intervals.

Task Problem Specification

The task problem determines how to determine the actual solution from the task bits. Generally, you can use a decomposable problem (where each task bit has the same weight as any other), or a nondecomposable one. You can also determine that the task should be biased (lean heavily towards an answer of 3) or unbiased (equal probability on all answers). These can be specified by the `-nondecomposable` and `-biased` switches.

First a total is determined from the task bits, and then it is compared against two cutoffs. If the total is less than the first cutoff, the answer is 1, if it is less than the second cutoff the answer is 2, otherwise the answer is 3. The `-nondecomposable` flag affects the cutoff points, and so does the `-biased` flag. Additionally, you can tweak the `-cutoff_friendly` and `-cutoff_hostile` flags to introduce whatever level of bias you please.

If a decomposable problem is used, the total is the product of the task bits. Otherwise, the following formula is used:

```
Total = 2*t1*t2*t3 + 2*t4*t5 + t6 + t7 + 2*t7*t8*t9
```

If the cutoffs are not specified, the following cutoff values will be used:

Friendly	Biased	Unbiased	Hostile	Biased	Unbiased
Decomposable	71	109	Decomposable	287	432
Nondecomposable	28	34	Nondecomposable	42	49

Appendix B – Meta-Matrix Input File for OrgAhead: Structure File

Each of the dependency matrices (i.e. person-person, person-task, task-resources, etc.) may not be read in via a file containing the original matrices, for both V1 and V2 modes of ORGAHEAD.

The input file is specified by the parameter `-sfn` or `-structure_file_name`:

```
c:\> OrgAhead –sfn matrices.txt
```

Output matrices are printed with the output according to the print flag `-pm` or `-print_matrices`:

```
c:\> OrgAhead -pm
```

The structure of the file, still under development and improvement, is as follows:
Line 1 contains 4 numbers separated by space or tab:
Number of Hierarchy Levels
Maximum People Per Level
Number of Tasks
Number of Resources (i.e. Task_Complexity or -tc)

For example:
`3 15 5 9` - gives the default maximum hierarchy of 3, default max people per level of 5, 5 tasks (i.e. 5 task sets or 5 task definitions), and 9 resources, or Task Complexity default.
Note: The current version of ORGAHEAD is V2.0.3, which allows only 3 levels. The V2.1 will allow for more than 3 levels.

Line 2 contains the number of people per level in the matrices contained in the file:
For example:
`3 4 8` – says 3 analysts, 4 managers, and 8 ceos are contained in the matrices. The people matrices must contain 3+4+8=15 rows or columns (depending on which axes is P)

After line 2, the matrices are specified by a token, on line by itself:
The two capital letter token describes the rows by columns:

```
TR = Task-by-Resource          PP = Person-by-Person
PT = Person-by-Task            PR = Person-by-Resource
TT = Task-by-Task (precedence)
```

So `TR` means that each row is a task and each column is a resource and has to be of the correct size. On the line immediately following the token, the matrix is presented.

So, continuing with the example, the TR matrix would be 5 x 9:

```
TR
100010010
101010100
101010101
101010100
000010010
```

In the future, a random spot will be denoted with an 'x' in place of a '1' or '0'. That means, when the org is generated, the 'x' will take a '1' or '0' randomly determined. This is not functioning yet. Also, the TT matrix is not used yet.

Matrices can have only white-space, letter or tab, or nothing in between each matrix entry; that is, as with TR example, you don't need to delimit the matrix entries with a space.

Additional task set information still needs to be entered via the `-ts` parameter; for example, you cannot yet add the cutoffs through the structure file ... yet.

Finally, any line beginning with the pound symbol, '#', will be treated as a comment line.

For the PP matrix, the links are supervisor ties. So the upper diagonal can be just zeroes. A "1" in row 5, column 2 means that the 5th person supervises the 2nd. Currently, lateral ties are ignored and should not be included as the results might be unpredictable.

Appendix C – Version 2 Details

INTRO:
In version 2 of OrgAhead, agent pointers to the task level are now treated as pointers to a resources level; hence AxR can change via annealing.

PARAMETERS:
```
-v2 0,1 : 1 means engage version 2 of ORGAHEAD, default: 0
```

Each task is defined on the command-line using `-ts #`, which stands for `-task_set`. A `-ts #` parameter can have the following old parameters apply to that task:
```
-pcf default: 109
-pch default: 432
-tf default: .33
-tn default: .34
-th default: .33
-rc "ones_and_zeros" ==> resources (explained below)
```

For instance:
"`-ts 0 -pcf 81 -pch 81 -tf .5 -tn .0 -th .5 `" will make task #0 (the first task) a binary majority task. If that is followed by "`-ts 1 <other parms>`", then a second task (#1) is created.

The number after `-ts` will force the creation of tasks below that number. So if you only have "`-ts 3 <parms>`", the ORGAHEAD will create tasks 0-2 as well with default values.

TASK X RESOURCES:
`-rc` following `-ts` will define the resources for that task specified by `-ts`. Example: `-ts 3 -rc "101010101"` means every other bit is considered in the calculation of the answer for fourth task (#3). If the size of the `-rc` string is null or less than Task_Complexity (which is now the resource pool), then the rest is filled with 1.

PERSON X TASK ASSIGNMENTS;
`-ta` following `-ts` will define the task assignments for that task to sets of strings defining each level. Example: `-ts 2 -ta "111 010 001"` means that Task 2 will be assigned to the first three analysts, the second manager, and the third CEOS. The firs and third managers and the first two CEOS are forcibly not assigned the task.

For random orgs, these are applied only if people appear in these positions. Otherwise, the assignment is a random, Bernoulli draw.

So if a there exists a fourth analyst, his or her assignment to task #2 is random.

A value 'n' greater than 1 will create an assignment for 'n' individuals on that level; this is just shorthand.

For instance: `-ta "3 3 3"` is the same as `-ta "111 111 111"`. These numbers can be 9 at max and can be used in cumulatively: `-ta "999 999 999"` implies the first 27 individuals of each level has the task defined by the previous `-ts`.

PERSON X RESOURCES ASSIGNMENTS:
Remember, the old task vector is now treated as the resource vector, so these linkages are defined using the old scheme (e.g. -analyst "0a0b0c0d").

MAX_TASKS is currently set to 10
MAX_PERSONS per level is currently set to 40

LIMITATIONS:
Tasks are worked on consecutively each round.
Neither the Task X Resources nor the Task x Person matrices change at this point.

CALCULATING DIFFERENT TASKS USING A COMMON MEMORY STRUCTURE:
If task #0 requires resources 110 and task #1 requires 011 and the agent sees all three resources (111), the agent's memory table has overlapping information. So let's say the agent sees 13 for task #0. The feedback process will record the feedback for 131, 132, and 133; the third position is like a "don't care". If then for task #1 the agent needs to answer for 23, he will sum up his responses from 123, 223, and 333 and pick the max from those.

If the agent doesn't have a required resource, it is treated as a "don't care". That is, the feedback is spread across all values of that resource and the answer takes into account all those values.

If an agent has no task assigned to him or her, s/he will use the entire memory table to formulate answers, restricted by the # of resources s/he sees.

SPEED ISSUES:
Having to process these "dont cares" slows down the program. Sparse TxR and AxT will cause "dont cares" to be used and the sparser these are (in conjunction with more people and bigger tasks) will significantly slow down the program.

PERFORMANCE MEASURES:
Only a single performance value is given, rather than separate ones for each task. Performance is the same as before, but this time it incorporates all of the tasks (i.e. divide by the number of tasks). The SD result, for now, is just an average SD over the tasks.

PRINTOUTS:
Preliminary assignments are shown to stdout.
The Task Structure is shown following the Organizational Structure when -v2 is 1.

Appendix D – New Performance Measures

Several new performance measures have been introduced into OrgAhead. These are *completion*, *time* (or duration), *certainty*, and *consensus*. These may be engaged with switch parameters:

```
usage:
OrgAhead –v2 -print_completion -print_time -print_certainty –print_consensus -
po
or:
OrgAhead –v2 –pcomp –ptime -pcert –pcons -po

-pcomp and –ptime require –po (or –print_organization)
```

-print_completion:

The *completion* measure refers to the degree to which agents, and the org, have the resources required for their assigned tasks. The degree of completion for agent *i* is the sum, over each task to which s/he is assigned (AT_{ij}), the resources that a) the task requires (TR_{ik}) and b) the agent possesses (AR_{ik}). Finally, take the final sum and divide by the sum of the number of resources required by each the task, such that a given resource may be counted more than once, if needed by more than one task.

