Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO Nº 06/2010

Aos 06 dias do mês de agosto de 2010 realizou-se a sessão pública de Defesa da Dissertação **"Otimização Distribuída de Restrições em Ambientes Dinâmicos,"** apresentada pelo aluno **Bruno Giacomet** como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Fabrício Enembreck PUCPR (Orientador)	_____ *(assinatura)*	APROVADO *(aprov/reprov.)*
Prof. Dr. Bráulio Coelho Avila PUCPR	_____	Aprovado
Prof. Dr. Edson Emílio Scalabrin PUCPR	_____	Aprovado
Prof. Dr. César Augusto Tacla UTFPR/CPGEI	_____	Aprovado

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado APROVADO (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Mauro Sérgio Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

OTIMIZAÇÃO DISTRIBUÍDA DE RESTRIÇÕES EM AMBIENTES DINÂMICOS

Aluno: Bruno Giacomet
Orientador: Prof. Dr. Fabrício Enembreck

Curitiba
Agosto/2010

BRUNO GIACOMET

OTIMIZAÇÃO DISTRIBUÍDA DE RESTRIÇÕES EM AMBIENTES DINÂMICOS

Trabalho apresentado ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para validação de créditos de dissertação de mestrado, sob a orientação do Prof. Dr. Fabrício Enembreck.

Curitiba

Agosto/2010

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

G429o 2010

Giacomet, Bruno
Otimização distribuída de restrições em ambientes dinâmicos / Bruno Giacomet ; orientador, Fabrício Enembreck. – 2010.
viii, 81 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2010
Bibliografia: f. 79-81

1. Inteligência artificial. 2. Inteligência artificial distribuída. 3. Restrições (Inteligência artificial). 4. Algoritmos. 5. Informática. I. Enembreck, Fabrício. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 006.3

RESUMO

Problemas onde ambientes sofrem alterações em sua definição no decorrer do tempo representam uma categoria de problemas amplamente pesquisada. Observa-se na literatura que o formalismo de otimização de restrições apresenta um desempenho eficiente no que diz respeito à modelagem de problemas do mundo real com robustez e naturalidade. Porém, uma das dificuldades encontradas é como otimizar a construção de uma nova solução onde relações entre variáveis distintas podem tanto surgir como desaparecer ou até mesmo se modificar durante sua execução. Este trabalho apresenta o *DynADOPT*, um algoritmo completo, baseado em técnicas clássicas de reuso de soluções para tratar da otimização distribuída de restrições em ambientes dinâmicos.

Palavras-chave: Inteligência Artificial; Sistemas Multi-Agente; Ambientes Dinâmicos.

ABSTRACT

Problems where environments may present changes in its definition during the resolution introduces a new research field, widely studied in the last few years. The constraint optimization formalism proved to be a powerful method to model real-world problems with robust and natural abstraction. However, one of the difficulties reached is how to adapt an optimal solution whenever a new relationship between two distinct variables may appear, be removed or even suffer changes during the problem solving execution. This work presents *DynADOPT*, a new complete algorithm based on classic strategies of solution reuse to solve distributed optimization of constraints in dynamic environments.

Keywords: Artificial Intelligence; Multiagent systems; Dynamic Environments.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

A utilização de técnicas computacionais para auxílio à tomada de decisão tem motivado inúmeros pesquisadores a proporem novas metodologias, principalmente no que diz respeito a formas de resolver problemas do cotidiano de diversos setores. Os avanços nas áreas de informática possibilitaram que pequenos dispositivos possuam capacidade de processamento suficiente para serem dotados de conhecimento. Dessa forma, é possível fazer uso destes artefatos na resolução de problemas de maneira eficaz, permitindo aos seres humanos a tomada de decisões de forma precisa e com a devida rapidez, essencial em muitas situações. A otimização distribuída de restrições (*Distributed Constraint Optimization Problems – DCOP*) é um modelo de formalização de problemas complexos, amplamente pesquisado atualmente. Essa formalização permite que o problema seja divido em pequenas partes, facilitando assim sua abstração e resolução. Outra importante característica desta abordagem é a possibilidade de atrelar às possíveis soluções, níveis de custo ou qualidade podendo desta forma ser possível avaliar os resultados de maneira mais natural. Juntamente com a formalização de DCOP, foram pesquisados vários métodos de resolução de problemas modelados neste formalismo. Essas abordagens utilizam vários conceitos diferentes, tais como *backtracking* e programação dinâmica.

Os métodos para resolução de *DCOP* são eficientes para resolução de problemas estáticos, isto é, problemas onde as funções de custo, que são as funções que avaliam as soluções candidatas, não sofrem alterações durante sua execução. Entretanto, essa característica não é observada em muitos casos. Devido a esta carência, surgiram os problemas de otimização de restrição distribuídos dinâmicos (*Dynamic Distributed Constraint Optimization Problems – DynDCOP*). Porém, os algoritmos propostos para solucionar esses problemas não fazem uso de técnicas de reaproveitamento de informações geradas por resoluções prévias como ponto de partida em busca de convergir para um estado ótimo, apresentando, geralmente desempenho insatisfatório. Logo, este trabalho apresenta um novo algoritmo que reutiliza informações de soluções anteriores e consegue obter uma nova solução de forma eficiente quando alterações nas restrições são detectadas.

1.1. *Motivação e Hipótese*

A utilização de métodos de resolução de *DCOP* possibilita a otimização de várias tarefas do mundo real, como agendamento de grades horárias, controle de tráfego aéreo, condução de locomotivas e otimização em redes de sensores. Um método eficaz de executar tarefas desta categoria é o *ADOPT*, um método de resolução de *DCOP* estático que apresenta bons resultados em problemas do mundo real. Porém, é importante notar que esses problemas citados facilmente encontram situações onde existe a necessidade de alterações em sua estrutura, com a finalidade de adaptar-se ao novo ambiente apresentando características dinâmicas. Na maioria dos casos onde se encontram estas modificações, muitas informações obtidas na resolução dos problemas passados poderiam ser utilizadas evitando desperdício de tempo e esforço computacional, atingindo soluções ótimas para os novos problemas.

1.2. *Objetivos*

Este projeto tem como objetivo principal apresentar uma solução computacional baseada em técnicas clássicas para resolução de *DCOP* adaptado para atuar com eficiência e de forma completa em ambientes com restrições dinâmicas. Outro importante objetivo presente no projeto é que o método de resolução tenha um desempenho superior em relação ao algoritmo *ADOPT* original. Supõe-se que estratégias que priorizem a utilização de informações obtidas previamente tornem possível o alcance desses objetivos.

1.3. *Resultados obtidos*

Ao final deste trabalho é apresentado o *DynADOPT*, um algoritmo completo capaz de resolver problemas onde dada uma resolução anterior é possível convergir de forma mais eficiente para uma nova solução, fazendo uso das informações encontradas previamente. Para isso, foram estudados conceitos e algoritmos, além de técnicas de reuso de soluções e raciocínio.

1.4. *Estrutura do documento*

O presente documento está organizado da seguinte forma: o próximo capítulo compreende a fundamentação teórica, ou seja, uma pesquisa detalhada sobre o que foi desenvolvido até o presente momento sobre o problema em questão. Após esta análise, é apresentado o *DynADOPT*, um algoritmo capaz de resolver problemas *DynDCOP*. Os resultados alcançados pela realização de experimentos são apresentados em seguida, assim como as métricas utilizadas e a elaboração do ambiente de simulação. Ao final do trabalho são apresentadas as conclusões do estudo e os planos para o futuro desta pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta as origens, conceitos e a formalização de problemas de otimização de restrições distribuída, *Distributed Constraint Optimization Problem* (*DCOP*), bem como a sua variação que visa tratar de problemas em ambientes dinâmicos, denominados problemas de otimização distribuída de restrições em ambientes dinâmicos, *Dynamic Distributed Constraint Optimization Problems* (*DynDCSP*). Também são apresentados os principais algoritmos e técnicas que objetivam resolver problemas formalizados com as técnicas mencionadas.

2.1. *Problemas de Satisfação e Otimização de Restrições*

O formalismo de satisfação de restrições (*Constraint Satisfaction Problem – CSP*) (Rossi, et al., 2007) consiste em representar problemas como um conjunto finito de variáveis com domínios discretos e um conjunto de restrições entre as variáveis do problema. Um CSP pode ser definido por um conjunto de variáveis, $X_1, X_2,..., X_n$ e um conjunto de restrições $C_1, C_2,..., C_m$ (Weiss, 1999). Cada variável possui um domínio não-vazio de possíveis atribuições. As restrições do problema contêm conjuntos de atribuições permitidas. A resolução de um *CSP* pode ser definida como um conjunto de valores atribuídos para as variáveis do problema onde não existem restrições violadas.

A partir do conceito de *CSP*, aliado à evolução da Inteligência Artificial, surgiram novas subáreas de pesquisa, entre elas a Inteligência Artificial Distribuída (*Distributed Artificial Inteligence - DAI*). A *DAI* (Ferber, 1999) caracteriza-se pela resolução de problemas de forma distribuída, ou seja, dividindo o problema em subproblemas menores, onde a solução de cada um desses subproblemas faz parte da solução global. Esse paradigma possibilitou aos pesquisadores abordar novas formas de modelagem de problemas e algoritmos de resolução, como o *CSP* Distribuído.

O *DCSP* (*Distributed Constraint Satisfaction Problem*) (Yokoo, et al., 1998) utiliza os mesmos princípios do *CSP* para representação de um problema. Porém, nessa abordagem, as variáveis são distribuídas entre agentes que se comunicam através de mensagens com a finalidade de satisfazer as restrições em busca de uma solução global a partir de pontos de vista locais. A pesquisa é motivada pela existência de problemas naturalmente modeláveis no mundo real por este formalismo, e que ao mesmo tempo exigem um enorme esforço computacional quando abstraídos para abordagens centrali-

zadas (Bessière, et al., 2001). Uma das principais vantagens dessa forma de representação é se utilizar da eficiência do processamento paralelo e distribuído (Yokoo, et al., 1998). Outra característica deste formalismo é a forma de representação visual. Um *DCSP* geralmente é representado na forma de um grafo, onde seus nós representam as variáveis contidas no problema e as arestas, por sua vez, são caracterizadas como as restrições que duas variáveis distintas podem conter (Hannebauer, 2002). Um exemplo da utilização deste formalismo pode ser constatado na Figura 1. Nela, é possível observar três agentes x_1, x_2 e x_3 onde cada um destes possui um domínio definido para suas respectivas variáveis (ex: $D_1 = \{1, 2\}$). Também é possível observar que o agente x_3 contém restrições com os demais agentes, onde seus valores devem ser diferentes para atingir uma solução satisfatória para o problema representado.

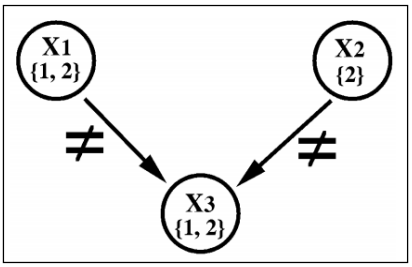

Figura 1 - Problema formalizado em DCSP (Yokoo, et al., 1998)

Conforme descrito anteriormente, o *DCSP* busca encontrar soluções satisfatórias quando existentes, porém, não é possível avaliar a qualidade da solução obtida. O *DCOP* (*Distributed Constraint Optimization Problem*) é uma extensão do *DCSP* capaz de estabelecer graus de qualidade ou custo da solução encontrada devido às funções de otimização utilizadas pelos mecanismos de busca (Lesser, et al., 2003).

A definição de um *DCOP* é semelhante ao de um *DCSP*. O seu formalismo é composto por um conjunto de n variáveis $V = \{v_1, v_2, ...v_n\}$, sendo que cada uma delas está associada a um único agente x_i e possui um domínio finito e discreto D_i. Apenas o agente correspondente tem o controle sobre sua variável e conhece o seu domínio. Este é responsável por escolher um valor d_i para sua variável tal que $d_i \quad D_i$. O objetivo de um *DCOP* é permitir um processo coordenado de escolha dos valores para as variáveis do problema com a finalidade de minimizar ou a função de custo do problema ou maximizar sua função de utilidade.

A principal diferença entre um *DCSP* e um *DCOP* é a definição das relações existentes entre as variáveis que compõe o problema. Em um *DCOP*, para cada par de

variáveis x_i e x_j, é dada uma função de custo f_{ij}: $D_i \times D_j \rightarrow N \cup \infty$. Estas funções de custo são análogas às restrições do *DCSP* e convenientemente na literatura recebem a mesma denominação.

Como alguns algoritmos de *DCOP* necessitam que os agentes obedeçam uma estruturação específica, será apresentada a técnica de pré-processamento para geração de pseudo-árvores, que estabelece uma estrutura organizacional entre os agentes.

2.2. *Pseudo-árvores*

Muitos algoritmos de *DCOP* têm a necessidade de estabelecer uma hierarquia entre os agentes, de modo a manter um fluxo de controle sobre as mensagens com a finalidade de evitar ciclos ou trocas de informações redundantes. Uma pseudo-árvore consiste na reorganização do grafo de restrições do problema em uma árvore, onde um agente pode ter apenas um pai e múltiplos filhos. Esta técnica foi proposta para satisfação de restrições em (Freuder, et al., 1985) e também utilizado em (Dechter, et al., 2006) e (Modi, et al., 2005). A construção desta é feita utilizando o algoritmo de busca em profundidade (*Depth First Search*). Um exemplo de uma pseudo-árvore é apresentado na Figura 2.

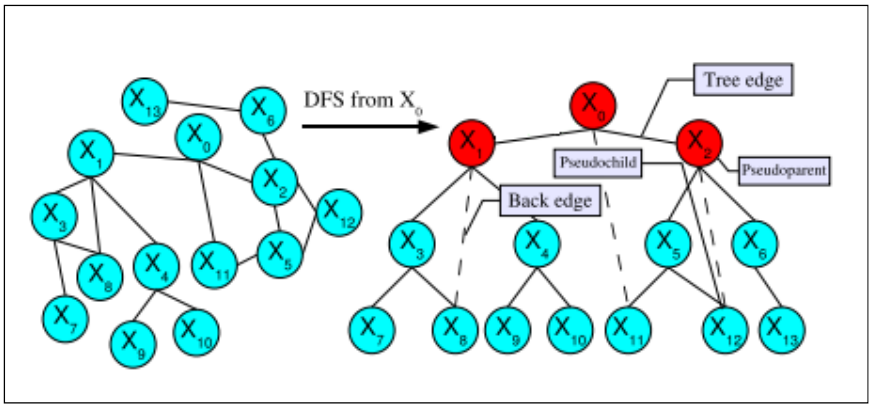

Figura 2 - Pseudo-árvore (Petcu, et al., 2006)

Analisando a Figura 2, é possível observar a hierarquia a partir de x_0 e entre os agentes na estrutura de pseudo-árvore, onde as relações entre os agentes são mantidas. Por exemplo, o agente x_0 e o agente x_{10} fazem parte do mesmo subproblema, assim como x_2 e x_{11}. Outra característica relevante são os agentes denominados pseudo-pai (*Pseudoparent*) e pseudo-filho (*Pseudochild*). Um pseudo-pai é um agente que contém uma relação com algum outro agente que já contém um pai direto. Este último é consi-

derado um pseudo-filho. Um exemplo deste tipo de relacionamento pode ser destacado ainda analisando a Figura 2, onde x_1 é o pseudo-pai do pseudo-filho x_8. É importante detectar e definir este tipo de ligação entre os agentes, pois em um esquema de comunicação *top-down*, onde o agente que representa a raiz da árvore processa suas informações e envia para os filhos e assim sucessivamente, pode haver vários casos onde a redundância das informações passadas pode comprometer a integridade do sistema.

Outra característica importante das pseudo-árvores é a possibilidade de isolar subproblemas, tornando-os menos complexos e possibilitando solucioná-los em paralelo. Um exemplo desta propriedade são os ramos derivados dos nós x_1 e x_2. Além da organização, uma das finalidades das pseudo-árvores baseadas em busca em profundidade é a de reduzir o tempo de busca nos nós do grafo (Freuder, 1985).

Esta técnica de estruturação do grafo de restrições é necessária para vários métodos de resolução de *DCOP*, sendo estes chamados de algoritmos de busca em árvores (*Tree Search Algorithms*), porém nem todos os métodos propostos necessitam deste tipo de pré-processamento. A seguir serão apresentados e descritos os principais algoritmos para resolução de *DCOP*.

2.3. *Métodos para resolução de DCOP*

Para encontrar a solução de problemas modelados como *DCOP*, foram desenvolvidos diversos algoritmos, utilizando diferentes técnicas afim de alcançar este objetivo. Estes algoritmos podem ser divididos em duas grandes ramificações dentro dos métodos de resolução: síncronos, e assíncronos.

Os métodos síncronos são aqueles que possuem uma ordem de execução, onde cada agente aguarda os demais para realizar o seu processamento. Por outro lado os algoritmos assíncronos possuem um comportamento onde todos os agentes tem a possibilidade de agirem livremente, sem a necessidade de aguardar o término de outros fatores ou agentes durante a sua execução.

Entre estes serão apresentados apenas os algoritmos de caráter assíncrono mais conhecidos na literatura. Esta pré-seleção se deve ao fato dos métodos síncronos possuírem um comportamento dispendioso em relação a processamento e tempo de execução, uma vez que os agentes envolvidos devem aguardar a sua vez na fila de prioridades.

