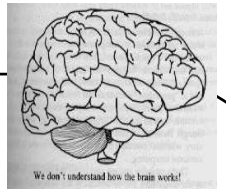
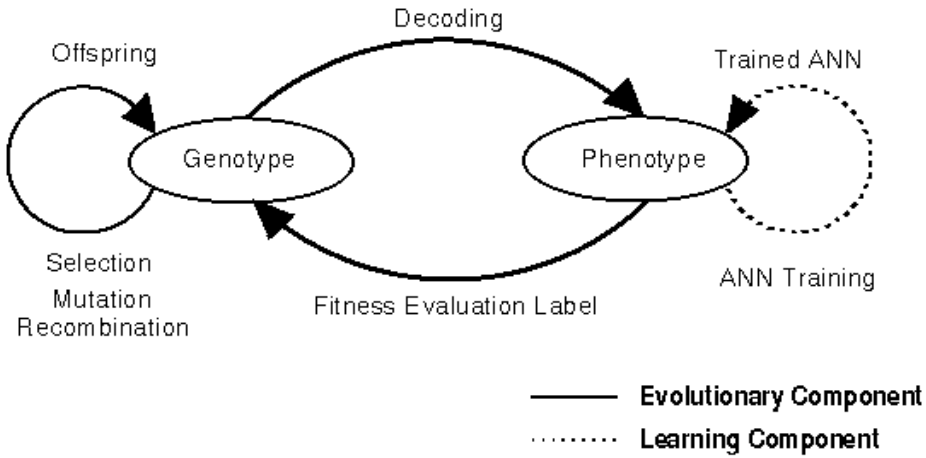
Modelos Conexionistas y Autómatas

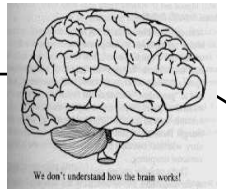


ALGORITMOS GENÉTICOS Y REDES NEURONALES

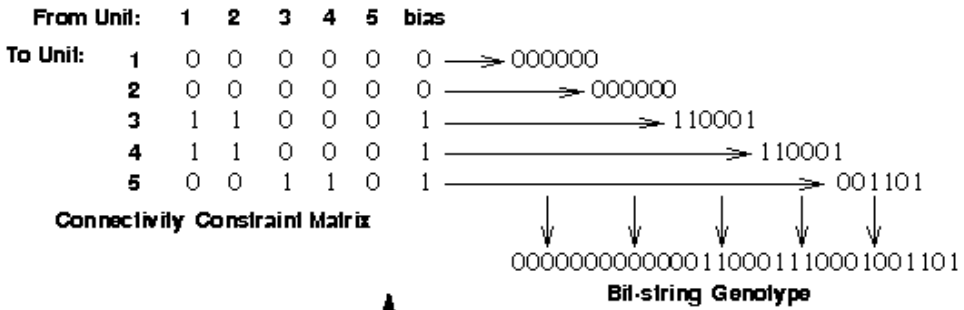
- **Mediante el uso de Algoritmos Genéticos (AG) se pude diseñar (optimizar) tanto la arquitectura como los pesos y parámetros más adecuados para una RN.**
- **Los AG son uno de los métodos más potentes para resolver problemas de optimización.**
- **Los AG son métodos multipunto: parten de una población (conjunto de RNs) en su búsqueda.**
- **El criterio a optimizar es la eficacia (fitness) individual de la red.**
- **A partir de la población inicial, mediante una serie de operaciones se genera la descendencia (offsprings).**
- **Entre las operaciones más usuales tenemos:**
  - **Mutación:**
    - **Descendiente igual al padre, pero con pequeñas variaciones.**
  - **Recombinación (crossover):**
    - **Propiedades proveniente de 2 (ó más) padres.**
    - **Selección de padres aleatoria, prefiriendose los mejores.**

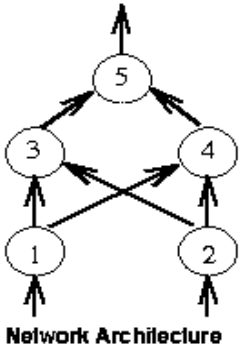


Modelos Conexionistas y Autómatas



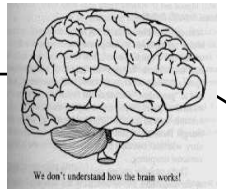
- **Para poder aplicar AG a un problema es necesario:**
  - » **Poder representar las soluciones como strings.**
  - » **Poder calcular la eficacia (fitness) de las soluciones.**
- **En las RN, se plantean por tanto dos tipos de problemas de optimización:**
  - » **Topología o Arquitectura de red (problema discreto):**
    - **Nº de unidades ocultas**
    - **Estructura de interconexionado**
  - » **Ajuste de parámetros (problema continuo):**
    - **Parámetros de aprendizaje**
    - **Pesos**