AT is the agent-task assignment matrix, *TR* is the task-resource requirement, and *AR* is the agent-resource acquisition matrix. There are n_T tasks, n_R resources, and n_A agents. $COMP_i$ is an *completion* measure for agent *i* across all required resources and tasks for the agent.[4]

$$(1) \quad COMP_i = \frac{\sum_{j=0}^{nT}(AT_{ij} \times \sum_{k=0}^{nR}(TR_{ik} \times AR_{ik}))}{\sum_{j=0}^{nT}(AT_{ij} \times \sum_{k=0}^{nR} TR_{jk})}$$

$$(2) \quad COMP_{overall} = \frac{\sum_{i=0}^{nA} COMP_{i\{num\}}}{\sum_{i=0}^{nA} COMP_{i\{den\}}}$$

$$(3) \quad COMP_{average} = \frac{\sum_{i=0}^{nA} COMP_i}{n_A}$$

The *overall* completion measure varies slightly from the *average*, which is simply the sum of completions over all agents and dividing by the number of agents. The *overall* measure refers to the completion status for the entire organization, not making as clear a distinction between the agents as the *average* measure does. The *overall* measure sums the requirements met across all agents, and then divides by the sum of requirements. The $COMP_{i\{num\}}$ refers to just the numerator portion of the $COMP_i$ equation and, similarly, the $COMP_{i\{den\}}$ refers to the denominator. A high completion ratio indicates that the requirements for each task are being met and, thus, the agent will learn each task properly. A low completion ratio will result in the agent basing his actions on too few

[4] The measure(s) are not captured at the person per task level; though, we can add that in if needed.

resources and inputs or those that are not relevant to his or her tasks, adversely affecting his performance.

To address the latter issue, having too many non-relevant tasks, we provide an *overflow* measure along with *completion*:

$$(3)\quad OVERFLOW_i = \frac{\sum_{j=0}^{nT}(AT_{ij} \times \sum_{k=0}^{nR}((1 - TR_{ik}) \times AR_{ik}))}{\sum_{j=0}^{nT}(AT_{ij} \times \sum_{k=0}^{nR} TR_{jk})}$$

$$(4)\quad OVERFLOW_{overall} = \frac{\sum_{i=0}^{nA} OVERFLOW_{i\{num\}}}{\sum_{i=0}^{nA} OVERFLOW_{i\{den\}}}$$

$$(5)\quad OVERFLOW_{average} = \frac{\sum_{i=0}^{nA} OVERFLOW_i}{n_A}$$

The calculation for *overflow* is virtually identical to *completion*, except that we tabulate the resources the task **does not** require. The ratio per agent or *overall* or *average* can easily be greater than one, which indicates an inefficient assignment of agents to tasks or agents to resources.[5]

Additional parameters:

```
-pcompp : -print_completion_personal (only)
-pcompo : -print_completion_overall (only)
-pcompt : -print_completion_task (only)
```

`-print_time:`

The *time* measure estimates about how long, in a real-time organization, it would take for an agent to receive and process his inputs, which are resources and subordinates. L_i or L_k refers to the level of agent *i* or *k* in the hierarchy, with 1 being the level of analyst and 0 the task input level. *AA* is the agent-to-agent reporting network (e.g. AA_{ij} means agent *j* reports to agent *i*). The calculation basically sums, over each task, the difference in levels between agent *i* and his subordinates who also work on his or her task and also sums the resources, required by the task, and the distance to those, which just depends on agent *i*'s level in the hierarchy. This summation is divided over the number of tasks the agent has giving an average *time* measure over all of an agent *i*'s tasks. $TIME_i$ is an average *time* measure for agent *i* over all the tasks to which s/he is assigned.[6]

$$(6)\quad TIME_i = \frac{\sum_{j}^{nT} AT_{ij} \times (\sum_{k=0}^{nA} AA_{ik} \times AT_{kj} \times |L_i - L_k| + \sum_{k=0}^{nR} L_i \times TR_{jk} \times AR_{ik})}{\sum_{j=0}^{nT} AT_{ij}}$$

$$(7)\quad TIME_{overall} = \frac{\sum_{i}^{nA} TIME_{i\{num\}}}{\sum_{i=0}^{nA} TIME_{i\{den\}}}$$

[5] A future addition might be a single inefficiency index that combines both *completion* and *overflow* measures.
[6] Again, it might useful to have the measure also at the per task level.

(8) $$TIME_{average} = \frac{\sum_{i}^{n_A} TIME_i}{n_A}$$

Additional Parameters:

`-tiw, -time_ineff_weight:`

There is a secondary parameter, which weights those resources or subordinates an agent has which are not relevant to a task. By default, it is set to 0.0 and does not appear in equation (6). If you use, for example, `-tiw 1.0`, resources and subordinates not relevant to the task *j* will also become added into the equation, increasing the overall *time* measures.[7]

`-print_certainty:`

Certainty measures the degree to which agent *i's* decision choice was clearly the correct answer. If his memory shows that other decisions were almost as likely to be chosen, then the agent is uncertain about the choice. If the decision is a clear winner, then *certainty* will be high.

Certainty is measured over a single relative efficiency cycle.[8] To explain briefly, if the efficiency cycle is 100, then after every 100 tasks worked on, an efficiency, or accuracy, report for the last 100 tasks is produced.

To briefly review, an agent has a set of inputs, resources or subordinates. In a cycle, an agent makes a decision, based on the past answers to these inputs has produced the correct result. The decisions are 1, 2, or 3, which are named friendly, neutral, and hostile, respectively. The agent refers to its memory and asks which decisions is the most likely correct given what the real correct answer was for the same inputs seen before. The number of times in the agent *j*'s memory that decision 1 was the correct answer for the inputs at time *t* is $tally_{jt1}$; the number of times 2 was correct answer is $tally_{jt2}$, and so forth.

The ratio of the maximum of these tallies to the sum of all of them gives us the *certainty* in the agent's answer. These ratios are summed for agent relevant tasks over the efficiency cycle and appropriately averaged. *effcycles* is set by the `-ec` or `-efficiency_cycle` parameter and is the window during which each set of efficiency, as well as *certainty*, measures are obtained. *Certainty* is reset at the beginning of each efficiency cycle. $CERT_i$ is an average certainty measure for agent *i* per task inputs seen.

(9) $$CERT_i = \frac{\sum_{t=0}^{effcycles} \sum_{j=0}^{n_T} AT_{ij} \times \frac{\max\{tally_{jt1..3}\}}{tally_{jt1} + tally_{jt2} + tally_{jt3}}}{effcycles \times AT_{ij}}$$

[7] The issue of non-relevant resources interfering with task decisions is still an open issue.
[8] However, this can also be changed such that the certainty cycle is independent of the efficiency cycle.

$$(10)\quad CERT_{overall} = \frac{\sum_{i=0}^{n_A} CERT_{i\{num\}}}{\sum_{i=0}^{n_A} CERT_{i\{den\}}}$$

$$(11)\quad CERT_{average} = \frac{\sum_{i=0}^{n_A} CERT_{i\{num\}}}{n_A}$$

`-print_consensus:`

The *consensus* measure provides the degree to which all of the agents' answers matched the organization's final answer for a task *i*.[9] $decision_{jti}$ refers to the answer agent *j* gave for task *i* at time *t* in the current efficiency cycle. $decision_{ORGti}$ refers to the organization's decision for task *i* at time *t*. *Consensus* is reset at the beginning of each efficiency cycle.[10]

$\vartheta\{decision_{jti} = decision_{ORGti}\}$ is an indicator function which yields 1 when the condition is met and 0 if not. $CONS_i$ is the consensus on a task *i*, only if it was performed. That is, the `-task_order_file`, or `-tof`, can specify the orders of tasks for each cycle and can prevent the org from working on some tasks. If the org does not work on a task, it is not counted in the consensus measures. $\vartheta\{working_i\}$ is an indicator function that yields 1 if the task *i* is being worked on in the current cycle. It is possible for a cycle to not include a task depending on the task orderings.

$$(12)\quad CONS_i = \frac{\sum_{t=0}^{effcycles} \sum_{j=0}^{n_A} \vartheta\{decision_{jti} = decision_{ORGti}\}}{effcycles \times n_A}$$

$$(13)\quad CONS_{overall} = \frac{\sum_{i=0}^{n_T} CONS_{i\{num\}} \times \vartheta\{working_i\}}{effcycles \times n_A \times \sum_{i=0}^{n_T} \vartheta\{working_i\}}$$

$$(14)\quad CONS_{average} = \frac{\sum_{i=0}^{n_T} CONS_i}{n_T}$$

[9] Currently the overall consensus equals the average consensus.
[10] We can also make the consensus cycle different from the efficiency cycle if need be.

