Algoritmi euristici ed evolutivi per l'ottimizzazione

Marco Baioletti

Dipartimento di Matematica e Informatica
Università degli Studi di Perugia
Via Vanvitelli, Perugia

17 febbraio 2009 / GALN 2009

Sommario

Sommario

Algoritmi ispirati da fenomeni naturali

- La Natura offre una vastissima gamma di esempi di capacità di ottimizzazione
- Da un lato i meccanismi evolutivi naturali portano alla creazione di individui particolarmente adatti al loro habitat
- Dall'altro esseri viventi estremamente semplici sono in grado insieme di svolgere compiti complessi (intelligenza "collettiva")
- Molti modelli sono stocastici e danno luogo ad algoritmi randomizzati
- Ovviamente solo alcuni aspetti naturali sono realmente interessanti dal punto di vista algoritmico

Sommario

Cenni Storici

- Studiati in maniera approfondita per la prima volta da Holland (1975)
- Delineati a grandi linee già nell'articolo di Turing *Computing Machinery and Intelligence* (1950)
- Molto utilizzati e studiati sia dal punto di vista teorico che applicativo
- Usando una rappresentazione binaria si potrebbero applicare a qualsiasi problema di ottimizzazione
- Si basano sulla teoria darwiniana dell'evoluzione

Selezione naturale

- Principio darwiniano della selezione naturale: sopravvivenza del più adatto
- Riproduzione: formazione di nuovi individui che ereditano il patrimonio genetico dai genitori
- Mutazioni genetiche casuali: indispensabili per l'evoluzione della specie
- "Scopo": ottenere popolazioni di individui adatti all'ambiente

Ingredienti per gli algoritmi genetici
Rappresentazione

- Individui rappresentati mediante il loro patrimonio genetico (genotipo)
- Cromosoma: stringa di lunghezza fissa L
- Ogni simbolo (gene) è preso da un alfabeto finito di elementi (alleli)
- Possibile presenza di vincoli: non tutte le stringhe rappresentano cromosomi validi

Ingredienti per gli algoritmi genetici
Fitness

- Definizione di una funzione fitness f
- Individui con alta fitness sono i più adatti all'ambiente: hanno maggiori probabilità di sopravvivere e di riprodursi
- Scopo dell'AG: trovare l'individuo (o gli individui) con la fitness maggiore
- La fitness potrebbe essere diversa dalla funzione da massimizzare (essere una trasformata monotona crescente)

Ingredienti per gli algoritmi genetici
Popolazione

- La popolazione viene inizializzata in qualche modo (ad esempio con N individui generati a caso)
- Il numero di individui è mantenuto fisso
- La popolazione è manipolata per un certo numero di generazioni mediante
  - Selezione
  - Riproduzione
  - Mutazione

Ingredienti per gli algoritmi genetici
Selezione

- La selezione avviene in modo probabilistico in base alla fitness
- Si estraggono N elementi secondo la distribuzione di probabilità

$$\pi_i = \frac{f(i)}{\sum_j f(j)}$$

- Si creano $\frac{N}{2}$ coppie
- Tanti altri possibili schemi di selezione

Ingredienti per gli algoritmi genetici
Riproduzione/Crossover

- Ogni coppia di elementi produce due nuovi “figli”
- Ogni figlio eredita parte del patrimonio da ognuno dei due genitori
- Lo schema più usato e più semplice è il crossover ad un punto
- Si sceglie un punto l a caso tra 1 e L
- Un figlio prende i geni da 1 a l-1 da un genitore e il resto dall’altro
- L’altro figlio al contrario

Esempio di crossover ad un punto

- Genitore 1=ABBBCDAC Genitore 2=BAAAADDA
- Scelgo il punto 4

Genitore 1	ABBBCDAC	Genitore 2	BAAAADDA
Figlio 1	ABBAADDA	Figlio 2	BAABCDAC

Ingredienti per gli algoritmi genetici
Crossover

- Sono possibili molti altri operatori di crossover (a due punti, ecc.)
- Particolare attenzione nel caso di presenza di vincoli
- Altro schema diffuso: selezione a caso tra i due genitori