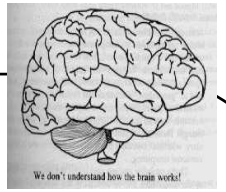
- **Clásicamente, la forma de resolver estos problemas es:**
  - » **Topología: Intuición del diseñador**
  - » **Parámetros aprendizaje: Prueba y error**
  - » **Pesos: Algoritmos de aprendizaje**
- **Mediante AG se pueden abordar todos estos problemas, aunque usualmente se combinan con los Algoritmos clásicos de RN.**

Modelos Conexionistas y Autómatas

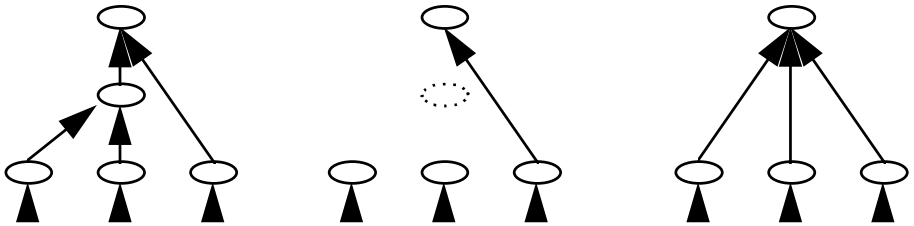


- **ENZO**
  - **Es un sistema evolutivo de optimización de RN que se utiliza en conjunción con SNNS (Stutgart Neural Network Simulator).**
  - **ENZO se encarga de optimizar únicamente la topología de la red, dejando el resto del trabajo al SNNS o al diseñador.**
  - **Actualmente sólo trabaja con redes que utilicen el algoritmo de descenso de gradiente rápido (RPROP).**
  - **Tener en cuenta que:**
    - **No optimiza los parámetros del aprendizaje.**
    - **No realiza aprendizaje de pesos.**
    - **Sólo utiliza redes tipo RPROP.**
  - **El ciclo evolutivo se compondrá de los siguientes pasos:**
    - **Preevolución:**
      - **Usando las especificaciones del usuario se genera una población de redes con una determinada densidad de conexionado.**
    - **Ciclo:**
      - **Selección de un padre, prefiriendo los mejores.**
      - **Generar un descendiente tras mutar un duplicado del padre.**
      - **Si se elige recombinación, añadir aleatoriamente uno o varios nodos ocultos del segundo padre.**
      - **Entrenar el nuevo miembro:**
        - **Usar decaimiento de pesos.**
        - **Podar pesos inútiles y reentrenar si procede.**
      - **Evaluar la eficacia del nuevo miembro:**
        - **Considerar criterios de diseño más importantes.**
      - **Insertar en la población, eliminando al peor.**
    - **Postevolución.**

