

Simulation explorer

Si vous ouvrez le module d'édition du modèle, vous ne pourrez que le consulter. Le modèle ne peut être édité que lorsque aucun autre module n'est ouvert.

Au cours d'une simulation, chaque acteur cherche à obtenir la plus grande valeur pour son but, défini comme

$$aim(a, s) = (1 - abs(GI(a))) * satisfaction(a, s) + GI(a) * (influence(a, s) - influence(a, a, s))$$

où

$s = (s_{r1}, \dots, s_{rn})$ est un état de l'organisation

$satisfaction(a, s) = \sum_{c \in A} \sum_{r \in R} solidarity(a, c) * stake(c, r) * effect_r(c, s_r)$, ce qu'il reçoit

$influence(a, b, s) = \sum_{r \in R; a \text{ controls } r} \sum_{b \in A} satisfaction(b, s)$, ce qu'il donne

Initial states : pour chaque relation, valeur initiale de son état et de ses bornes min et max (se substituent aux valeurs données dans Model editor, mais sans les modifier).

Actors' parameters : pour chaque acteur,

Distance min/max aim : indique la distance entre les situations qui procurent à l'acteur les valeurs minimales et maximales de son but (distance euclidienne entre les états des relations dont l'acteur dépend, pondérés par les enjeux).

Scope : détermine la capacité de l'acteur à discriminer les situations, et donc à sélectionner les règles qui sont applicables dans la situation courante ; avec la valeur 1, toutes les règles sont toujours applicables ; avec la valeur 3, l'acteur pourra distinguer les situations mauvaises, moyennes et bonnes. Les valeurs élevées prolongent les simulations

Tenacity : plus la ténacité de l'acteur est grande, plus il a tendance à explorer, au détriment de l'exploitation ; les valeurs élevées rallongent donc sensiblement la durée des simulations.

Group Identification : investissement de l'acteur dans l'organisation.

Repartition of reward : répartition de la récompense entre la dernière et l'avant dernière règle : récompense dernière / récompense avant-dernière.

Type of rules :

Self-learning : les règles sont récompensées proportionnellement à ce qu'elles ont fait gagner ou perdre ; cette récompense est ajoutée selon la formule $force := al * force + (1 - al) * récompense$, où $al \in [0, 1]$ croît progressivement avec le taux d'exploitation.

Simple : les règles sont récompensées de façon forfaitaire d'un montant **reward**, indépendamment de ce qu'elles ont fait gagner ou perdre ; cette récompense est ajoutée à la force de la règle, jusqu'à un certain plafond ; les règles sont progressivement oubliées du facteur **oblivion**.

Oblivion : facteur d'oubli des règles, utilisé uniquement avec les règles simples.

Reward : récompense utilisé uniquement avec les règles simples.

Action range : valeur de référence de l'importance des actions de l'acteur. Plus c'est élevé, plus l'exploration de l'acteur est énergique, utilisé uniquement avec les règles simples.

N'oubliez pas de faire **accept** pour valider les valeurs de ces paramètres.

Partie gauche

Normal stakes, Normal solidarities : variation ou non (option **fuzzy**) des valeurs des enjeux et solidarités pour chaque simulation.

With constraints : application ou non des contraintes entre les relations.

Number of steps : nombre maximum de pas de chaque simulation ; une simulation qui s'arrête avant ce nombre de pas **converge**, dans le cas contraire elle ne le fait pas.

Number of runs : nombre de simulations.

Accept : pour valider ces valeurs.

Run : lance les simulations. Indiquer le répertoire (par défaut celui contenant le modèle), dans lequel sera créé un répertoire <nom du modèle>-simulation-<dateHeure>;
Ce répertoire contiendra les résultats des simulations.

Open previous simulation : permet de restaurer les paramètres et les résultats d'une simulation antérieure en ouvrant, dans le répertoire contenant les résultats le fichier (nom du modèle)-SimulationInitialParameters.xls qui lui est associé.

View results : accès à la visualisation des résultats, voir ci-dessous.