Genitore 1	ABBBCDAC	Genitore 2	BAAAADDA
Figlio 1	AABAADAA	Figlio 2	BBABCDDC

Ingredienti per gli algoritmi genetici
Mutazione

- Si sceglie a caso una o più posizioni dei cromosomi “figli” e si alterano casualmente

  Figlio 1 AABAADAA Figlio 2 BBABCDDC

  Figlio 1 ABBCADDA Figlio 2 DAABCDAC

- Particolare attenzione nel caso di presenza di vincoli

Uno schema di algoritmo genetico

```
init_population()
for i:=1 to max_gen
    evaluation()
    selection()
    crossover()
    mutation()
    replacement()
end
return best element
```

Parametri

- p_c probabilità con cui applicare il crossover
- p_m probabilità con cui applicare la mutazione
- N dimensione popolazione
- max_gen numero di generazioni
- ...

Applicazioni

- Esistono un numero impressionante di applicazioni degli algoritmi genetici
- Si possono usare sia per l'ottimizzazione di funzioni a variabile reale sia per ottimizzazione combinatorica
- Gli algoritmi genetici sono anche stati implementati per problemi decisionali (ad esempio SAT)

AG e TSP

- Un cromosoma è un elenco di $L = n$ città (senza duplicati)
- La fitness è una qualsiasi funzione monotona decrescente della lunghezza del percorso
- Il problema è come scegliere gli operatori di crossover (e mutazione)
- Dai due genitori (123456) e (532146) si potrebbero generare i due figli (123146) e (532456) che non sono permutazioni

AG e TSP
Possibili soluzioni

- “Riparare” cromosomi non validi
- “Eliminare” cromosomi non validi
- Accettare (ma penalizzare) cromosomi non validi
- Usare operatori di crossover “ad hoc”
- Cambiare rappresentazione

AG per ottimizzazioni numeriche

- Per massimizzare una funzione di una o più variabili reali (soggetta a vincoli) non conviene usare una rappresentazione binaria con crossover e mutazione "classici"
- I cromosomi sono invece numeri in virgola mobile
- Un operatore di crossover aritmetico è la media pesata dei genitori
- Un operatore di mutazione è la mutazione non–uniforme

$$x'_k = x_k + \Delta(t, x_k)$$

ove Δ è un numero casuale che
  - garantisce che x'_k cada nell'intervallo di definizione
  - ha maggiore probabilità di assumere valori vicini allo 0

Giustificazioni teoriche

- Esistono giustificazioni teoriche per spiegare il buon comportamento degli AG
- La più semplice giustificazione usa il cosiddetto Teorema dello schema
- Uno schema per cromosomi binari è una stringa di $0, 1, \star$ ($\star$ rappresenta 0 o 1)
- Il teorema dice che *buoni schemi tendono ad essere sempre più presenti nella popolazione*