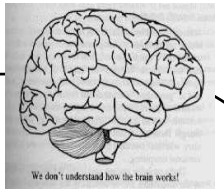
Modelos Conexionistas y Autómatas

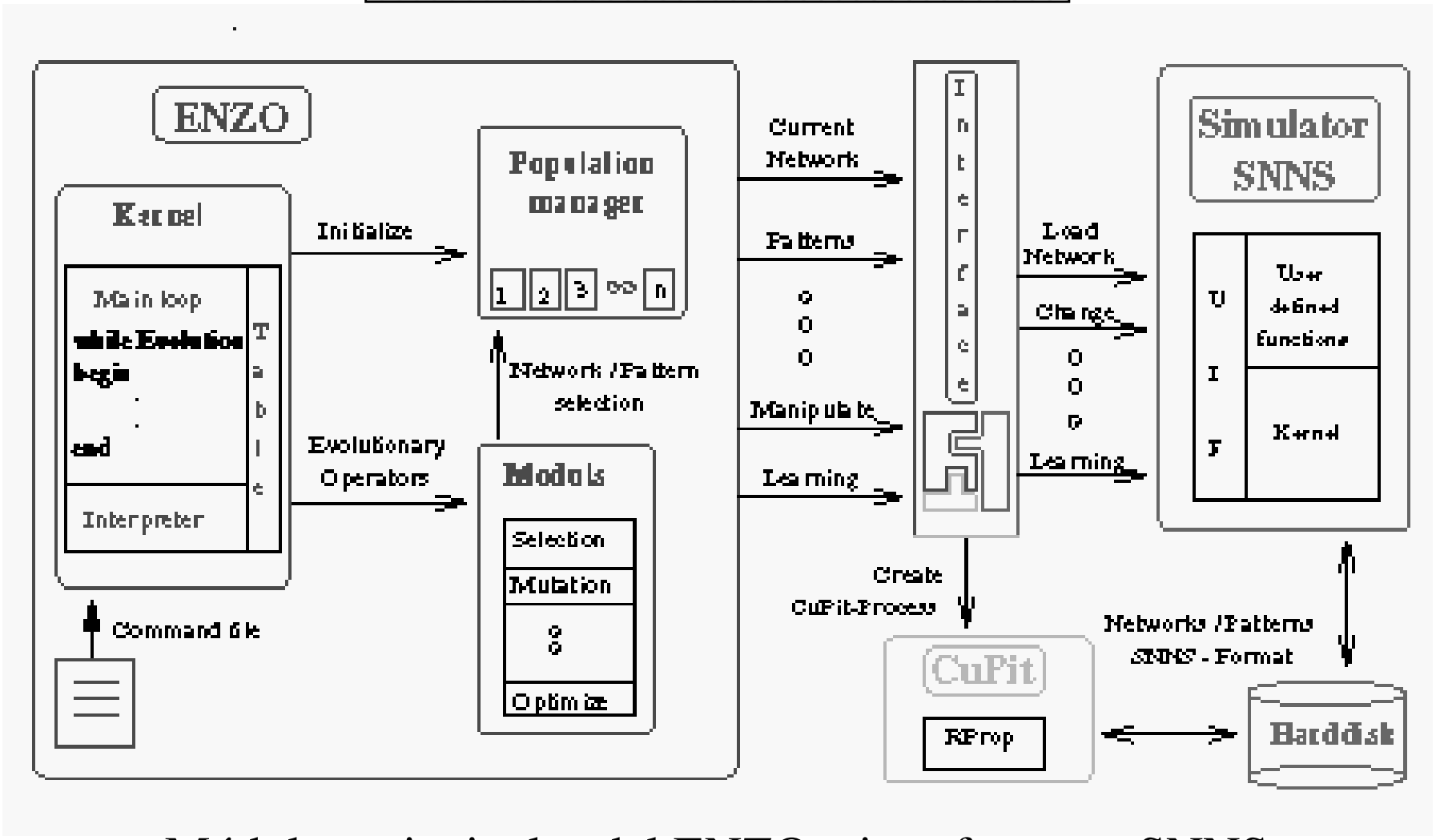


- **Mutaciones:**
  - » **Links:**
    - **Borrado de links (soft pruning): Opción: Más probable el de menor peso ($p_{del} \cdot N_{\theta}(0,w)$)**
    - **Añadido de links: Igua probabilidad (p_{add})**
    - **No se evalúan directamente los efectos de pruning, lo hará la propia evolución.**
  - » **Borrado de Unidades:**
    - **Mayor impacto en la red.**
    - **Dos heurísticas: PWU y Algoritmo Bypass**
    - **PWU: Preferir unidades débiles.**
      - **Ordena unidades según conectividad relativa (conexiones activas / conexiones máximas).**
      - **Borra preferiblemente las débilmente conectadas.**
      - **Complementa al soft-pruning.**
    - **Algoritmo Bypass:**
      - **Aproxima la función no lineal implementada por una lineal**
      - **Aumenta la proporción de éxitos.**

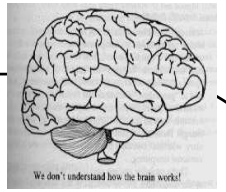


- **Conclusiones:**
  - » **Al utilizar 2 técnicas de búsqueda:**
    - **Descendiente rápido del gradiente, para optimización local de pesos.**
    - **Evolución, para optimización global de la topología.**
  - » **evita quedar atrapado en mínimos locales.**
  - » **Acelera el aprendizaje y la calidad de los descendientes mejora el promedio de su topología.**
  - » **Trata de eliminar las unidades de entradas redundantes.**





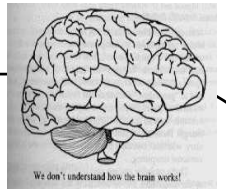
Módulos principales del ENZO e interface con SNNS



