# Learning Action Strategies for Planning Domains using Genetic Programming

John Levine and David Humphreys

johnl@inf.ed.ac.uk

http://www.aiai.ed.ac.uk/~johnl

Centre for Intelligent Systems and their Applications
School of Informatics, University of Edinburgh

# Introduction

- Domain-independent planning is extremely hard
- Domain-specific control knowledge can help
- Control knowledge can be learnt from examples
- Control knowledge can be expressed as rules
- A collection of ordered rules is called a policy
- A good policy can solve planning problems without searching
- Our aim is to find policies for planning domains using genetic programming
- Our system for doing this is called L2Plan (Learn to Plan)

# Previous Results

- L2Act (Khardon, 1999):

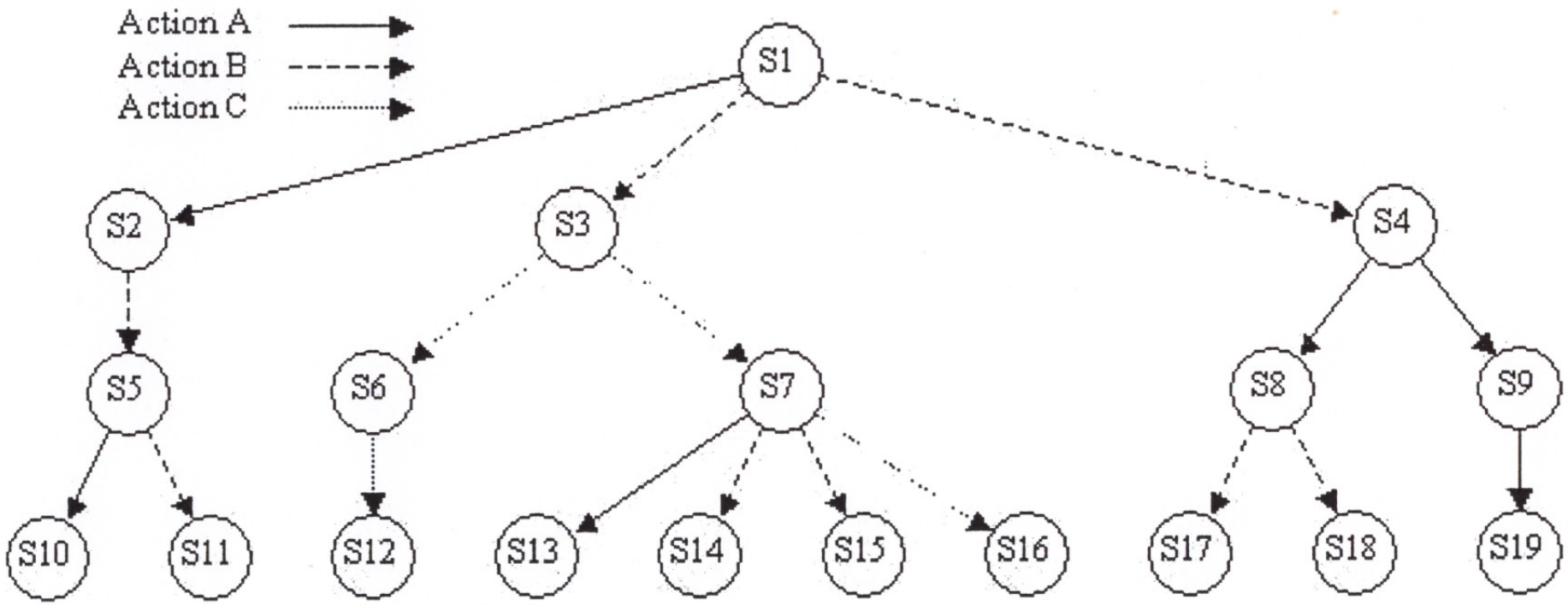
  "roughly 80 percent of the problems of the same size [as the training examples] (8 blocks), and 60 percent of the larger problems (with 20 blocks)."

