Universidade de São Paulo
Mestrado em Ciência da Computação

Instituto de Matemática e Estatística
Disciplina Planejamento

Modelo de ações para o problema ROADEF sobre Linha de Montagem de Carros

Karina Valdivia Delgado
kvd@ime.usp.br

13 de Outubro de 2003

1 Problema

Ler a especificação do problema da competição Roadef 2005 e construir um modelo preliminar das ações necessárias para resolver o problema proposto como um problema de planejamento de ações.

2 Descripção da solução

Decidimos criar uma ação $montar$ para cada tipo de carro, por exemplo se os carros podem ter a característica 1 e a característica 2 temos que ter uma ação montar característica 1,montar característica 2, montar característica 1 e 2 e montar sim características.
Em cada um das ações perguntamos se a cor do carro é a mesma e excedeu do limite então adiciono 1 a $wash$ que é contador de limpezas feitas e inicializo o contador $carmountcolor$.
Se não se excedeu o limite só incremento $carmouncolor$.
Se a cor do carro é diferente então temos que incrementar $wash$ e a cor atual é a cor do carro que estamos montando.
Também se tem que fazer a verificação se existe penalidade ou não pelo montagem do componente i.
No estado inicial os carros não estão montados e no estado final os carros tem que estar montados todos.
As funçoes usadas são as mostradas na tabela 1.
Para ter os valores iniciais de P1 e P2 temos que ler o arquivo de restrições de ratio.
Do arquivo de vehiculos leemos o identificador do carro, a cor de carro, e as características que precisa e também aquelas que não precisa.

Função	Descripção
(actualcolor)	Tem a cor do carro anterior (a cor atual)
(carmountcolor)	Quantidade de carros pintados com a mesma cor sequencialmente.
(spraygunswash)	Quantidade predefinida de cada quanto se deve fazer a limpeza das pistolas.
(wash)	Quantidade de trocas de cor feitas
(comp1rest)	Quantidade de carros que deven ser producido sim o componente 1 para que um carro possa ser producidos com o componente 1 sim penalidade.
(penaltycomp1)	Quantidade de penalidades do componente1
(P1)	É o P da relação 1/P do componente 1.
(comp2rest)	Quantidade de carros que deven ser producido sim o componente 2 para que um carro possa ser producidos com o componente 2 sim penalidade.
(penaltycomp2)	Quantidade de penalidades do componente2
(P2)	É o P da relação 1/P do componente 2
...	...

Tabela 1: Descripção das funcões

3 Restrições

A soluçao proposta tem as seguintes restrições:

- A produção do dia anterior não foi tomada em conta
- Só foi considerado ratio 1/P para os componentes
- A penalidade para cada componente deve ser dada.
- O númeoro de trocas de cor tem uma penalidade de 100
- Temos que criar uma açào por cada combinação possível de componentes, o qual fica exponencial

4 Exemplo

Estado inicial:
spraygunswash=20
P1=5
P2=8
A penalidade do componente 1 é 1000, a penalidade do componente2 e 1 e a penalidade por cada troca de cor é 100
Estado final:
Todos os carros montados.

Carro	Cor	Componente1	Componente2
carro997	Green	Sim	Não
carro998	Green	Não	Sim
carro999	Red	Não	Não
carro1000	Red	Não	Sim

