Um Problema de Escalonamento de Trens Usando Programação por Restrições

Bruno Pereira Damasceno[1]
Claudio Cesar de Sá[1]
Milton Roberto Heinen[1]

[1] Universidade do Estado Santa Catarina – UDESC
Departamento de Ciência da Computação – DCC
Centro de Ciências Tecnológicas – CCT
Campus Universitário Prof. Avelino Marcante s/n
Bloco F 2o. Andar – Sala 209
89.223-100 – Joinville – SC – Brasil

brunopereiradamasceno@gmail.com.br,{claudio,miltonh}@joinville.udesc.br

Resumo. *Neste artigo, Programação por Restrições é aplicada ao problema do Escalonamento de Trens. O objetivo consiste em encontrar o menor tempo para que trens em estações em dois extremos cruzem um percurso pré-definido, evitando o cruzamento com trens em direções opostas. Após modelagem e implementação, os resultados de tempo levantados são aceitáveis, dada a complexidade desta classe de problema. Este resultado fortalece a CP como uma teoria atrativa a ser aplicado a problemas reais.*

1. Introdução

Um objetivo recorrente em Inteligência Artificial (IA) é a especificação de metodologias e técnicas que resolvam problemas específicos, cujas soluções apresentem características do comportamento humano "*inteligente*". Os objetivos atuais da IA estão distantes da definição original de inteligência para uma máquina, formulada por Turing, a qual exige senso comum em sua avaliação, logo, uma definição contestada por vários. A IA atual visa estabelecer uma tecnologia capaz de suportar o desenvolvimento de programas com bom desempenho em tarefas que exijam sofisticação cognitiva em um domínio especializado. Um programa que atenda a essa condição é considerado inteligente. Essa definição de "inteligência" almeja um sistema com a capacidade de aprender novos conhecimentos a partir de um conhecimento básico e, finalmente, reproduzi-los com desempenho semelhante a um especialista.

Neste contexto, os problemas que exijam algum tipo de habilidade humana, quanto a estruturação de uma solução algoritmica, os quais conduzam há um significativo número de espaço de estados, raciocínio diversos, incertezas, múltiplas avaliações, manutenção de verdades, dinâmica temporal, evolução, etc, são de interesse da IA. Logo, a habilidade mental típica do ser humano na solução de um problema, torna um indicativo de que um dado problema é ou não de interesse da IA.

Neste artigo, um problema adaptado de um clássico da IA, significativo quanto memória utilizada e espaço de estados, é atacado por uma técnica conhecida como a Programação por Restrições (PR) [Jaffar and Lassez 1987]. A PR se preocupa

em resolver problemas combinatoriais típicos das classes NP e E [Rossi et al. 2006, Dechter 2003].

O problema apresentado é o de escalonamento de trens, com múltiplos trens e estações. A proposta é de, em um conjunto de estações conectadas entre si, minimizar o tempo para que trens nas duas estaçoes extremas deste conjunto atravessem este trecho. Deve ser considerado que trens que se movimentam em direções opostas não podem cruzar ao mesmo tempo um trecho entre duas estaçoes adjacentes.

Este artigo está organizado de acordo com: na seção 2 uma revisão–resumo do contexto da PR e suas motivações. Na seção 3 segue por uma análise e modelagem do problema. Na seção 4 é discutido aspectos técnicos desta solução escrito na linguagem ProLog. Na seção 5 alguns testes são exibidos afim de constatar a eficiência da PR.

2. Fundamentação Conceitual

Nesta seção é apresentada a classe dos *Problemas de Satisfação de Restrições* (PSR), e uma síntese dos objetivos da *Programação por Restrições*, como uma das técnicas dirigida a resolver os PSRs.

2.1. Problemas de Satisfação de Restrições

Um problema combinatorial clássico é apresentado por um conjunto de variáveis de um sistema, as quais serão instanciadas por objetos de domínios, segundo um conjunto de relações, as quais representam o relacionamento entre os objetos. A tarefa combinatorial é dada pela ação de instanciar estes objetos as variáveis, de tal modo que todas as relações sejam satisfeitas.

