Global Path Planning Utilizdo Particle Swarm Optimizacin

Joe Cabezas Arnaldo Gaspar

June 27, 2011

Motion Planning
Global Path Planning

- Encontrar una o más rutas para que un agente en particular logre llegar desde una región inicial p_{inicial} a otra región objetivo p_{objetivo}

Motion Planning
Global Path Planning

- Encontrar una o más rutas para que un agente en particular logre llegar desde una región inicial p_{inicial} a otra región objetivo p_{objetivo}
- Considerando obstáculos, condiciones de movimiento, balance, etc.

Motion Planning
Global Path Planning

- Encontrar una o más rutas para que un agente en particular logre llegar desde una región inicial p_{inicial} a otra región objetivo p_{objetivo}
- Considerando obstáculos, condiciones de movimiento, balance, etc.
- Corresponde a un problema de optimización que puede incluir criterios como distancia, tiempo, energía, etc.

Motion Planning
Global Path Planning

- Encontrar una o más rutas para que un agente en particular logre llegar desde una región inicial p_{inicial} a otra región objetivo p_{objetivo}
- Considerando obstáculos, condiciones de movimiento, balance, etc.
- Corresponde a un problema de optimización que puede incluir criterios como distancia, tiempo, energía, etc.
- Lo primero es resolver el problema en un entorno estático, y posteriormente crear un plan para un entorno dinámico.

Motion Planning
Global Path Planning

- Encontrar una o más rutas para que un agente en particular logre llegar desde una región inicial p_{inicial} a otra región objetivo p_{objetivo}
- Considerando obstáculos, condiciones de movimiento, balance, etc.
- Corresponde a un problema de optimización que puede incluir criterios como distancia, tiempo, energía, etc.
- Lo primero es resolver el problema en un entorno estático, y posteriormente crear un plan para un entorno dinámico.
- Nuestro objetivo es resolver el Global Path Planning

Particle Swarm Optimization

Conceptos

1. El enjambre de partículas corresponde al conjunto de S partículas
2. Cada partícula es una eventual solución y corresponde a una ruta
3. A cada solución se le calcula una velocidad de movimiento respecto a su mejor posición histórica y respecto a la mejor posición global
4. El objetivo es que cada partícula tome movimientos relativos al mejor del grupo y a su propia información de mejor posición usando la velocidad

Representación
Variables

1. El mapa está representado como una matriz con coeficientes binarios

$$Map = \begin{bmatrix} (1,1) & \cdots & (1,m) \\ \vdots & \ddots & \vdots \\ (n,1) & \cdots & (n,m) \end{bmatrix}$$

2. Una partícula corresponde a una solución del problema, es decir una ruta r de longitud l

$$r = \{(p_{1x}, p_{1y}), \cdots, (p_{lx}, p_{ly})\}, \quad |r| = l$$

3. Todas las combinaciones de rutas posibles dentro del mapa representan el dominio de cada ruta
4. La longitud de k la velocidad es fija, y cada punto se calcula con k incrementos de $\delta = \lfloor l/k \rfloor$ sobre la ruta p.

$$v = \{(p_{\delta x}, p_{\delta y}), \cdots, (p_{k\delta x}, p_{k\delta y})\}, \quad |v| = k$$

Representación
Variantes

1. Cada ruta p será construida pseudo aleatoreamente
2. La velocidad de la partícula en el inicio corresponderá a sus pivotes
3. Para evaluar la velocidad entre las partículas se actualizan los pivotes de las velocidades en vez de la posición.
4. La actualización de la posición corresponde a la generación de nuevas rutas que pasen por los pivotes antes calculados.

Representación
Función Objetivo

Función Objetivo

$$\min F(r) = lenght(r) + C(1 + lenght(r)^{\alpha}) \quad (1)$$

1. $lenght(r)$ es el largo de la ruta
2. C es el número de colisiones
3. α es el nivel de penalización por rutas cortas con colisiones

Particle Swarm Optimization

Parámetros de la velocidad de la partícula

Posición y velocidad de la partícula

$r = \{p_1, \ldots, p_l\}$
$\mathrm{v} = \{\ \mathrm{v}_1, \ldots, v_k\}$

1. La ruta r se vá generando pseudo aleatoreamente
2. Por ende, todas las rutas tienen largos distintos.
3. Sin embargo, la longitud del vector velocidad es constante dado como parámetro
4. En este caso difiere respecto a la velocidad respecto a la del PSO original.

Particle Swarm Optimization

Parámetros de la velocidad de la partícula

Actualizacíon de velocidad y posición de la partícula

$v_{i+1} = \omega v_i + \rho_g \phi_g (v(g) - v(x_i)) + \rho_p \phi_g (v(p_i) - v(r_i))$
$r_{i+1} =$ Construir ruta a partir de v_{i+1}

Los parámetros son:

1. ω que pondera el efecto de la velocidad actual
2. ϕ_g parámetro para favorecer la explotación
3. ϕ_p parámetro para favorecer la exploración
4. $\rho_g, \rho_p \sim U(0, 1)$
5. g la mejor posición global
6. p_i la mejor posición de la partícula i
7. r_i posición de la partícula i (la ruta i).

