UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITƒCNICA SUPERIOR

Introducción a ProGen

Herramientas de la inteligencia artificial

Introducción

- En estas prácticas:
  - Aprenderemos a usar una herramienta de PG
- No queremos tener que programar todo un sistema de PG para aplicar PG a nuestro problema
- ¿Qué hacemos?
  - Utilizamos alguna librería, API, Framework o entorno SW que nos ahorre este trabajo

Introducción

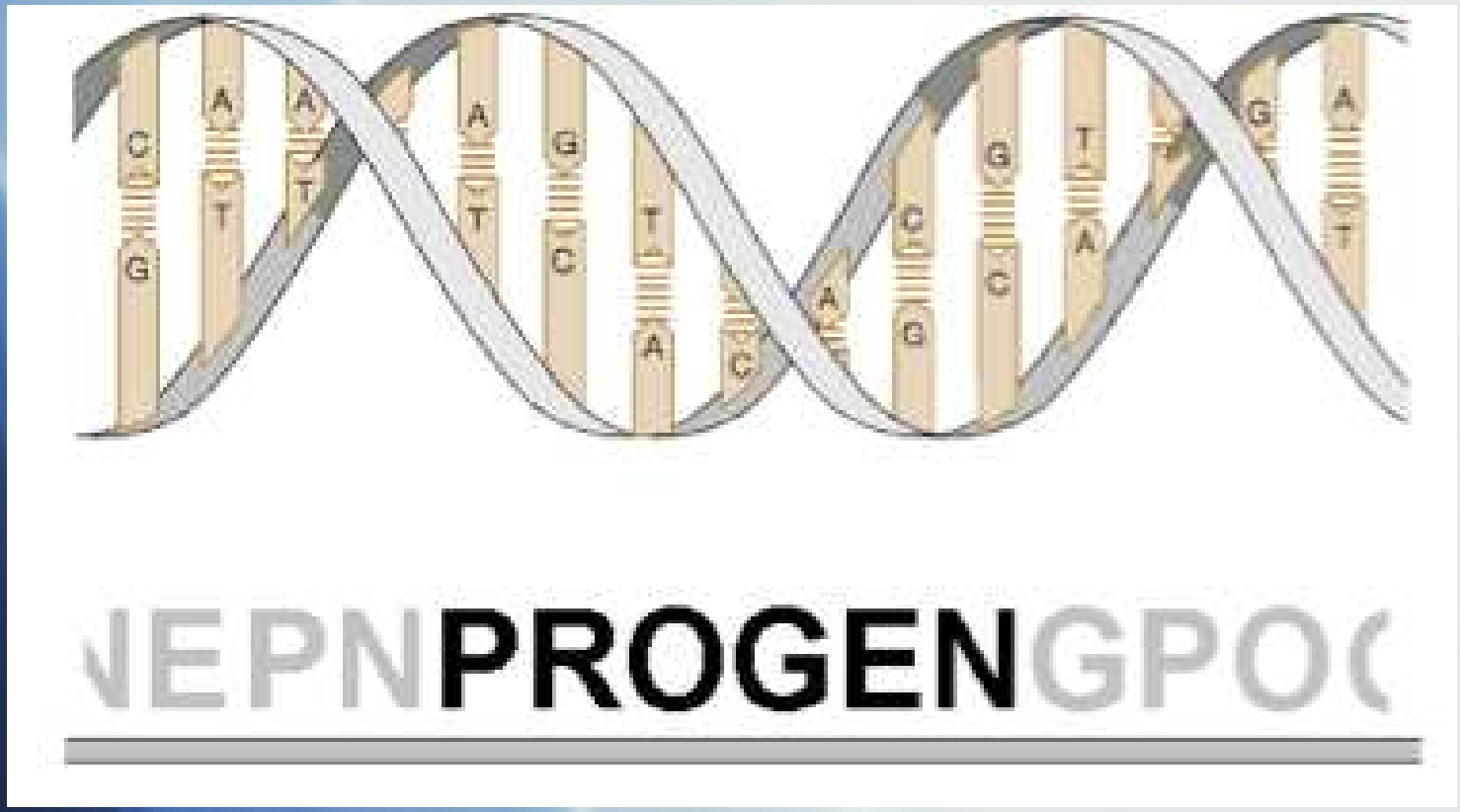
- ¿Y qué opciones tenemos?
  - Lilgp
  - ECJ
  - BeagleGP
  - GPC++
  - EO Evolutionary Computation Framework
  - JGProg
  - JGAP
  - DGPF
  - …