Lifetime Measures

At the end of a lifecycle, or a single simulation run, of ORGAHEAD, a lifetime set of summary measures of the above will be produced. *MEASURE* is one of the aforementioned measures: *COMP*, *OVERFLOW*, *CERT*, or *CONS*. *samples* is the number of times a measure is taken. Remember, measures are taken at either at the output of an organization (for `-pcomp` and `-ptime`) or at the end of an efficiency cycle (for `-pcert` and `-pcons`). *s* denotes a particular sampling; samplings occur at regular intervals. $n_A\_or\_n_T$ refers to the limit of the summation depending on the measure.

$$(15)\quad MEASURE_{lifetime\_overall} = \frac{\sum_{s=0}^{samples}\sum_{i=0}^{nAornT} MEASURE_{i\{num\}s}}{\sum_{s=0}^{samples}\sum_{i=0}^{nAornT} MEASURE_{i\{den\}s}}$$

$$(16)\quad MEASURE_{lifetime\_average} = \frac{\sum_{s=0}^{samples}\sum_{i=0}^{nAornT} MEASURE_{is}}{samples \times n_A\_or\_n_T}$$

$$(17)\quad MEASURE_{overall\_average} = \frac{\sum_{s=0}^{samples} MEASURE_{overall,s}}{samples}$$

$$(18)\quad MEASURE_{average\_average} = \frac{\sum_{s=0}^{samples} MEASURE_{average,s}}{samples}$$

lifetime_overall sums the numerator and denominator of a measure independently and produces a final ratio; as if, we do not differentiate the activities of the agents or when the activities occur.

lifetime_average calculates each average behavior of the agent, or task, across their population and time; as if we do not differentiate the behavior across time or per organization structure.

overall_average and *average_average* are merely the averages of the overall and average measures.

Appendix E – An Exhaustive List of OrgAhead Parameters

OrgAhead has a simple lookahead feature. When its time to change an org, instead of seeing if relative efficiency has exceeded some threshold, a lookahead process is used. An org. will contemplate itself undergoing some change and the resulting efficiency increase/decrease incurred, and if considered profitable, the org. will itself undergo that change. This program uses simulated annealing to determine when a change is profitable.

Almost every aspect of the simulator is specifiable as a command-line argument. The current and proposed arguments (vis model 1) are described below. The arguments correspond to creating new triggers for studying organizational adaptation and creating alternative performance functions. Items in boxes are interdependent (i.e. if you change one significantly, you have to consider changing some or all of the others). Additional comments and pointers appear in italics.

-help, ?, usage_help, full_help: Display Command Information
Display information about this command, which includes a command description with examples, plus a synopsis of the command line parameters. If you specify -full_help rather than -help complete parameter help is displayed if it's available.

-simulation_times, st: integer = 1
Specifies how many times to simulate the org. If more than 1 is specified, prints out mean and standard deviation for several efficiency statistics.

Annealing Parameters:

-cooling_ratio, cr: real = 0.9
The ratio by which the annealer's temperature drops when cooling.
-changes_between_coolings, cbc: integer = 1
How many changes the organization can undergo between temperature coolings.
-task_limit, tl: integer = 50000
If the organization does this many tasks, the program quits. If set to 0, the program will not quit no matter how many tasks are done. This is dangerous because (depending on -freezing_partition) a simulation can continue forever.

Since one cooling occurs per change, the number of coolings will depend on the change cycle (i.e. how often changes occur) and the total task limit. The –cr .9 gives a full cooling for –tl 50000 but lower coolings would be required for shorter task limits, which is typical in authors' experiments (i.e. we normally use a faster cooling depending on the task limit)

-medeiro_efficiency_threshold, met: real
If efficiency ever drops by more than this much, then increase temperature. (Actually, this should only happen as a result of an environmental change that causes the organization to perform poorly.)
-medeiro_efficiency_ratio, mer: real = 2.0
If the -medeiro_efficiency_threshold is exceeded, multiply temperature by this ratio. If zero, temperature is instead set to initial temperature.
-medeiro_cycle, mcy: integer
If specified, indicates that temperature should be raised periodically. Indicates how often temperature should be raised.
-medeiro_cycle_ratio, mcr: real = 2.0
If the -medeiro_cycle number is specified, indicates the ratio by which to increase temperature. If zero, temperature is instead set to initial temperature.

The medeiro parameters are employed when we desire multiple coolings within a single run of OrgAhead, reflecting empirical conditions in which risk-taking abruptly occurs at levels equal to or near initial conditions.

Annealing Cost Parameters:

-initial_partition, ip: real = 0.9
On a scale of 0 to 1, how much of the range of uphill costs should be accepted when the simulation starts.
-freezing_partition, fp: real = 0.1
On a scale of 0 to 1, how much of the range of uphill costs should be accepted when the simulation ends. This is used as a means for ending the simulation. As the annealer's temperature cools, the actual range of costs accepted slips lower and lower, approaching 0 asymptotically. When the actual range slips below this value, the simulation ends. Setting this value to 0 causes simulation to continue until -task_limit is reached.

A –fp 0.1 can potentially result in premature freezing (i.e. simulation ends before the end of task limit). To avoid this, we recommend setting –fp 1e-700 (i.e. 10^{-700}, a number very, very close to zero).

-theoretical_delta_efficiency, tde: real = 0.50
The maximum change in efficiency allowable, theoretically. (Note this should range between 0 and 1, not 0 and 100 (percent) like most other efficiency parameters.)
-theoretical_delta_resources, tdr: real = 7.0
The maximum change in resources allowable, theoretically.
-theoretical_delta_analysts, tda: real = 1.0
The maximum change in analysts allowable, theoretically.
-theoretical_delta_managers, tdm: real = 1.0
The maximum change in managers allowable, theoretically.
-theoretical_delta_CEOs, tdc: real = 1.0
The maximum change in CEOs allowable, theoretically.
-stodginess_factor, sf: real = 0.0
This factor is added to the cost of any hypothetical organization when comparing it to the real org. So a positive factor will make orgs more resistant to being changed.
-weight_efficiency, we: real = 100.0
How strongly efficiency (or rather, inefficiency, which is -1 * efficiency) should weigh in determining the cost of an org.
-weight_resources, wr: real = 0.0
How strongly the number of resources (totaled over everyone) should weigh in determining the cost of an org.
-weight_analysts, wa: real = 0.0
How strongly the number of analysts should weigh in determining the cost of an org.
-weight_managers, wm: real = 0.0
How strongly the number of managers should weigh in determining the cost of an org.
-weight_CEOs, wc: real = 0.0
How strongly the number of CEOs should weigh in determining the cost of an org.
-weight_people, wp: real = 0.0
-weight_inverse_efficiency, wie: real = 1.0
How strongly the reciprocal of the efficiency weighs in determining cost
-weight_efficiency_people, wep: real = 0.0
How strongly the ratio between inefficiency and people weighs
-weight_people_efficiency, wpe: real = 0.0
How strongly the ratio between people and efficiency weighs
-weight_efficiency_resources, wer: real = 0.0
How strongly the ratio between inefficiency and resources weighs
-weight_resources_efficiency, wre: real = 0.0
How strongly the ratio between resources and efficiency weighs

Typically, we run our experiments such that efficiency (i.e. performance of organization at each efficiency window) determines the risk-taking cost. However, users may want alternative definitions to risky costs, such as massive organizational growth or downsizing.