A esta classe de problemas combinatoriais é conhecida como *Problemas de Satisfação de Restrições* (PSRs). A resolução dos PSR's constituem em encontrar valores as variáveis respeitando ou satisfazendo suas restrições. Deste modo, um PSR é tipicamente um problema *NP-Completo* [Rossi et al. 2006]. O desafio de todo o processamento por restrições está em gerar algoritmos que resolvam esta classe de problemas em um tempo computacional aceitável. Invariavelmente, alguns destes problemas NP, podem apresentar uma complexidade espacial consideravel, assim passam para classe P-SPACE [Sipser 1996]. Dado este aspecto combinatorial e de complexidade NP, esta passa ter interesse por outras àreas da pesquisa que lidam buscas *heurísticas*, tais como a Computação Evolutiva (CE), ou ainda buscas *completas* com a IA clássica e a Pesquisa Operacional (PO), etc. Em geral, destacam-se os problemas de escalonamentos, planejamento, roteamento, contenção, alocação, etc, atacando-os com ferramentas da PR [Rossi et al. 2006].

2.2. Programação por Restrições

O objetivo da Programação por Restrições (PR) é resolver problemas por exploração das restrições encontrando valores que satisfaçam uma solução [Apt 2003]. A PR provê uma abordagem declarativa para resolução de problemas.

A tarefa principal de um algoritmo PR está em encontrar soluções (valores para as variáveis) que obedeçam às regras impostas pelas restrições. Um resultado é dito *consistente* quando atende a estes critérios [Barták 1999]. Para aplicações da PR existem algoritmos que se utilizam dos conceitos advindos da Pesquisa Operacional (PO), da Programação em Lógica (PL), entre outros [Barták 2007].

2.3. Elementos da Programação por Restrições

Alguns fundamentos da PR se relacionam via restrições e declarações sobre um problema. Os problemas devem ser modelados, categorizados, ou seja, representados de forma a fazer com que se possa aplicar um determinado método de busca para encontrar a(s) solução(ões). Diversas são as técnicas encontradas na IA para a modelagem e resolução de problemas [Russell and Norvig 2010]. Formalmente um *modelo* em PR, segue uma notação dada pela tupla (V, D, R), onde:

V**:** um conjunto de variáveis usadas na modelagem do problema, $\{x_1, ..., x_n\}$;
D**:** um conjunto domínio(s), $\{D_1, ..., D_n\}$ em que as variáveis de V podem assumir valores;
R**:** tem-se no de m conjunto de restrições um mapeamento do tipo $(V \times D)^m \rightarrow V$. Assim, uma restrição é dada por: $r_j(x_1, ..., x_n)$

Logo, encontrar uma solução de um modelo em PR é logicamente expresso por:

$$\exists x_1 \exists x_2 ... \exists x_n (r_1(x_1, ..., x_n) \wedge r_2(x_1, ..., x_n) \wedge ... \wedge r_m(x_1, ..., x_n)) \quad (1)$$

Onde a sua interpretação lógica consistente é uma resposta ao problema. A descrição de sua solubilidade é análoga ao mundo de Herbrand [Russell and Norvig 2010]. Cada restrição é aplicada a um subconjunto de variáveis, visando a satisfatibilidade em de seus valores. Uma atribuição é dita *consistente* se esta não violar nenhuma restrição. Assim, uma solução é encontrada quando todas as variáveis possuirem um valor consistente [Apt 2003, Dechter 2003, Rossi et al. 2006].

Assim, encontrar uma solução para um problema (p) em termos de (V, D, R), resume-se em encontrar uma construção de um *modelo* (M_p) para este problema. Logo, um modelo deve ser especificado por esta tupla, tal que $M_p = (V, D, R)$. Leia-se: *um modelo M para o problema p.* Em resumo, busca-se uma consistência da equação 1, computando-se sobre M_p de modo recursivo, aplicando suas restrições, propagação e expansão, sistematicamente sobre um procedimento de busca. Detalhes: [Apt 2003].