Particle Swarm Optimization

Consideraciónes respecto a las restas

Actualizacíon de velocidad y posición de la partícula

$v_{i+1} = \omega v_i + \rho_g \phi_g (v(g) - v(r_i)) + \rho_p \phi_g (v(p_i) - v(r_i))$
$r_{i+1} =$ Construir ruta a partir de v^{t+1}

Se calculan las diferencias entre los pivotes (velocidades)

1. $v(g) - v(r_i)$, la velocidad se calcula como
2. $v(p_i) - v(r_i)$

Donde la velocidad corresponde a un punto en el plano con componentes x, y

$$v_x = \omega v_{ix} + \rho_g \phi_g (v(g_x) - v(r_x)) + \rho_p \phi_g (v(p_x) - v(r_x)) \quad (2)$$
$$v_y = \omega v_{ix} + \rho_g \phi_g (v(g_y) - v(r_y)) + \rho_p \phi_g (v(p_y) - v(r_y)) \quad (3)$$

Particle Swarm Optimization

Algoritmo en Pseudo Código para la Inicialización del enjambre

Data: $S, \omega, \Phi_g, \Phi_p$
Result: Mejor particula s_i.
for $i = 1$ **to** S **do**
    $r_i =$ Inicializar la posicion de la particula aleatoreamente;
    $p_i \leftarrow r_i$;
    $v_i =$ Inicializar la velocidad con los pivotes de la particula;
    if $f(p_i) < f(g)$ **then**
        $g \leftarrow p_i$

Código C++ de la inicialización

```
void Swarm::initialize(){

}
```

Código C++ de la inicialización

```
void Swarm::initialize(){
 for(unsigned int i=0; i < this->population.size(); i++) {



 }
}
```

Código C++ de la inicialización

```
void Swarm::initialize(){
 for(unsigned int i=0; i < this->population.size(); i++) {
 Particle &p = this->population[i];
 p.initialize();
 p.evaluateFitness();
 p.initVelocity();




 }
```

Código C++ de la inicialización

```
void Swarm::initialize(){
 for(unsigned int i=0; i < this->population.size(); i++) {
 Particle &p = this->population[i];
 p.initialize();
 p.evaluateFitness();
 p.initVelocity();
 p.setBestPosition(p.getPosition());
 p.setBestVelocity(p.getVelocity());
 p.setBestFitness(p.getFitness());



 }
```

Código C++ de la inicialización

```
void Swarm::initialize(){
 for(unsigned int i=0; i < this->population.size(); i++) {
 Particle &p = this->population[i];
 p.initialize();
 p.evaluateFitness();
 p.initVelocity();
 p.setBestPosition(p.getPosition());
 p.setBestVelocity(p.getVelocity());
 p.setBestFitness(p.getFitness());
 if(p.getFitness() <= this->bestFitness){
 this->setBestFitness(p.getFitness());
 this->bestParticle = i;
 }

 }
```

Particle Swarm Optimization

Pseudo Code Algorithm para la iteración del enjambre

Data: $S, \omega, \Phi_g, \Phi_p, I$
$i = 0$ **Result**: Mejor particula g.
while $i < I$ **do**
  for $i = 1$ **to** S **do**
    $r_p, r_g \sim U(0,1)$;
    $v_i \rightarrow \omega v_i + \Phi_p r_p(v(p_i) - v(r_i)) + \Phi_g r_g(v(g) - v(r_i))$;
    $x_i \rightarrow$ Generar ruta con pivotes en v_i;
    if $f(r_i) < f(p_i)$ **then**
      $p_i \rightarrow r_i$;
      if $f(p_i) < f(g)$ **then**
        $g \rightarrow p_i$;
      end
    end
  end
  $i = i + 1$;
end
return g ;

Código C++ de la iteración

```
void Swarm::iterate() {
```

Código C++ de la iteración

```
void Swarm::iterate() {
   while (iteration < this->iterations) {

       iteration++;
   }
```

Código C++ de la iteración

```
void Swarm::iterate() {
   while (iteration < this->iterations) {
        for (unsigned int i = 0; i < this -> population.size(); i++)  {









        }

        iteration++;
    }
```

Código C++ de la iteración

```
void Swarm::iterate() {
   while (iteration < this->iterations) {
        for (unsigned int i = 0; i < this -> population.size(); i++)  {

            Particle &p = this->population[i];
             p.updateVelocity(this->population[this->getBestParticle()]
              .getBestVelocity());
             p.updatePosition();
             p.initVelocity();
             p.evaluateFitness();









        }

        iteration++;
    }
```

Código C++ de la iteración

```
void Swarm::iterate() {
   while (iteration < this->iterations) {
        for (unsigned int i = 0; i < this -> population.size(); i++)  {

            Particle &p = this->population[i];
             p.updateVelocity(this->population[this->getBestParticle()]
              .getBestVelocity());
             p.updatePosition();
             p.initVelocity();
             p.evaluateFitness();

             if (p.getFitness() < p.getBestFitness()) {
                  p.setBestPosition(p.getPosition());
                  p.setBestVelocity(p.getBestVelocity());
                  p.setBestFitness(p.getFitness());
             }




        }

        iteration++;
   }
```

Código C++ de la iteración

```
void Swarm::iterate() {
   while (iteration < this->iterations) {
        for (unsigned int i = 0; i < this -> population.size(); i++)  {

            Particle &p = this->population[i];
             p.updateVelocity(this->population[this->getBestParticle()]
              .getBestVelocity());
             p.updatePosition();
             p.initVelocity();
             p.evaluateFitness();

             if (p.getFitness() < p.getBestFitness()) {
                  p.setBestPosition(p.getPosition());
                  p.setBestVelocity(p.getBestVelocity());
                  p.setBestFitness(p.getFitness());
             }

              if (p.getFitness() < this -> bestFitness) {
                   this -> bestFitness  = p.getFitness();
                   this -> bestParticle = i;
               }
        }

        iteration++;
   }
```

Ejecución del programa

```
./psomp.bin -map maps/complexchico.dat  -iteraciones 1  -particulas 1 -pivotes 20  -omega 1 -phip 2 -phig 3 -alpha 10
```

Conclusiones

1. Las rutas aleatoreas no aseguran estabilidad en el algoritmo
2. En instancias grandes la generación de rutas aleatoreas no converge
3. Se requiere un cambio de representación para el problema.