Annealing Move Parameters:

-hustin_window, huw: integer = 10
How many temperatures to base Hustin probabilities on.
-hustin_probability, hup: real = 0.1
The minimum probability a move can achieve from Hustin. Also, the maximum value, which is computed as 1 - hustin_probability.
-enable_augmenting , eh: switch
If set, then the organization is allowed to augment people.
-enable_move-to-other-problem , ef: switch
If set, then the organization is allowed to move-to-other-problem people.
-enable_add_person_person, eapp: switch
If set, then the organization is allowed to add connections between people.
-enable_change_person_person, ecpp: switch
If set, then the organization is allowed to change connections between people.
-enable_delete_person_person, edpp: switch
If set, then the organization is allowed to delete connections between people.
-enable_add_person_task, eapt: switch
If set, then the organization is allowed to add connections between people and tasks.
-enable_change_person_task, ecpt: switch
If set, then the organization is allowed to change connections between people and tasks.
-enable_delete_person_task, edpt: switch
If set, then the organization is allowed to delete connections between people and tasks.

By default, all the move classes are enabled. To only allow personnel changing, do: -ef -eh.
NOTE: you generally want to use the default which means all change are enabled.
To allow only connection changing, do: -ecpp -ecpt
To do both personnel and connection changing, do: -ef -eh -ecpp -ecpt
To do both, except disallowing fires, do: -eh -ecpp -ecpt

Duration Parameters:
NOTE: these set the windows that are used to determnine when efficiency/accuracy is measured, when it is measured, the level of training, forgetting, they also affect when "churn" such as turnover, reassignment can occur. So the hypothetical is for the model of the organization is trying to do lookahead on.

-hypothetical_efficiency_period, hep: integer = 500
How many tasks a hypothetical organization performs after training. The hypothetical org's efficiency is computed from these tasks.
-hypothetical_training_period, htp: integer = 500
How many tasks a hypothetical organization gets to train on.

So the hypothetical organization = the old organization + a change, this is how many times it runs with the proposed change before the manager starts thinking about performance
If you knew whether the manager made errors in how they thought about the future you would change these, if you do not use the defaults of 500.

-efficiency_cycle, ec: integer = 500
Periodically this program performs an efficiency check. It prints and resets efficiency statistics on the org. This parameter specifies the size of the period.

Periodic performance measures occurs every efficiency cycle. An –ec 500 means perform (and report) a efficiency check every time the org works on 500 tasks.

-memory_cycle, mc: integer = 100
Periodically this program resets everyone's memory, in order to compute everyone's relative efficiency. Specifies the size of this period.

This is how much the individual can retain and is a generally a good default

-change_cycle, cc: integer = 500
Periodically this program attempts to change the org's structure. Specifies the size of this period.

This is how often is churn going on – this is dynamism, this is turnover. This has to be set relative to the efficiency cycle. So if efficiency is 100 and you have so much churn that multiple changes occur even during on performance cycle then set this to say 50, if it is a low churn organization with little change going on you might set it to 200

-training_period, tp: integer = 500
Specifies how many tasks to 'train' the organization on. Until the organization has done this many tasks, no absolute efficiency values are reported, and no changes are possible. Also, no new people added on later may be fired if they have less than this much experience.

If highly trained you might use 500 if poorly trained you might use 100 or 200.

-rand_seed, rs: integer
Specifies a seed number for the random number generator. If unspecified, the random number generator will be seeded based on the current time. (This exists mainly for debugging purposes. if you give OrgAhead the same parms, it will still come up with different results, unless you include -rand_seed x, where x is a constant integer.) Since the training period signifies how much initial experience each person has, and the memory cycle signifies 1X to half of how much current experience each person has, you will want the memory cycle to be about one half the training period.

Output Parameters:

These parameters determine what information the simulator prints. All output goes to stdout. If no parameters are specified then only the final organization structures and efficiency statistics are printed, once for each simulation. If more than one simulation occurs, some overall efficiency statistics are also printed.

-print_task, pt: switch
If set, the task bits and solution are printed for each task. This can yield heavy output if many tasks are done.
-print_person, pp: switch
If set, each person's resources and experience are printed during each efficiency check. This can yield very heavy output.
-print_organization , po: switch
If set, the organization structure is printed when each simulation begins, ends, and every time a change in the organization occurs. Otherwise, the organization structure is only printed at the end of each simulation.
-print_efficiency, pe: switch
If set, the org's efficiency and efficiency of each person are printed whenever the organization performs an efficiency check.
-print_change, pc: switch
If set, this program prints out each change of the organization as it occurs, including some annealing statistics.

Organization Parameters:

These specify the initial organization structure, as well as constraints on possible future organization structures.
-analysts, a: string = ""

Specifies the resource access structure, that is, the network the analysts use to view the tasks.

-managers, m: string = ""

Specifies the network the managers use to supervise the analysts.

-ceos , c: string = ""

Specifies the network the top managers or CEOs use to supervise the managers. If no analysts, managers, or CEOs are specified (all are given as empty strings), this program will create a random organization for each simulation.

-president, p: switch

Specifies if the organization should be overlooked by a 'president'. He oversees all CEOs as well as any unsupervised analysts or managers, and makes the organization's decisions based on experience.

-SOP, s: switch

If set, indicates that everyone should follow an SOP, that is, when making a decision, they should ignore their experience, and pick the most commonly occurring number in their resource pattern.

Note: if SOP is on training is ignored in terms of its impact on performance – but the simulation will still run that many time periods.

-max_people, mp: integer = 15

Indicates the maximum number of people on a single level.

-max_resources, mr: integer = 7

Indicates the maximum number of resources a person may use.

This is the "cognitive limit" on how much an individual can use. If set to 7 the individual sees 2 to the 7^{th} different patterns. All other time windows are set relative to this.

-drop_resource, dr: key first, last, random, keyend = last

When a person is assigned more resources than they can handle, they must drop one. This parameter indicates which resource to drop.

To start OrgAhead with a voting team, use -a "A B C D E F G H I"
For managed team, use -a "A B C D E F G H I" -m "ABC DEF GHI"
For hierarchy, use -a "A B C D E F G H I" -m "ABC DEF GHI" -c "ABC"
For random start, use -a "" -m "" -c ""

Task Parameters:

These specify what kind of tasks the organization must solve, and how each solution should be generated.

-task, t: *key primary, switch, toggle, glide, keyend = primary*

This program has two sets of task parameters, primary and secondary. This parameter indicates how to use them. If set to 'primary', only the primary task parameters are used, if set to 'switch', the task parameters switch from primary to secondary at some point in each simulation. If set to 'toggle', the task periodically toggles between primary and secondary task parameters. If set to 'glide', the task gradually glides from primary to secondary parameters through each simulation.

-task_complexity, tc: integer = 9

How many task resources are used.

Task complexity is one of the more crucial parameters to OrgAhead as it defines the size of the problem space. A binary task (i.e. –tf .5 –tn .0 –th .5) will yield a problem of size two to the power of task complexity. For example, if –tc 9 and binary task, then the possible combinations of tasks that the organization sees is $2^9 = 256$. Note that default –tf, -tn, -th produce a trinary task with problem space size 3^9. Consider the implications on other parameters such memory (i.e. –mc). If an agent sees all 9 task bits and has a default memory of 100, then the agent will remember less than half of the possible combinations and remember, that agents need to see multiple instances of combinations in order to learn effectively. While, typically, agents will not see all 9 task bits, the –tc parameter does set an upper bound. The more tasks bits and/or subordinate an agent has, the less effect his or her experience and memory will be. On the otherhand, seeing more information allows an agent to be more accurate about the "real" answer to the task.

-nondecomposable, n: switch
Determines the formula used to compute the correct answer from the task bits. If not set, the task bits are multiplied, and the total is compared to the cutoffs. If specified, the following non-linear formula is used:

Total = 2*t1*t2*t3 + 2*t4*t5 + t6 + t7 + 2*t7*t8*t9

And the total is compared to the cutoffs to yield a solution. This flag may not be set if -task_complexity is a value other than 9.