En esta asignatura siempre hemos usado este

Lilgp

- Desventajas:
  - Se programa en C (en la uc3m los alumnos suelen estar mucho más cómodos en Java)
  - Tiene bugs
  - Nadie lo mantiene desde hace unos nueve años

Este año…



ProGen

- Motor de PG desarrollado en la Carlos III (Grupo EVANNAI)
- Escrito íntegramente en Java
- Software Libre (próximamente)
- Autores:
  - César Estébanez
  - Alberto Vegas
  - Pablo Alonso
  - Elena Frau
  - David Fernández
  - Iñaki Reta

ProGen y ProGen 2

- Llevamos unos meses trabajando en ProGen 2
- ProGen creció de forma descontrolada
- ProGen 2: rediseñado y reescrito “from scratch”
- Empezaremos a usar ProGen 2 en cuanto esté disponible (muy pronto)
- De momento, empezaremos con ProGen

ProGen: Ventajas para vosotros

- Fácil de usar
- Aún en fase de desarrollo
- Proyectos para ampliarlo constantemente en los próximos años
- Utilizado en investigación
- Acceso (inmejorable) a los desarrolladores
- Vosotros mismos podéis participar en su desarrollo

ProGen: Introducción

- Primero, descargarlo de Aula Global
- Instalación: no hay que hacer nada.
- Manejo: Eclipse resulta muy útil (editor, precompilador, debugger)
- ¿Linux o Windows?

ProGen: Manual de usuario

- En el directorio ProGen/doc/
- Es una versión bastante desfasada, pero contiene mucha información interesante.
- Hay una guía de inicio rápido que explica en cuatro páginas cómo empezar a trabajar con ProGen.
- En cualquier caso, repito: es un borrador.

ProGen: Familiarizarse (I)

- Explorar estructura:
  - Archivos de configuración (y de salida):
    - Sobre todo master_file.cfg
      - Le dice a ProGen qué problema (de los muchos que podemos tener codificados en ProGen al mismo tiempo) vamos a ejecutar
    - Los demás archivos los veremos más adelante

ProGen: Familiarizarse (II)

- Explorar estructura:
  - El directorio src:
    - Dos clases importantes:
      - ProGen.java
      - Experimenter.java
    - Dos paquetes fundamentales:
      - progen
      - userprogram
    - Dentro de userprogram:
      - Los problemas que el usuario codifica para que ProGen los resuelva. Aquí estará nuestro código

ProGen: Familiarizarse (III)

- Explorar estructura:
  - El directorio src:
    - El paquete progen:
      - Contiene:
        - el núcleo de ProGen.
        - Los operadores genéticos.
        - El repositorio de funciones y terminales.
        - El output.
        - Y otras cosas.

ProGen: Familiarizarse (III)

- Explorar estructura:
  - El directorio src:
    - El paquete userprogram:
      - Varios ejemplos preprogramados (muchos más en la versión de desarrollo).
      - Además de un Skeleton
      - Cuando acabe el curso habrá más.
      - Hoy trabajaremos con el de la regresión.