Restrições

As restrições conduzem há um *encolhimento* no espaço de possibilidades (de estados) na busca por uma solução. A ordem pela qual as restrições são impostas não é relevante, mas sim, que ao final da conjunção dos termos seja atribuído o valor *verdadeiro*. As restrições possuem propriedades importantes a serem citadas [Rossi et al. 2006], tais como:

- Elas constituem uma *informação parcial*, haja vista que esta não pode, por si só, determinar o valor das variáveis do problema;
- As restrições são *aditivas*. Por exemplo: uma restrição $r_1 : X + Y \geq Z$ pode ser adicionada a uma outra restrição $r_2 : X + Y \leq W$;
- As restrições raramente são *independentes*. Geralmente compartilham variáveis, pois tratam sob um mesmo modelo. A combinação das restrições r_1 e r_2 resulta na obtenção de uma expressão algébrica do tipo: $Z \geq X + Y \leq W$;
- As restrições são ainda *não-direcionais*. Considerando a restrição $X+Y = Z$, esta pode ser utilizada para determinar a sua forma equivalente em $X \quad (X = Z - Y)$ ou em $Y \quad (Y = Z - X)$;

- As restrições são de natureza *declarativa* pelo fato de apenas denotarem as relações que devem ser asseguradas entre variáveis sem especificar um procedimento computacional para estabelecer esse relacionamento.

Afim de tratar essas restrições, variáveis, temos linguagens dirigidas há modelos construídos sob o paradigma da PR. Essas linguagens possuem suporte a diversos tipos de domínios. Dentre elas destacam-se as restrições booleanas, domínios finitos, intervalos reais e termos lineares. Os mais utilizados são: inteiros, booleanos, reais (potencialmente infinito), conjuntos, intervalos, etc. Detalhes encontram-se [Apt 2003].

2.4. Modelagem da Programação por Restrições

A partir das definição de modelo da PR, com os conceitos de *restrições*, *domínios* e *variáveis* de um problema, evidencia-se uma aderência desta com o paradigma da *Programação em Lógica* (PL) [Rossi et al. 2006, Marriott and Stuckey 1998]. O mecanismo de busca embutido nas linguagens com PL é natural e imediato visando a exploração, e consequentemente, a *redução* ou *filtragem* do espaço de estados. Assim, este trabalho resolve um problema com a PR, mas utilizando uma linguagem específica da PL, com uma biblioteca e mecanismo de busca modificado ao paradigma da PR. Estas linguagens da PR são conhecidas como *solvers* de M_p. O *solver* aqui utilizado é o $ECliPSe$ (ver `http://eclipseclp.org/`).

Estes aspectos da PR torna-a atrativa sob os seguintes requisitos metodológicos na resolução de problemas diversos [Barták 2007]:

- Adequação a representação do conhecimento, caso este seja construído sob alguma notação matemática;
- Rápida prototipação, consequentemente baixo custo de desenvolvimento;
- Visão declarativa de suas restrições, possibilitando uma facilidade quanto aos testes e depuração;
- Flexibilidade na codificação dos algoritmos por abstrair características de programação em lógica.

Estes requisitos são imediatos ao se realizar uma prototipação, uma prova de conceito, etc, de um problema de complexidade exponencial, no caso os problemas da classe NP. As seções que se seguem, descrevem o problema aqui resolvido, sua modelagem, implementação e resultados.

3. Modelagem do Problema do *Escalonamento de trens*

O problema de escalonamento de trens pode ser considerado como uma variação do problema de escalonamento de trabalhos, conhecido em inglês como *job shop*. Neste tipo de problema, existe um conjunto de tarefas a serem completadas, com restrições sobre os recursos utilizados pelas tarefas. Tarefas podem possuir dependência, ou seja, existem tarefas que são pré-requisitos para outras. O principal objetivo é minimizar o tempo de conclusão das tarefas. No caso dos trens, as tarefas podem ser associadas aos veículos, e os recursos como os trilhos e estações.

Uma das resoluções para o problema genérico do *job shop* foi proposta por Sanja Petrovic e Carole Fayad (2006), utilizando algoritmos genéticos. No artigo escrito, utiliza-se lógica fuzzy para a divisão do trabalho em operações, chamados lotes. Cada

lote é, então, alocado em um recurso. Usa-se uma função de desempenho para a medição da qualidade da solução, e também é usada mutação no algoritmo. Ao final, comparando diferentes métodos de alocação, o algoritmo genético foi mais eficiente que os métodos "tradicionais", porém menos eficiente que o próprio método com inicialização randômica.