-primary_biased, pb: switch
If set, the primary task will be biased; if not set, the primary task will be unbiased. Unbiased means that an answer of 1 (friendly) is about as likely as an answer of 3 (hostile), biased makes the answer of 3 more likely than 1.

-primary_cutoff_friendly, pcf: integer
How small the sum of the task bits must be to yield a friendly answer (of 1).

-primary_cutoff_hostile, pch: integer
How large the sum of the task bits must be to yield a hostile answer (of 3). If these cutoffs are specified, they override the -biased switch, otherwise, default cutoffs are used depending on if the task is biased or decomposable. These cutoffs must be specified if the task complexity is a value other than 9. The default cutoffs are as follows:

In the binary task setup, -pcf and –pch will be equal; that is, there is no middle partition of the solution space.

The following table displays the default cutoff settings for a trinary task:

Friendly	**Biased**	**Unbiased**	**Hostile**	**Biased**	**Unbiased**
Decomposable	71	109	**Decomposable**	287	432
Nondecomposable	28	34	**Nondecomposable**	42	49

-primary_duration_mean, pdm: real = 5000
This declares for how many tasks the primary task parameters should be used. After this many tasks, if the task parameter is 'switch' or 'toggle', the secondary task parameters are used. If the task parameter is 'glide', this specifies how long before the secondary task. Parameters have totally subsumed the primary task parameters. If the -task parameter is 'primary', this parameter is ignored.

Primary and secondary (below) durations allow the user to specify different solution criteria for a subset of the tasks, or organizational life cycle. Additional parameters allow the user to specify when and how often each criterion (primary or secondary) will take effect (e.g. toggle from one to another, switch back and forth, etc.)

-primary_duration_variance, pdv: real = 0
Specifies the variance for how many tasks the primary task parameters should be used. The duration is chosen along a Normal distribution using this mean and variance. If set to 0, the mean is used as a constant.
Leave this at 0 unless you want waves to occur in the task.

-secondary_biased, sb: switch
Works like primary_biased, except applies to secondary task.
e.g, if you know that the secondary task is biased more or less that the first then you set this to illustrate that – so for example, if in December you switch to most flu patients then you might go from an unbiased to a biased task - bias sets the proportion of cases that have a particular answer.

-secondary_cutoff_friendly, scf: integer
Works like secondary_cutoff_friendly, except applies to secondary task

-secondary_cutoff_hostile, sch: integer
Works like secondary_cutoff_hostile, except applies to secondary task

Only set these if the secondary task is trinary and if you want it to be biased

-secondary_duration_mean, sdm: real = 5000
-secondary_duration_variance, sdv: real = 0
Specifies the variance for how many tasks the secondary task parameters should be used. The duration is chosen along a Normal distribution using this mean and variance. If set to 0, the mean is used as a constant.
This sets when it switches back to the primary task

-task_friendly, tf: real = 0.33
Indicates the probability that a task bit will be 1.
-task_neutral, tn: real = 0.34
Indicates the probability that a task bit will be 2.
-task_hostile, th: real = 0.33
Indicates the probability that a ta*sk bit will be 3. These three numbers must total 1.*

Specifying all three probabilities implies a trinary task bit. A binary task is typically set up using –tf .5 –tn .0 -.th .5, but not necessarily; the user need only specify two bits at 50% and the third at 0% so –tf .5 –tn .5 –th .0 is also a binary task and will yield identical behavior assuming the cutoffs are adjusted accordingly.

Although this program defaults to trinary unbiased tasks, you can configure it to binary unbiased tasks, by eliminating the posibility that a '2' will be selected. Use the following parameters:
-tf 0.5 -tn 0.0 -th 0.5 -pcf 100 -pch 100

For a biased binary task (only 4 3's guarantee a 3 answer), do:
-tf 0.5 -tn 0.0 -th 0.5 -pcf 80 -pch 80
and for a heavily biased binary task (only 3 3's guarantee a 3 answer):
-tf 0.5 -tn 0.0 -th 0.5 -pcf 25 -pch 25

Hiring Parameters:

These parameters apply to all aspects of augmenting or adding new people to the org.
-hiring_dormancy, hd: integer = 0

Using the dormancy thesis in time units, if greater than the change cycle it stalls hiring if less than it hires at most once every change cycle in general use the default of 0.

After augmenting someone, the organization may not change itself for this many tasks. If a value less than change_cycle is given, then -change_cycle is used.

-hiring_analyst_probability, hap: real = 0.33
Specifies the probability that, when the organization augments new people, they will be analysts.
-hiring_manager_probability, hmp: real = 0.34
Specifies the probability that, when the organization augments new people, they will be managers.
-hiring_ceo_probability, hcp: real = 0.33
Specifies the probability that, when the organization augments new people, they will be CEOs.
These last three parameters must total 1.
-first_person_gets_random, fpgr: switch
When someone is augmented on an empty level, he usually gets all the resources from a particular level or colleague (depending on the circumstances). If this switch is set, he gets one random resource from same level or person instead of all the resources.

If you wish augments to occur on every level with equal probability, you can do: -hap 0.33 -hmp 0.33 -hcp 0.33

Hiring Analyst Parameters:

These parameters only apply to augmenting analysts.

-hiring_analyst_mean, ham: real = 1.0

Specifies how many analysts should be augmented at a time.

-hiring_analyst_distribution, had: switch

If specified, indicates that the number of analysts augmented should be determined from a Poisson distribution using the mean. Otherwise the number of analysts augmented should always be the mean.

-hiring_analyst_resource, har: key none, random, best, worst, busiest, laziest, keyend = random

Specifies what resource should be given to each new analyst. (they can only be given tasks). They can receive no resource, or a random task, or be given a resource by their most efficient, least efficient, busiest or laziest colleague.

-hiring_analyst_keep, hak: switch

Specifies that when a new analyst gets a resource from a colleague, does that colleague keep the resource, or does he lose it? Lose is the default

-hiring_analyst_supervisor, has: key best, worst, random, laziest, busiest, keyend = random

Specifies which supervisor each new analyst should get. Each analyst may be assigned a random supervisor, or one with the highest or lowest efficiency, or the one with the most or least resources (busiest/laziest). Default is random.

Hiring Manager Parameters:

These parameters only apply to augmenting managers.

-hiring_manager_mean, hmm: real = 1.0

Specifies how many managers should be augmentd at a time.

Keep the default of 1 as no reason to think they leave in clumps

-hiring_manager_distribution, hmd: switch

If specified, indicates that the number of managers augmented should be determined from a Poisson distribution using the mean. Otherwise, the number of managers augmented should always be the mean.

-hiring_manager_resource, hmr: key none, random, task, person, best, worst, busiest, laziest, keyend = random

Specifies what resource should be given to each new manager. (they can be given tasks or analysts). They can receive no resource, or a random resource (either person or task), or a random person, or a random task, or be given a resource by their most efficient, least efficient, busiest or laziest colleague.

-hiring_manager_keep, hmk: switch

Specifies that when a new manager gets a resource from a colleague, does that colleague keep the resource, or does he lose it?

-hiring_manager_supervisor, hms: key best, worst, random, laziest, busiest, keyend = random

Specifies which supervisor each new manager should get. Each manager may be assigned a random supervisor, or one with the highest or lowest efficiency, or the one with the most or least resources (busiest/laziest).

Hiring CEO Parameters:

These parameters only apply to augmenting CEOs.

-hiring_ceo_mean, hcm: real = 1.0

Specifies how many CEOs should be augmented at a time.

-hiring_ceo_distribution, hcd: switch

If specified, indicates that the number of CEOs augmented should be determined from a Poisson distribution using the mean. Otherwise the number of CEOs augmented should always be the mean.

-hiring_ceo_resource, hcr: key none, random, task, person, best, worst, busiest, laziest, keyend = random

Specifies what resource should be given to each new ceo. (they can be given tasks, analysts, or managers). They can receive no resource, or a random resource (either person or task), or a random person, or a random task, or be given a resource by their most efficient, least efficient, busiest or laziest colleague.