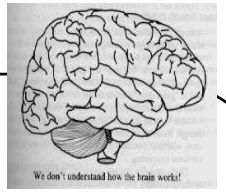
xtc.cmd

```
# sample command-file for tc-problem                        --- TR, 22.08.95 ---
# use the seed == 814288352 to receive optimal result
# in about 20 generations !
#
#########################
#        files          #
#########################
network      tc.net         # name of network-file
learnpattern tc.pat         # name of learnpattern-file
testpattern  tc.pat         # name of testpattern-file
historyFile  hst            # append this suffix to history files
netDestName  tc_erg         # save nets with this prefix
#########################
#        modules        #
#########################
# ----- initialize and pre-evolution ----------------------------------------
genpopNepo     # initialize nepomuk
loadSNNSPat    # load SNNS-Pattern
initPop        # create and initialize Population
optInitPop     # optimize initial population
# ----- stop-evolution ------------------------------------------------------
stopErr        # stop if something's wrong with parents/offsprings
stopIt         # stop after maxGenerations
# ----- selection -----------------------------------------------------------
preferSel      # use random prefer selection
```

Modelos Conexionistas y Autómatas

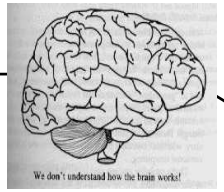



```
# ----- mutation ----------------------------------------------------------
simpleMut      # do simple mutation
mutUnits       # delete or add hidden units
mutLinks       # delete or add links
mutInputs      # cut off input units
# ----- optimize ----------------------------------------------------------
nullWeg        # delete useless nets
relearn        # scale weights before relearning
learnSNNS      # do learning via SNNS-Function
#learnCupit    # do learning via CuPit / switch with learnSNNS
prune          # use pruning
cleanup        # remove dead units
nullWeg        # delete useless nets
learnSNNS      # do re-learning via SNNS-Function
#learnCupit    # do learning via CuPit / switch with learnSNNS
# ----- evaluate ----------------------------------------------------------
learnRating    # increase fitness if training patterns were not learned
topologyRating # increase fitness dependent on topology
# ----- history -----------------------------------------------------------
histSimple    # write standard information
histFitness   # write fitness information
histWeights   # write weight and unit information
Xhist         # show fitness in a X-Window
# ----- survival ----------------------------------------------------------
fittestSurvive  # sort nets by fitness (the lower the better)
# ----- post-evolution ----------------------------------------------------
saveAll         # save networks
```




```
#########################
#     parameters        #
#########################
maxGenerations 30   # stop after  x  generations
popsize   30        # population size
gensize   10        # new members each generation
preferfactor 3.0    # preferfactor for better nets
# ----- optimization ----------------------------------------------------
learnModul learnSNNS    # use standard SNNS-learning
#learnModul learnCupit    # use CuPit - learning
learnfct   Rprop                  # learning function to be set
learnparam 0.00 20.99 5.0 0.0 0.0  # and parameters
maxtss     0.01                   # stop learning if error is smaller
maxepochs  100                    # stop after x epochs
shuffle 1                         # shuffle patterns
relearnfactor 0.5                 # scale weights before relearning by x
threshold 0.2                     # minimal treshhold for pruning
thresholdStart 0.5                # Start with this treshold
pruneEndGen 30                    # use minimal treshold from this generation
# ----- mutation --------------------------------------------------------
probadd 0.2               # probability of adding a link
probdel 0.1               # probability of removing a link
probMutUnits 0.1          # probability of unit mutation
probMutUnitsSplit 0.9     # probability relation between adding and deleting
probMutInputs      0.6    # probability of input unit mutation
probMutInputsSplit 0.6    # probability relation between adding and deleting
```

Modelos Conexionistas y Autómatas




```
# ----- evaluation ------------------------------------------------------
                          # linear sum of:
weightRating   20.0       # number of weights times x
unitRating     20.0       # number of units times x
inputRating   200.0       # number of inputs units times x
tssRating      30.0       # mean error  times x
noLearnRating 200.0       # add x if training patterns were not learned
#testHamThresh   0.2       # Treshold for hamming distance
#testHamRating  10.0       # Add x if over treshold
# ----- history ---------------------------------------------------------
Xgeometry 650 10 600 330   # size and position of X-Window
Xcoord 0.0 0.0 30.0 600.0  # size of the coordinate system
# ----- post-evolution --------------------------------------------------
saveNetsCnt 5             # save x nets
# eof
```