Outro artigo, escrito por Cook e Applegate (1991), propõe executar uma série de cortes nos "planos" do problema de forma a restringi-lo um pouco mais, reduzindo o tempo total para encontrar uma solução. Apresenta-se uma série de possíveis cortes, reduzindo o tempo total de resolução em até mil vezes.

Woodruff (1997) propõe a utilização de um algoritmo baseado no problema de programação disjuntiva (origem teórica do job shop) para o escalonamento de trens no metrô de Amsterdã, na Holanda. Mesmo sem construir efetivamente a solução, o algoritmo conseguiu em ambiente de testes ajustar o escalonamento de trens ajustado para os horários de pico. Por fim, o artigo de Caprara et al (2005) propõe uma solução para um problema bem parecido com o proposto, com algumas variáveis a mais. Utilizando restrições como quantidade máxima de trens em uma estação em determinado tempo e manutenção dos vagões, propõe- se uma solução para quando já existe um conjunto de trens escalonados e deseja-se incluir outros nos intervalos.

O problema consiste em encontrar o menor tempo para a travessia de trens em uma sequência de estações. As estações se conectam sequencialmente, formando um trecho único entre todas as estações. Assim, existe um único caminho entre as estações nos dois extremos, como demonstra a Figura 1.

Figura 1. Conexão entre estações

Os trens estão posicionados nas duas estações extremas. Cada trem possui um tempo de saída da estação inicial, e deve percorrer todo o trecho entre as estações até a estação no outro extremo do trecho. É necessário observar que trens que trafegam em direções opostas não devem se cruzar em um determinado trecho entre duas estações adjacentes, ou seja, um trecho só poderá ser ocupado por trens trafegando em um mesmo sentido. Trens que partem na mesma direção partindo da mesma estação não podem sair no mesmo instante. A Figura 2 exemplifica estas restrições.

Considerando que o trecho custa 10 unidades de tempo para sua travessia, e o trem A parte no tempo 0. O trem B, que parte na mesma direção de A, deve sair com no mínimo uma unidade de tempo de diferença, ou seja, no tempo 1. Como o trilho se encontra ocupado por A, o trem C deve sair somente quando A completar a travessia, ou seja, no tempo 10. Caso o trem B esteja no meio da passagem quando A chegar ao outro lado, o trem C deve aguardar a passagem de B também , para então poder atravessar o trecho.

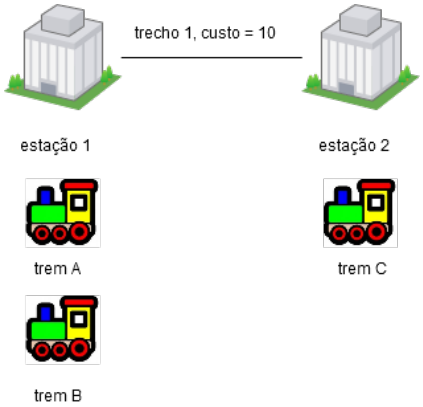

Figura 2. Restrição sobre um trecho

- Como uma proposta para a solução do problema, é utilizado um vetor unidimensional de valores, onde as posições representam o tempo de saída de cada trem em cada estação. As posições estão organizadas de forma a armazenar os trens sequencialmente, de forma alternada. Ou seja, sendo A e B as estações extremas, os trens serão armazenados na seguinte ordem: $T = \{1A, 1B, 2A, 2B, 3A, 3B, ..., NA, NB\}$, sendo N o número total de trens. O número de trens em cada estação extrema é igual.

  Cada posição de T representa os tempos de saída de cada trem, de forma que o trem 1 do lado A, por exemplo, seja composto dos tempos $1A = \{T1, T2, T3, ..., Tn\}$, com n sendo o número total de estações no trajeto.