-hiring_ceo_keep, hck: switch

Specifies that when a new ceo gets a resource from a colleague, does that colleague keep the resource, or does he lose it?

Adding Connection Parameters:
These parameters apply to all types of connection adding between people.

-adding_threshold, at: real = 100.0
Specifies how much the org's relative efficiency should change before adding connections. (Not used in OrgAhead).
Note; this reflects how "reactive" the unit is, very reactive you might set this to 25 not reactive or following sops – set it to 300

-adding_when, aw: key rises, sinks, rises_or_sinks, keyend = rises
Specifies how the org's relative efficiency should change in order to add connections. The organization can do this when its efficiency rises, sinks, or does either, by more than the adding threshold. (Not used in OrgAhead).

-adding_dormancy, ad: integer = 0
After adding connections, the organization may not change itself for this many tasks. If a value less than change cycle is given, then change_cycle is used.
This is set in time units and if you use a value less than the change cycle then it will change the people-to-problem at most every change cycle, if you use value longer than the change cycle it will slow down how often it makes this type of change. Default of 0 works fine which means change at change cycle

The next 6 measures have to add to 1, here is an example of how to set them,

	Degree of family orientation, interaction of all to all		
	Low <= 1 std below mean	Medium	High >= 1 std above mean
Dynamism or change in organization			
Low <= 1 std below mean	Set all of the 6 to the default (apx 1/6th)	Set people to task to ½ people to people aatp=amtp=actp= 1/9th amap=acmp=acap= 2/9th	Set people to task to 3 people to people aatp=amtp=actp= 1/12th amap=acmp=acap= 3/12th
medium	Set people to task to twice the people to people aatp=amtp=actp= 2/9th amap=acmp=acap= 1/9th	Set all of the 6 to the default (apx 1/6th)	Set people to task to ½ people to people aatp=amtp=actp= 1/9th amap=acmp=acap= 2/9th
High >= 1 std above mean	Set people to task to twice the people to people aatp=amtp=actp= 3/12th amap=acmp=acap=1/122th	Set people to task to twice the people to people aatp=amtp=actp= 2/9th amap=acmp=acap= 1/9th	Set all of the 6 to the default (apx 1/6th)

These next 3 are set based on dynamism

-adding_analyst_task_probability, aatp: real = 0.17
Specifies the probability that when the organization adds connections, they will be from analysts to tasks.

-adding_manager_task_probability, amtp: real = 0.17
Specifies the probability that when the organization adds connections, they will be from managers to tasks.

-adding_ceo_task_probability, actp: real = 0.16
Specifies the probability that when the organization adds connections, they will be from CEOs to tasks.

For these 3 – people-to-task - since dynamism is based on a 6 point scale

-adding_manager_analyst_probability, amap: real = 0.17
Specifies the probability that when the organization adds connections, they will be from managers to analysts.
-adding_ceo_analyst_probability, acap: real = 0.16
Secifies the probability that when the organization adds connections, they will be from CEOs to analysts.
-adding_ceo_manager_probability, acmp: real = 0.17
Specifies the probability that when the organization adds connections, they will be from CEOs to managers.

These six parameters should total 1. If you want connections to be added between any two levels with equal probability, you can specify:
-aatp 0.16 -amtp 0.16 -actp 0.16 -amap 0.16 -acap 0.16 -acmp 0.16

Note: if you do not specify one it uses the default value listed above

Adding Connection Parameters: (Analyst-Task)
These parameters are used whenever analyst-task connections are being added.

-adding_analyst_task_mean, aatm: real = 1.0
Specifies how many analyst-task connections should be added at a time.
Note; leave this at 1, as we don't if they are firing in droves
-adding_analyst_task_distribution, aatd: switch
If specified, indicates that the number of analyst-task connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean.
-adding_analyst_task_orphan, aato: real = 0.5
Specifies the probability that the task being connected should be an 'orphan' task, i.e. unsupervised by every analyst.
-adding_analyst_task_superior, aats: key best, worst, random, busiest, laziest, keyend = random
Specifies which analyst should acquire a task. The best, worst, busiest, laziest, or a random analyst can be chosen.
-adding_analyst_task_inferior, aati: key random, keyend = random
Specifies which task the analyst should acquire.

Adding Connection Parameters: (Manager-Task)
These parameters are used whenever manager-task connections are being considered.

-adding_manager_task_mean, amtm: real = 1.0
Specifies how many manager-task connections should be added at a time.
Note; leave this at 1 as we don't if they are firing in droves
-adding_manager_task_distribution, amtd: switch
If specified, indicates that the number of manager-task connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean. *Note: this is on*
-adding_manager_task_orphan, amto: real = 0.5
Specifies the probability that the task being connected should be an 'orphan' task; i.e: unsupervised by every manager. *Note: this is 0 according to SME*
-adding_manager_task_superior, amts: key best, worst, random, busiest, laziest, keyend = random
Specifies which manager should acquire a task. The best, worst, busiest, laziest, or a random manager can be chosen.
-adding_manager_task_inferior, amti: key random, keyend = random
Specifies which task the manager should acquire.

Adding Connection Parameters: (Manager-Analyst)
These parameters are used whenever manager-analyst connections are being considered.

-adding_manager_analyst_mean, amam: real = 1.0
Specifies how many manager-analyst connections should be added at a time.
-adding_manager_analyst_distribution, amad: switch
If specified, indicates that the number of manager-analyst connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean. *Note: this is on*
-adding_manager_analyst_orphan, amao: real = 0.5
Specifies the probability that the analyst being connected should be an 'orphan' analyst, i.e: currently unsupervised by every manager. *Note: this is 0 according to SME*
-adding_manager_analyst_superior, amas: key best, worst, random, busiest, laziest, keyend = random
Specifies which manager should acquire a analyst. The best, worst, busiest, laziest, or a random manager can be chosen.
-adding_manager_analyst_inferior, amai: key best, worst, random, busiest, laziest, keyend = random
Specifies which analyst the manager should acquire. The best, worst, busiest, laziest, or a random analyst can be chosen.

Adding Connection Parameters: (CEO-Task)
These parameters are used whenever CEO-task connections are being considered.

-adding_ceo_task_mean, actm: real = 1.0
Specifies how many CEO-task connections should be added at a time.
-adding_ceo_task_distribution, actd: switch
If specified, indicates that the number of CEO-task connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean. *Note: this is on*
-adding_ceo_task_orphan, acto: real = 0.5
Specifies the probability that the task being connected should be an 'orphan' task, i.e: unsupervised by every ceo. *Note: this is 0 according to SME*
-adding_ceo_task_superior, acts: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire a task. The best, worst, busiest, laziest, or a random ceo can be chosen.
-adding_ceo_task_inferior, acti: key random, keyend = random
Specifies which task the ceo should acquire.

Adding Connection Parameters: (CEO-Analyst)
These parameters are used whenever CEO-analyst connections are being considered.

-adding_ceo_analyst_mean, acam: real = 1.0
Specifies how many CEO-analyst connections should be added at a time.
-adding_ceo_analyst_distribution, acad: switch
If specified, indicates that the number of CEO-analyst connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean. *Note: this is on*
-adding_ceo_analyst_orphan, acao: real = 0.5
Specifies the probability that the analyst being connected should be an 'orphan' analyst, i.e: unsupervised by every ceo. *Note: this is 0 according to SME*
-adding_ceo_analyst_superior, acas: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire a analyst. The best, worst, busiest, laziest, or a random ceo can be chosen.
-adding_ceo_analyst_inferior, acai: key best, worst, random, busiest, laziest, keyend = random
Specifies which analyst the ceo should acquire. The best, worst, busiest, laziest, or a random analyst can be chosen.

Adding Connection Parameters: (CEO-Manager)
These parameters are used whenever CEO-manager connections are being considered.

