Learning Coordination Strategies in Computer Go

Myriam Abramson

Coordination strategies

- Coordination is an essential trait of problem solving
- Complex problems can be reformulated as coordination problems
- Go is the quintessential example of coordination strategy
  - Stones receive their meaning from other stones
  - There is no intrinsic value in a move by itself

Reinforcement Learning

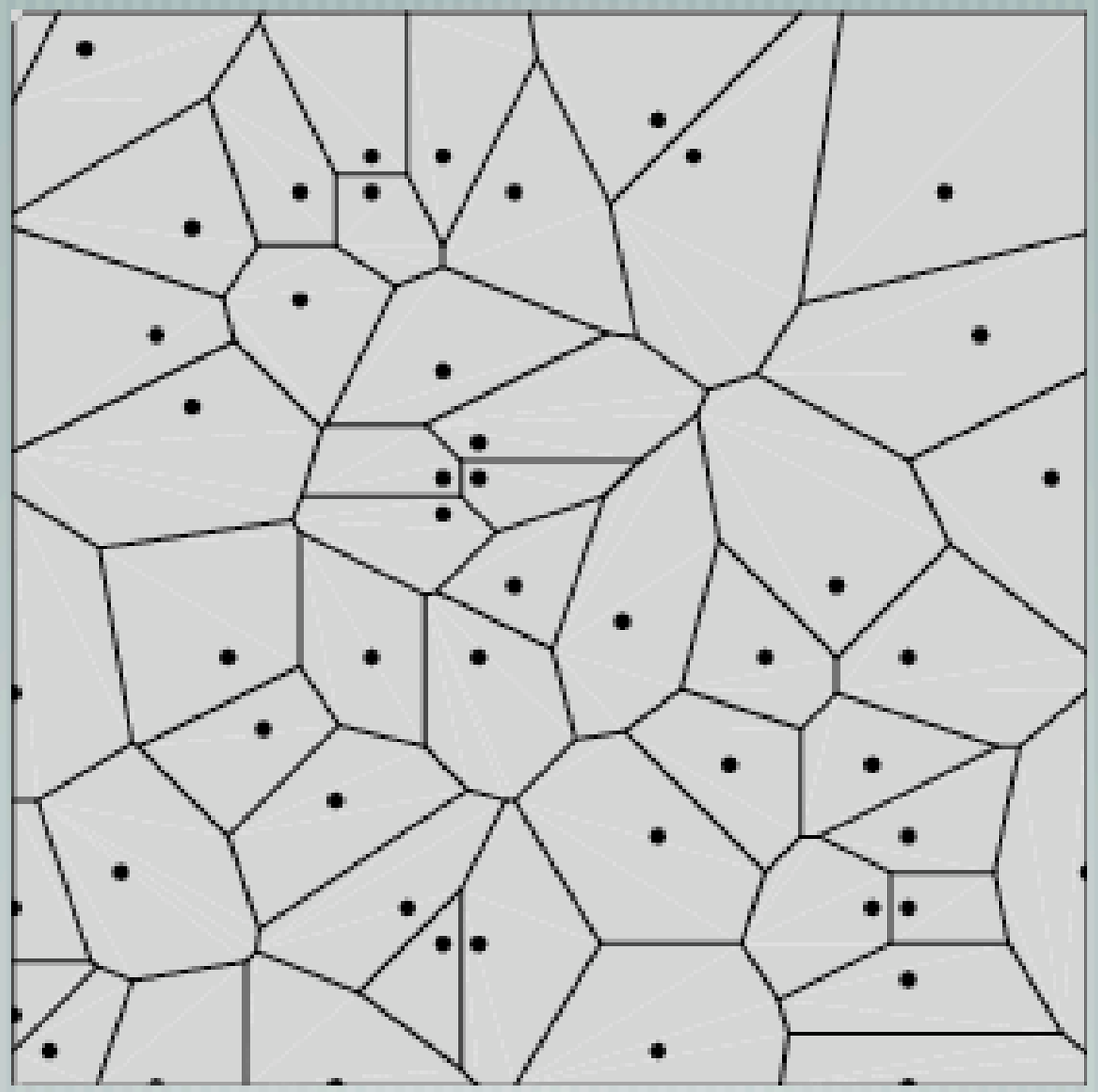
- A good learning methodology for coordination strategies
- Learn the value of a state from the value of successor states