  Uma solução possível, para um trem em cada extremo, partindo nos tempos $TA = 0$ e $TB = 5$, e 3 estações com tempo $t = 10$, seria $Q = \{0, 15, 25, 5, 15, 25\}$. O trem do lado A parte da estação inical no tempo $T1 = 0$, chegando na estação intermediária em $T = 10$. Como o trem B partiu em $T = 5$, chegará somente na estação intermediará em $T = 15$, e A deverá aguardar até este tempo para poder utilizar o segundo trecho. Ambos partem da estação intermediária no tempo $T = 15$, chegando ao mesmo tempo ao final do percurso em $T = 25$.

- Sendo $T(i, j)$ o tempo de saída do trem i da estação j, E o número total de estações, $R(j, j-1)$ o tempo entre as estações adjacentes j e $j-1$, define-se que o tempo para a saída da próxima estação seja maior ou igual que o tempo para alcançar a estaçao anterior, mais o tempo do trecho entre elas. O tempo para a saída da estaçao inicial será 0.

$$\forall_{i,j} : 1 < j \leq E \tag{2a}$$

$$T(i, j) \geq T(i, j-1) + R(j, j-1) \tag{2b}$$

- É necessário restringir o trecho para que ele não possa ser ocupado ao mesmo tempo por trens de direções opostas. Seja $T1(i, j)$ o tempo do trem de ida i na estação j, $T2(k, j)$ o tempo do trem de volta k na mesma estação j, E o número total de estações, $R(j, j-1)$ o tempo entre as estações adjacentes j e $j-1$:

$$\forall_{i,j,k} : 1 < j \leq E \tag{3a}$$
$$T1(i,j) \geq T1(i,j-1) + R(j,j-1) \vee \tag{3b}$$
$$T2(i,j-1) \geq T2(i,j) + R(j,j-1) \tag{3c}$$

Se as restrições forem verdadeiras,os trens não irão se cruzar em direções opostas, e seguirão até a estação no outro extremo. Esse processo é realizado entre todos os elementos do vetor T.

- O cálculo do custo é feito sobre os tempos de chegada do último trem que sai de cada lado. Sendo TA e TB os tempos de chegada dos trens A (último trem de ida) e B (último trem de volta) na estações nos extremos opostos ao de saída, o custo é a soma dos dois tempos.

$$Custo = TA + TB \tag{4a}$$

Cada instância diferente da lista de tempos T possuirá um custo diferente. Será escolhida a melhor solução dentre todas as instâncias, que neste caso é a que possui o menor valor para $Custo$. Uma solução é válida quando satisfaz todas as restrições. Este conjunto de fórmulas é a essência dos algoritmos apresentados nas seções seguintes.

4. Implementação

A implementação foi desenvolvida em Prolog, através de restrições sobre a lista de tempos T. O tamanho da lista é de $Q \times E$, sendo Q a quantidade de trens ao total e E a quantidade de estações. O problema consiste em escalonar os cruzamentos dos trens de forma a minimizar o tempo de chegada dos últimos trens que saem das estações extremas, de forma a atender as restrições descritas na seção anterior. Apenas para reafirmá-las, são elas:

- O tempos de um trem em determinada estação é o tempo até a estaçao anterior mais o trecho a ser percorrido
- Dois trens em direções opostas não podem utilizar o mesmo trecho ao mesmo tempo.

As restrições são feitas na lista a partir de várias estruturas de repetição, de forma a iterar sobre os tempos de saída dos trens (armazenados em uma lista separada), tempos de duração entre estações (também estão em uma lista separada), e a própria lista com as restrições. Assim, facilita-se a postagem das restrições ao diminuir a quantidade de código para tal. Para a restição dos trens em sentidos opostos, foi utilizado o predicado `disjunctive`, que é utilizado para garantir que duas ou mais tarefas não se sobreponham no tempo, ou seja, exatamente o que se deseja atingir neste trabalho. Ao final, utiliza-se o predicado `search` para efetuar a busca no espaço de estados e encontrar os valores das variáveis, de forma a minimizar o custo final.