2.3.1. ADOPT

O *ADOPT* (*Asynchronous Distributed Constraint OPTimization*) é um algoritmo *DCOP* assíncrono (Modi, et al., 2003) e foi o primeiro método completo a ser proposto que garante uma solução global ótima em um ambiente de comunicação assíncrona entre os agentes. A idéia geral do *ADOPT* é fazer com que os agentes concorrentemente alterem os limites mínimos da solução global. Assim como outros algoritmos de *DCOP*, é necessário que os agentes sejam organizados em pseudo-árvores. Dessa forma, os agentes comunicam os valores atribuídos às suas variáveis para os agentes conectados com prioridade inferior (filhos e pseudo-filhos) e comunicam os limites mínimos obtidos aos agentes de prioridade alta (pai). Como toda a comunicação é assíncrona, o *ADOPT* pode explorar as vantagens da computação concorrente quando possível.

O algoritmo inicia com cada agente atribuindo um valor para sua variável e o comunicando para todos os seus descendentes utilizando mensagens do tipo *VALUE*. Após essa fase, os agentes aguardam e respondem as mensagens recebidas. Quando recebido uma mensagem *VALUE*, o agente armazena esse valor vindo de um de seus ancestrais em seu contexto corrente (*Currentview*) e reporta para o seu pai o limite mínimo baseado em seu contexto através de uma mensagem do tipo *VIEW*. Um detalhe importante a ser considerado, é o descarte de uma mensagem *VIEW* quando existem incompatibilidades entre o contexto recebido e o corrente. Isso se deve ao fato do agente pai ou do agente filho ter um uma visão mais atualizada devido a ancestrais em comum. O limite mínimo é obtido através de um procedimento denominado subida da encosta (*Hill climb*). Neste, o agente estima através de seu contexto corrente o custo do valor no domínio que minimiza a sua função de otimização local e o envia para seu pai. Sempre que o agente recebe uma mensagem de seu pai do tipo *VALUE* e detecta uma inconsistência em sua visão, o seu limite mínimo é descartado, pois o valor armazenado se torna inválido. Para uma melhor compreensão dos procedimentos que compõe o *ADOPT*, é apresentando na Figura 3 o pseudocódigo do algoritmo.

```
#   xi: variável local
#   di: valor atual da variável de xi
#   c(d): lower bound atual da pseudo-árvore iniciada no agente filho se xi escolher d

0   Inicio
1     CurrentContext ← {};
2     ∀d ∈ Di, faça
3       c(d) ← 0;
4     fim faça;
5     BACKTRACK;

6   Ao receber (VALUE,(xj,dj))
7     adiciona (xj,dj) em CurrentContext;
8     se context(d,xi) é incompatível
            com CurrentContext
9           ∀d ∈ Di, faça
10          c(d) ← 0;
11          fim faça; fim se;
12    BACKTRACK;

13  Ao receber (View,context,cost)
14    d ← valor de xi em context;
15    se context é compatível
            com CurrentContext U (d,xi)
16      c(d) ← max(c(d),cost);
17      se c(d) foi alterado
18        BACKTRACK;
19      fim se; fim se;

20  BACKTRACK
21    ∀d ∈ Di, faça
22    e(d) ← EstimaMenorCusto(xi, CurrentContext U (xi, d) + c(d));
23    Escolha d que minimize e(d);
24      di ← d;
25    fim faça;
26    ENVIA(VALUE(xi, di))) para todos os descendentes conectados
27    ENVIA(VIEW(CurrentContext, e(di)) para o pai
```

Figura 3 - Pseudocódigo do ADOPT

De acordo com a Figura 3, o *ADOPT* é dividido em quatro procedimentos principais: inicialização, recebimento de *VALUE* (linha 6), recebimento de *VIEW* (linha 13), e procedimento de subida da encosta (linha 20). Na fase de inicialização sua visão local e a sua variável são definidas como vazios e os custos dos valores do domínio são atribuídos com zero. No procedimento de recebimento de *VALUE* o agente adiciona o valor enviado pelo seu pai em sua visão local. Caso houver uma mudança no contexto, ou seja, o valor do agente emissor for diferente de um previamente armazenado, os custos para as variáveis do domínio do agente corrente não são mais válidas e por este motivo devem ser descartados, além de atualizar seu contexto local. O procedimento de recebimento de *VIEW* estabelece a recepção da visão local de um filho. Nesta fase o agente atribui o valor indicado no contexto recebido, caso este seja compatível com a sua visão local, o agente deve atribuir para o custo de sua variável o maior custo entre o atual e o recebido. É importante definir o conceito de visão compatível: ocorre quando uma vari-

ável aparece em duas visões diferentes e possuem o mesmo valor. Contextos que não possuem as mesmas variáveis também podem ser considerados compatíveis. Esta condição é necessária para que os agentes não alterem os custos armazenados baseados em visões antigas e inválidas. O último procedimento a ser definido é o de subida da encosta. Este é chamado em todos os demais métodos que compõe o algoritmo, e é responsável por estimar os menores custos dos valores do domínio do agente, também como o envio das mensagens de *VALUE* e *VIEW* para os demais agentes do problema que possuem relação com o corrente. Note que no procedimento de recebimento de *VIEW*, a subida da encosta somente é chamada caso o custo do valor seja alterado. A seguir é definido um problema modelado com *DCOP* e apresentada a sua resolução utilizando os mecanismos da abordagem proposta pelo *ADOPT*. Este consiste em encontrar a associação A tal que custo seja mínimo.

$$F(A) = \sum_{xi,xj \in V} f_{ij}(d_i, d_j) \quad (1)$$

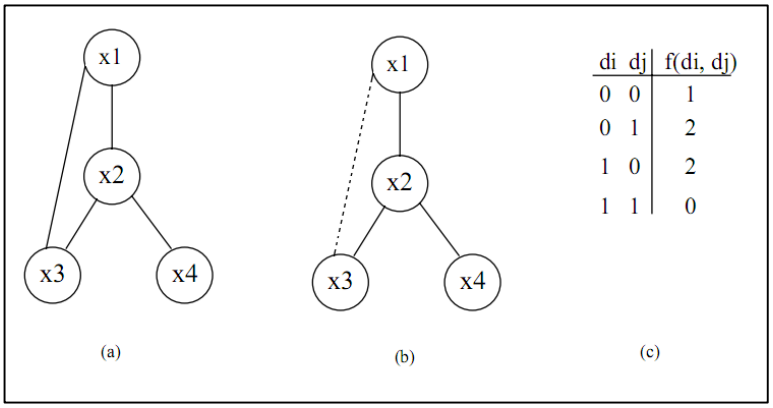

Figura 4 - Exemplo de problema DCOP, adaptado de (Modi, et al., 2005)

O exemplo proposto na Figura 4 consiste em encontrar uma solução para o grafo composto de quatro agentes (Figura 4.a). Conforme mencionado, o *ADOPT* necessita que os agentes envolvidos sejam hierarquizados em uma ordem de prioridade, logo para resolver essa condição o grafo é organizado na forma de pseudo-árvore (Figura 4.b). Outra necessidade é definir as funções de custo entre os agentes para que possam ser avaliados os valores de seus domínios (Figura 4.c). Na seqüência é demonstrada a execução do ADOPT objetivando resolver o problema proposto.

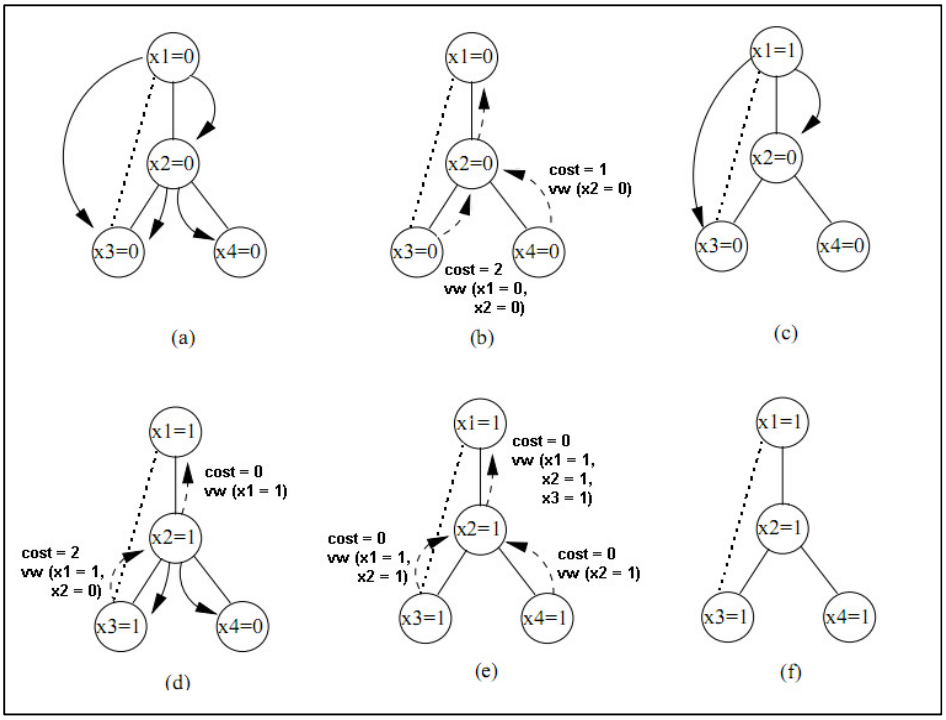

Figura 5 - Exemplo de execução do *ADOPT* adaptada de (Modi, et al., 2005)

O exemplo da execução do *ADOPT* contido na Figura 5, demonstra as ações tomadas pelo algoritmo durante a resolução do problema. Como se trata de uma abordagem assíncrona pode haver mais de uma resolução para o problema. Isso se deve ao fato da ordem das mensagens não seguir nenhuma seqüência previamente estabelecida. Logo, o exemplo apresentado descreve uma destas possíveis resoluções.

O algoritmo inicializa sua execução com os agentes atribuindo um valor (geralmente o primeiro valor contido no domínio) para suas variáveis e os propagando para seus descendentes. Na Figura 5.a os agentes são todos inicializados com *0* e enviam este para todos os seus filhos. Recebendo o valor de seu pai (Figura 5.b), x_2 adiciona em sua visão local e escolhe para sua variável o valor *0*, pois de acordo com a função de custo do problema $f(0, 0) = 1$ e $f(0,1) = 2$, e propaga a sua visão para x_1. Ao mesmo tempo, x_3 e x_4 realizam o mesmo procedimento com os valores recebidos de seus ancestrais e ambos determinam que *0* é o melhor valor para o contexto, enviando a visão local para o agente pai. No momento que o agente x_1 recebe a visão de seu filho, atribui o valor de sua variável com o contido no contexto ($x_1 = 0$) e atualiza o custo desse valor para o mais alto entre o custo atual e o recebido na mensagem. Em uma nova estimativa, x_1 decide que o melhor valor para si é *1*, propagando a decisão para todos os filhos (Figura

5.c). Estes por sua vez, com o contexto alterado zeram seus custos locais realizam novas estimativas para escolha de suas variáveis. Logo, x_2 e x_3 decidem que o valor ótimo baseado no contexto recebido por cada é *1* (Figura 5.d). Este valor é propagado por x_2 para x_4 que também altera sua variável para *1* (Figura 5.e). A execução encerra quando o custo do melhor valor estimado é igual ao custo do valor atual do contexto (Figura 5.f). Analisando sua execução é possível perceber o grande volume de mensagens trocadas entre os agentes bem como o pouco processamento local necessário para a resolução do problema.

As vantagens principais deste algoritmo são a flexibilidade de sua arquitetura e o pouco esforço computacional local necessário para sua execução. Porém ele pode apresentar ineficiência em ambientes com muitos agentes ou um número elevado de restrições devido ao grande número de trocas de mensagem. Uma troca de contexto em um nó raiz ou próximo dele causa um grande número alterações nos agentes que estão contidos no ramo do subproblema. Assim como a idéia deste projeto, existiram outras adaptações baseadas no *ADOPT* com o objetivo de aperfeiçoar seu desempenho em ambientes específicos.

2.3.2. ADOPT-ng

Uma das deficiências exploradas do *ADOPT* aborda o modelo de comunicação, sobretudo a transmissão das mensagens de custo, que não provê informações necessárias para os agentes tomarem as melhores decisões. Com isto, os agentes necessitam de um número maior de ciclos e de trocas de mensagens para alcançar um estado estável em seu subproblema. Outro aspecto negativo do *ADOPT* é a necessidade de uma organização dos agentes em forma de uma pseudo-árvore, podendo apresentar elevado custo computacional para a construção da árvore a partir do grafo de restrições do problema.

Para resolver tais problemas, (Silaghi, et al., 2006) propuseram uma nova versão do algoritmo, chamada *ADOPT-ng*. A principal diferença entre a versão do *ADOPT-ng* e a original está no sistema de mensagens, que é baseada em um novo tipo de *nogood*. Essa idéia tem o objetivo de eliminar a necessidade da etapa de pré-processamento por não exigir uma ordenação dos agentes, apresentando uma melhoria do desempenho do algoritmo. Isso deu origem a um novo tipo de *nogood*, chamado *nogood* valorado, no qual a informação sobre o custo de uma atribuição é acoplada ao *nogood*.

O *ADOPT-ng* substitui a mensagem de custo do *ADOPT* original por uma mensagem de *nogood*, onde *c* é o custo mínino em um dado conjunto *N* de atribuições de variáveis distintas. Com estas informações, um agente pode adicionar um custo de atribuição *v* a partir dos *nogoods* recebidos, sendo possível estimar um custo mínimo para as atribuições conflitantes. Através desta adição, é possível maximizar a eficiência do esquema de busca acelerando a resolução do algoritmo.

Além da mensagem de *nogood*, o *ADOPT-ng* também contém outro tipo de mensagem, chamada *add-link*. Uma mensagem de *add-link* anuncia o interesse em uma variável contida no *nogood*, cujo agente remetente não é vizinho do agente destinatário. A idéia desta mensagem é minimizar de maneira mais eficiente o custo total das atribuições no *nogood*. O processo de reconstrução de uma solução parcial contida no *nogood* é mais eficiente que o *ADOPT* tradicional, pois somente as variáveis contidas no *nogood* irão alterar seus valores.

2.3.3. BnB-ADOPT

Outra proposta de um algoritmo derivado do *ADOPT* é o *BnB-ADOPT* proposto por Yeoh (Yeoh, et al., 2008). A idéia desta abordagem é substituir o método de busca originalmente utilizado no *ADOPT* por um que possibilite um aumento no desempenho do algoritmo.

O *ADOPT* utilizada em sua estratégia a busca do melhor primeiro (*best-first search*). Esta idéia apresenta bons resultados para a convergência do algoritmo, pois seu objetivo é convergir para a melhor solução possível. Porém, para garantir um baixo consumo de memória, necessita que as soluções abandonadas sejam eliminadas. A proposta do *BnB-ADOPT* é substituir a busca do melhor primeiro pela busca em profundidade com melhor custo (*depth-first branch-and-bound search*). Este método de busca acarreta em um baixo consumo de memória e evita a reconstrução de soluções parciais. Infelizmente esta estratégia explora também várias soluções desnecessárias, desperdiçando processamento e aumentando o espaço de estados a ser analisado.

2.3.4. DPOP

O *DPOP* (*Dynamic Programming Optimization Protocol*) utiliza uma nova estratégia no que diz respeito à forma como os agentes analisam suas restrições e como se comunicam, o *DPOP* (Petcu, et al., 2005) é baseado nos princípios da programação di-

nâmica, se destacando entre os algoritmos baseados em técnicas de *backtracking* propostos até então. Trabalhando de maneira assíncrona e fazendo uso das técnicas de hierarquização dos agentes em árvores, o *DPOP* resolve problemas de otimização utilizando um número linear de mensagens.

Basicamente, o algoritmo pode ser divido em duas fases: a fase de cálculo de utilidades para os valores do domínio, e a fase de atribuições pelos agentes tendo em vista as restrições previamente avaliadas.

A primeira fase, onde são calculados os custos das possíveis atribuições dos agentes para suas respectivas variáveis, tem início nos nós-folha da pseudo-árvore do problema. Nesta etapa, um agente envia para seu pai direto uma mensagem de *UTIL* contendo os melhores custos do agente atrelado ao conjunto de atribuições para o subproblema. Desta forma, ao receber uma mensagem desta categoria, o agente pode facilmente calcular o custo ótimo para o subproblema nele iniciado.

A segunda fase da execução do algoritmo consiste na atribuição de valores às variáveis por seus respectivos agentes. Esta etapa tem início no agente pai (raiz da árvore) que na fase anterior recebeu as mensagens de *UTIL* de seus filhos. O agente então escolhe o valor que minimize o custo global e comunica aos seus descendentes diretos sua decisão através de mensagens do tipo *VALUE*. Estes, por sua vez, realizam o procedimento de maneira similar. Escolhem um valor em seus respectivos domínios, baseados na escolha dos pais e no conjunto atribuições recebidos anteriormente. Após a atribuição ser concluída, o agente propaga a escolha junto com a mensagem oriunda de seu ancestral para seus descendentes. O procedimento é finalizado quando os nós-folha realizam suas devidas atribuições.

Apesar de o *DPOP* ser capaz de resolver problemas de otimização em tempo polinomial com um número linear de mensagens, o tamanho das mensagens também possui esse comportamento. Dessa forma, situações onde os problemas são super-restritos, o tamanho das mensagens pode ser consideravelmente expressivo. Esse aspecto poderia inviabilizar a utilização do *DPOP* em ambientes onde o consumo de memória é limitado. Em função desta deficiência, Petcu e Faltings (Petcu, et al., 2006) apresentam uma versão híbrida do *DPOP*, denominada *Memory Bound DPOP* (*MB-DPOP*), que permite limitar o consumo de memória exigido. (Petcu, et al., 2006) também propuseram uma versão parcialmente centralizada do *DPOP* para aumentar a eficiência do processo de busca.