- For example, temporal difference learning updates:
  - $V(s) = V(s) + \alpha[r + V(s') - V(s)]$ TD(0)
  - $Q(s,a) = Q(s,a) + \alpha[r + Q(s',a') - Q(s,a)]$ Sarsa

Vector quantisation

- Unsupervised learning scheme to find underlying structure
- Knowledge is represented by a set of prototypes
- Competitive learning - nearest prototype wins
- Move winning prototype closer to actual input
- Self-organising map:
  - Update neighbourhood of winner towards input

Example prototypes (2D)



SLVQ

- Sarsa + Learning Vector Quantisation
- Store (m, a, Q, α) with each prototype
- Update action-values Q by Sarsa algorithm
- Update prototypes m using vector quantisation rule

$$m_t = \arg\max_m \ similarity(s_t, m)$$

$$\Delta Q(m_t, a) = \alpha_{m_t} \left[r_t + \gamma Q(m', a') - Q(m_t, a) \right]$$

$$m\left(t+1\right) = m\left(t\right) + \Delta Q(m_t, a) \left[s_t - m_t \right]$$

Applying SLVQ to Computer Go

- State is represented by vector of influence function values
- Fuzzy contrast model used to measure similarity
- Initial prototypes selected from GnuGo games
- SLVQ + softmax exploration

Influence function

- Influence(B) = +1.0
- Influence(W) = -1.0
- Influence(E) = 1/8 $\sum_i$ Influence(i) for all 8-connected i
- Repeat until equilibrium
- Equivalent to solving DP for random 8-connected walk

Similarity measure

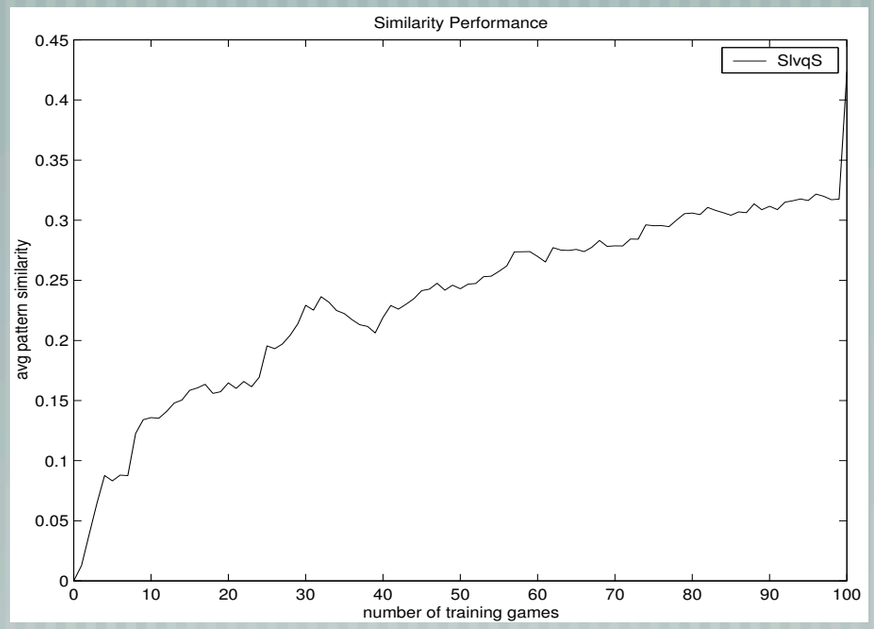


Figure 5.2: Similarity of the codebook vectors

$$Closeness(P,R) = \sum_i \sum_j min\left\{p_{ij}, r_{ij}\right\}$$

$$Contrast(P,R) = \sum_i \sum_j max\left\{p_{ij} - r_{ij}, 0\right\}$$

$$S\left(P,R\right) = Closeness(P,R) - \alpha Contrast(P,R) - \beta Contrast(R,P)$$