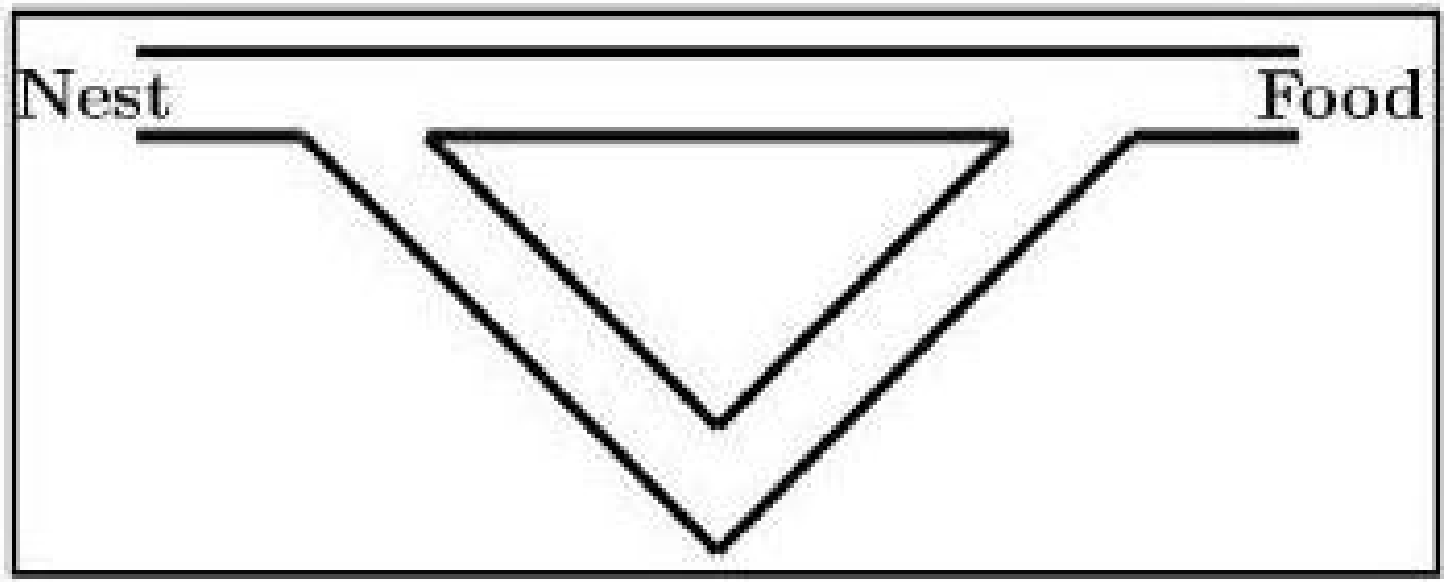
Sommario

Cenni storici

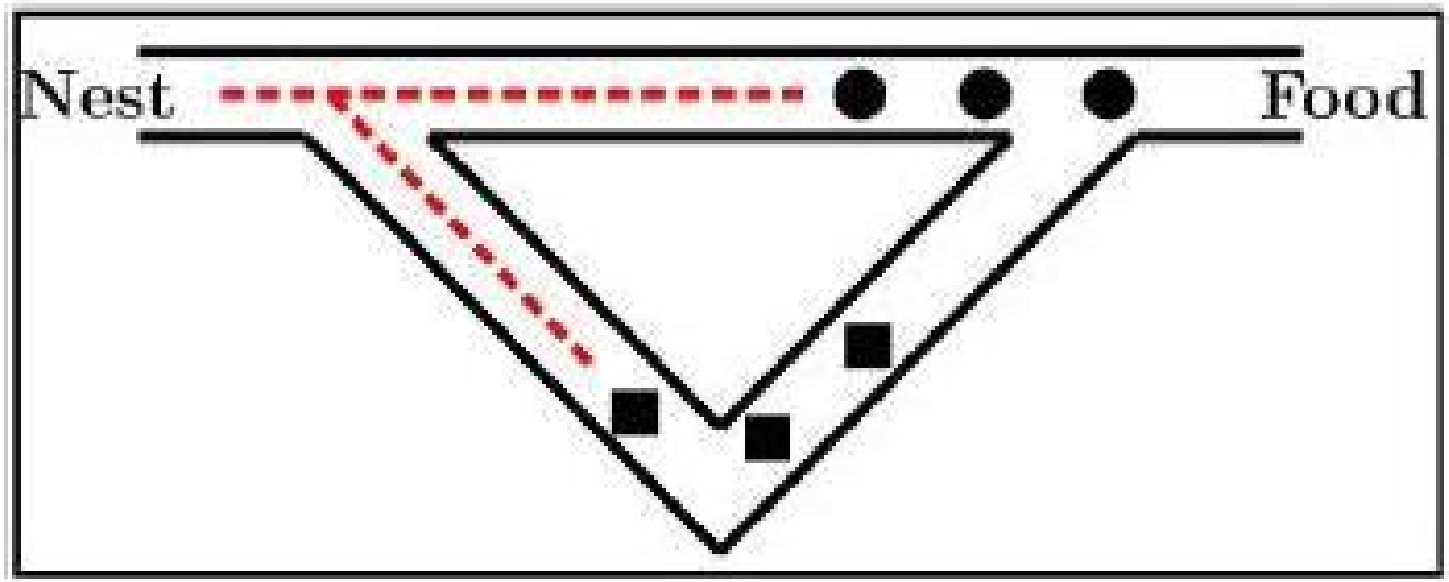
- Ideata da Dorigo nei primi anni ’90 a partire da studi sul comportamento di colonie di formiche “reali”
- Utilizzato in molti problemi di ottimizzazione combinatorica
- Si basa sul concetto di stigmergia