- EvoCK (Aler, Borrajo and Isasi, 2001):

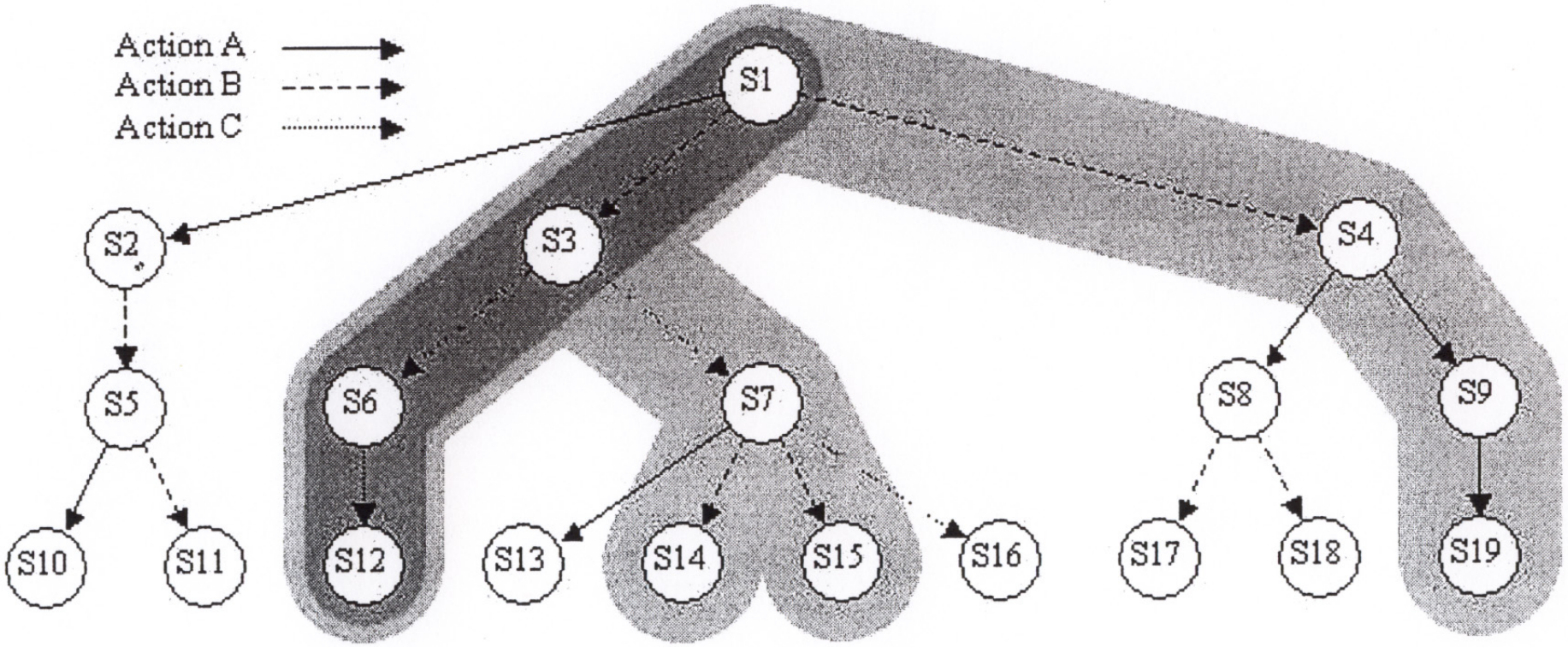
| Problems Solved | | | |
|---|---|---|---|
| 5 goals | 10 goals | 20 goals | 50 goals |
| 95% | 85% | 73% | 38% |

- Incomplete generalisation from examples

# Planning as a Tree Search



# Policy Restricted Planning



# Example Planning Domain

```
(define (domain blocksworld)
(:predicates (clear ?x)
             (on-table ?x)
             (on ?x ?y))
(:action move-block-to-block
  :parameters (?bm ?bf ?bt)
  :precondition (and (clear ?bm) (clear ?bt) (on ?bm ?bf))
  :effect (and (not (clear ?bt)) (not (on ?bm ?bf))
               (on ?bm ?bt) (clear ?bf)))
(:action move-block-to-table
  :parameters (?bm ?bf)
  :precondition (and (clear ?bm) (on ?bm ?bf))
  :effect (and (not (on ?bm ?bf)) (on-table ?bm) (clear ?bf)))
(:action move-table-to-block
  :parameters (?bm ?bt)
  :precondition (and (clear ?bm) (clear ?bt) (on-table ?bm))
  :effect (and (not (clear ?bt)) (not (on-table ?bm))
               (on ?bm ?bt))))
```