Opponents

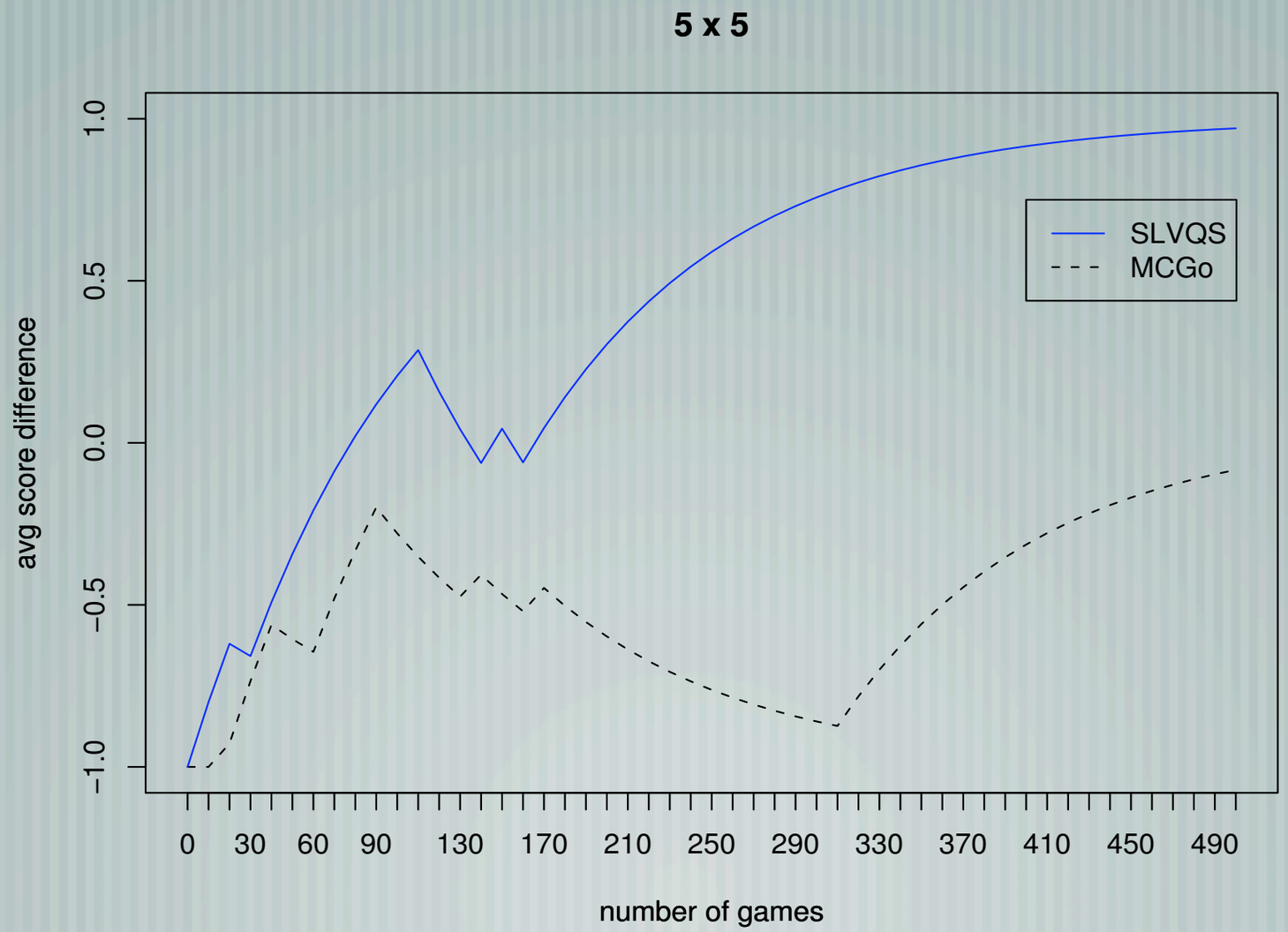
- Random
- Nearest Neighbour: Play towards nearest prototype
- Heuristic: 1-ply search over sum of influence values
- Minimax: 2-ply search over sum of influence values
- Wally
- Monte Carlo Go: don't understand