Synthesis results : synthèse des résultats : moyenne et écarts types des simulations.

State analysis : ouvre le module State analysis ; dans la liste *Significant states*, "Convergence's state" correspond à la moyenne de l'état des relations en fin simulations.

Save report : crée un fichier de nom
(nom du modèle)_(date)(heure)Synthesis.rtf

Clear report : enlève du rapport les courbes qui y ont été ajoutées.

Fenêtre View results de visualisation des résultats

Option **Convergence** : proportion des simulation qui ont convergées (Synthesis results ne prend en compte que celles qui ont convergées).

Option Actor

Experiment : Choisir soit toutes les simulations pour un seul acteur, soit une simulation particulière avec tous les acteurs.

Variable : selon le choix, soit un histogramme, soit une courbe dont l'abscisse est le nombre de pas et l'ordonnée la but ou l'ambition de l'acteur.

Avec le choix "toutes les simulations", la courbe en rouge soutenu correspond à la valeur moyenne.

Les \oplus marquent le point terminal d'une simulation.

En positionnant le curseur dessus (ou sur tout emplacement d'une courbe), vous aurez l'indication de l'acteur concerné. Possibilité de zoomer.

Option Relation : idem que l'option Actor.

ZZZ : Le chargement des fichiers contenant les résultats de simulation peut provoquer un débordement mémoire (cf. message sur la console) ; dans ce cas, mieux vaut relancer SocLab.

Specific Run

Appel la fonction `core.orgNew.specificRun_1()` que vous pouvez programmer selon les calculs que vous souhaitez réaliser.

L'algorithme Self-learning

```
repeat          // The Global Simulation Loop
  foreach actor a:          //all actors see the same world
    a.action = a.selectAction ( )
  foreach actor a:          //they don't act in turn
    performAction (a.action)
Until (foreach actor a: a.ambitiont ≤ a.aimt)

//SelectAction
  // Perception of aim and updating gap
  aimt(a, s) = (1 - abs(GI(a))) * satisfaction(a, s) + GI(a) * (influence(a, s) - influence(a, s))
  deltaAim = aimt - aimt-1
  gap = (ambitiont-1 - aimt) / (ambitiont-1 - minAim)
  // Updating ambition
  if (ambitiont-1 > aimt)
    if ((gap < (11 - tenacity) / 10) && (deltaAim == 0))
      ambitiont = ambitiont-1 - ((1 - ExpRt-1) * (ambitiont-1 - aimt + 1)) / 100
    else ambitiont = ambitiont-1 - ((1 - ExpRt-1) * gap / 50 //about
  else //a is already satisfied
    ambitiont = ambitiont-1 + ((aimt - ambitiont-1) / 100)
  // Updating exploration rate
  ExpRIns = 0.1 + (0.8 / (1 + eslope * (gap - abscissa)))
  ExpRt = ExpRt-1 * ExpRt-1 + (1 - ExpRt-1) * ExpRIns
  // Updating action_range
  action_range = 2 * ExpRt //about
  // Updating strength of last and penultimate selected rules
  // SRt stands for the Rule Selected at time t
  SRt-1.strengtht = (1 - ExpRt) * SRt-1.strengtht-1 + ExpRt * autonomy * deltaAim
  SRt-2.strengtht = SRt-2.strengtht-1 + ExpRt * (1 - autonomy) * deltaAim
  // Forgetting bad rules
  if (SRt-2.strengtht < 0) RuleBase.remove(SRt-2)
  // Selecting the set M of rules applicable at time t
  M.clear ( )
  foreach rule R in RuleBase
    if (distance(R.situation, CurrentSituation < closenessThreshold)
      M.add(R)
  // Selecting an action
  if (M.isEmpty())
    SRt = (CurrentSituation(), (atRandom( ) * action_range), 0)
    RuleBase.add (SRt)
  else
    SRt = ChooseOneOfThreeRulesWithMaximumStrength (M)
  return (SRt.action)
```