Tabela 2: Exemplo

```
( define ( domain cars )
        ( :requirements :conditional-effect :fluents: typing : equality:
                                                    negative-preconditions)
        ( :types car color charact)
        ( :predicates
                (carneedcolor ?x - car  ?y -color)
                (ismount ?x - car)
                (carneedcharac ?i - charact ?x - car)
        )
        ( :function
                (actualcolor)
                (carmountcolor)
                (spraygunswash)
                (wash)

                (comp1rest)
                (penaltycomp1)
                (P1)

                (comp2rest)
                (penaltycomp2)
                (P2)

        )
        (:action mountwhitoutcomp
                :parameters (?i1 - charact  ?i2 -charact  ?x - car  ?c - color )
                :precondition ( and
                        (carneedcolor ?x  ?c)
                        (not(carneedcharac ?i1 ?x))
                        (not(carneedcharac ?i2 ?x))
                        (not(ismount ?x - car)))
                )
                :effect ( and
                        (not(carneedcolor ?x  ?c))
                        (ismount ?x - car)
                        (decrease (comp1rest)(1))
                        (decrease (comp2rest)(1))

                        when(and
                        (= ?c ?actualcolor)
                        (= carmountcolor (- (spraygunswash) 1)))

                        (and
                        (assign (carmountcolor) (0))
```

```
                (increase (wash)(1)))

                when (and
                (= ?c ?actualcolor)
                (< carmountcolor (- (spraygunswash) 1)))
                (increase (carmountcolor)(1))

                when
                (not(= ?c ?actualcolor))
                (assign (actualcolor) (c))
                (increase (wash)(1)))
        )
)
(:action mountcomp1
        :parameters (?i1 - charact  ?i2 -charact  ?x - car  ?c - color )
        :precondition ( and
                (carneedcolor ?x  ?c)
                (carneedcharac ?i1 ?x)
                (not(carneedcharac ?i2 ?x))
                (not(ismount ?x - car)))
        )
        :effect ( and
                (not(carneedcolor ?x  ?c))
                ( not (carneedcharac ?i1 ?x) )
                (ismount ?x - car)

                (decrease (comp2rest)(1))
                when
                (> comp1rest 0)
                (increase (penaltycomp1)(1)))

                (assign comp1rest (- (P1) 1))

                when(and
                (= ?c ?actualcolor)
                (= carmountcolor (-(spraygunswash) 1)))

                (and
                (assing (carmountcolor) (0))
                (increase (wash)(1)))

                when (and
                (= ?c ?actualcolor)
                (< carmountcolor (-(spraygunswash) 1)))
                (increase (carmountcolor)(1))

                when
                (not(= ?c ?actualcolor))
                (assign (actualcolor) (c))
                (increase (wash)(1)))
        )
)

(:action mountcomp2
```

```
          :parameters (?i1 - charact  ?i2 -charact  ?x - car  ?c - color )
          :precondition ( and
                  (carneedcolor ?x  ?c)
                  (carneedcharac ?i2 ?x)
                  (not(carneedcharac ?i1 ?x))
                  (not(ismount ?x - car)))
          )
          :effect ( and
                  (not(carneedcolor ?x  ?c))
                  ( not (carneedcharac ?i2 ?x) )
                  (ismount ?x - car)

                  (decrease (comp1rest)(1))
                  when
                  (> comp2rest 0)
                  (increase (penaltycomp2)(1)))

                  (assign comp2rest (-(P2) 1))

                  when(and
                  (= ?c ?actualcolor)
                  (= carmountcolor (- (spraygunswash) 1)))

                  (and
                  (assing (carmountcolor) (0))
                  (increase (wash)(1)))

                  when (and
                  (= ?c ?actualcolor)
                  (< carmountcolor (- (spraygunswash) 1)))
                  (increase (carmountcolor)(1))

                  when
                  (not(= ?c ?actualcolor))
                  (assign (actualcolor) (c))
                  (increase (wash)(1)))
           )
)

(:action mountcomp1comp2
          :parameters (?i1 - charact  ?i2 -charact  ?x - car  ?c - color )
          :precondition ( and
                  (carneedcolor ?x  ?c)
                  (carneedcharac ?i1 ?x)
                  (carneedcharac ?i2 ?x)
                  (not(ismount ?x - car)))
          )
          :effect ( and
                  (not(carneedcolor ?x  ?c))
                  ( not (carneedcharac ?i1 ?x) )
                  ( not (carneedcharac ?i2 ?x) )
                  (ismount ?x - car)
```

```
                         when
                         (> comp1rest 0)
                         (increase (penaltycomp1)(1)))

                         when
                         (> comp2rest 0)
                         (increase (penaltycomp2)(1)))

                         (assign comp1rest (-(P1) 1))
                         (assign comp2rest (-(P2) 1))

                         when(and
                         (= ?c ?actualcolor)
                         (= carmountcolor (-(spraygunswash) 1)))
                         (and
                         (assing (carmountcolor) (0))
                         (increase (wash)(1)))

                         when (and
                         (= ?c ?actualcolor)
                         (< carmountcolor (-(spraygunswash) 1)))
                         (increase (carmountcolor)(1))

                         when
                         (not(= ?c ?actualcolor))
                         (assign (actualcolor) (c))
                         (increase (wash)(1)))
                 )
         )

)

(define (problem CARS-4-2)
(:domain CARS)
(:init (=wash  0)
   (=penaltycomp1 0)
   (=penaltycomp2 0)
   (=spraygunswash 20)
   (=P1  5)
   (=P2  8)
   (not(ismount 997))
   (carneedcolor 997 Green)
   (carneedcharac 1 997)
   (not(carneedcharac 2 997))

   (not(ismount 998))
   (carneedcolor 998 Green)
   (not(carneedcharac 1 998))
   (carneedcharac 2 998)

   (not(ismount 999))
   (carneedcolor 999 Red)
   (not(carneedcharac 1 999))
   (not(carneedcharac 2 999))
```

```
   (not(ismount 1000))
   (carneedcolor 1000 Red)
   (not(carneedcharac 1 1000))
   (carneedcharac 2 1000)
(:goal
   (ismount 997)
   (ismount 998)
   (ismount 999)
   (ismount 1000)
:metric minimize (+(+(*(1000)(penaltycomp1))(*(100)(wash)))(penaltycomp2))
)
```