Stigmergia

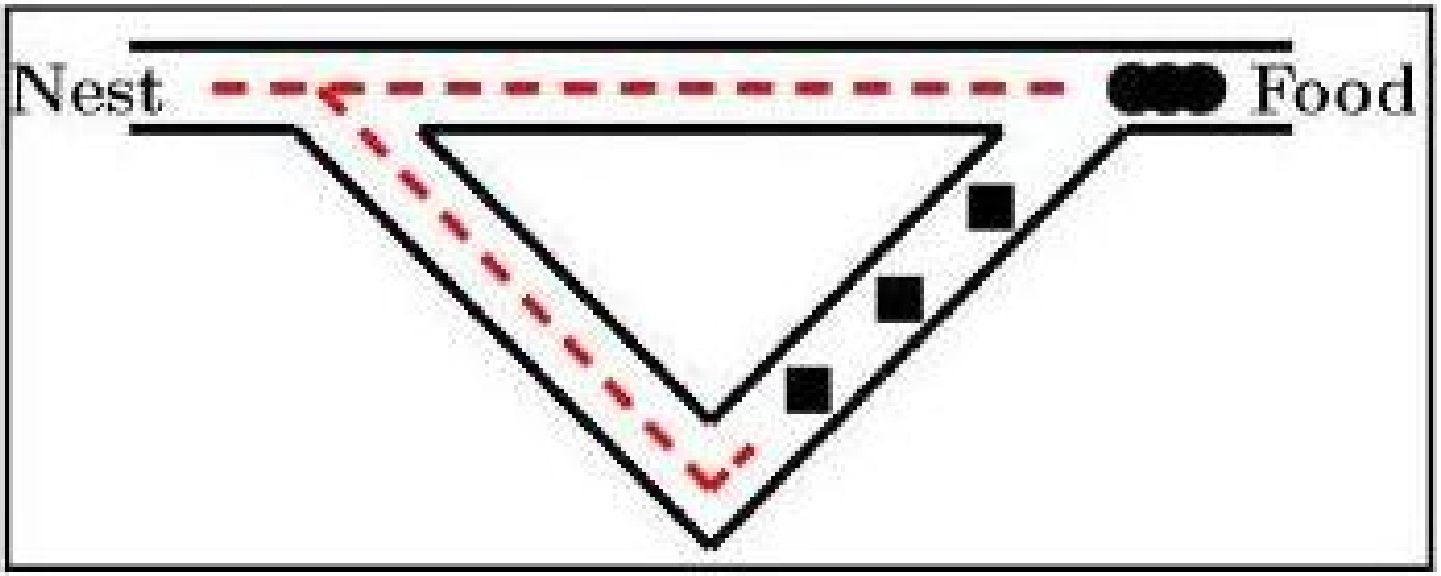
- Le formiche quando si muovono rilasciano sul terreno il feromone
- Nella ricerca del cibo tendono a seguire percorsi con maggior quantità di feromone
- Avendo trovato il cibo, quando rientrano al nido rilasciano una quantità maggiore di feromone
- Il feromone evapora