2.3.5. OptAPO

Grande parte dos algoritmos que tratam problemas de *DCOP* é baseada em trocas de informações locais entre os agentes. O *OptAPO* (*Optimal Asynchronous Partial Overlay*) (Mailler, et al., 2004) utiliza uma abordagem diferenciada em relação à organização adotada pelos agentes, pois não utiliza estrutura de pseudo-árvores e pela forma de como são tomadas as decisões para alterar o valor de suas respectivas variáveis. Essas ações se baseiam em sessões de mediação que têm como objetivo unificar o conhecimento de uma parte do problema tomando decisões locais, buscando uma solução parcial para o problema global.

No algoritmo, cada agente contém seu custo F_i e um custo estimado F_i* que o possibilita analisar o problema e iniciar ou não uma sessão de mediação. Estas sessões estão dividas em dois tipos básicos: as mediações ativas e passivas. As mediações ativas têm como finalidade atualizar os valores de F_i e F_i* dos agentes envolvidos. As mediações passivas, por sua vez, têm como objetivo estimar com maior precisão os valores de F_i*. Em outras palavras, as sessões ativas visam alterar os valores escolhidos pelos agentes para alcançar uma solução ótima para o subproblema, e as sessões passivas têm a finalidade de aumentar a visão dos agentes de menor prioridade. A prioridade entre os agentes é definida dinamicamente através do tamanho da *good_list* de cada um, esta sendo definida como uma lista contendo as visões que o agente contém sobre o problema.

Em uma sessão de mediação ativa, o agente mediador faz uma requisição para os demais agentes envolvidos na sessão para que retornem uma avaliação do custo de suas restrições baseadas em suas visões. Os agentes que já estiverem atendendo requisições de outras sessões respondem com uma mensagem de *wait* e é desligado da sessão corrente. Isso garante o paralelismo no algoritmo. Quando os agentes respondem a requisição do mediador, este conduz uma busca do tipo *Branch and Bound* visando encontrar a combinação de valores que minimize F_i. Em uma mediação passiva o agente mediador busca apenas compreender os resultados dos agentes de prioridade alta, que conseqüentemente possuem um conhecimento maior do problema.

Outros métodos de resolução de *DCOP* foram desenvolvidos, entretanto, este trabalho se limita a apresentar os principais métodos. O leitor interessado é convidado a consultar as referências (Yokoo, et al., 1998), (Petcu, et al., 2006), (Bowring, et al.,

2008), (Petcu, et al., 2006), (Petcu, et al., 2006), que apresentam outros métodos menos conhecidos.

2.4.*Problemas de Otimização de Restrição Distribuídos Dinâmicos*

Até o momento, todos os métodos e formalismos apresentados são específicos para tratamento de problemas estáticos, ou seja, não abordam problemas em que restrições ou variáveis podem se alterar ou deixar de existir no ambiente durante a sua execução. Devido a essa deficiência, foram formalizados os problemas de otimização de restrições distribuídos dinâmicos (*Dynamic Distributed Constraint Optimization Problems – DynDCOP*) (Schiex, et al., 1993).

O *DynDCOP* pode ser considerado uma seqüência $\mathcal{P}_n$ de *DCOPs*, sendo $\mathcal{P}_{i+1}$ um problema resultante de uma alteração em $\mathcal{P}_i$ (Bessière, 1991), onde estas alterações são denominadas restrições, ou seja, uma nova restrição é adicionada ao grafo de relações entre os agentes, ou uma relaxação é realizada, o que significa que uma restrição foi removida ou reduzida no ambiente. Junto com essa nova formalização de problemas, foram propostos algoritmos com o objetivo de solucioná-los. Os objetivos principais desses algoritmos além de obviamente obter uma solução para os problemas é realizar a busca focando eficiência e estabilidade (Verfaillie, et al., 1994). Eficiência neste contexto significa que o método precisa ser rápido na convergência. Sistemas que operam em tempo real necessitam de abordagens que levam em consideração este comportamento. A estabilidade por sua vez, está relacionada às soluções obtidas que devem levar em consideração informações essenciais armazenadas nas execuções anteriores. Em sua grande maioria, os algoritmos que tratam *DynDCOP* são adaptações de métodos propostos para *DCOP*, porém a literatura apresenta também soluções especialmente desenvolvidas para abordar problemas *DynDCOP*.

2.5.*Estratégias de resolução em ambientes Dinâmicos*

Conforme mencionado anteriormente, vários algoritmos que objetivam resolver problemas *DynDCOP* são baseados em métodos desenvolvidos que atuam em problemas estáticos. As alterações em sua grande maioria visam capacitar o algoritmo eleito a reutilizar as soluções anteriores de maneira eficiente. Desta forma, serão apresentadas algumas destas técnicas (Verfaillie, et al., 2005).

Os métodos de reuso de soluções passadas podem ser divididos em duas grandes subáreas: as abordagens *reativas* e *pró-ativas*. Para compreender as diferenças entre estas abordagens, primeiramente é necessário definir as soluções produzidas na resolução de um problema. Para um *DCOP*, *soluções consistentes* são aquelas que têm o menor custo possível, um grau de qualidade máximo ou que suas utilidades estejam contidas dentro de um intervalo previamente estabelecido (Verfaillie, et al, 2005). Por sua vez, soluções não-ótimas ou incompletas são consideradas inconsistentes. A seguir serão apresentadas as abordagens para otimização de algoritmos de *DCOP* para ambientes dinâmicos.

2.5.1. Abordagens Reativas

Neste tipo de abordagem estão reunidos os métodos que não utilizam conhecimento para as possíveis alterações, porém isto não significa que estes métodos não antecipem escolhas. Eles apenas não fazem uso das possíveis direções para soluções futuras. As técnicas pertencentes a este grupo são subdividas em *reuso de soluções* (*solution reuse*) e *reuso de raciocínio* (*reasoning reuse*).

O *reuso de soluções* é utilizado em situações onde o problema anterior foi solucionado, mas por uma alteração no problema onde a solução atual não é mais válida. O princípio desta técnica é que um problema *P'*, resultante de um problema *P* tenha uma solução *S'* relativamente próxima de *S*. A seguir são apresentadas as principais famílias de métodos que utilizam esta abordagem:

Métodos de busca em árvore: consiste em utilizar um método de busca em profundidade considerando como heurística a solução anterior;

Métodos de busca local: estes métodos têm a maior eficácia em contextos dinâmicos, pois as soluções anteriores podem ser simplesmente utilizadas como estado inicial para uma nova busca;

Métodos de re-atribuição de variáveis: este é semelhante ao método de busca local. A diferença entre estes é a forma de alteração da solução antiga para obter a resolução do novo problema. O princípio desta técnica é identificar restrições não satisfeitas, remover a atribuição de pelo menos uma variável e

re-atribuir as demais com a finalidade de encontrar o conjunto de variáveis cujo custo é ótimo. Caso a atribuição não produza tal custo, é realizado o procedimento de *backtracking*.

As técnicas de *reuso de raciocínio* são mais utilizadas em casos onde a solução para o problema anterior não é consistente. Estas fazem uso das informações contidas nas restrições analisadas e resolvidas do problema *P,* e por uma mudança ocorrida em sua definição resulta em um novo problema *P'*. Se esta mudança for uma relaxação no grafo, conseqüentemente as resoluções para as restrições não são mais confiáveis. A idéia básica desta técnica é que como *P'* é semelhante a *P*, várias destas restrições se mantém válidas. Evitar a reavaliação destas é uma maneira eficiente de aperfeiçoar a busca por uma nova solução. A seguir são apresentadas as principais famílias de métodos que fazem uso desta abordagem:

Métodos baseados em grafos: estes métodos apenas consultam o grafo quando uma atribuição é questionável. Quando uma restrição *c* é removida do problema, todas as avaliações feitas com *c* são questionáveis e também removidas;

Métodos baseados em justificativas: os métodos que utilizam esta abordagem analisam as justificativas armazenadas com cada avaliação das restrições, para determinar quando estas se tornam inválidas. Estas justificativas são as restrições que produziram o resultado. Quando uma destas é removida do problema, todas as avaliações, que contém nas justificativas as restrições que não fazem mais parte do problema, se tornam inválidas;

Métodos baseados em explicação: método semelhante ao baseado em justificativas, porém neste método são armazenadas juntamente com as avaliações, todas as restrições que logicamente fazem parte destas.

2.5.2. Abordagens Pró-Ativas

Ao contrário da abordagem apresentada anteriormente, as técnicas pró-ativas para otimizar algoritmos de *DCOP* para atuar em ambientes dinâmicos utilizam todo o

conhecimento disponível para produzir uma solução, objetivando que esta resista ou se adapte da melhor forma possível a alterações no ambiente. Os métodos desta categoria podem ser divididos em duas famílias distintas:

> Soluções robustas: são métodos que visam construir uma solução que resista a mudanças no ambiente, isto é, continue sendo uma solução mesmo após alterações no problema;
>
> Soluções flexíveis: estas técnicas têm como objetivo encontrar uma solução que possa ser facilmente modificada para se tornar a solução do novo problema.

É possível notar uma clara diferença em relação às informações necessárias para a utilização das abordagens pró-ativas e reativas. Como as abordagens reativas apenas reagem às mudanças do ambiente, não visam construir soluções com informações sobre futuras mudanças. Isto significa que estas técnicas necessitam de poucos dados para sua execução, reduzindo o uso de memória. Por outro lado, os métodos pró-ativos conseguem produzir soluções que podem resistir a mudanças futuras ou flexíveis o suficiente para serem facilmente adaptadas para isto. A desvantagem neste caso é a utilização de memória e tempo computacional elevado para avaliar as combinações produzidas por estes modelos.

2.6. *Métodos para resolução de DynDCOP*

Nesta seção são abordados os algoritmos desenvolvidos para trabalhar com problemas de comportamento dinâmico. Alguns destes foram baseados em técnicas previamente utilizadas em *DCOPs* estáticos. Porém existem métodos fundamentados em outras propostas. A seguir são apresentados alguns dos mais conhecidos algoritmos para resolução de *DynDCOP*, tanto baseados em técnicas estáticas, como desenvolvidos especialmente para esta finalidade.

2.6.1. DynCOAA

Este algoritmo foi desenvolvido para atuar em problemas dinâmicos e com o intuito de resolver o problema de agendamento de entradas e saídas de navios em vários portos do mundo.

O *DynCOAA* (*Dynamic Constraint Optimization Ant Algorithm*) (Mertens, et al., 2006), caracteriza-se por ter sido totalmente desenvolvido para solucionar problemas em ambientes dinâmicos, ao contrário de grande parte dos métodos propostos para este fim, que são baseados em algoritmos que visam atuar em problemas estáticos. Outra característica relevante é a forma de organização das partes do problema. O grafo de restrições é composto por variáveis, seus domínios e as restrições entre elas, e os agentes por sua vez, se movimentam por este grafo com a finalidade de solucionarem o problema.

A idéia principal do algoritmo, é utilizar o princípio da otimização por colônia de formigas (*Ant Colony Optimization – ACO*) (Dorigo, et al., 1997). O ambiente representa o problema a ser solucionado e os agentes (formigas) atuam sobre esse, cooperando entre si. Esta abordagem torna os algoritmos baseados em *ACO* naturalmente adaptados para problemas dinâmicos. Basicamente, a idéia do *ACO* é que os agentes devem produzir retornos positivos para os demais quando uma solução é encontrada. Este retorno é passado indiretamente através dos feromônios, depositados no ambiente. A qualidade da solução está diretamente ligada à intensidade do feromônio empregado pelo agente na solução encontrada. Para evitar que os agentes sigam uma tendência de escolherem o mesmo valor para suas variáveis, as decisões podem ser tomadas de maneira estocástica com alguma probabilidade, ou seja, o *DynCOAA* não pode ser considerado um método completo de resolução de *DynDCOP*, pois é incapaz de garantir a melhor solução.

O algoritmo apresenta uma grande vantagem no que diz respeito a trocas de mensagens e a independência entre os agentes, pois a maior parte das informações necessárias para as tomadas de decisão são armazenadas no próprio ambiente.

2.6.2. DynDBA

Uma estratégia de solucionar problemas de *DynDCOP* amplamente estudada consiste em adaptar algum algoritmo desenvolvido para resolver *DCOP* para atuar em

ambientes dinâmicos. Um dos métodos que surgiram nos estudos foi o *DynDBA*, um algoritmo derivado do *Distributed Breakout Algorithm* (*DBA*) (Yokoo, et al., 1996).

O *DBA* é uma versão otimizada do *Breakout Algorithm* (*BA*) (Morris, 1993), desenvolvida para ambientes distribuídos. Este algoritmo pode ser definido como parcialmente assíncrono, pois agentes vizinhos não podem alterar os valores de suas variáveis concorrentemente. Outro fator considerado é o tratamento de mínimos locais, este definido como um estado onde não existe nenhum valor que resulte em uma solução melhor do que a atual para o problema. Como o *DBA* trabalha de maneira distribuída, os agentes conseguem apenas detectar situações de quase mínimos locais, onde em uma restrição nenhum valor pertencente ao domínio da variável do agente resulta em uma solução de menor custo do que a atual.

No algoritmo, os agentes possuem dois modos de funcionamento, o *wait_ok?* e o *wait_improve?*. No modo *wait_ok?* o agente armazena os valores enviados pelos vizinhos em sua visão local. Após o recebimento de todos os agentes relacionados, é realizada uma avaliação com base nas atribuições contidas em sua visão. Se existir um valor que resulte em uma possível melhora no custo local, o agente então o envia para seus vizinhos e altera seu modo de funcionamento para *wait_improve?*. No modo *wait_improve?* o agente aguarda o recebimento das mensagens do tipo *improve* de todos os seus vizinhos. Se ele constata uma situação de quase mínimo local, são aumentados os pesos das restrições envolvidas. Se o agente recebe a confirmação de que sua atribuição reduz o custo da restrição de forma mais eficaz, este altera o valor de sua variável, envia mensagens de *ok* para seus vizinhos e retorna ao modo *wait_ok?*. Apesar do mecanismo de mensagens apresentar pouca flexibilidade, (Mailler, 2005) propôs a adição de alguns procedimentos para uma atuação mais eficiente do DBA em ambientes onde restrições podem surgir ou desaparecer durante sua execução.

O *Dynamic Distributed Breakout Algorithm* (*DynDBA*) é a versão do *DBA* adaptada para ambientes dinâmicos (Mailler, 2005). Neste método, foram propostos dois novos procedimentos: a adição e remoção de restrições. A adição de restrições consiste em agregar uma nova restrição no problema. Para isso é necessário que os agentes que se relacionam através desta, ordenem os modos de comportamento dos agentes com suas mensagens evitando incompatibilidades na comunicação. O procedimento de remoção de restrições elimina todos os vizinhos e mensagens pendentes da restrição a ser removida do problema.

Apesar de ser adaptado para problemas dinâmicos, o *DynDBA* não utiliza técnicas de *DynDCOP* como reconstrução eficiente de soluções abandonadas através de reuso de soluções e reuso de raciocínio.

2.6.3. S-DPOP

O *S-DPOP* (*Self-Stabilizing Dynamic Programing*) proposto em (Petcu, et al., 2005), trouxe uma nova metodologia no que diz respeito a algoritmos que tratam *DynDCOP*, denominados algoritmos auto-estabilizantes (*self-stabilizing algorithms*). Um algoritmo é considerado auto-estabilizante se sempre consegue convergir para uma solução partindo de qualquer configuração inicial. Este algoritmo é resultado de uma adaptação do *DPOP* para trabalhar com esses conceitos. A seguir são apresentados mecanismos do algoritmo *S-DPOP*.

A idéia central do algoritmo é combinar métodos de geração de pseudo-árvores que mantenham a topologia caso haja alterações no problema, protocolos para propagação de mensagens de utilidade e de valor, agindo concorrentemente com a finalidade de garantir seu caráter auto-estabilizante. Estes métodos têm como objetivo principal reutilizar as informações anteriores para aumentar o desempenho do algoritmo.

A necessidade de um método para geração de pseudo-árvores é reduzir o impacto de uma alteração na topologia do problema. Para isso é desejável que a nova árvore possua um alto grau de semelhança com a do problema anterior, podendo assim ser reutilizada a maior parte possível da resolução do problema passado. A forma utilizada neste algoritmo é a busca em profundidade auto-estabilizante (*self-stabilizing depth-first search*) (Collin, et al., 1994). Para encontrar as informações que ainda permanecem válidas é necessário apenas detectar as semelhanças entre a nova árvore e a do problema anterior. As informações relevantes estão contidas nas mensagens de utilidade enviadas dos filhos para os pais, iniciando a propagação nos nós-folha e concluindo no nó raiz. O procedimento auto-estabilizante para propagação de mensagens de utilidade tem como finalidade evitar que o agente recalcule a matriz com os pesos de cada atribuição, se o algoritmo detectar que o problema atual não apresente nenhuma alteração neste pronto da organização dos agentes. O procedimento auto-estabilizante que trata das mensagens de valor trabalha de maneira semelhante ao que trata das mensagens de utilidade. Os agentes enviam mensagens partindo do nó raiz tendo como destino final os nós-folha,

com o objetivo de analisar atribuições que não necessitam ser reavaliadas, evitando assim desperdício computacional.

O algoritmo auto-estabilizante *S-DPOP* tem como objetivo principal atuar em problemas dinâmicos onde dados quaisquer conjuntos de atribuições ou contextos anexados aos agentes é possível convergir para uma solução ótima. Porém o algoritmo realiza envio de mensagens para certificação de agentes que não necessitam realizar alterações em suas informações, tanto na fase de propagação de utilidade quanto na fase de propagação de atribuição. Outro fator que pode acarretar em perdas no desempenho é a reconstrução da pseudo-árvore a cada alteração no problema, pois em ambiente com muitos agentes a geração de uma nova árvore pode ser consideravelmente dispendiosa. O autor ainda propôs outros métodos derivados do *S-DPOP*, entre eles o *RS-DPOP* (Petcu, et al., 2007) que trabalha de forma análoga ao seu antecessor, porém adaptado para problemas onde as modificações ocorrem em tempos indeterminados.