Como exemplo, a solução encontrada para três estações com trechos de custo $c = 10$ e dois trens de cada lado (totalizando quatro trens), saindo no tempo mínimo $t = 0$, é 44, conforme mostra a tabela abaixo:

A lista de solução final fica da seguinte forma: $T = \{0, 11, 21, 0, 11, 21, 1, 12, 22, 1, 12, 22\}$.

Tabela 1. Exemplo Ilustrativo

Trem	Estação 1	Estação 2	Estação 3
$A1$	0	11	21
$B1$	21	11	0
$A2$	1	12	22
$B2$	22	12	1

4.1. Formulação do Problema das *Escalonamento de trens*

Para formalizar o Problema em PLR deve ser identificado um conjunto de variáveis, um domínio, e restrições, definidos por:
Lema 4.1. As variáveis são os elementos do vetor da Definição **??**.
Lema 4.2. As restrições para esta modelagem são definidas em **??** e **??**.
Lema 4.3. O domínio das soluções se encontra no conjunto dos números naturais (N) e está restrita ao limite de $1 \leq x \leq k$, onde x é um valor da solução para o problema do *Escalonamento de trens* , e k é um valor arbitrário. Idealmente, o valor de k deve ser infinito, mas isto aumentaria muito o espaço de soluções para as buscas, aumentando significativamente o tempo para encontrar-se a melhor solução.

4.1.1. Discussão sobre Possíveis Soluções

Não foi encontrada na literatura uma implementação específica para o escalonamento de trens. Existem trabalhos semelhantes, desenvolvidos por Woodruff, que são voltados especificamente para o problema de escalonamento de trabalhos. Para este trabalho, a solução precisou ser adaptada para atender aos requisitos específicos do problema dos trens. Devido a falta de espaço, este código é omitido.

Como a solução de Woodruff não atende as necessidades impostas neste trabalho, foi construído um novo código, utilizando alguns dos predicados da solução para escalonamento de trabalhos.

4.2. Modelagem da Minimização das Soluções

A minimização da solução final consiste em analisar todas as soluções válidas para o problema, pela soma dos valores nas posições do vetor referentes ao tempo final dos dois últimos trens que fazem o cruzamento, de cada lado. O predicado `search` é empregado para encontrar todas soluções possíveis. Com uma comparação entre os resultados encontrados, escolhe-se a de menor valor, tratando-se de uma minimização. O código completo deste experimento pode ser obtido com os autores.

5. Resultados

O experimento consistiu em implementar o método do item anterior. A implementação foi realizada com a versão 6.0 do ECL^iPS^e[1] `http://www.eclipseclp.org`. A biblioteca utilizada para PLR é a *ic*, disponível no mesmo pacote de instalação do ECL^iPS^e.

[1] ECL^iPS^e: *solver* para Programação por Restrições, baseado na linguagem ProLog. Esta permite a utilização de bibliotecas para CP, do inglês CLP *Constraint Logic Programming* e outras.

Os testes consistem em executar as instâncias válidas. Com uma configuração inicial dos tempos entre as estações e dos tempos de início de cada trem, o tempo de processamento é computado a partir da geração de todas soluções, e em seguida a sua minimização.

Foi necessário executar uma mesma solução para testar as heurísticas de busca, com o resultado sendo o da Tabela 5. Para os testes, foi escolhida uma instância média, com 10 trens e 10 estações, sendo o tempo de início de todos os trens $Tini = 0$, tempo de carregamento $Tcarga = 10$, e tempo entre estações $Ttrecho = 20$.

Tabela 2. Variações de parâmetros do *search*: seleção de variável e escolha no domínio

Testes	first_fail	anti_first_fail	occurrence	most_constrained	max_regret
indomain_min	$0,12$	$0,12$	$0,49$	$0,57$	$0,38$
indomain_max	$0,06$	$0,06$	$0,06$	$0,03$	$0,01$
indomain_media	$0,06$	$0,06$	$0,09$	$0,09$	$0,13$
indomain_split	$0,13$	$0,11$	$0,13$	$0,11$	$0,14$
indomain_random	$0,31$	$0,3$	$0,3$	$0,33$	$0,33$
indomain_interval	$0,53$	$0,48$	$0,51$	$0,53$	$0,51$

A Tabela 5 apresenta alguns destes valores quando executados em uma máquina padrão com 2.5 GHz de velocidade de CPU, 4 GB de memória principal. Todos os valores são arrendondados em duas casas decimais, com exceçao da média, que utiliza três casas decimais. Foram utilizados os parametros indomain_min e first_fail para execução dos testes.