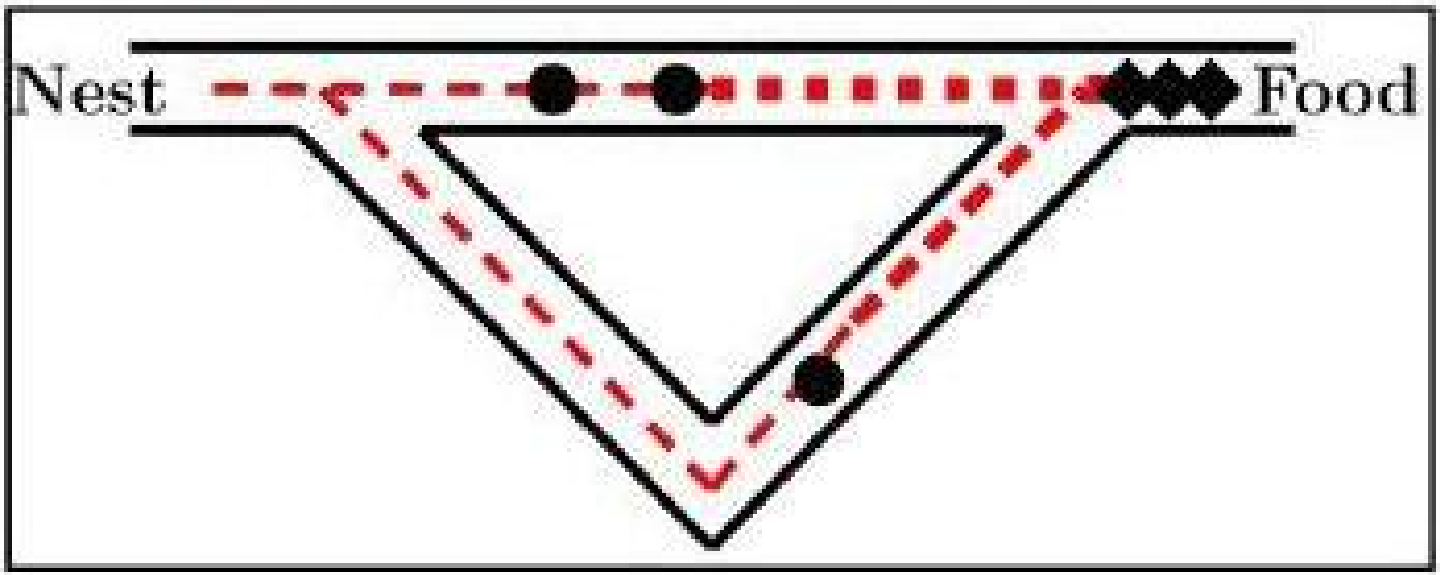
Feromone assente



All’inizio le formiche si muovono a caso



Quelle che scelgono il percorso più breve arrivano (ovviamente) prima al cibo



Tornando indietro rinforzano la traccia che sarà seguita da altre formiche

Formiche artificiali
Grafo

- Scopo “ideale”: trovare il massimo della funzione $f : S \rightarrow \mathcal{R}$
- Gli elementi di S sono assimilabili a percorsi in un grafo (N, A)
- N è l’insieme degli stati
- A è l’insieme delle transizioni ammissibili
- (opzionale) c_{rs} è il costo della transizione $r \rightarrow s$
- Ω è l’insieme dei vincoli del problema (restrizioni sui possibili percorsi)

Formiche artificiali
Formiche e terreno

- Ogni "formica"
  - parte da uno stato iniziale
  - ha una condizione di terminazione del percorso
  - crea un elemento di S un componente alla volta
  - ha una memoria in cui tiene traccia del proprio percorso
- Colonia di m formiche
- Il "terreno" consiste in una funzione $\tau : A \rightarrow \mathcal{R}$
- τ_{rs} è la quantità di feromone depositata sulla transizione $r \rightarrow s$

Formiche artificiali
Movimento delle formiche

- La formica si muove sul grafo in maniera probabilistica

$$p_{s|r} = \frac{\tau_{rs}^{\alpha} \eta_{rs}^{\beta}}{\sum_z \tau_{rz}^{\alpha} \eta_{rz}^{\beta}}$$

- η_{rs} è una funzione euristica: indica quanto è "promettente" passare da r a s
- α e β sono parametri di importanza

Formiche artificiali
Aggiornamento del feromone

Il feromone può essere aggiornato

- "step–by–step": durante la costruzione della soluzione
- in maniera "differita": una volta costruita la soluzione $\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau_{rs}$ (la quantità è proporzionale alla bontà della soluzione)
- dal "demone": si aggiornano solo i feromoni delle componenti che partecipano alle migliori soluzioni