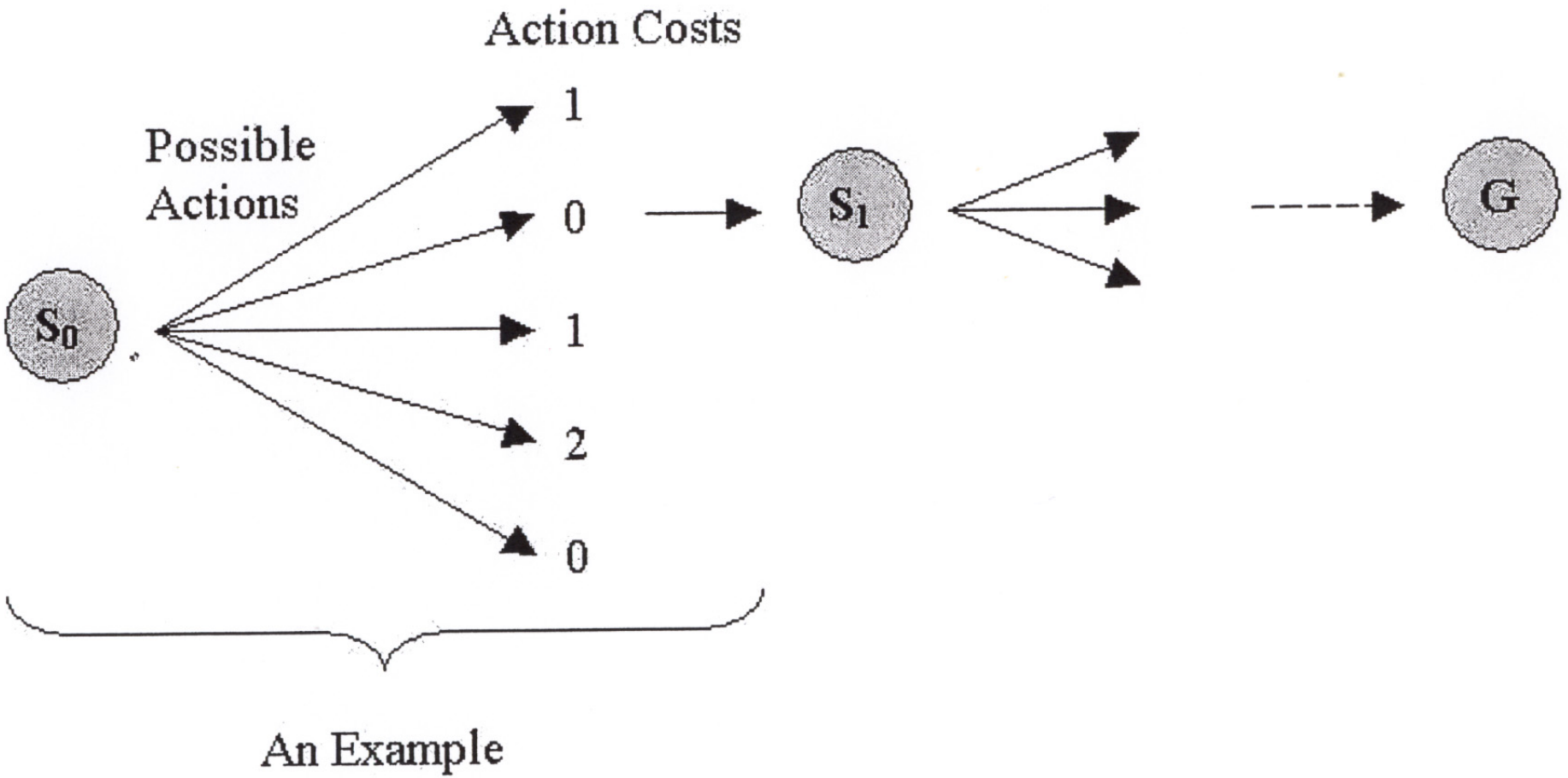
# What is a Policy?

```
(define (policy blocks1)
(:rule make_well_placed_block_1
  :condition (and (on ?bm ?bf) (wp ?bt))
  :goalCondition (and (on ?bm ?bt))
  :action move-block-to-block ?bm ?bf ?bt)
(:rule make_well_placed_block_2
  :condition (and (wp ?bt))
  :goalCondition (and (on ?bm ?bt))
  :action move-table-to-block ?bm ?bt)
(:rule make_well_placed_block_3
  :condition (and (on ?bm ?bf))
  :goalCondition (and (on-table ?bm))
  :action move-block-to-table ?bm ?bf)
(:rule move_non_wp_block_to_table
  :condition (and (on ?bm ?bf) (not (wp ?bm)))
  :goalCondition (and )
  :action move-block-to-table ?bm ?bf))
```