Opponent comparison

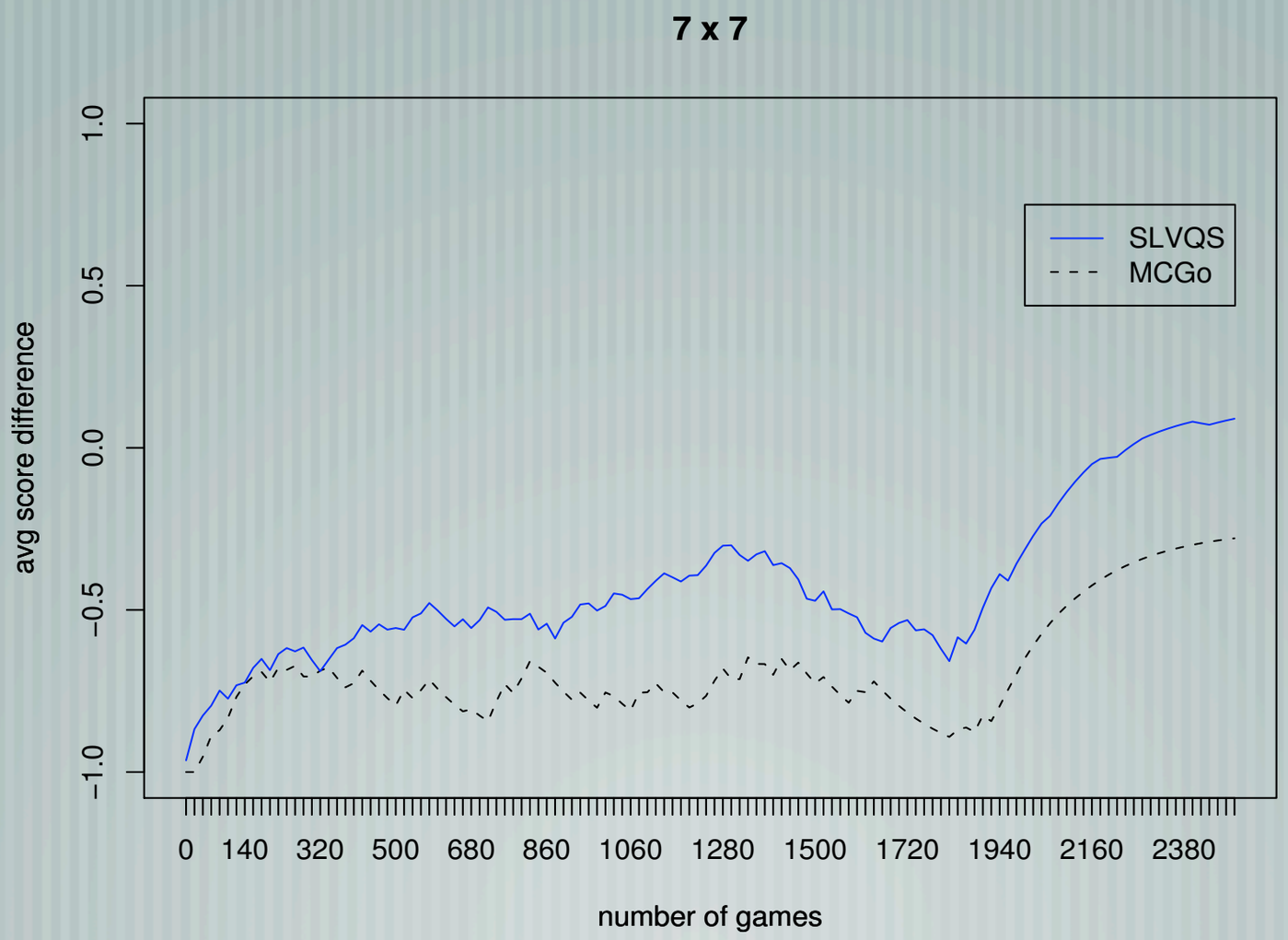
7x7 results averaged over 100 games

black\white	NN	Random	Heuristic	Minimax	Wally
NN	-1.0	1.0	-0.18	-0.10	-0.57
Random	-1.0	0.75	-0.75	-0.74	-0.79
Heuristic	0.13	1.0	0.47	-1.0	-0.62
Minimax	1.0	1.0	1.0	-0.05	-0.23
Wally	1.0	1.0	-0.09	0.48	0.55

Results .v. Minimax (5x5)



Results .v. Wally (7x7)



Extensions

- Continuous valued version of SLVQ
  - e.g. for problems like Mountain Car
- Tabu search for efficient exploration
- Learning by parts

Learning by parts

- Decompose the board into overlapping squares
- Train an agent for each square, using SLVQ
- Overall policy is selected stochastically using Gibbs distribution over each agent's evaluation.
- Performs slightly better than whole-board SLVQ

Conclusions

- Tabular Sarsa is a simple, effective online learning algorithm
- Nearest neighbour representation allows Sarsa to be applied effectively in Go
- Selecting prototypes from real games is a good idea
- Twiddling the prototypes by SLVQ may possibly perhaps improve performance

Demo

Who wants to play??