2.6.4. Dynamic AWC

Uma abordagem para adaptar soluções em ambientes em tempo real foi proposta em (Hattori, et al., 2006), utilizando uma estratégia baseada no *AWC* (*Asynchronous Weak-Commitment*) (Yokoo, et al., 1998), algoritmo para resolução de *DCSP*, onde a idéia é limitar o escopo e reduzir o tempo da busca. Um problema em tempo real pode ser definido como um problema onde mudanças ocorrem constantemente por fatores inesperados durante sua execução. Logo, com estas modificações, as soluções fatalmente se tornam obsoletas no decorrer do processo.

Uma característica importante do *AWC* é a busca da solução ótima local e não global, pois, segundo os autores, em um ambiente em tempo real não é importante uma convergência ótima da solução como um todo, e sim obter um conjunto de atribuições que satisfaz o problema em tempo hábil. Por este motivo o agente que foi adicionado ou sofreu alterações em suas restrições foca sua avaliação apenas em agentes vizinhos, reduzindo sensivelmente a comunicação e tempo de execução. Outra característica do algoritmo é que o agente em questão sempre tenta maximizar a utilidade da atribuição, ou seja, atingir a melhor solução local.

A principal desvantagem deste algoritmo é o elevado número de troca de informações quando o ambiente é caracterizado como um grafo super-restrito, ou seja, um grafo onde cada nó possui relação com todos os demais. Neste caso, o agente modifica-

do envia mensagens para todos os seus vizinhos, ou seja, todos os nós do problema acarretando em perda na eficiência da resolução.

2.6.5. DCDCOP

O algoritmo *DCDCOP* (*Dynamic Complex Distributed Optimization Problems*) desenvolvido por Khanna (Khanna, et al., 2009), propôs a idéia de utilizar índices de insatisfação entre agentes para resolver problemas dinâmicos. Na medida com que os agentes trocam mensagens, analisam esses índices de insatisfação com o objetivo de aceitar ou não as informações para a convergência.

A idéia principal do algoritmo é atribuir grupos de variáveis e através de buscas *branch-and-bound* alcançar uma solução local para o agente. Note que a estratégia de atribuir mais de uma variável do problema por agente é utilizada exclusivamente nesta abordagem. Após a convergência local são trocadas mensagens junto com os índices de insatisfação. Um índice de insatisfação pode ser definido como uma medida para saber o quão longe um agente está de um estado consciente. Em outras palavras, este índice demonstra qual o custo da solução local. Quando um agente recebe uma mensagem, ele compara o índice de insatisfação com seu próprio e caso receba um índice menor, aceita as informações e as utiliza para reavaliar suas variáveis. Caso o índice recebido seja maior que o atual, ignora a mensagem. Com este algoritmo, é encerrada a apresentação dos principais métodos encontrados na literatura para resolução de *DynDCOPs*. Porém, é interessante fazer um comparativo entre as abordagens apresentadas neste capítulo com a finalidade de analisar as estratégias adotadas em cada algoritmo.

2.7. *Comparação entre métodos de resolução DynDCOP*

Conforme descrito anteriormente, após a discussão sobre os algoritmos para resolução de *DynDCOP* mais relevantes encontrados na literatura, é possível estabelecer uma comparação entre as abordagens. Para esta comparação foram utilizados alguns parâmetros, são estes: estratégia adotada, abordagem reativa ou pró-ativa, sincronismo, e se o algoritmo é completo ou não. É importante lembrar que a abordagem, reativa ou pró-ativa, não diz respeito ao comportamento dos agentes e sim sobre a metodologia utilizada para tratar problemas dinâmicos, definidos na seção 2.5 deste capítulo. Outro detalhe importante é que mesmo sendo previamente definido que não seriam apresentados algoritmos síncronos neste estudo, nem todos os discutidos possuem características

completamente assíncronas. A seguir, na Figura 6, é apresentado a tabela que faz a comparação entre os algoritmos.

Algoritmo	Estratégia	Abordagem	Sincronismo	Completo
DynCOAA	Utiliza a estratégia de colônia de Formigas (agentes), percorrendo o ambiente com a finalidade de alcançar a solução (ambiente).	Reativo	Assíncrono	Não
DynDBA	Utiliza procedimentos de interpretação e tratamento de mensagens devido ao comportamento dos agentes, definidos pelos modos de funcionamento *wait_ok?* e *wait_improve?*.	Reativo	Parcialmente assíncrono	Sim
S-DPOP	Faz uso de protocolos de checagem de valor e utilidade agindo concorrentemente, aliado à rotinas auto-estabilizantes para geração de árvores semelhantes às anteriores.	Reativo	Assíncrono	Sim
Dynamic AWC	Limita o escopo e reduz o tempo de busca do algoritmo, objetivando alcançar a solução local ao invés da solução global do problema.	Reativo	Assíncrono	Não
DCDCOP	Utilizar mecanismos de busca *branch-and-bound* para convergir para uma solução local interna ao agente, aliado com trocas de informações entre agentes utilizando índices de insatisfação como métrica da qualidade dessas mensagens.	Reativo	Assíncrono	Sim

Figura 6 - Comparação entre algoritmos *DynDCOP*

Na Figura 6, é possível observar algumas características em comum a todos os algoritmos estudados. Todos trabalham baseados em abordagens reativas, isto é, não fazem uso de rotinas que avaliam futuras alterações para a escolha da solução. Apesar do algoritmo *S-DPOP* conter em sua metodologia as árvores auto-estabilizantes, estas não realizam nenhuma avaliação para construir uma árvore que suporte futuras alterações. Outra característica comum entre os algoritmos é o fato de todos possuírem características assíncronas no comportamento de seus respectivos agentes. Com exceção do algoritmo *DynDBA*, todos os demais métodos são totalmente assíncronos. A característica mais interessante desta análise é a completude dos algoritmos. Com exceção do *DynDBA*, que não utiliza nenhum procedimento de reutilização de informações previamente obtidas para uma nova convergência, todos os demais métodos completos para resolução de *DynDCOP*, utilizam grandes rotinas adicionais ou avaliação externa das soluções ou informações para garantir a completude do algoritmo, sendo o *S-DPOP*

com protocolos concorrentes de checagem de valor e utilidade e o *DCDCOP* com os índices de insatisfação de mensagens trocadas, aliado ao agrupamento de variáveis em agentes. Logo, é possível observar a carência de métodos completos para resolução de *DynDCOP* que não possuam rotinas e avaliações adicionais afim de garantir o caráter ótimo do algoritmo.

2.8. *Considerações finais*

O capítulo discorrido, apresentou as definições de *DynDCOP*, bem como suas origens e formalizações. Foram discutidos também, algoritmos e métodos para resolução de problemas, tanto para *DCOP* como para *DynDCOP*. A seguir serão abordados os aspectos da proposta de elaborar um algoritmo para problemas dinâmicos. Também serão discutidos os casos de utilização tratados, objetivos, formas de validação, aplicações e resultados.

3. DYNADOPT

Após a discussão sobre métodos de formalização e resolução de *DCOP* e *DynDCOP*, este capítulo se dedica a apresentar o novo algoritmo proposto. Este, conforme mencionado anteriormente, tem como principal finalidade agir em ambientes com restrições dinâmicas, isto é, que podem sofrer alterações ao longo do tempo. Este capítulo discute os princípios fundamentais do *DynADOPT* e também sua implementação, seu funcionamento, vantagens, desvantagens e casos de utilização.

3.1. *Princípios do DynADOPT*

O *DynAdopt* (*Dynamic Distributed Constraint OPTimization)* é um algoritmo baseado no método *ADOPT* (Modi, et al., 2003) e surgiu como uma adaptação especializada para o tratamento de problemas com restrições dinâmicas. A escolha do *ADOPT* além de ser um pré-requisito para o nosso trabalho está baseada nas características que beneficiam a sua adaptação para trabalhar em problemas dinâmicos, tais como flexibilidade da arquitetura, baixo esforço computacional, funcionamento simples, comunicação local e caráter assíncrono. A característica de flexibilidade da arquitetura considera que o algoritmo não possui uma ordem fixa de comunicação entre os agentes, nem fases pré-definidas durante a sua execução como é o caso dos algoritmos *DPOP* e *OPT-APO*. O baixo esforço computacional se deve às rotinas de cálculo presentes nos agentes, as quais contêm poucos passos. O funcionamento simples do algoritmo por sua vez, diz respeito à sua estrutura geral do comportamento dos agentes, enumerando poucos tipos de mensagens e informações armazenadas. A comunicação de informações locais feitas pelo algoritmo também contribui positivamente para a sua adaptação, pois facilita a compreensão do algoritmo e impede a propagação de atribuições e contextos desnecessários. E por fim o caráter assíncrono do *ADOPT* se mostra como um fator chave para resolução de problemas dinâmicos, pois o fato de não existir uma seqüência de execução entre os agentes, permite que o algoritmo trabalhe diretamente nos subespaços de busca onde há necessidade. Outro fator relevante para a escolha é devido a sua implementação estar previamente concluída pelo autor do presente trabalho, facilitando a elaboração do *DynADOPT*.

A idéia central da proposta consiste em aproveitar o máximo possível de informações obtidas na resolução do problema no instante anterior à mudança do ambiente.

Logo, é interessante que variáveis e restrições que não sofreram alterações permaneçam estáveis, aumentando assim a eficiência e diminuindo o esforço empregado em analisar regiões desnecessárias. Para isso, é essencial focar os esforços no tratamento das novas arestas do problema com uma estratégia que não entre em conflito com a filosofia do algoritmo.

A técnica mais apropriada para a adaptação consiste basicamente em reiniciar a execução do algoritmo em agentes estrategicamente posicionados em relação à restrição que sofreu algum tipo de modificação, mantendo as visões, custos e atribuições previamente obtidos. Esta abordagem além de manter a estratégia original do algoritmo, permite que o sistema aproveite as informações previamente armazenadas pelos agentes, evitando assim a propagação de valores e contextos desnecessários. Para isso, o objetivo proposto conta com as visões atuais dos nós, possibilitando que estes alcancem a solução com mais eficiência. No *ADOPT* original, quando o problema é reiniciado, as informações dos contextos previamente obtidos são abandonadas, tornando a execução mais trabalhosa. Estes dados são essenciais para uma convergência eficiente, pois carregam os valores dos agentes que se relacionam entre si, permitindo que cada um avalie o custo de sua própria variável com um conhecimento maior do ambiente. Outra vantagem é a possibilidade de uma alteração no problema atingir agentes estabilizados apenas se necessário. Depois de abordadas as fundamentações do *DynADOPT* são apresentados os detalhes de implementação da proposta.

3.2. *Implementação*

A implementação do *DynADOPT* segue a estrutura básica do *ADOPT* no que diz respeito ao comportamento dos agentes e ao sistema de mensagens assíncronas. A primeira execução do algoritmo é extremamente semelhante ao algoritmo original. Os agentes previamente hierarquizados atribuem um valor inicial às suas respectivas variáveis, e através das trocas de mensagens assíncronas com valores, visões e custos com seus vizinhos diretos, que possuem uma restrição direta com o agente, chegam a uma convergência global, alcançando a solução com o menor custo. Neste momento, o do estado final da execução, os agentes possuem todas as informações sobre os custos de seus vizinhos diretos em sua visão local. Caso haja a necessidade de uma nova resolução do problema, ou seja, a solução encontrada não é mais válida, o *ADOPT* reinicia sua execução reiniciando todas as estruturas mantidas pelos agentes, perdendo todo o co-

nhecimento sobre sua convergência. A idéia do *DynADOPT* é iniciar uma nova resolução tomando como ponto de partida o estado final já conhecido do problema. Conforme apresentado anteriormente, esse estado é mais próximo da nova solução que o estado inicial (Verfaillie, et al., 2005). Com a finalidade de demonstrar uma maior clareza sobre o funcionamento do *DynADOPT*, é apresentado na Figura 7, o pseudocódigo do algoritmo.

```
#   xi: variável local
#   di: valor atual da variável de xi
#   c(d): lower bound atual da pseudo-árvore iniciada no agente
    filho se xi escolher d

0   Inicio
1     se FirstResolution é verdadeiro
          CurrentContext ← {};
2         ∀d ∈ Di, faça
3             c(d) ← 0;
4         fim faça; fim se;
5     BACKTRACK;

6   Ao receber (VALUE,(xj,dj))
7     adiciona (xj,dj) em CurrentContext;
8     se context(d,xi) é incompatível
           com CurrentContext
9         ∀d ∈ Di, faça
10        c(d) ← 0;
11        fim faça; fim se;
12    BACKTRACK;

13  Ao receber (View,context,cost)
14    d ← valor de xi em context;
15    se context é compatível
           com CurrentContext U (d,xi)
16      c(d) ← max(c(d),cost);
17      se c(d) foi alterado
18        BACKTRACK;
19      fim se; fim se;

20  BACKTRACK
21    ∀d ∈ Di, faça
22    e(d) ← EstimaMenorCusto(xi, CurrentContext U (xi, d) +
    c(d));
23    Escolha d que minimize e(d);
24      di ← d;
25    fim faça;
26    ENVIA(VALUE(xi, di))) para todos os descendentes conecta-
    dos
27    ENVIA(VIEW(CurrentContext, e(di)) para o pai
```

Figura 7 – Pseudocódigo do *DynADOPT*

Analisando a Figura 7 é possível notar que o *DynADOPT* possui uma estrutura bastante similar ao *ADOPT*, com exceção da rotina de inicialização, específica do *DynADOPT*, que evita o desperdício de informação. Ambos são divididos nos seguintes procedimentos: inicialização (linha 0), recebimento de *VALUE* (linha 6), recebimento de *VIEW* (linha 13), e subida da encosta (linha 20). Estes procedimentos tal como a execução do *ADOPT* encontra-se detalhados no capítulo de fundamentação teórica desde documento com exceção da rotina de inicialização. Esta possui uma condição especifica

para o *DynADOPT* que possibilita o início da resolução de um novo problema sem a necessidade de excluir todas as informações presentes nas estruturas dos agentes obtidos em suas execuções anteriores. Na Figura 8 é apresentada a rotina de inicialização do *DynADOPT*. Nela é possível observar a condição que impede a remoção das informações obtidas execuções anteriores (linha 1). Caso o algoritmo se encontre na primeira execução do problema, todas as estruturas dos agentes são reiniciadas e os custos estimados zerados. Caso contrário, a resolução atual é proveniente de uma alteração no problema antigo, estes dados são mantidos, e a rotina de subida da encosta é chamada, aonde acontece a avaliação de todas as restrições dos agentes Quando uma função de custo tem seus pesos alterados, automaticamente são explorados. Caso esta possibilite uma nova atribuição mais vantajosa, é enviada aos vizinhos. Se não houver modificações relevantes, o algoritmo simplesmente ignora a avaliação e mantém seu valor e seu custo. Esta estratégia permite que os agentes não necessitem de uma orientação sobre onde se localizam as modificações, poupando os esforços de uma estrutura externa para tal propósito.

```
0   Inicio
1     se FirstResolution é verdadeiro
          CurrentContext ← {};
2         ∀d ∈ D_i, faça
3             c(d) ← 0;
4         fim faça; fim se;
5     BACKTRACK;
```

Figura 8 - Rotina de incialização do *DynADOPT*

Com os pseudocódigos devidamente formalizados e descritos, é possível apresentar e discutir a implementação do sistema de avaliação do *DynADOPT*, juntamente com as estruturas adjacentes de geração de grafos e modificação de arestas. A Figura 9 demonstra o diagrama de classes do sistema desenvolvido. Nele é notável notar que a principal estrutura utilizada no projeto foi a do agente do algoritmo *ADOPT*. Isso se deve ao fato do novo algoritmo estar baseado neste, onde toda a sua estrutura, bem como sistema de mensagens, estratégia, comunicação e organização dos agentes permanecerem as mesmas. Para o envio de mensagens entre os nós do problema, é utilizada uma caixa de mensagens, visível a todos os agentes onde é possível consultar e publicar as devidas mensagens para os demais, seguindo a filosofia original do *ADOPT*. Outro detalhe interessante é a estrutura ambiente, utilizada pelos algoritmos. Nela estão localizadas as listas agentes, funções de custo e o grafo que representa o problema formando

uma espécie de ambiente comum, permitindo que um objeto devidamente capacitado, possa interagir com o grafo do problema tanto para sua geração como para sua modificação. As classes que definem esses objetos de alteração das arestas são o *ModificadorDeRestricoes*, e suas classes especializadas, *AdicionadorDeRestricoes* e *RemovedorDeRestricoes*. Como todos os elementos do problema estão situados no ambiente, os mecanismos de alteração das arestas atuam diretamente sobre as estruturas do grafo e funções de custo. Dessa forma os dados e informações contidos nos agentes não sofrem quaisquer modificações.

Assim como as modificações do problema, a classe responsável pela geração de grafos aleatórios também atua sobre o ambiente. Basicamente, o *GeradorGrafo* constrói uma matriz quadrada, atribuindo valores diferentes de zero que representam as ligações entre os agentes do problema. Dessa forma, é possível estabelecer uma indexação para as funções de custo e indicar sua localização no grafo através de seu identificador.