# Example Problem

```
(define (problem large-a)
  (:domain blocksworld)
  (:length 6)
  (:objects 1 2 3 4 5 6 7 8 9)
  (:init (on 3 2) (on 2 1) (on-table 1)
         (on 5 4) (on-table 4) (on 9 8)
         (on 8 7) (on 7 6) (on-table 6)
         (clear 3) (clear 5) (clear 9))
  (:goal (and (on 1 5) (on-table 5) (on 8 9)
              (on 9 4) (on-table 4) (on 2 3)
              (on 3 7) (on 7 6) (on-table 6)
              (clear 1) (clear 8) (clear 2))))
```

# Generating Training Examples



# L2Plan: Implementation

- Population of randomly generated policies
- Run each policy on training examples
- Fitness function:

$$F(p_i) = \frac{1}{1 + \left(\sum_{j=1}^{n} C(p_i, e_j)\right)/n}$$

- Tournament selection
- Generational algorithm with elistism
- Various crossover and mutation operators
- Mutation-based local search

# Genetic Operators

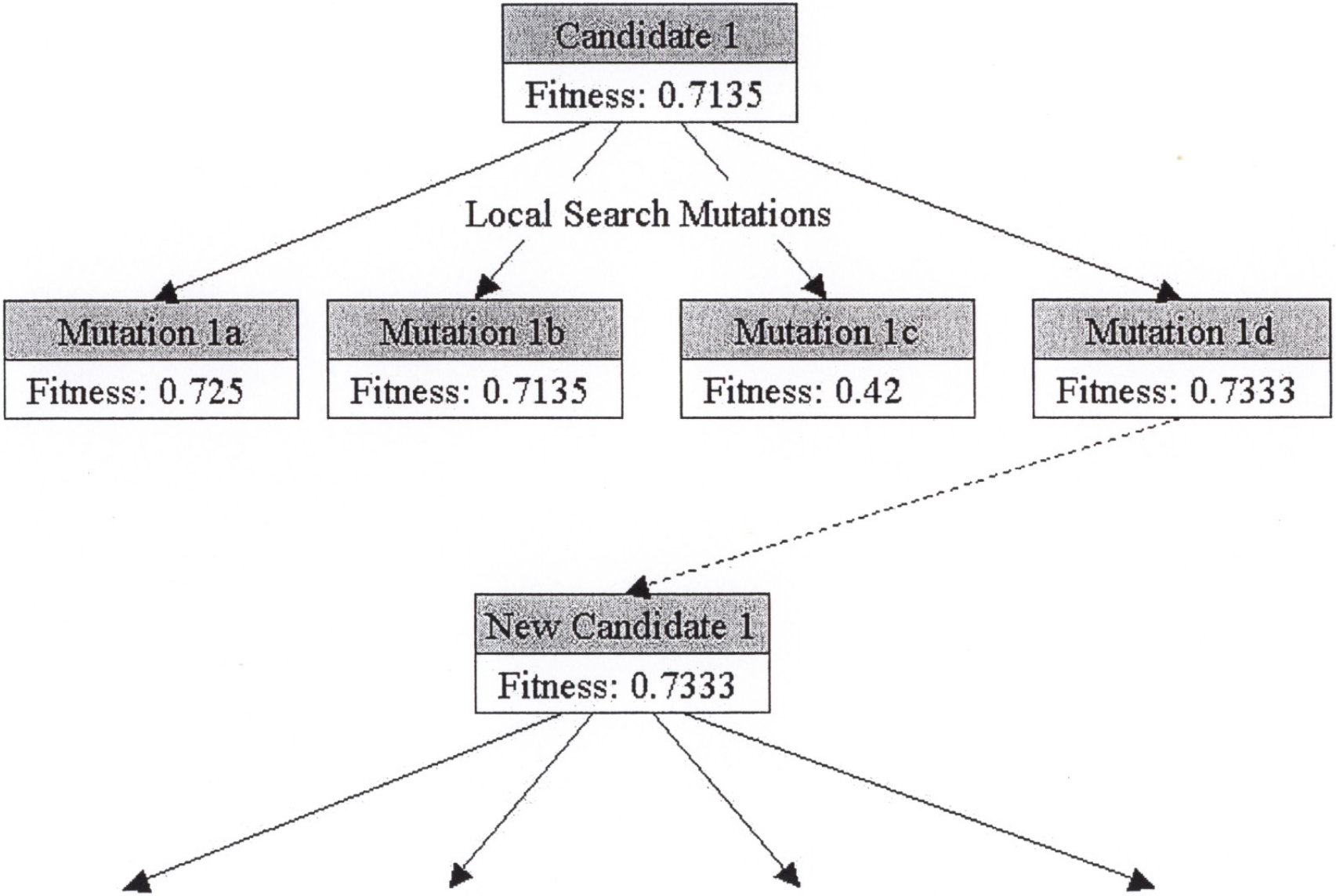
Crossover operators:

- Single point rule level crossover
- Single rule swap crossover
- Similar action rule crossover

Mutation operators:

- Rule addition mutation
- Rule deletion mutation
- Rule swap mutation
- Rule condition mutation

# Local Search



# Evaluating the Policies

Two forms of policy restricted planning:

- breadth-first planning
- first-action planning

Metrics tracked by the policy tester:

- number of test problems solved
- number of test problems solved optimally (fewest actions)
- number of extra steps taken on average
- number of states examined during the search

# Blocks World Results

- We used 30 5-block training problems
- This gave us 135 training examples

| | fi rst-action planning | | | | breadth-fi rst planning | | | |
|---|---|---|---|---|---|---|---|---|
| | Sol | Opt | Extra | Nodes | Sol | Opt | Extra | Nodes |
| 5 blocks | 100 | 100 | 0.00 | 5.76 | 100 | 100 | 0.00 | 7.14 |
| 10 blocks | 100 | 86 | 0.15 | 12.51 | 100 | 94 | 0.07 | 17.60 |
| 15 blocks | 100 | 65 | 0.57 | 21.02 | 100 | 88 | 0.15 | 41.12 |
| 20 blocks | 100 | 46 | 0.91 | 29.63 | 100 | 84 | 0.21 | 99.02 |
| hand-coded | 100 | 34 | 1.26 | 29.98 | 100 | 100 | 0.00 | 197.42 |

- Better results than L2Act and EvoCK
- Learnt policy out-performs hand-coded policy under fi rst-action planning

# Briefcase Domain Results

- We used 30 2-object, 5-city training problems
- This gave us 167 training examples

| | fi rst-action planning | | | | breadth-fi rst planning | | | |
|---|---|---|---|---|---|---|---|---|
| | Sol | Opt | Extra | Nodes | Sol | Opt | Extra | Nodes |
| 2 objects, 5 cities | 100 | 95 | 0.05 | 6.05 | 100 | 100 | 0.00 | 9.37 |
| 2 objects, 10 cities | 100 | 96 | 0.04 | 6.87 | 100 | 100 | 0.00 | 11.97 |
| 4 objects, 5 cities | 100 | 80 | 0.20 | 10.38 | 100 | 100 | 0.00 | 27.98 |
| 4 objects, 10 cities | 100 | 76 | 0.25 | 12.91 | 100 | 100 | 0.00 | 62.04 |
| hand-coded | 100 | 74 | 0.28 | 12.94 | 100 | 100 | 0.00 | 68.76 |

- Very good performance obtained
- Slightly out-performs the hand-coded policy

# Conclusions and Further Work

Conclusions from this study:

- Successful demonstration that GP can learn policies
- Comparable performance with systems using human-coded knowledge for these two simple domains

Further work:

- Application to more domains required
- Support for typed version of PDDL required
- Investigation of the use of description logic in the rules
- Further planning strategies beyond first-action or breadth-first: e.g. repeated first-action planning with reinforcement of better plans