Tabela 3. Tempo de execução (parâmetros do *search*:)

Trens $\times$ Estações	Ex1	Ex2	Ex3	Ex4	Ex5	Média
10×10	$0,03$	$0,02$	$0,01$	$0,01$	$0,02$	$0,018$
10×15	$0,04$	$0,04$	$0,05$	$0,04$	$0,04$	$0,042$
10×20	$0,07$	$0,08$	$0,08$	$0,08$	$0,07$	$0,076$
15×10	$0,13$	$0,11$	$0,13$	$0,11$	$0,14$	$0,124$
15×15	$0,28$	$0,30$	$0,32$	$0,29$	$0,26$	$0,290$
15×20	$0,44$	$0,46$	$0,41$	$0,43$	$0,43$	$0,434$
20×10	$0,84$	$0,86$	$0,90$	$0,91$	$0,89$	$0,880$
20×15	$5,4$	$5,66$	$6,01$	$5,78$	$5,68$	$5,706$
20×20	$25,79$	$26,33$	$26,13$	$26,64$	$26,76$	$26,330$

Embora tenha-se obtido tempos aceitáveis, a estatística dos tempos obtidos demonstram a complexidade exponencial deste problema. Tratando-se de uma otimização, este é um NP-difícil, do inglês, *NP-hard* [Sipser 1996]. Com o aumento da quantidade de trens e estações, os tempos crescem por um fator exponencial. Mesmo com instâncias pequenas, devido a natureza do problema, logo se começa a notar as diferenças de tempos de execução. Encontrar a melhor combinação não é o fator mais complicado neste experimento, mas sim encontrar *todas* as combinações, obtendo a soluçao ótima.

Em estratégias de buscas por melhoramentos como algoritmos genéticos, *simulated annealing*, são interessantes se o objetivo fosse encontrar uma única solução [Russell and Norvig 2010]. Contudo, no problema aqui proposto, a complexidade do controle em evitar as soluções duplicadas por estes métodos, deixaria de ser uma abordagem viável neste problema. Assim, a completude quanto as soluções existentes de um dado problema, torna a PLR como uma técnica atrativa.

6. Conclusão

Neste artigo foi aplicado a Programação por Restrições (PR) ao problema do *Escalonamento de trens* , onde deve-se encontrar o menor tempo para a passagem dos trens sobre um trecho de trilhos. A modelagem do problema foi construída de forma a encontrar todas as combinações válidas para o problema. Dentre todas soluções encontradas, uma minimização é aplicada para encontrar a melhor combinação das travessias dos trens sobre os trilhos da solução. Na realização dos experimentos constatou-se que o tempo para encontrar o valor da maximização é desprezível para instâncias pequenas, em torno de 16 trens e 15 estações ao total. Assim, para estes problemas combinatoriais, a Programação por Restrições (PR) pode ser aplicada a problemas do mundo real, onde possa se necessário escalonar as saídas de trens de forma a diminuir o tempo total necessário para que todos os trens atravessem o trecho sugerido. Alternativas a estudos futuros e abertos, destacam-se: *hardwares* especializados, paralelismo e concorrência na resolução do problema.

Referências

Apt, K. (2003). *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA.

Barták, R. (1999). Constraint programming - what is behind? In *In Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC99*, pages 7–15.

Barták, R. (2007). *On-line guide to constraint programming*. Availble in http://kti.ms.mff.cuni.cz/ bartak/constraints/index.html. Acess in 23/03/2007.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In *POPL*, pages 111–119.

Marriott, K. and Stuckey, P. J. (1998). *Introduction to Constraint Logic Programming*. MIT Press, Cambridge, MA, USA.

Rossi, F., Beek, P. V., and Walsh, T., editors (2006). *Handbook of constraint programming*. Elsevier.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Sipser, M. (1996). Introduction to the theory of computation. *SIGACT News*, 27(1):27–29.