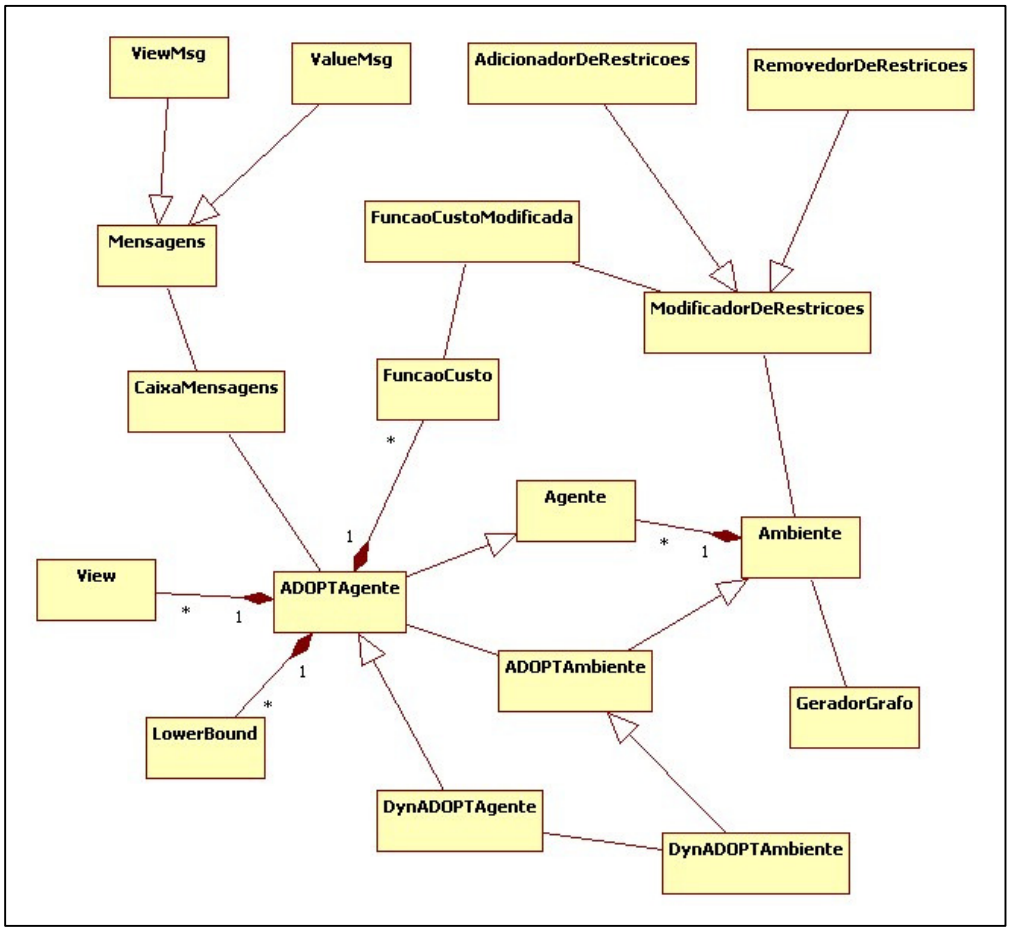

Figura 9 - Diagrama de classes do sistema *DynADOPT*

Com o objetivo de oferecer uma visão sistêmica de todo o sistema de simulação utilizado, é apresentado na Figura 10 o diagrama de atividades, que demonstra todo o fluxo de dados empregado. Inicialmente é gerado o grafo que será utilizado como base para a execução total. Logo após são criadas as instâncias dos agentes e funções de custo, possibilitando a geração da pseudo-árvore, essencial para a resolução correta do problema pelo *DynADOPT*. Com os elementos necessários criados e carregados, é dado o início à resolução do problema. Como o *DynADOPT* baseia todo o comportamento dos agentes no *ADOPT*, e este não possui detecção de término, a solução utilizada para esta questão foi a de monitorar quando os agentes deixam de enviar mensagens, em outras palavras, encontram-se estabilizados. Quando a solução é encontrada, é analisado se o sistema alcançou o fim de sua e execução, caso contrário são chamadas as rotinas para a alteração do problema.

A fase de alteração do problema inicia com a avaliação do número de arestas que deverão ser processadas nesta etapa. É escolhida uma função de custo que ainda não tenha sido alterada neste processamento e modificada, conforme os parâmetros previamente informados. Esta rotina prossegue elegendo e alterando as arestas do problema até que o número estimado de funções de custo modificadas para esta etapa seja alcançado.

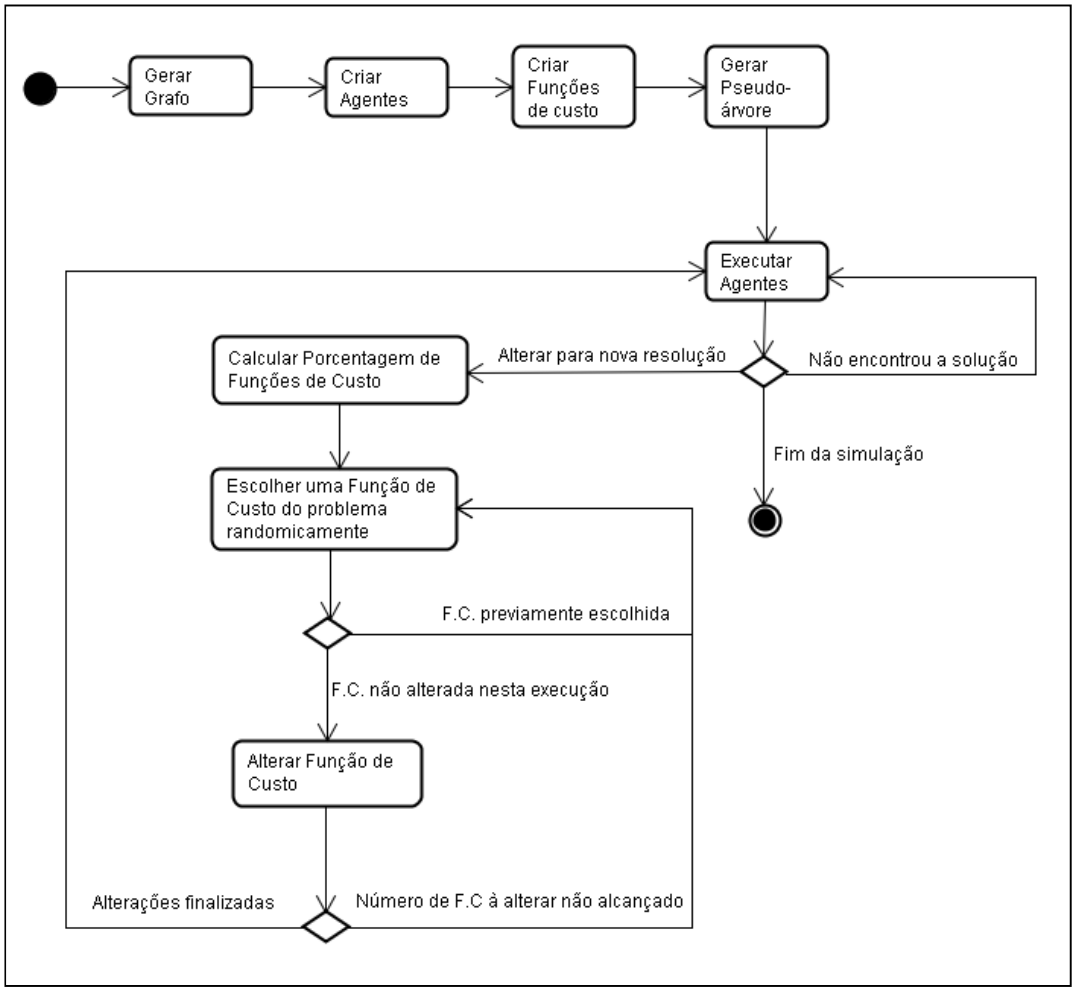

Figura 10 - Diagrama de Atividades do Sistema

Um detalhe importante sobre as rotinas de modificação é que cada aresta do problema pode ser modificada apenas uma vez a cada nova resolução. Isso é devido à forma de parametrização de como as alterações são controladas, informando a porcentagem desejada. Os pseudocódigos das rotinas de geração de grafos e modificação de funções de custo encontram-se definidas e detalhadas no capítulo de resultados e experimentos deste trabalho. Com os critérios de implementação devidamente detalhados, são apresentados na seqüência os casos de utilização do *DynADOPT* abordados no projeto e suas características.

3.3. *Casos de utilização*

A idéia da utilização do *DynADOPT* parte do princípio que o problema no momento em que sofre algum tipo de alteração está estabilizado, ou seja, sua solução foi

obtida. Desta forma a nova resolução sempre parte de um estado próximo ao ótimo, permitindo que locais onde não houveram alterações, a princípio, permaneçam estabilizados. É importante ressaltar que casos onde ocorrem alterações em problemas não-estabilizados não serão abordados neste trabalho. Outro fator importante no momento é definir cada uma das modificações do ambiente detalhadamente.

As alterações consideradas válidas no trabalho são apenas aquelas onde envolvem as restrições do problema. Especificamente foram estudadas alterações no custo das atribuições de uma restrição e também sua remoção por completo. Adição e remoção de agentes, atualização nos domínios dos mesmos ou quaisquer eventos relacionados aos nós (agentes) presentes no grafo de restrições do problema não são cobertos por este projeto. Também não é abordada a adição de novas restrições no grafo *DynDCOP*.

A primeira modificação estudada diz respeito às funções de custo, onde os pesos para atribuições são atualizados ou ocorre adição e remoção de novas condições entre variáveis no problema. Em outras palavras algumas atribuições entre duas variáveis no problema representam um custo tanto local como global diferente do problema anterior.

Tabela 1 - Função de custo dinâmica

xi	*xj*	*f(xi, xj)*
0	0	1
0	1	2
1	0	2
1	1	0

(a)

xi	*xj*	*f(xi, xj)*
0	0	<u>0</u>
0	1	2
1	0	2
1	1	<u>1</u>

(b)

No caso das funções de custo sofrerem alterações, não existe a necessidade da geração de uma nova árvore, pois a topologia do problema não se altera. As Tabelas 1.a e 1.b apresentam um exemplo desde caso. É possível notar que a modificação acontece apenas no custo do conjunto de atribuições contido na definição da função. Um detalhe importante a ser ressaltado é de que o domínio não se altera, pois esta condição está fora do escopo do projeto. A Figura 11 demonstra um caso onde a função de custo do problema representado se altera.

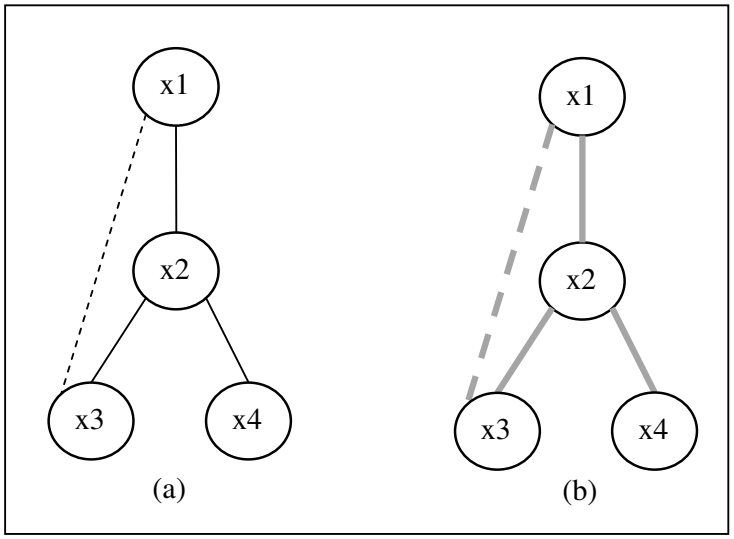

Figura 11 - Grafo com função de custo dinâmica

Na Figura 11.a é mostrada a pseudo-árvore de um *DCOP* que utiliza a função de custo apresentada na Tabela 1.a. Analogamente, a Figura 11.b utiliza a função definida na Tabela 1.b. Analisando a figura é possível concluir que este tipo de alteração em um *DCOP* não altera a topologia de sua estrutura, conseqüentemente, não existe a necessidade da geração de uma nova árvore para este caso. Outra conclusão que pode ser obtida na análise é o forte impacto que este tipo de modificação acarreta no problema, necessitando uma grande troca de valores dos agentes para atingir um estado ótimo. Graças ao comportamento dinâmico e inteligente do *DynADOPT* de manter o conhecimento prévio do problema, sua utilização apresenta um grande desempenho para este caso de dinamismo em *DynDCOP*, assim como os demais contidos na seqüência.

Outro caso de alteração considerada no estudo é a remoção de uma restrição do problema. Nesta situação, a proposta consiste em anular a função de custo da determinada aresta onde não existe mais uma condição, ou seja, todos os custos da função passam a ser nulos (valor *0*) e não apresentam mais um impacto direto na solução local e global do problema. A idéia desta abordagem é a de evitar a reconstrução da pseudo-árvore aumentando a eficiência do algoritmo. O raciocínio parte do princípio de que, se não existe uma restrição entre duas variáveis, estas podem assumir quaisquer valores sempre resultando em um custo nulo (zero). Outro detalhe é que o agente não perde conexão com o resto do problema caso haja alguma outra restrição, podendo agir normalmente em relação a outros nós do problema cujo qual se relacione. Conforme descrito anteriormente, os pseudocódigos referentes às rotinas de modificações das arestas dos

problemas estão definidos no capítulo de experimentos do documento. Um exemplo de função de custo nula é apresentado na Tabela 2. É notável a possibilidade de ambos os agentes envolvidos atribuírem quaisquer valores para suas respectivas variáveis sem acréscimo no custo da solução.

Tabela 2 - Função de custo nula

xi	*xj*	*f(xi, xj)*
0	0	0
0	1	0
1	0	0
1	1	0

Conforme mencionado, esta alteração não necessita da reconstrução da pseudo-árvore, pois a topologia do problema é mantida. Esta afirmação pode ser observada na Figura 12. Analisando a figura, é possível perceber que as demais relações entre os agentes que contém restrições anuladas permanecem válidas. Nós com apenas uma aresta como o *11*, continuam conectados ao problema, porém, não contém mais a capacidade de impor um custo ao mesmo. A possibilidade de descartar o agente que não influi no grafo de restrições não é considerada, pois se encontra fora dos objetivos do trabalho. É importante destacar que os custos da função objetivo tornam-se nulos apenas na aresta entre os dois agentes afetados pela remoção da aresta em questão. Outro detalhe interessante é que mesmo que as modificações das arestas sejam locais, permitem que os agentes explorem um espaço de estados maior, pois agentes que possuem relações com diversos agentes no problema, como o agente *7*, tendem a realizar atribuições com um grau de complexidade menor. Este tipo de alteração é chamado na literatura de relaxação do grafo (Bessière, 1991).

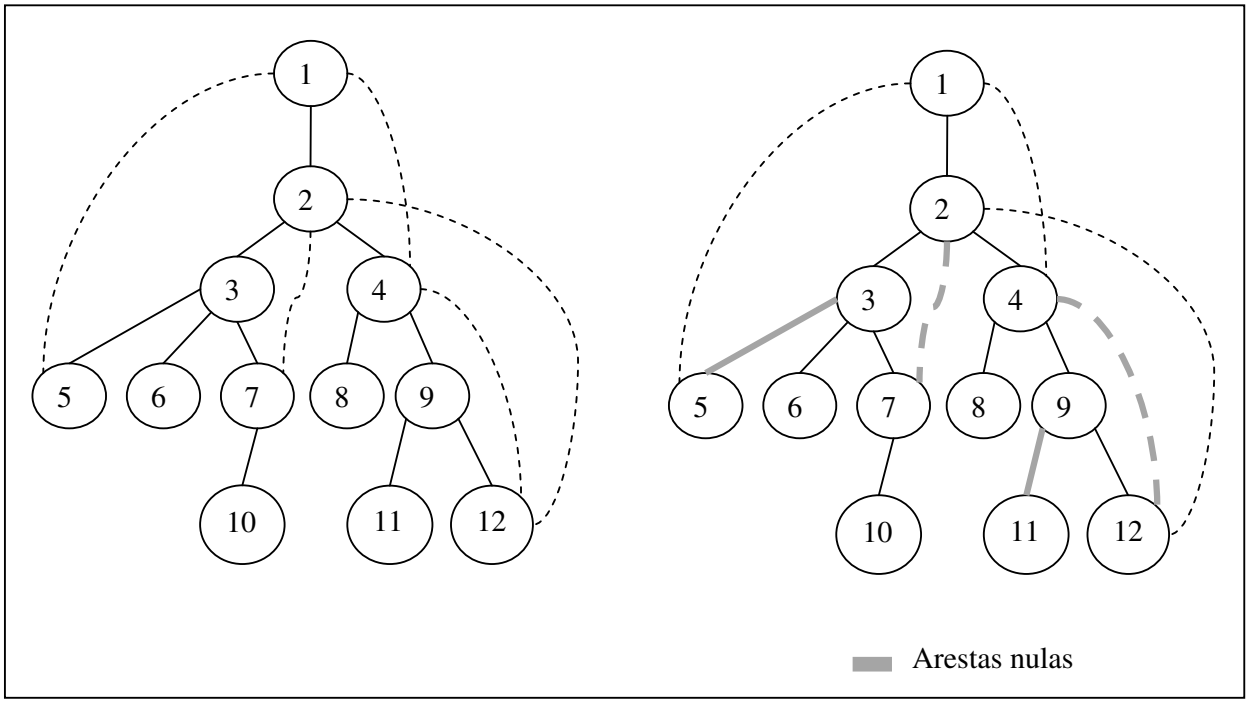

Figura 12 - Grafo com restrições anuladas

Apesar de não considerada como requisito para o desenvolvimento deste trabalho, nós apresentamos a seguir um tipo de alteração não tratada pela nossa abordagem: a adição de novas restrições. Esta diz respeito à inclusão de novas restrições no grafo (Bessière, 1991). Esta alteração é considerada um caso particular, pois em muitos casos, a pseudo-árvore se torna um grafo com a inclusão de novas arestas, perdendo a hierarquia e fluxo de mensagens entre os agentes, tornando-se inconsistente e sendo necessária a sua reconstrução. Nas Figuras 13 e 14, um exemplo da adição de uma nova restrição do problema e o impacto causado pela mesma é apresentado.