-adding_ceo_manager_mean, acmm: real = 1.0
Specifies how many CEO-manager connections should be added at a time.
-adding_ceo_manager_distribution, acmd: switch
If specified, indicates that the number of CEO-manager connections to be added should be determined from a Poisson distribution using the mean. Otherwise the number of connections added should always be the mean. *Note: this is on*
-adding_ceo_manager_orphan, acmo: real = 0.5
Specifies the probability that the manager being connected should be an 'orphan' manager, i.e: unsupervised by every ceo. *Note: this is 0 according to SME*
-adding_ceo_manager_superior, acms: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire a manager. The best, worst, busiest, laziest, or a random ceo can be chosen.
-hiring_ceo_manager_inferior, acmi: key best, worst, random, busiest, laziest, keyend = random
Specifies which manager the ceo should acquire. The best, worst, busiest, laziest, or a random manager can be chosen.

Changing Connection Parameters:
The next 6 have to add to 1 and you should set them up in the same way as the add connections
And specify all nine of them
These parameters apply to all types of connection changing between people.

-changing_dormancy, cd: integer = 0
After changing connections, the organization may not change itself for this many tasks. If a value less than change_cycle is given, then change_cycle is used.
-changing_analyst_task_probability, catp: real = 0.17
Specifies the probability that when the organization changes connections, they will be from analysts to tasks.
-changing_manager_task_probability, cmtp: real = 0.17
Specifies the probability that when the organization changes connections, they will be from managers to tasks.
-changing_manager_analyst_probability, cmap: real = 0.17
Specifies the probability that when the organization changes connections, they will be from managers to analysts.
-changing_ceo_task_probability, cctp: real = 0.16
Specifies the probability that when the organization changes connections, they will be from CEOs to tasks.
-changing_ceo_analyst_probability, ccap: real = 0.16
Specifies the probability that when the organization changes connections, they will be from CEOs to analysts.
-changing_ceo_manager_probability, ccmp: real = 0.17
Specifies the probability that when the organization changes connections, they will be from CEOs to managers. These six parameters should total 1.

If you want connections to be changed between any two levels with equal probability, you can specify:
-catp 0.16 -cmtp 0.16 -cctp 0.16 -cmap 0.16 -ccap 0.16 -ccmp 0.16

Changing Connection Parameters: (Analyst-Task)

These parameters are used whenever analyst-task connections are being considered.
-changing_analyst_task_mean, catm: real = 1.0
Specifies how many analyst-task connections should be changed at a time.
-changing_analyst_task_distribution, catd: switch
If specified, indicates that the number of analyst-task connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.
-changing_analyst_task_superior, cats: key best, worst, random, busiest, laziest, keyend = random

Specifies which analyst should acquire a task. The best, worst, busiest, laziest, or a random analyst can be chosen.

-changing_analyst_task_inferior, cati: key random, keyend = random

Specifies which task the analyst should acquire.

-changing_analyst_task_remove, catr: key inferior, superior, keyend = superior

If set to 'superior', the analyst that acquired the task loses some other task. If set to 'inferior', the task acquried by the analyst becomes unsupervised by some other analyst (it 'loses' the analyst.)

-changing_analyst_task_loser, catl: key best, worst, random, busiest, laziest, keyend = random

Depending on the setting of -changing_analyst_task_remove, deletes the connection between the best/worst/laziest/busiest/random analyst and the task just acquired, or a random task and the analyst just acquired.

Changing Connection Parameters: (Manager-Task)

These parameters are used whenever manager-task connections are being considered.

-changing_manager_task_mean, cmtm: real = 1.0

Specifies how many manager-task connections should be changed at a time.

-changing_manager_task_distribution , cmtd: switch

If specified, indicates that the number of manager-task connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.

-changing_manager_task_superior, cmts: key best, worst, random, busiest, laziest, keyend = random

Specifies which manager should acquire a task. The best, worst, busiest, laziest, or a random manager can be chosen.

-changing_manager_task_inferior, cmti: key random, keyend = random

Specifies which task the manager should acquire.

-changing_manager_task_remove, cmtr: key inferior, superior, keyend = superior

If set to 'superior', the manager that acquired the task loses some other task. If set to 'inferior', the task acquried by the manager becomes unsupervised by a some other manager (it 'loses' that manager).

-changing_manager_task_loser, cmtl: key best, worst, random, busiest, laziest, keyend = random

Depending on the setting of -changing_manager_task_remove, deletes the connection between the best/worst/laziest/busiest/random manager and the task just acquired, or a random task and the manager just acquired.

Changing Connection Parameters: (Manager-Analyst)

These parameters are used whenever manager-analyst connections are being considered.

-changing_manager_analyst_mean, cmam: real = 1.0

Specifies how many manager-analyst connections should be changed at a time.

-changing_manager_analyst_distribution, cmad: switch

If specified, indicates that the number of manager-analyst connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.

-changing_manager_analyst_superior, cmas: key best, worst, random, busiest, laziest, keyend = random

Specifies which manager should acquire an analyst. The best, worst, busiest, laziest, or random manager can be chosen.

-changing_manager_analyst_inferior, cmai: key best, worst, random, busiest, laziest, keyend = random

Specifies which analyst the manager should acquire. The best, worst, busiest, laziest, or random analyst can be chosen.

-changing_manager_analyst_remove, cmar: key inferior, superior, keyend = superior

If set to 'superior', the manager that acquired the analyst loses some other analyst. If set to 'inferior', the analyst acquried by the manager becomes unsupervised by some other manager (the analyst 'loses' the manager).

-changing_manager_analyst_loser, cmal: key best, worst, random, busiest, laziest, keyend = random

Depending on the setting of -changing_manager_analyst_remove, deletes the connection between the best/worst/laziest/busiest/random manager and the analyst just acquired, or the best/worst/laziest/busiest/random analyst and the manager just acquired.

Changing Connection Parameters: (CEO-Task)

These parameters are used whenever CEO-task connections are being considered.

-changing_ceo_task_mean, cctm: real = 1.0
Specifies how many CEO-task connections should be changed at a time.

-changing_ceo_task_distribution, cctd: switch
If specified, indicates that the number of CEO-task connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.

-changing_ceo_task_superior, ccts: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire a task. The best, worst, busiest, laziest, or random ceo can be chosen.

-changing_ceo_task_inferior, ccti: key random, keyend = random
Specifies which task the ceo should acquire.

-changing_ceo_task_remove, cctr: key inferior, superior, keyend = superior
If set to 'superior', the ceo that acquired the task loses some other task. If set to 'inferior', the task acquried by the ceo becomes unsupervised by some other ceo (it 'loses' the ceo).

-changing_ceo_task_loser, cctl: key best, worst, random, busiest, laziest, keyend = random
Depending on the setting of -changing_ceo_task_remove, deletes the connection between the best/worst/laziest/busiest/random ceo and the task just acquired, or a random task and the ceo just acquired.

Changing Connection Parameters: (CEO-Analyst)
These parameters are used whenever CEO-analyst connections are being considered.

-changing_ceo_analyst_mean, ccam: real = 1.0
Specifies how many CEO-analyst connections should be changed at a time.

-changing_ceo_analyst_distribution, ccad: switch
If specified, indicates that the number of CEO-analyst connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.

-changing_ceo_analyst_superior, ccas: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire an analyst. The best, worst, busiest, laziest, or a random ceo can be chosen.

-changing_ceo_analyst_inferior, ccai: key best, worst, random, busiest, laziest, keyend = random
Specifies which analyst the ceo should acquire. The best, worst, busiest, laziest, or a random analyst can be chosen.

-changing_ceo_analyst_remove, ccar: key inferior, superior, keyend = superior
If set to 'superior', the ceo that acquired the analyst loses some other analyst. If set to 'inferior', the analyst acquired by the ceo becomes unsupervised by some other ceo (the analyst 'loses' the ceo).

-changing_ceo_analyst_loser, ccal: key best, worst, random, busiest, laziest, keyend = random
Depending on the setting of -changing_ceo_analyst_remove, deletes the connection between the best/worst/laziest/busiest/random ceo and the analyst just acquired, or the best/worst/laziest/busiest/random analyst and the ceo just acquired.

Changing Connection Parameters: (CEO-Manager)
These parameters are used whenever CEO-manager connections are being considered.

-changing_ceo_manager_mean, ccmm: real = 1.0
Specifies how many CEO-manager connections should be changed at a time.

-changing_ceo_manager_distribution, ccmd: switch

If specified, indicates that the number of CEO-manager connections to be changed should be determined from a Poisson distribution using the mean. Otherwise the number of connections changed should always be the mean.