Toma de contacto. Compilación y ejecución: Regression

- Editar master_file.cfg:
  - prp_experiment_file: src/userprogram/Regression/Regression.txt
  - prp_experimenter: off
  - Las demás opciones del master file no se usan de momento
- Directorio /src/userprogram/Regression:
  - Regression.java:
    - Función de fitness:
    - Recibe un individuo.
    - Computa su fitness y lo devuelve:
      - IMPORTANTE: ProGen siempre intenta MINIMIZAR el fitness
    - Datos internos:
      - Accesibles desde la función de fitness y desde las funciones y Terminales
  - Regression.txt:
    - parámetros de la ejecución

Toma de contacto. Compilación y ejecución: Regression

- Volvemos al directorio raíz de ProGen:
  - Compilar: $ ant
  - Ejecutar: $ ant ProGen
- Familiarizarse con el Output
  - Salida por pantalla
  - Salida a ficheros
  - (Los distintos tipos de salida se pueden configurar en el master_file)

Toma de contacto: el Skeleton

- /src/userprogram/skeleton
- Sirve como base para programar nuestras propias aplicaciones.
- Contiene la estructura de las funciones que debemos implementar además de comentarios muy útiles.
- Echadle un vistazo.

Implementando problemas: La Regresión Simbólica

- Seguiremos todos los pasos de creación de una aplicación real de PG con ProGen
- PASO 1: Identificar el problema
  - el GP-program intentará “descubrir” o aproximar la curva $x^4 + x^3 + x^2 + x$

    mediante una serie de puntos pertenecientes a la misma
  - Terminales: {X, ¿etc?}
  - Funciones: {*, /, +, -, ¿etc?} (recordar propiedad closure)
  - Raw Fitness: suma de los errores cometidos en cada punto (recordad que queremos minimizar el fitness)

Implementando problemas: La Regresión Simbólica

- PASO 2: Parámetros de la ejecución:
  - Parámetros de la ejecución: definir tamaño población, número de generaciones, limitaciones de tamaño de los árboles, método de generación de individuos, etc.
  - Fundamental: Los Function Sets
    - Qué funciones y terminales usa cada function set
      - Definidos en el repositorio de funciones o en nuestro directorio
    - Qué function set usa cada RPB y cada ADF
  - Fundamental: Los operadores genéticos (y el elitismo)
  - Error común: No olvidar el prp_project_name

Implementando problemas: La Regresión Simbólica

- PASO 3: Crear nuestras funciones o terminales (si es que usamos alguna que no esté en el repositorio)
  - Lo más cómodo es copiar el .java de una del repositorio y después modificarlo
  - Tendremos que cambiar el package (ahora es userprogram.nuestroprograma) y también importar progran.functions.Function
  - El constructor: `super(2, "int$$int$$int", "-");`
  - Método execute():
    - Recibe un array de hijos: PGNode[] children
    - Puede ejecutarlos: children[0].execute(uProgram, stack);
    - Debe devolver un objeto del tipo indicado en el interfaz

Implementando problemas: La Regresión Simbólica

- PASO 4: Escribir la función de fitness:
  - Recibe un individuo que tenemos que evaluar
  - Operaciones importantes:
    - El método estático SetVariable(nombreVar, value)
      - Nota: El nombre de la variable se escribe tal como aparece en el function set del archivo de parámetros (no como el nombre de su clase)
    - El método ind.execute(userprogram)
    - return fitness;
  - Métodos initialize(), uninitialize(), postProcessBest():
    - No los usaremos en este problema

Implementando problemas: La Regresión Simbólica

- PASO 5: Ejecución, análisis y Parameter Tunning:
  - Desde el directorio raíz de ProGen:
    - $ ant
    - $ ant ProGen
  - Observar output y archivos de salida
  - ¿A qué soluciones llega? ¿Encuentra la función objetivo?
  - Tuneado de parámetros

Fin de la primera sesión

- En la próxima sesión implementaremos un ejemplo basado en side effects: El problema de la hormiga artificial
  - En este caso los conceptos cambian totalmente.
  - La lógica subyacente a la PG es más complicada de entender.
  - Y los operadores y la función de fitness un poquito más enrevesados.
- ¿Alguna pregunta sobre la sesión de hoy?