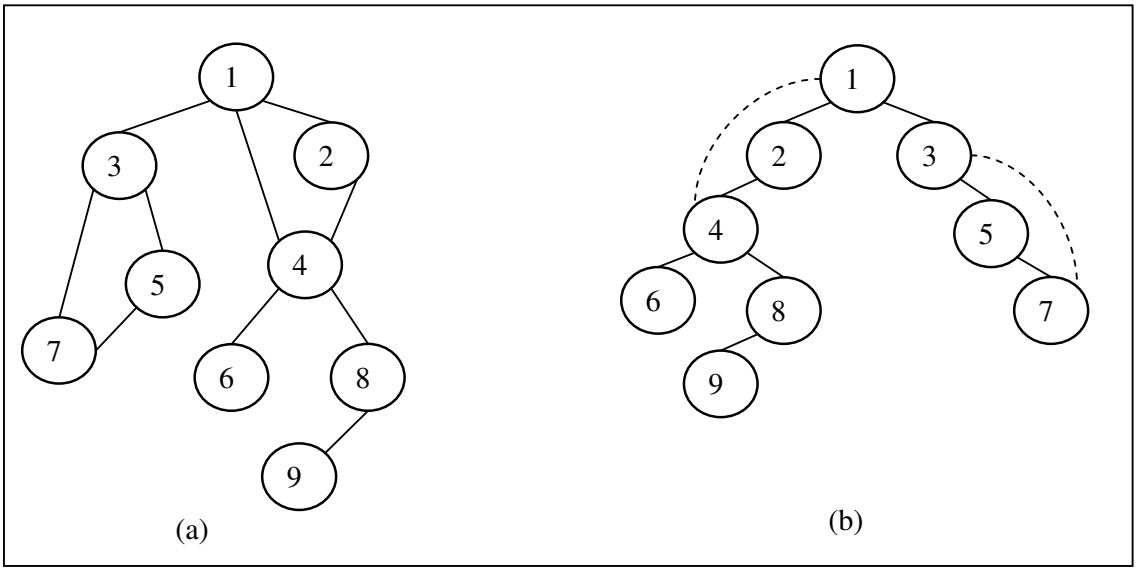

Figura 13 - Problema *DCOP*

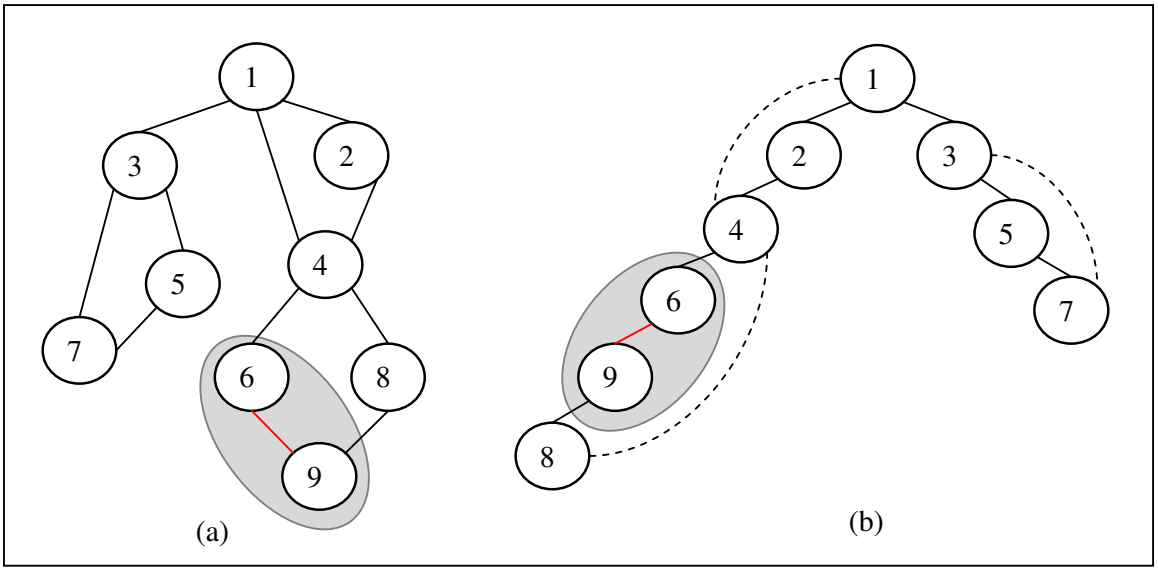

Figura 14 - Grafo DCOP com restrição adicionada

Nas Figuras 13.a e 13.b, é apresentado um grafo de restrições de um problema *DCOP* e sua respectiva pseudo-árvore. A necessidade da reconstrução da árvore é atestada nas Figuras 14.a e 14.b, onde esta é visivelmente diferente da gerada a partir do problema original. A restrição adicionada localiza-se entre os agentes *6* e *9*, causando um ciclo na região do grafo. Este caso introduz uma dificuldade adicional em relação aos demais casos propostos. O agente *9* no problema anterior tinha como pai o agente 8 e com a adição de uma nova aresta no novo grafo tornou-se filho de *6.* Toda sua visão

local está baseada em uma situação hierárquica do grafo em que não existe o novo cenário. Portanto além do problema de uma visão inválida em termos de contexto, existe uma não-conformidade também se tratando da estrutura em que o conhecimento foi obtido, não sendo possível um aproveitamento por parte do algoritmo *DynADOPT*.

Para uma melhor compreensão sobre a execução do algoritmo e o seu comportamento perante os casos apresentados, são discutidos nas seções seguintes os detalhes de execução do novo algoritmo proposto.

3.4.*Execução*

Uma vez apresentadas as idéias principais, pseudocódigos, rotinas de funcionamento e casos de utilização abordados é possível apresentar e analisar um exemplo de execução do algoritmo *DynADOPT*. Para uma maior facilidade de compreensão o caso proposto para esta discussão será baseado no exemplo utilizado na apresentação do *ADOPT*, mostrado a seguir.

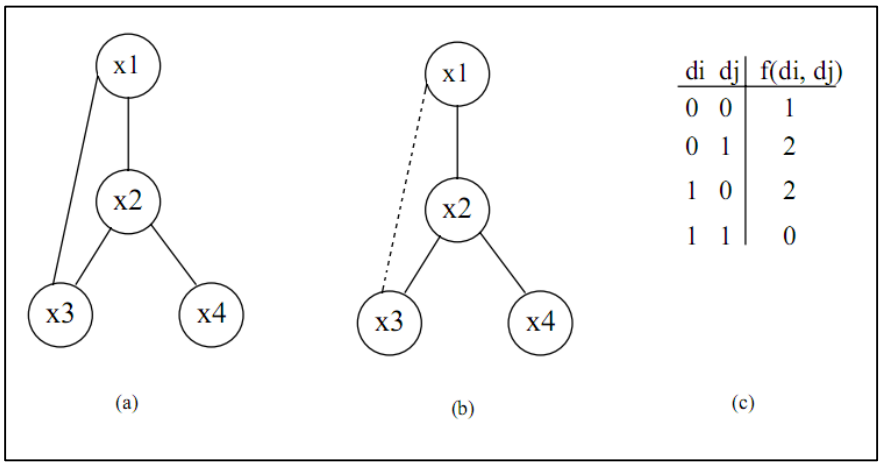

Figura 15 - Exemplo de problema *DCOP*

O caso apresentado na Figura 15.a consiste em encontrar uma solução para o grafo com quatro agentes. Conforme dito anteriormente o *DynADOPT* também se utiliza dos métodos de geração de pseudo-árvore para a sua execução (Figura 15.b). Finalmente na Figura 15.c é definida a função de custo vigente do problema. A próxima figura demonstra detalhadamente a resolução do problema proposto.

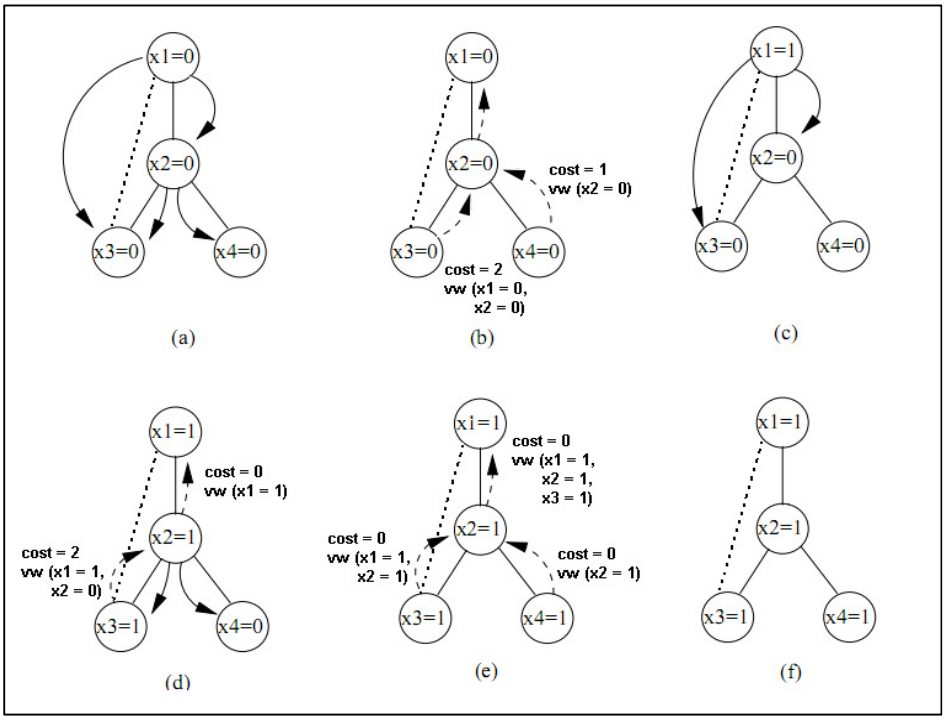

Figura 16 - Exemplo da execução inicial do *DynADOPT*

Analisando a Figura 16 nota-se com facilidade que a execução e resolução seguem de forma semelhante ao *ADOPT*. Isso se deve ao fato de que na inicialização do *DynADOPT*, os agentes não dispõem de nenhuma informação sobre os demais envolvidos no problema. Logo, após as sucessivas mensagens trocadas, a convergência é obtida, resultando no custo global da solução *1*. Para uma ampla compreensão do funcionamento do algoritmo nos cenários abordados no trabalho, serão apresentados dois diferentes tipos de alterações impostas, baseados neste mesmo problema, cada um com características diferentes, conforme os casos de utilização descritos anteriormente.

3.4.1. Modificação de restrições

O primeiro comportamento dinâmico discutido modifica os custos das tuplas das restrições entre duas variáveis d_i e d_j, controladas por dois agentes distintos. Um detalhe a ser definido para maior clareza nos modelos presentes na seqüência é o direcionamento das arestas das funções de custo, definido por $f(d_i, d_j)$, onde $i < j$. Em outras palavras, a restrição entre os agentes x_1 e x_3 é sempre avaliada por $f(d_1, d_3)$.

Dando continuidade à execução do algoritmo, presente nas Figuras 15 e 16, depois que a solução inicial foi encontrada, o problema resolvido sofreu uma alteração em

suas restrições e originou um novo caso onde a solução não é mais válida, conforme demonstrado na Figura 17.

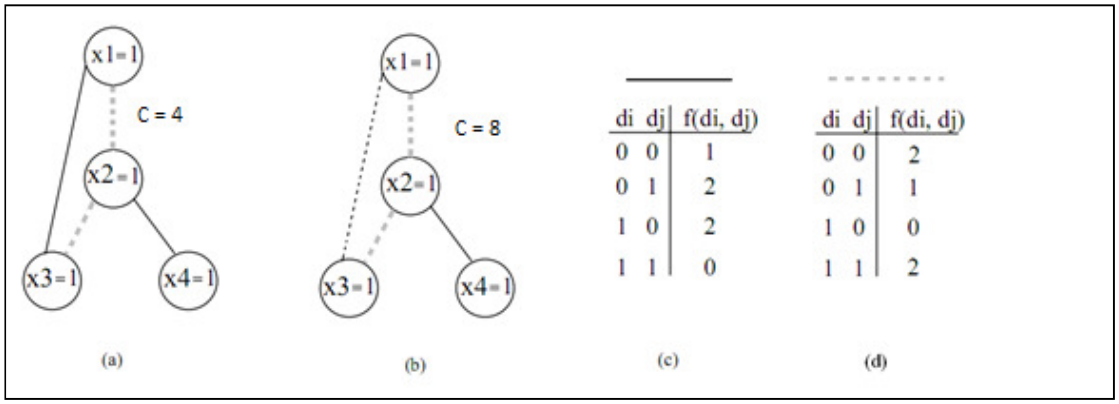

Figura 17 - Exemplo de problema *DCOP* modificado

De acordo com a Figura 17, é possível observar uma alteração em algumas arestas do problema resultando de uma nova função de custo, vigente em duas das quatro arestas do problema. Devido às combinações resultarem na alteração do custo global de 4 para 8, a solução antiga não é mais válida, necessitando uma nova execução.

Apesar do conjunto de atribuições não ser mais o que representa a melhor solução, toda visão dos agentes sobre seus vizinhos, ainda pode ser aproveitada pelo *DynA-DOPT* para facilitar a busca da nova resolução. A seguir é apresentada a execução do algoritmo, dado um problema inicial previamente resolvido (Figura 17.a) e o novo cenário, originado do problema anterior com uma modificação atribuída em suas arestas (Figura 17.b).

Os agentes do problema iniciam a nova execução com o valor da última solução obtida. Isso é essencial para o funcionamento correto do algoritmo, pois é este valor que os demais agentes possuem em suas respectivas visões locais e utilizam para reavaliar as suas restrições com os devidos vizinhos. Em seguida os agentes iniciam o processo de reavaliação de suas restrições e caso encontrem algum custo inválido, enviam mensagens para seus respectivos vizinhos com os novos valores para seus pesos e atribuições. De forma análoga ao *ADOPT*, a execução deste algoritmo é assíncrona e devido a esta característica, existem diversas seqüências de execução possíveis, logo a Figura 18 apresenta uma dessas possibilidades.

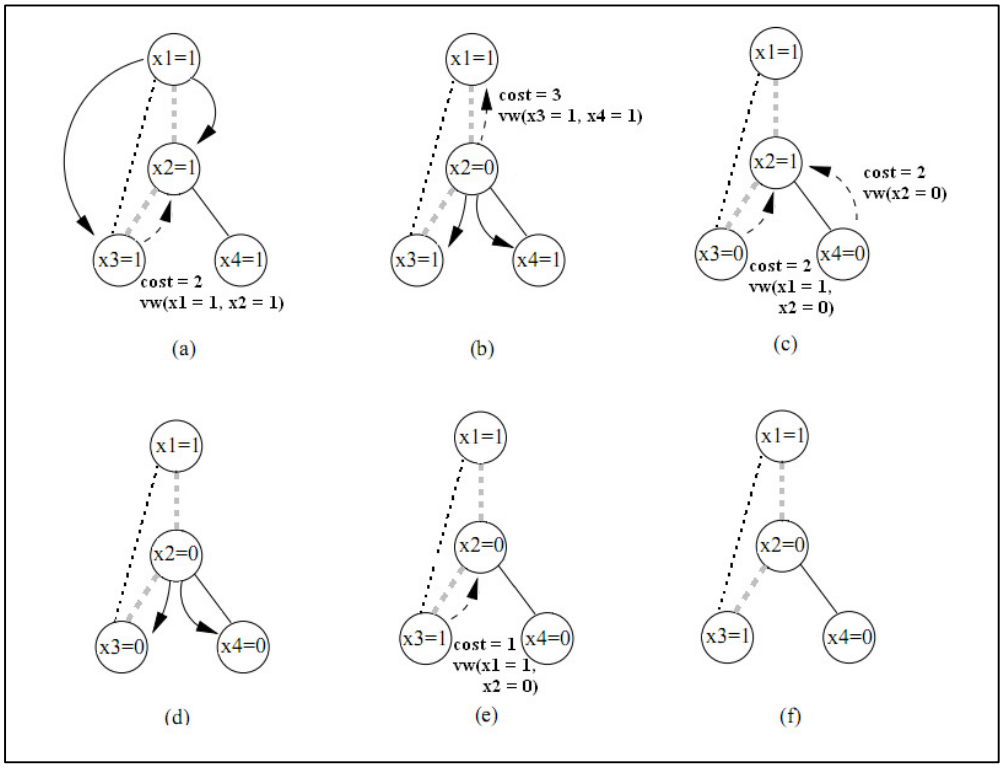

Figura 18 - Exemplo da execução do *DynADOPT* em problemas de modificação de restrições

O algoritmo inicia nova execução com os últimos valores válidos para cada variável conforme mencionado anteriormente sendo todos com o valor *1* (Figura 18.a). Os agentes então reavaliam suas restrições caso encontrem novos custos a serem explorados emitem mensagens para seus vizinhos, seguindo a idéia original do *ADOPT*. Na Figura 18.b é possível observar que o agente x_2 alterou o valor de sua variável e propagou o custo *3* para x_1 e enviando o novo valor para seus filhos x_3 e x_4. Esta alteração se deve às novas funções de custo que conectam x_2 aos demais agentes x_1 e x_3. Ainda no exemplo, na Figura 18.c, x_3 alterou seu valor em conseqüência da mensagem recebida de x_2, enviando essa alteração com a sua devida visão para seu pai, assim como x_4. Estas mensagens, acarretam na nova modificação da variável controlada por x_2 (Figura 18.d), propagando novamente para seus filho x_3 e x_4, até a convergência do algoritmo (Figuras 18.e e 18.f), originando uma solução válida para o novo problema, sem a necessidade de uma execução completa, ou seja, partida do estado inicial (sem nenhum conhecimento prévio do problema). A próxima situação abordada, é a de restrições que ao invés de modificarem os custos das possíveis atribuições, as anula por completo, resultando no

desaparecimento da aresta no que diz respeito ao custo local e conseqüentemente global de problemas *DynDCOP*.

