Optimización por Enjambre de Partículas en Jason-CArtAgO

Arturo Márquez Flores

Centro de Investigación en Inteligencia Artificial, Universidad Veracruzana

Introducción

En este trabajo se presenta una implementación del algoritmo de optimización por enjambre de partículas en Jason-CArtAgO. Dicha implementación hace uso de herramientas de Jason [1] como las acciones internas programables en Java y comunicación mediante actos de habla [2]. Además, utilizando el marco de CArtAgO [3], esta implementación se basa en el paradigma de agentes y artefactos.

Por otro lado, en PSO los agentes involucrados son muy limitados. Esta implementación busca establecer una base que permita desarrollar agentes más complejos y con diferentes representaciones de conocimiento y paralelizables, con el fin de poder resolver problemas de optimización no triviales.

PSO

El algoritmo PSO es un método de optimización estocástica basado en poblaciones propuesto por Kennedy y Eberhart en [4]. Cuenta con una *población* de N soluciones potenciales que contienen una posición - x_t^i, una velocidad - v_t^i y una mejor posición personal - p_t^i.

El algoritmo inicializa las posiciones iniciales de manera uniformemente aleatoria sobre el espacio de búsqueda y en cada iteración, y para cada partícula, realiza una actualización de los valores de velocidad, posición y mejor posición personal de acuerdo con las siguientes ecuaciones:

$$v_{t+1}^i = \omega v_t^i + \phi_1 R_{1,t}^i (p_t^i - x_t^i) + \phi_2 R_{2,t}^i (g_t - x_t^i)$$

$$x_{t+1}^i = x_t^i + v_t^i$$

$$p_{t+1}^i = \begin{cases} x_{t+1}^i & \text{if } f(x_{t+1}^i) < f(p_t^i) \\ p_t^i & \text{if } f(x_{t+1}^i) \geq f(p_t^i) \end{cases}$$

El algoritmo también actualiza el mejor global del total de partículas, g_t, en cada iteración.

Jason

La implementación en Jason nos permite aprovechar dos aspectos principales. Primero la comunicación basada en actos de habla. En este caso, estas son útiles para que el agente central *central* solicite al resto de los agentes *particula* que se inicialicen y actualicen su velocidad y posición. Además cada partícula puede comunicar su mejor personal al agente central. Jason también permite aprovechar la versatilidad del lenguaje de propósito general Java para crear acciones internas que proveen de extensibilidad al lenguaje. En esta implementación se utilizan las acciones internas para realizar las siguientes funciones:

Acciones Internas

- **individuoAleatorio.java** - Regresa una nueva posición aleatoria en forma de lista, con cada valor de entrada en cierto rango de valores permitidos por el espacio de búsqueda.
- **nuevaVelocidad.java** - Regresa una nueva velocidad en forma de lista aplicando la ecuación para la actualización de velocidad, tomando como argumentos los parámetros, la posición, el mejor personal, y el mejor global.
- **velocidadAleatoria.java** - Regresa una nueva velocidad aleatoria en forma de lista, con cada valor de entrada en cierto rango de valores.
- **nuevaPosicion.java** - Regresa una nueva posición en forma de lista aplicando la ecuación para la actualización de posición, tomando como argumentos la posición antigua y la velocidad.
- **sphere.java** - Regresa un valor numérico dada una posición. Esta es un ejemplo de las funciones objetivo que el algoritmo puede minimizar.

CArtAgO

Para llevar a cabo sus objetivos, los agentes interactúan con un artefacto que funciona como un enlace entre los agentes y el usuario. Esto permite establecer una conexión con los agentes y el ambiente. En este caso el artefacto le permite al usuario definir los parámetros del algoritmo e inicializar un experimento.

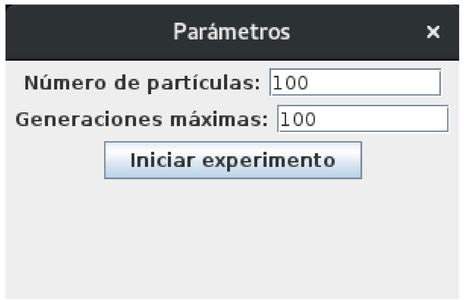

Fig. 1: GUI

Agentes

Los agentes en este proyecto son tres:

Agentes

- **parameters.asl** - Define las creencias que contienen la información sobre los parámetros para la actualización de la velocidad. Todas las partículas heredan estas creencias mediante la cláusula *include*.
- **central.asl** - Se encarga de recibir los parámetros del GUI, crear la población, calcular el mejor global, comunicarlo a las partículas y, mediante un acto de habla, pedirle a las partículas que realicen la actualización de los valores.
- **particula.asl** - Implementa las acciones internas para inicializar y actualizar los valores y comunica su mejor personal al agente central.

Restuldados

En el Cuadro 1 se muestra que esta implementación minimiza exitosamente la función de ejemplo $y = x_1^2 + x_2^2$.

Parámetros	Valor Inicial	Valor Final
10 partículas 10 iteraciones	0.153578	0.017855
50 partículas 50 iteraciones	0.039257	0.0000...
100 partículas 100 iteraciones	0.003065	0.0000...

Conclusiones

Jason-CArtAgO permite hacer una implementación eficiente del PSO. Aunque los resultados no son concluisivos sobre la capacidad del algoritmo para resolver problemas más complejos, las herramientas de Jason-CArtAgO tiene ventajas cualitativas importantes. La ventaja más relevante es que esta implementación permite realizar extensiones para agentes más complejos y así poder resolver problemas no triviales. Por otro lado, la implementación permite la paralelización de los agentes, lo cual tendría un efecto positivo sobre la eficiencia computacional, sobre todo si se trata de agentes más complejos.

Referencias

[1] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge.
Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology).
John Wiley & Sons, Inc., USA, 2007.

[2] Anand S. Rao.
Agentspeak(l): Bdi agents speak out in a logical computable language, 1996.

[3] Alessandro Ricci, Michele Piunti, and Mirko Viroli.
Environment programming in multi-agent systems: an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems, 23(2):158–192, Sep 2011.

[4] James Kennedy and Russell C. Eberhart.
Particle swarm optimization.
In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.