Il feromone evapora: $\tau_{rs} \leftarrow (1 - \rho)\tau_{rs}$

Descrizione di un semplice algoritmo ACO

```
init_phero()
while not criterion
        for i = 1 to m
            init_ant(i)
            while not terminate(i)
                    choose_component(i)
                    update_state(i)
                    update_phero_step(i)
            end
            update_phero_delayed(i)
        end
        update_phero_daemon()
end
```

Applicazioni

- TSP
- QAP
- Job–shop scheduling
- AntNet
- Vehicle routing
- Colorazione di grafi
- ...

ACO e TSP

- Componenti: archi tra città
- Vincolo: ogni formica deve visitare tutte le città una sola volta
- $\eta_{ij} = \frac{1}{d_{ij}}$ (d_{ij} distanza euclidea)
- Aggiornamento del feromone $\Delta_{ij}^{k} = \frac{1}{L^{+}}$

Varianti

- Ant System: prime versioni di ACO, aggiornamento solo differito
- Ant Colony System:
  - scelta del componente: con probabilità q quello che massimizza $\tau_{rs}^{\alpha}\eta_{rs}^{\beta}$, altrimenti a caso con prob. proporzionale a $\tau_{rs}^{\alpha}\eta_{rs}^{\beta}$
  - aggiornamento del feromone mediante demone (migliori soluzioni)
  - aggiornamento del feromone step–by–step con diminuzione (per favorire l'esplorazione)
- Max–Min Ant System: feromone limitato in $[\tau_{min}, \tau_{max}]$ e aggiornato mediante demone (global best e iteration best)
- Hypercube Framework: feromone limitato in $[0, 1]$

Sommario

Cenni storici

- Ideata da Kennedy e Eberhart nel 1995
- Si ispira al comportamento degli stormi di uccelli, banchi di pesci, sciami di insetti
- Modelli inizialmente studiati in ambiti diversi dall'ottimizzazione
  - Computer Graphics
  - Modelli comportamentali di gruppi di animali
- Adatto ad ottimizzazione numerica

Sciami

- Il comportamento di uno stormo è molto complesso
- Coordinamento senza leader, subentrano fattori di comunicazione e di imitazione
- Un modello semplice usa le seguente regole
  1. Separazione: evitare collisione con i vicini (repulsione a corto raggio)
  2. Allineamento: puntare verso la posizione media dello stormo
  3. Coesione: puntare verso la posizione media dei vicini (attrazione a lungo raggio)

Particelle

- La particella i al tempo t assume una posizione $X_i^t \in \mathcal{R}^D$ e una velocità $V_i^t \in \mathcal{R}^D$
- Esiste una funzione di fitness $f : S \rightarrow \mathcal{R}$, ove $S \subset \mathcal{R}^D$, che valuta la bontà della posizione delle particelle
- Ogni particella ha una memoria in cui tiene traccia della migliore posizione raggiunta
- Ogni particella conosce la migliore posizione correntemente assunta dai suoi vicini (comunicazione tra particelle)
- Perciò è in grado di ricordare la migliore posizione raggiunta dai suoi vicini

Aggiornamento velocità e posizione

- P_i^t migliore posizione raggiunta dalla particella i
- B_i^t migliore posizione raggiunta da particelle dai vicini di i
- velocità

$$V_{ij}^{t+1} = V_{ij}^t + \phi_1 r_1 (P_{ij}^t - X_{ij}^t) + \phi_2 r_2 (B_{ij}^t - X_{ij}^t)$$

- posizione

$$X_{ij}^{t+1} = X_{ij}^t + V_{ij}^{t+1}$$

Aggiornamento velocità e posizione
Interpretazione

- Il termine $(P_{ij}^t - X_{ij}^t)$ è chiamato termine cognitivo
- Il termine $(B_{ij}^t - X_{ij}^t)$ è chiamato termine sociale
- ϕ_1, ϕ_2 sono parametri di importanza (in letteratura è diffuso l'uso di $\phi_1 = \phi_2 = 2$)
- r_1, r_2 sono numeri casuali in $[0, 1]$
- le velocità sono vincolate nell'intervallo $[-V_{max}, V_{max}]$