-changing_ceo_manager_superior, ccms: key best, worst, random, busiest, laziest, keyend = random
Specifies which ceo should acquire a manager. The best, worst, busiest, laziest, or a random ceo can be chosen.

-changing_ceo_manager_inferior, ccmi: key best, worst, random, busiest, laziest, keyend = random
Specifies which manager the ceo should acquire. The best, worst, busiest, laziest, or a random manager can be chosen.

-changing_ceo_manager_remove, ccmr: key inferior, superior, keyend = superior
If set to 'superior', the ceo that acquired the manager loses some other manager. If set to 'inferior', the manager acquired by the ceo becomes unsupervised by some other ceo (the manager 'loses' the ceo).

-changing_ceo_manager_loser, ccml: key best, worst, random, busiest, laziest, keyend = random
Depending on the setting of changing_ceo_manager_remove, deletes the connection between the best/worst/laziest/busiest/random ceo and the manager just acquired, or the best/worst/laziest/busiest/random manager and the ceo just acquired.

Deleting Connection Parameters:
The next 6 have to add to 1 and you should set them up in the same way as the add connections
And specify all nine of them
These parameters apply to all types of connection removal.

-deleting_threshold, dt: real = 100.0
Specifies how much the org's relative efficiency should change before deleting connections. (Not used in OrgAhead).

-deleting_when, dw: key rises, sinks, rises_or_sinks, keyend = sinks
Specifies how the org's relative efficiency should change in order to delete connections. The organization can do this when its efficiency either rises, sinks, or does either, by more than the -deleting_threshold. (Not used in OrgAhead).

-deleting_dormancy, dd: integer = 0
After deleting connections, the organization may not change itself for this many tasks. If a value less than change_cycle is given, then change_cycle is used.

-deleting_analyst_task_probability, datp: real = 0.17
Specifies the probability that when the organization deletes connections, they will be from analysts to tasks.

-deleting_manager_task_probability, dmtp: real = 0.17
Specifies the probability that when the organization deletes connections, they will be from managers to tasks.

-deleting_manager_analyst_probability, dmap: real = 0.17
Specifies the probability that when the organization deletes connections, they will be from managers to analysts.

-deleting_ceo_task_probability, dctp: real = 0.16
Specifies the probability that when the organization deletes connections, they will be from CEOs to tasks.

-deleting_ceo_analyst_probability, dcap: real = 0.16
Specifies the probability that when the organization deletes connections, they will be from CEOs to analysts.

-deleting_ceo_manager_probability, dcmp: real = 0.17
Specifies the probability that when the organization deletes connections, they will be from CEOs to managers. These six parameters should total 1.

If you want connections to be deleted between any two levels with equal probability, you can specify:
-datp 0.16 -dmtp 0.16 -dctp 0.16 -dmap 0.16 -dcap 0.16 -dcmp 0.16

Deleting Connection Parameters: (Analyst-Task)

These parameters are used whenever an analyst-task connections are being considered.

-deleting_analyst_task_mean, datm: real = 1.0
Specifies how many analyst-task connections should be deleted at a time.
-deleting_analyst_task_distribution, datd: switch
If specified, indicates that the number of analyst-task connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Deleting Connection Parameters: (Manager-Task)
These parameters are used whenever manager-task connections are being considered.

-deleting_manager_task_mean, dmtm: real = 1.0
Specifies how many manager-task connections should be deleted at a time.
-deleting_manager_task_distribution, dmtd: switch
If specified, indicates that the number of manager-task connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Deleting Connection Parameters: (Manager-Analyst)
These parameters are used whenever manager-analyst connections are being considered.

-deleting_manager_analyst_mean, dmam: real = 1.0
Specifies how many manager-analyst connections should be deleted at a time.
-deleting_manager_analyst_distribution, dmad: switch
If specified, indicates that the number of manager-analyst connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Deleting Connection Parameters: (CEO-Task)
These parameters are used whenever CEO-task connections are being considered.

-deleting_ceo_task_mean, dctm: real = 1.0
Specifies how many CEO-task connections should be deleted at a time.
-deleting_ceo_task_distribution, dctd: switch
If specified, indicates that the number of CEO-task connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Deleting Connection Parameters: (CEO-Analyst)
These parameters are used whenever CEO-analyst connections are being considered.

-deleting_ceo_analyst_mean, dcam: real = 1.0
Specifies how many CEO-analyst connections should be deleted at a time.
-deleting_ceo_analyst_distribution, dcad: switch
If specified, indicates that the number of CEO-analyst connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Deleting Connection Parameters: (CEO-Manager)
These parameters are used whenever CEO-manager connections are being considered.

-deleting_ceo_manager_mean, dcmm: real = 1.0
Specifies how many CEO-manager connections should be deleted at a time.
-deleting_ceo_manager_distribution, dcmd: switch

If specified, indicates that the number of CEO-manager connections to be removed should be determined from a Poisson distribution using the mean. Otherwise the number of connections deleted should always be the mean.

Firing Parameters:
These parameters apply to all aspects of dropping people from the org. Dropping is similar to move-to-other-problem , except it is not considered a voluntary move by the organization ; it is more like the person was taken out by the opposing forces.

-firing_probability, np: real = 0.0
Specifies the probability that someone should be dropped at each effeciency check,
-firing_analyst_probability, nap: real = 0.33
Specifies the probability that when the organization drops people, they will be analysts.
-firing_manager_probability, nmp: real = 0.34
Specifies the probability that when the organization drops people, they will be managers.
-firing_ceo_probability, ncp: real = 0.33
Specifies the probability that when the organization drops people, they will be CEOs. These three parameters must total 1.

Firing Analyst Parameters:
These parameters only apply to dropping analysts.

-firing_analyst_mean, nam: real = 1.0
Specifies how many analysts should be dropped at a time.
-firing_analyst_distribution, nad: switch
If specified, indicates that the number of analysts dropped should be determined from a Poisson distribution using the mean. Otherwise the number of analysts dropped should always be the mean.
-firing_analyst_victim, nav: key best, worst, random, busiest, laziest, keyend = random
Specifies which analyst should get dropped. The organization can nuke the best, worst, busiest, or laziest analyst, or can pick one randomly to nuke.
-firing_analyst_resources, nar: key none, best, worst, random, busiest, laziest, keyend = none
Specifies how a dropped analyst's tasks should be redistributed among the remaining analysts. The dropped analyst's tasks can go to the best, worst, busiest, laziest, or a random analyst.

Firing Manager Parameters:
These parameters only apply to dropping managers.

-firing_manager_mean, nmm: real = 1.0
Specifies how many managers should be dropped at a time.
-firing_manager_distribution, nmd: switch
If specified, indicates that the number of managers dropped should be determined from a Poisson distribution using the mean. Otherwise the number of managers dropped should always be the mean.
-firing_manager_victim, nmv: key best, worst, random, busiest, laziest, keyend = random
Specifies which manager should get dropped. The organization can nuke the best, worst, busiest, or laziest manager, or can pick one randomly to nuke.-firing_manager_resources, nmr: key none, best, worst, random, busiest, laziest, keyend = none
Specifies how a dropped manager's resources should be redistributed among the remaining managers. The dropped manager's resources can go to the best, worst, busiest, laziest, or a random manager.

Firing CEO Parameters:
These parameters only apply to dropping CEOs.

-firing_ceo_mean, ncm: real = 1.0
Specifies how many CEOs should be dropped at a time.
-firing_ceo_distribution, ncd: switch

If specified, indicates that the number of CEOs dropped should be determined from a Poisson distribution using the mean. Otherwise the number of CEOs dropped should always be the mean.

-firing_ceo_victim, ncv: key best, worst, random, busiest, laziest, keyend = random

Specifies which ceo should get dropped. The organization can nuke the best, worst, busiest, or laziest ceo, or can pick one randomly to nuke.-firing_ceo_resources, ncr: key none, best, worst, random, busiest, laziest, keyend = none

-firing_ceo_resources:

Specifies how a dropped CEO's resources should be redistributed among the remaining CEOs. The dropped CEO's resources can go to the best, worst, busiest, laziest, or a random ceo.