3.4.2. Remoção de restrições

O outro caso estudado neste trabalho é observar como o algoritmo proposto trabalha em ambientes que possuem uma parte das arestas removidas ao longo das sucessivas execuções. Para isto foi necessário elaborar uma metodologia com a finalidade da representação deste cenário sem alterar a estrutura em que os agentes se encontram. A solução encontrada é reduzir para zero todos os custos das restrições selecionadas. Desta forma qualquer conjunto de atribuições não representará nenhum impacto perante a solução global do problema. A Figura 19, demonstra uma situação onde duas restrições do problema contido na Figura 15 são removidas. É possível notar que as funções de custo x_1-x_2 e x_2-x_3 não impactam mais no custo da solução, pois quaisquer atribuições efetuadas pelos agentes conectados por estas arestas acarreta no custo local 0. É possível observar também que apesar do fato das restrições não existirem mais do ponto de vista de custos, ainda preservam a estrutura principal do grafo permitindo que os agentes mantenham a mesma hierarquia e o fluxo de informação original.

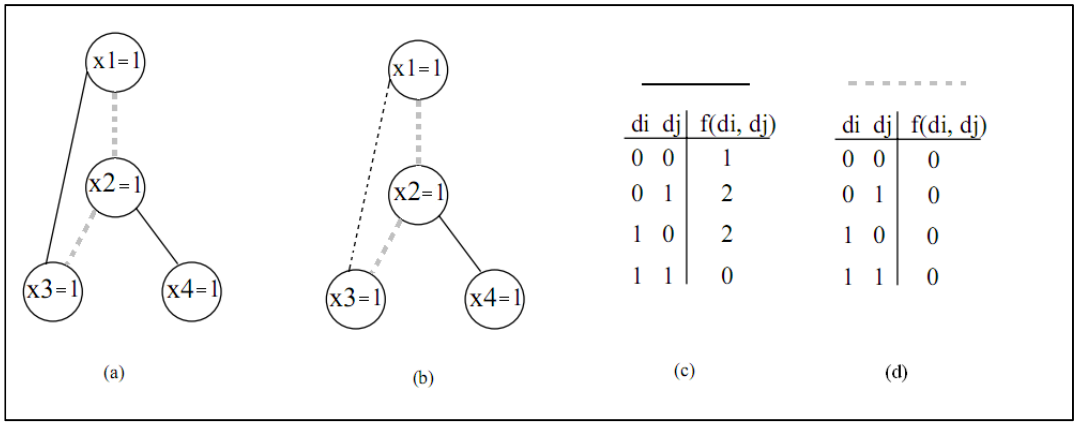

Figura 19 - Problema *DynDCOP* com remoção de restrições

A Figura 20 apresenta um exemplo da execução do DynADOPT, considerando o caso abordado na Figura 19, sendo este uma derivação do problema que consta nas Figuras 15 e 16. Nele é possível observar a convergência em um problema onde duas funções de custo tiveram seus pesos anulados. O exemplo também demonstra uma situação

bastante comum em problemas dinâmicos, onde o novo modelo não apresenta alterações significativas a ponto de acarretar em algum impacto na solução atual.

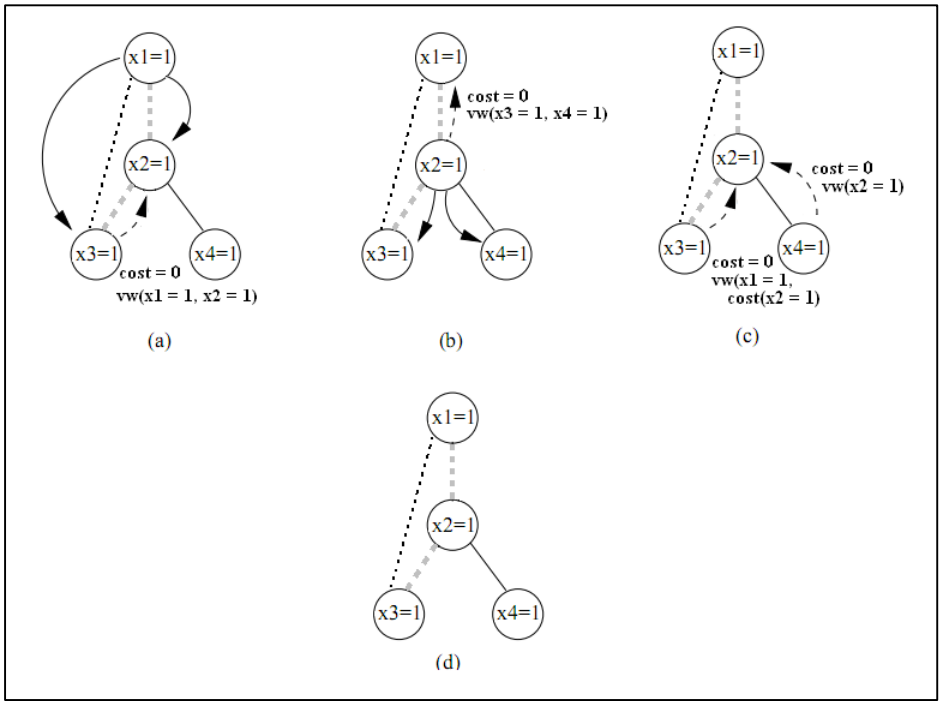

Figura 20 - Execução do DynADOPT em problema de remoção de restrições

A execução contida na Figura 20 inicia com o agente x_1 reavaliando e enviando o valor atual para seus filhos x_2 e x_3 que seguem a mesma rotina juntamente com x_4 onde rapidamente o algoritmo converge e a solução atual é mantida, pois representa o melhor custo para o modelo. Este exemplo demonstra a forma como o *DynADOPT* otimiza a resolução de problemas onde a solução atual persiste como a de menor custo. Isso ocorre devido à rápida convergência do algoritmo, pois a estratégia utilizada descarta o processamento de atribuições já contidas nas visões dos agentes ou custos maiores que o atual. Em outras palavras, o *DynADOPT* apenas explora novos estados que apresentarem melhores custos locais propagados pelos agentes.

3.5. *Considerações finais*

O presente capítulo apresentou os detalhes de funcionamento do algoritmo *DynADOPT*. Foram abordadas suas características de metodológicas e estruturais, bem

como os casos de utilização propostos e exemplos de execução. Também foram mencionados os casos onde o algoritmo possui um grande desempenho.

Os ambientes dinâmicos abordados no estudo abrangem situações onde as funções de custo que conectam os agentes têm seus custos modificados ou totalmente anulados, representado a remoção destes. Outros casos mencionados que não fazem parte do projeto, como modificações nas variáveis do problema, agentes e adição de restrições no grafo também foram discutidas, pois deverão ser abordadas em trabalhos futuros.

Outro aspecto do *DynADOPT* discutido foi a forma como o algoritmo utiliza técnicas de reuso de soluções para convergir com maior eficiência, além dos mecanismos que evitam o aumento do volume de informações entre os agentes durante as novas resoluções. Na seqüência serão abordadas as rotinas, algoritmos, formas e métricas de avaliação para a análise dos resultados obtidos através dos experimentos realizados com o *DynADOPT*, comprovando a sua eficiência algorítmica em problemas dinâmicos.

4. RESULTADOS E EXPERIMENTOS

Uma vez discutidos os aspectos funcionais do *DynADOPT* é possível apresentar os resultados obtidos através dos experimentos realizados com a finalidade de comprovar a eficácia do novo algoritmo proposto. Esta fase do documento descreve todo o trabalho empenhado na criação de um ambiente de simulação, com rotinas e métricas e análise de avaliação, abordando desde as metas estabelecidas até a apresentação dos resultados encontrados.

4.1. *Objetivos*

Os experimentos têm como principal objetivo comprovar a superioridade do desempenho do *DynADOPT*, sobre o algoritmo *ADOPT* em problemas dinâmicos. Para esta avaliação serão utilizadas métricas como o número de ciclos e trocas de mensagens e o volume de dados enviados, desempenhadas por ambos os algoritmos para resolução de um determinado problema. Além disso, são definidos também os casos de problemas dinâmicos utilizados nos estudos e avaliação. A seguir, são apresentadas as metodologias de avaliação empregadas nos experimentos realizados.

4.2. *Metodologia de avaliação*

Uma das grandes dificuldades em avaliar abordagens dinâmicas é o desenvolvimento de rotinas específicas para representar este tipo de ambiente. Logo, um dos passos fundamentais para a obtenção de resultados foi a criação de um ambiente de testes, possibilitando a geração de grafos aleatórios, dadas as definições básicas do problema desejado. Outra necessidade levantada é como impor alterações controladas dentro dos casos gerados e meios de garantir uma reprodução fiel do problema para que vários algoritmos possam obter soluções utilizando diferentes abordagens. Logo, é necessário um detalhamento sobre os grafos adotados bem como a forma eleita para o controle do dinamismo dos problemas.

4.2.1. Grafos

Os problemas *DynDCOP*, assim como os *DCOP*, podem ser representados através de grafos onde as arestas podem ser vistas como as restrições e os nós como os a-

gentes, conforme mencionado na fundamentação teórica deste documento. Nos experimentos executados com o *DynADOPT*, houve a necessidade de se obter grafos de diferentes tamanhos e densidades, objetivando analisar o comportamento do método em ambientes variados. A técnica utilizada para suprir essa necessidade foi a elaboração de um algoritmo gerador de grafos baseados em parâmetros informados, servindo como base de todo o sistema de avaliação adotado.

O sistema gerador de grafos trabalha distribuindo aleatoriamente um número informado de arestas dentro de um conjunto finito e de nós. Para obter o número de restrições desejadas utiliza-se a seguinte equação (Cormen, et al., 2003):

$$|E| = \frac{d * v(v-1)}{2} \qquad (2)$$

De acordo com a Equação 1, basta informar a quantidade de nós que o grafo deverá possuir (*v*) junto com a densidade (*d*) desejada para a geração de um novo problema, onde |*E*| é o tamanho do conjunto de arestas do modelo. Utilizando-se desta abordagem para construção de grafos, não existe a necessidade da elaboração de grafos manualmente que podem levar a criação de problemas com as mesmas características comprometendo a qualidade e o andamento dos experimentos. Na Figura 21, é possível observar o pseudocódigo da rotina de geração de grafos randômicos informando o número de vértices e a densidade desejada.

```
#   d: densidade informada para construção do grafo
#   v: número de vértices do grafo
#   NArestas: número de arestas do problema
#   matAdj: matriz de adjacência que representa o grafo

0   Início
1     matAdj ← InicializaMatrizAdjacencia();
2     NArestas ← (d * v *(v - 1)) / 2;
3     enquanto NArestas > 0 faça
4       Escolhe x, y randômico;
5       se matAdj(x, y) = 0 faça
6         matAdj(x, y) ← 1;
7         decrementa NArestas;
8       fim se; fim faça;
9   Fim
```

Figura 21 - Pseudocódigo da geração de grafos

Da mesma forma que a geração dos grafos, o controle das modificações, que dão ao ambiente o comportamento dinâmico desejado, também precisa ser feito aleatoriamente sem a interação humana.

4.2.2. Controle de alterações

A forma com a qual se alteram os problemas durante a execução do algoritmo possui um papel muito importante dentro do ambiente de experimentos pois, simula a ocorrência de cenários dinâmicos. Logo, foi necessário o desenvolvimento de pequenos mecanismos que simulam o dinamismo através de alterações nas restrições do problema. As alterações eleitas para simular o dinamismo dos problemas dos experimentos são modificação e remoção de arestas. Ambas as alterações trabalham com porcentagens baseadas na densidade inicial do grafo. Em outras palavras a cada resolução dentro de uma simulação, uma porcentagem do número de arestas do problema será escolhida para ser alterada conforme a definição do ambiente (modificação ou remoção). A seguir serão abordados detalhadamente cada um destes esquemas para alterações no grafo de restrições dos problemas *DynDCOP*.

4.2.2.1. Modificação de arestas

Na modificação de arestas, a idéia principal é acrescer gradualmente o custo da atribuição local escolhida na execução anterior no grupo de restrições selecionadas. Para efeitos práticos, este acréscimo será de *1*. Para se obter uma maior clareza sobre a simulação desta característica dinâmica, o pseudocódigo dessa rotina pode ser analisado na Figura 22. Nela, é possível compreender com mais detalhes o processo de modificação arestas do problema. É interessante notar na linha 5 a rotina para encontrar a função de custo que é representada na matriz de adjacência, onde existe uma condição para que a função não seja modificada duas vezes na mesma execução. Outro detalhe importante, apresentado na linha 7, é o custo acrescido da tupla que representa o custo da atribuição das duas variáveis conectadas, forçando que o algoritmo tenha que buscar uma nova solução.

```
#   M: número de arestas a serem modificadas
#   matAdj: matriz de adjacência do grafo
#   v1, v2: valores atribuídos à variáveis dos agentes conec-
    tados pela função de custo
#   F: função de custo
#   f: tupla da função de custo

0   Início
1     M ← p * númeroArestas(matAdj);
2     ∀m ∈ M faça
3       Escolhe x, y aleatoriamente;
4       se matAdj(x, y) > 0 faça
5         F ← encontraFuncaoCustoNãoModificada(matAdj(x, y));
6         f ← encontraTupla(F, v1, v2);
7         f ← custo(f)+ 1;
8       fim se; fim faça;
9   Fim
```

Figura 22 - Pseudocódigo da modificação de arestas

O objetivo da modificação de restrições é estudar o comportamento do *DynA-DOPT* em situações que a solução encontrada sofra um desgaste, necessitando uma nova reavaliação na maioria das novas resoluções. A seguir são apresentadas as definições dos casos em que restrições desaparecem do problema.

4.2.2.2. Remoção de arestas

O processo de adaptação a alterações que provocam remoção de arestas, deve permitir ao *DynADOPT* fazer uso eficiente dos recursos computacionais necessários para que a solução atual seja mantida ou outra solução de melhor qualidade seja obtida, se esse for o caso. Para isso é preciso estudar uma forma que seja possível realizar estas remoções sem quebrar a estrutura do problema, necessitando a geração de novas pseudo-árvores, pois conforme visto no capítulo de metodologia deste trabalho, a geração de uma nova pseudo-árvore acarreta em quebras na hierarquia impactando negativamente no desempenho do reuso das informações e performance do algoritmo. A forma escolhida para solucionar este problema consiste em anular os custos das restrições escolhidas, de modo que quaisquer valores atribuídos às variáveis relacionadas não impactem no custo local e conseqüentemente no custo global do problema. O pseudocódigo com a rotina de remoção de arestas utilizada nos experimentos pode ser vista na Figura 23.

O procedimento para remoção de arestas é muito semelhante ao da modificação, pois o processo principal de cálculo das arestas selecionadas é o mesmo. As linhas 5, 6 e 7 contém as rotinas específicas da remoção de arestas. A linha 5 busca a função de custo não removida. As linhas 6 e 7 realizam a anulação das tuplas da função encontrada.

```
#   R: número de arestas à serem removidas
#   matAdj: matriz de adjacência do grafo
#   v1, v2: valores atribuídos à variáveis dos agentes conec-
    tados pela função de custo
#   M: conjunto das restrições à serem modificadas
#   r: restrição contida no conjunto de restrições
#   F: função de custo
#   f: tupla da função de custo

0   Início
1     R ← p * númeroArestas(matAdj);
2     ∀r ∈ M faça
3       Escolhe x, y randômico;
4       se matAdj(x, y) > 0 faça
5         F ← encontraFuncaoCustoNãoRemovida(matAdj(x, y));
6         ∀f ∈ F faça
7           f ← 0;
8       fim faça; fim se; fim faça;
9   Fim
```

Figura 23 - Pseudocódigo da remoção de arestas

4.2.3. Métricas

Para a avaliação da solução proposta, serão utilizados os critérios encontrados na literatura para análise de desempenho de algoritmos *DCOP*. São estes: número de ciclos de execução, número de mensagens enviadas e volume de dados enviados (*throughput*) pelos agentes.

O número de ciclos representa o total de vezes em que cada agente é executado. Esta definição se deve ao fato do caráter assíncrono do algoritmo, que em determinados casos, exige um grande processamento por parte de apenas alguns agentes do grafo, enquanto os agentes restantes aguardam o término da execução. Para facilitar a visualização dos resultados, será utilizado o número total de ciclos, ou seja, o total de vezes em que um agente do problema foi executado.

O número de mensagens corresponde ao total de vezes em que os agentes enviam mensagens aos seus vizinhos próximos. De forma análoga à quantidade de ciclos, a quantidade de mensagens também será analisada como o número total de ocorrências dentro de uma execução.

Finalmente, o volume de dados enviados, mede a quantidade de informação que um agente transmite para um vizinho. Esta métrica é essencial para analisar o gasto de recursos absorvidos pelo algoritmo durante a execução em ambientes distribuídos.

Após a compreensão dos critérios definidos para esta etapa, bem como os casos que serão abordados, é possível apresentar os resultados dos experimentos realizados com o *DynADOPT* em ambientes dinâmicos.

4.3.*Resultados*

Os resultados obtidos através dos experimentos realizados têm como objetivo comparar o desempenho do *DynADOPT* com o *ADOPT* original. Para facilitar o entendimento, esta parte do trabalho será divida em duas etapas: apresentação dos resultados de problemas modificados e apresentação dos resultados dos problemas com remoção de arestas. Dentro destas divisões, serão analisados os dados obtidos para cada problema solucionado.