Vicinato

- Esistono vari metodi per definire quando due particelle possono comunicare
- Si forma un grafo di “vicinanza”
- Si distinguono topologie statiche e dinamiche
- Tra le topologie statiche troviamo
  - Global best: grafo completo
  - Local best: anello (grado $K = 2$)
  - grafi con grado uniforme $K > 2$

Uno schema di algoritmo PSO

```
init_swarm()
for t:= 1 to max_gen
    update_best_positions()
    for i:=1 to m
        update_velocity(i)
        update_position(i)
    end
end
return best element
```

Varianti

- Fattore di inerzia su V_{ij}^t

$$V_{ij}^{t+1} = \omega V_{ij}^t + \phi_1 r_1 (P_{ij}^t - X_{ij}^t) + \phi_2 r_2 (B_{ij}^t - X_{ij}^t)$$

- Coefficienti di costrizione

$$V_{ij}^{t+1} = \chi (V_{ij}^t + \phi_1 r_1 (P_{ij}^t - X_{ij}^t) + \phi_2 r_2 (B_{ij}^t - X_{ij}^t))$$

ove

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \quad \phi = \phi_1 + \phi_2 > 4$$

- Fully Informed Particle Swarm: il fattore sociale è la media delle posizioni di tutti i vicini

Applicazioni numeriche

- L'ambito naturale di applicazione del PSO è l'ottimizzazione numerica
- E' stato applicato in svariati problemi, dall'apprendimento delle reti neurali, all'analisi di immagini e di video, dalla progettazione di reti elettriche, al controllo automatico, ...
- Esistono versioni in grado di lavorare anche con problemi multi–obiettivo

Ottimizzazione combinatorica

- E' possibile discretizzare PSO
- Dalla velocità V_{ij}^t è possibile calcolare un valore $S(V_{ij}^t)$ in $[0, 1]$ che è usato come probabilità che X_{ij}^t sia 1 (anzichè 0): PSO binario
- In maniera analoga è possibile generare posizioni appartenenti ad insiemi discreti: PSO multivalore

Sommario

Altri algoritmi euristici ed evolutivi

- Programmazione genetica
- Strategie evolutive
- Algoritmi memetici
- Simulated annealing
- Metodi di ricerca locale
- ...

Conclusioni

- Sono stati presentati tre metodi euristici per l'ottimizzazione
  - Algoritmi genetici
  - Ant Colony Optimization
  - Particle Swarm Optimization
- Si tratta di metodi che non garantiscono l'ottimalità delle soluzioni trovate
- Producono (spesso) ottime soluzioni
- Sono (sovente) (molto) più efficienti dei metodi classici

Bibliografia
Algoritmi genetici

- Z. Michalewicz, 1999, Genetic Algorithms + Data Structures = Evolution Programs, Springer–Verlag.
- D. E. Goldberg, 1989, Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer Academic Publishers, Boston, MA.
- D. Whitley, A genetic algorithm tutorial. Statistics and Computing 4, 65-85, 1994

Bibliografia
Ant Colony Optimization

- M. Dorigo, T. Stützle, 2004. Ant Colony Optimization, MIT Press
- M. Dorigo, C. Blum, Ant colony optimization theory: A survey. Theoretical Computer Science Volume 344, Issues 2-3, Pages 243–278, 2005
- O. Cordon, F. Herrera, T. Stützle, A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. Mathware and Soft Computing, 9(2–3):141–175, 2002

Bibliografia
Particle Swarm Optimization

- M. Clerc. Particle Swarm Optimization. ISTE, 2006
- J. Kennedy and R. C. Eberhart. Swarm Intelligence. Morgan Kaufmann. 2001
- R. Poli, J. Kennedy, T. Blackwell, Particle Swarm Optimisation: an overview, Invited review paper for the first issue of the Swarm Intelligence Journal, 1(1):33–57, June, 2007