Modelagem Preliminar em PDDL para o Problema do Sequenciamento de Carros Proposto pelo ROADEF 2005

A primeira grande dificuldade foi realmente entender a idéia do problema de sequenciamento de carros para poder assim fazer um mapeamento mais adequado para um problema de planejamento. Uma das coisas mais importantes é que é preciso fazer todas as permutações possíveis para se saber qual o melhor escalonamento dos carros. Outras propostas podem ser feitas em um espaço de busca guiado por uma heurística qualquer, mas por se tratar de uma modelagem preliminar, precisamos forçar o nosso planejador a fazer uma busca completa.

Assim sendo, na modelagem proposta o planejador não pára na primeira vez que encontra um goal. Forçamos ele a fazer uma busca em largura completa, ou seja, precisamos encontrar todas as possíveis maneiras de se chegar ao goal. Isso porque, para cada plano que encontra o goal, temos no fim o cálculo da quantidade de restrições de alta e baixa prioridade violadas e a quantidade de vezes que se muda de cor no refil, ou seja, não é um problema clássico de planejamento e sim um problema de otimização de custos.

Uma outra importante observação é que não há restrição de tempo para as ações. Na descrição é apenas citado que o programa não deve demorar mais que 600 segundos para encontrar uma resposta. Isso deve ser levado fortemente em consideração quando se for tentar fazer algo realmente implementável.

Foram detectados vários problemas nessa descrição, tais como a ausência de uma definição melhor da função violação, que ficou bastante abstrata. É preciso ter um histórico das ações já realizadas, além de ser necessário a representação e manipulação do escalonamento do dia anterior para melhor implementá-la. É necessário também colocar na situação inicial as taxas N/P de cada restrição.

Uma tentativa de descrição no formato PDDL é proposto abaixo. Mesmo com os problemas citados, acredito que a descrição abaixo deva servir como base para uma possível implementação bem definida do problema ROADEF 2005 no formato PDDL. Por ser uma versão preliminar, estamos tratando em um nível bem abstrato, sem nos preocuparmos também com a corretude da sintaxe

```
(define (domain roadef)

        (:requirements: typing : fluents: equality)
        (:types carro cor alta_restrição baixa_restrição)
        (:predicates

                (cor_do_carro ?car – carro  ?color –cor)
                (alta_restrição ?car –carro  ?high_constraint – alta_restrição)
                (baixa_restrição ?car –carro  ?low_contraint – baixa_restrição)
                (carro_montado ?car - carro))

        (:functions

                (penalidade_alta)  /* Acumulador de Restrições Altas */
                (penalidade_baixa)  /* Acumulador de Restrições Baixas*/
                (penalidade_cor)   /* Acumulador de Mudança de Cor */
                (ultima_cor_refil ?color - cor )  /* Guarda a última cor do refil */
                (violação_alta ?h_constraint  -alta_restrição)
                (violação_baixa ?l_constraint  -baixa_restrição))


        (:action  montar_carro  /*Ação Única */

           :parameters (?car – carro  ?color - cor ?high –alta_restrição ?low –
baixa_restrição)

           :precondition (and (cor_do_carro ?car ?color)
                              (not (carro_montado ?car)))

           :effect (and  (when (not (= (ultima_cor_refil color))
                                  (and (increase (?penalidade_cor 1))
                                  (set (?ultima_cor_refil color)))
                         (when (=(violação_alta ?high))
                                  (and(increase (?penalidade_alta 1 ))))
                         (when (=(violação_baixa ?low))
```

```
                                        (and(increase (?penalidade_baixa 1 )))
                          (carro_montado ?car)))
```

/*Tem como efeito o carro montado e caso ocorra alguma violação de restrição e/ou necessidade de mudança na cor do refil, os mesmos são computados nos respectivos acumuladores */

)

Um exemplo de problema:

```
(define (problem roadef)
(:domain roadef )
(:objects
          amarelo azul vermelho – cor
          carro1 carro2 carro3 – carro
          restr1 restr2 – alta_restrição
          restr3 restr4 – baixa_restrição)
(:init

        (cor_do_carro carro1 azul)
        (alta_restrição carro1  restr1)
        (= not(carro_montado carro1))

        (cor_do_carro carro2 vermelho)
        (alta_restrição carro2  restr2)
        (baixa_restrição carro2  restr3)
        (= not(carro_montado carro2))

        (cor_do_carro carro3 azul)
        (alta_restrição carro3  restr2)
        (baixa_restrição carro3  restr4)
        (= not(carro_montado carro3))

        (= (penalidade_cor) 0)  /*Todas as penalidades são iniciadas com zero*/
        (= (penalidade_alta) 0)
        (= (penalidade_baixa) 0)
)

(:goal
```

```
        (and(carro_montado carro1)
        (carro_montado carro2)
        (carro_montado carro3))
)

(:metric minimize (penalidade_alta)
        subminimize (penalidade_cor)            /* Estratégia 1*/
        subsubminimize (penalidade_baixa))
                                                        ou
(:metric minimize (penalidade_cor)
        subminimize (penalidade_alta)            /* Estratégia 2*/
        subsubminimize (penalidade_baixa))
)
```

Obs1: Por falta de um grande conhecimento sobre o PDDL foram inventadas algumas coisas tais como “set” (seta o valor da função) e “subminimize” (dentre os planos mínimos, tenta fazer mais um minimização).

Obs2: Note que o plano-solução será algum do tipo montar_carro1, montar_carro3, etc, ou seja, será a melhor seqüência que satisfizer a estratégia 1 ou a estratégia 2. Montar um carro significa colocá-lo na linha de produção. Por isso creio que é necessária apenas uma ação