4.3.1. Modificação de restrições

Para a análise das modificações de restrições em problemas dinâmicos, foram escolhidos três casos e apresentadas comparações das métricas definidas anteriormente. Para esta classe de experimentos, consideraram-se simulações de duzentas sucessivas resoluções. Em outras palavras, os algoritmos solucionavam problemas derivados dos anteriores o montante definido de vezes em seqüência. Logo, os gráficos mostrados utilizam como dados o número de execuções realizadas por unidade correspondente à métrica, unitário para número de ciclos e mensagens e *bytes* para volume de dados. A apresentação dos resultados é agrupada por problema solucionado para facilitar a visualização e interpretação dos dados obtidos. A seguir serão demonstradas e discutidas as comparações alcançadas.

4.3.1.1. Problema de modificação de arestas 1

O primeiro problema solucionado pelo *DynADOPT* apresentado tem como objetivo analisar e discutir como o algoritmo se comporta em problemas complexos. O grafo apresenta 5 nós e densidade 1, ou seja, todos os agentes possuem uma relação com os demais agentes do problema. Além disso, a cada ciclo de resolução 80% das arestas do problema serão alteradas conforme descrito anteriormente. A seguir são apresentados os resultados obtidos no problema.

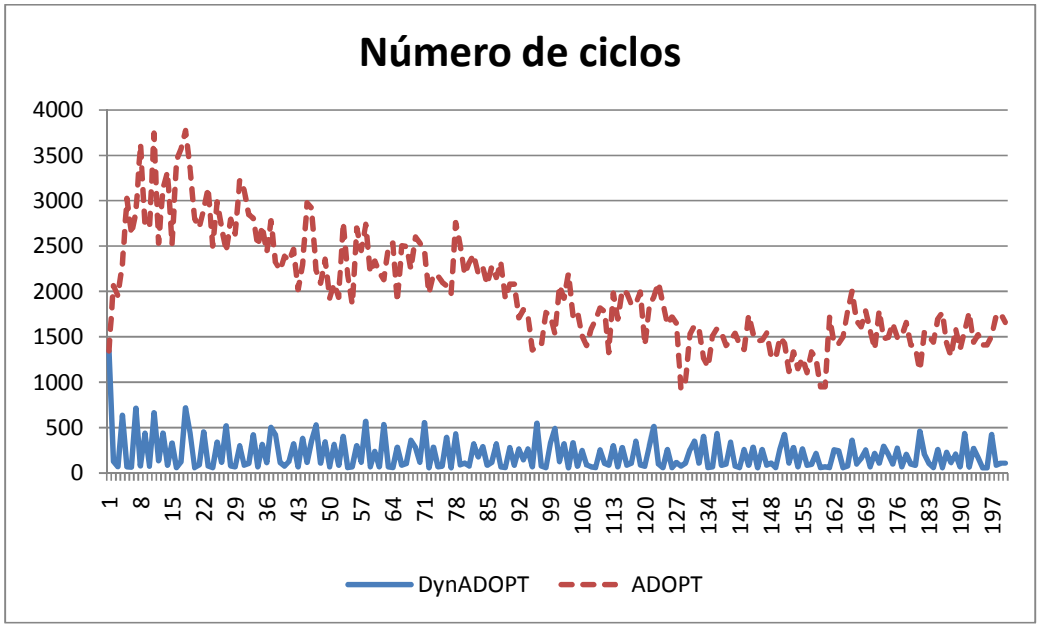

Figura 24 – Número de ciclos obtido na resolução do problema de modificação de arestas 1

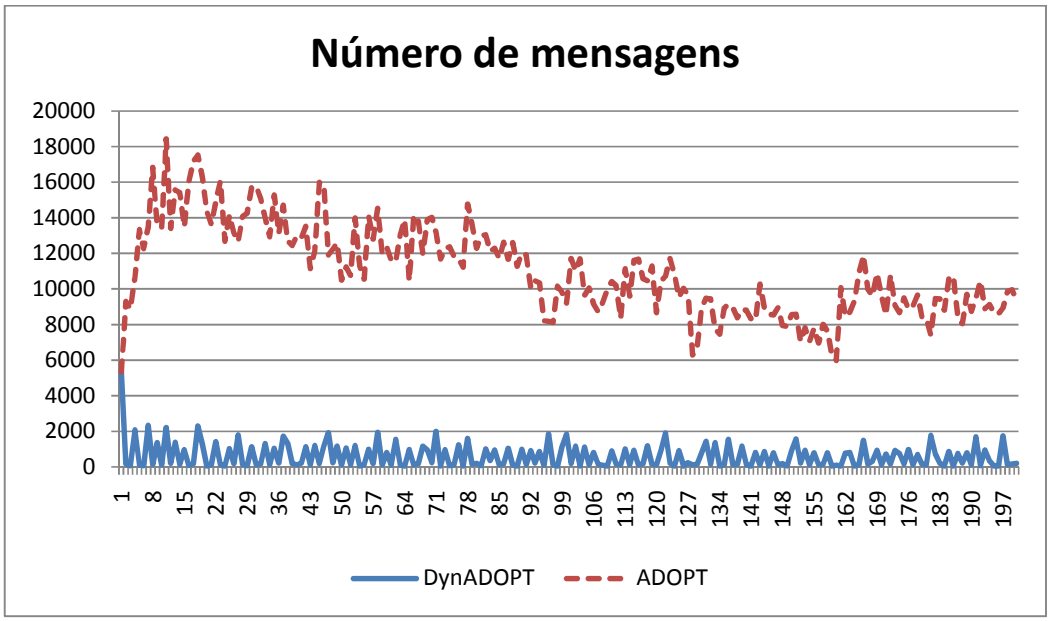

Figura 25 - Número de mensagens obtido na resolução do problema de modificação de arestas 1

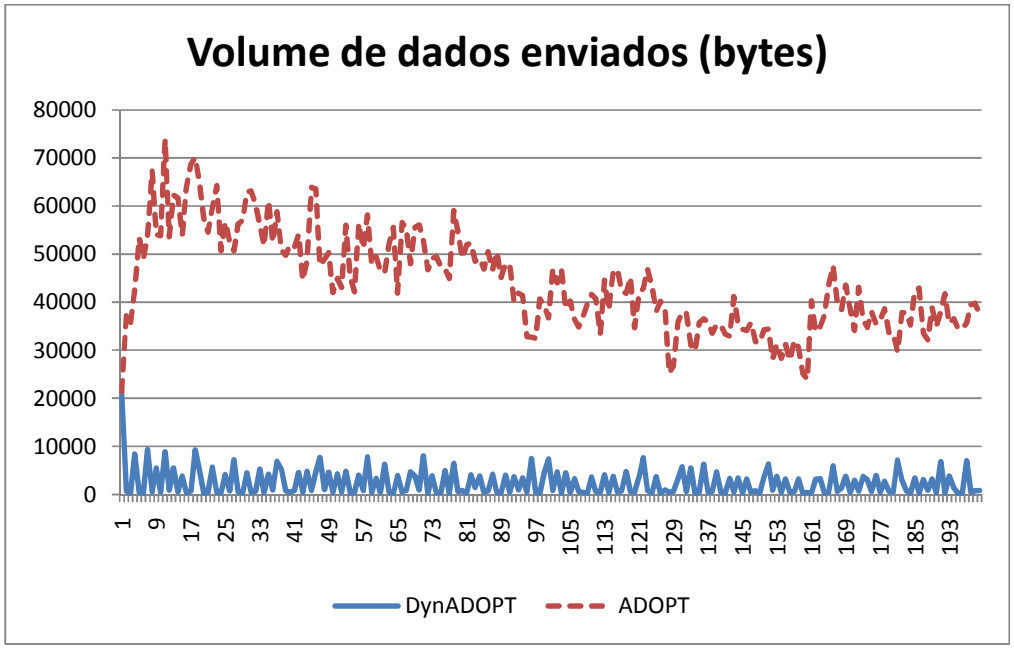

Figura 26 – Volume de dados obtido na resolução do problema de modificação de arestas 1

Analisando os gráficos das Figuras 24, 25 e 26, é possível notar claramente o desempenho superior do *DynADOPT* em relação ao *ADOPT*. Todos eles possuem um comportamento bastante similar, pois as três métricas utilizadas estão fortemente conectadas. A cada ciclo, um número linear de mensagens é enviado aos agentes, pois como o problema é completo (densidade 1), ou seja, todos os agentes enviam mensagens para todos os demais. O volume de dados nada mais apresenta do que o tamanho das mensagens, sendo este também linear graças à visão dos agentes que possuem todos os demais nós do problema. Outra característica notável nos gráficos é o comportamento do *ADOPT* durante a simulação. Nas primeiras execuções, o algoritmo apresenta uma forte queda no desempenho, que diminui no decorrer da simulação. Esta queda na performance se deve ao fato de um grande número de arestas (80%) apresentarem um custo diferente logo nas primeira execuções, forçando os agentes a uma troca mais intensa de informações para a convergência. Esta troca de informações diminui com o tempo, pois as atribuições alteradas pelo ambiente de simulação geram instâncias de problemas muito semelhantes às anteriores. Logo, as alterações tendem a possuir um comportamento homogêneo com o tempo. Este padrão pode ser observado também nos demais casos de modificação de arestas apresentados na seqüência.

4.3.1.2. Problema de modificação de arestas 2

O próximo problema aborda um contraste em relação ao primeiro modelo onde é possível observar a convergência dos algoritmos em um problema semelhante, porém com uma complexidade inferior tanto em sua estrutura principal quanto no dinamismo sofrido. O caso apresentado possui 5 agentes, densidade 0,8 e 30% das restrições alteradas a cada resolução.

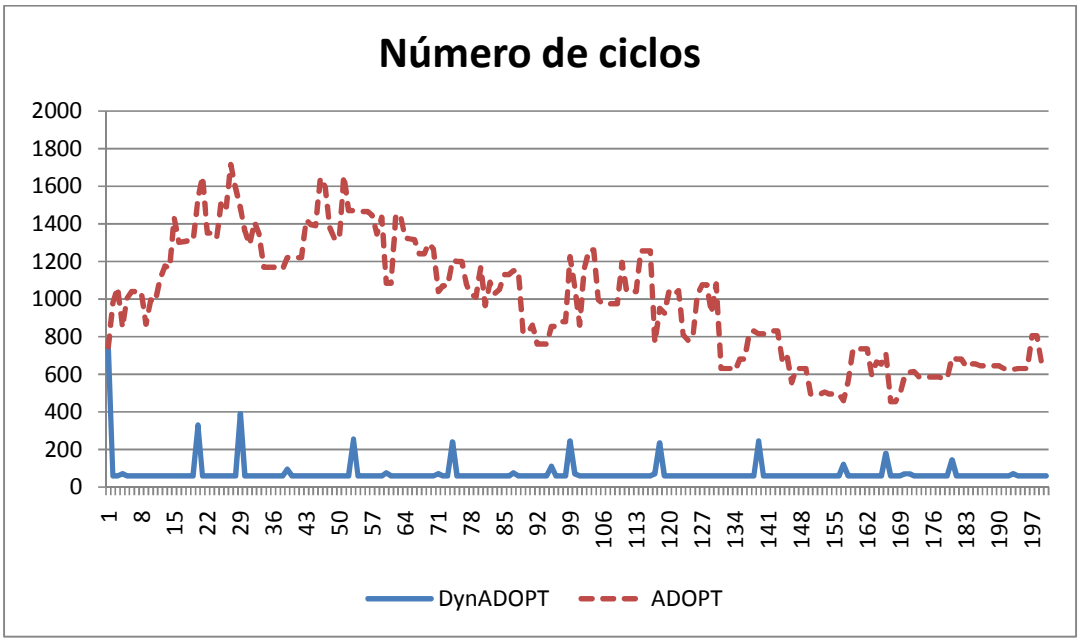

Figura 27 - Número de ciclos obtido na resolução do problema de modificação de arestas 2

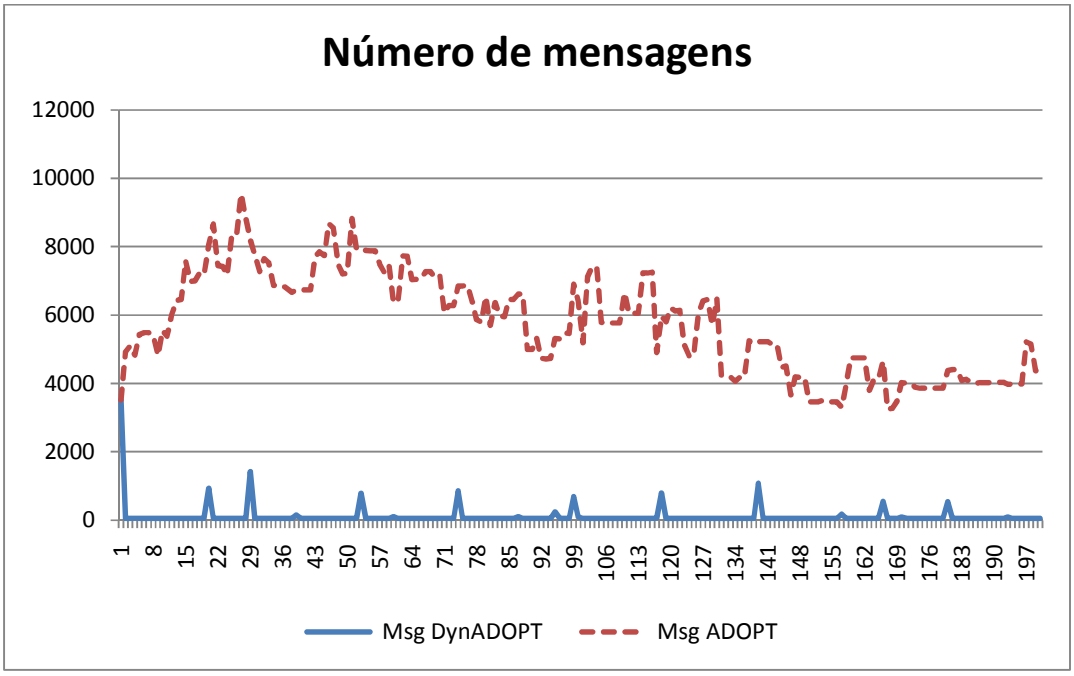

Figura 28 - Número de mensagens obtido na resolução do problema de modificação de arestas 2

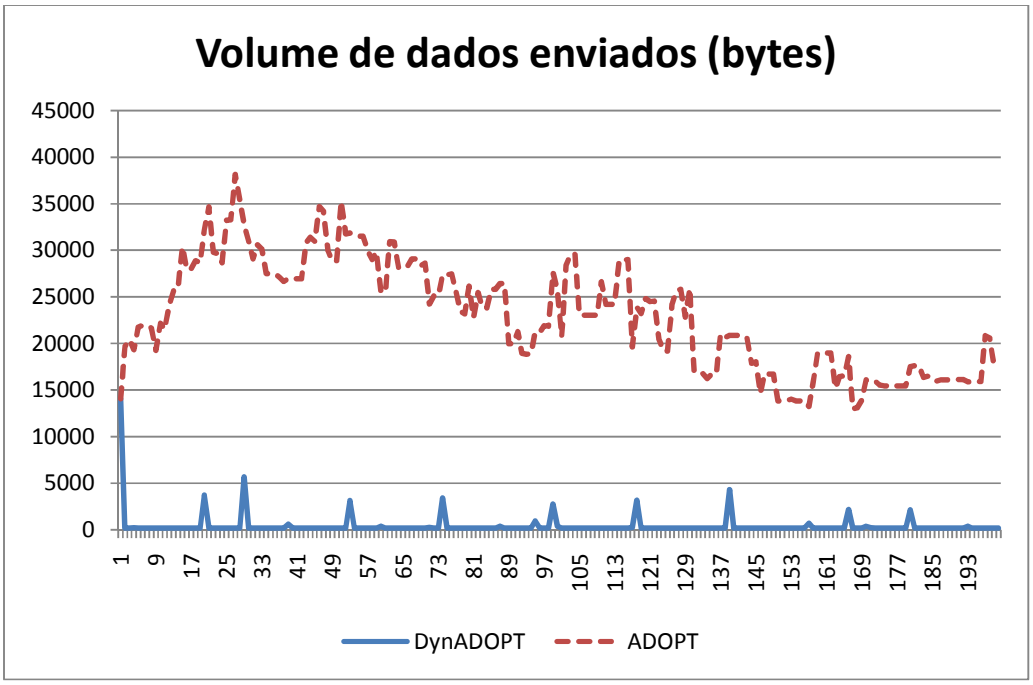

Figura 29 - Volume de dados obtido na resolução do problema de modificação de arestas 2

Assim como no problema 1, os gráficos das Figuras 27, 28 e 29 que comparam os resultados das convergências dos algoritmos do problema 2 também apresentam uma simetria entre si, devido a grande dependência entre as métricas, conforme citado no problema anterior. Porém, o comportamento do *ADOPT* neste cenário é mais estável em relação ao primeiro, pois a curva apresenta uma suavidade maior entre as alterações. Isso se deve às arestas, que têm seus custos modificados entre as resoluções ser consideravelmente menor em relação ao problema 1. Com isso, o novo ambiente, resultado do antigo problema resolvido com alterações impostas, apresenta um elevado grau de semelhança com o grafo original. Este fenômeno também pode ser observado nos resultados do *DynADOPT*, onde houve uma estabilização em sua performance em relação ao problema 1. Outra constatação observada é que o número de arestas impacta fortemente no desempenho da resolução do *DynADOPT* e seu algoritmo original. Esta característica pode ser evidenciada com mais detalhes no caso apresentado a seguir, onde um problema com mais agentes e menos restrições apresenta a convergência com desempenho superior aos outros casos apresentados.

4.3.1.3. Problema de modificação de arestas 3

O terceiro e último caso estudado sobre modificação de restrições aborda um problema com 8 agentes, 0,5 de densidade e 30% de restrições modificadas a cada alteração. O objetivo da discussão sobre este problema é constatar que um grafo com mais agentes, porém menos restrições entre eles, contribui positivamente para a eficiência do modelo a ser resolvido pelo sistema, como pode ser visto abaixo.

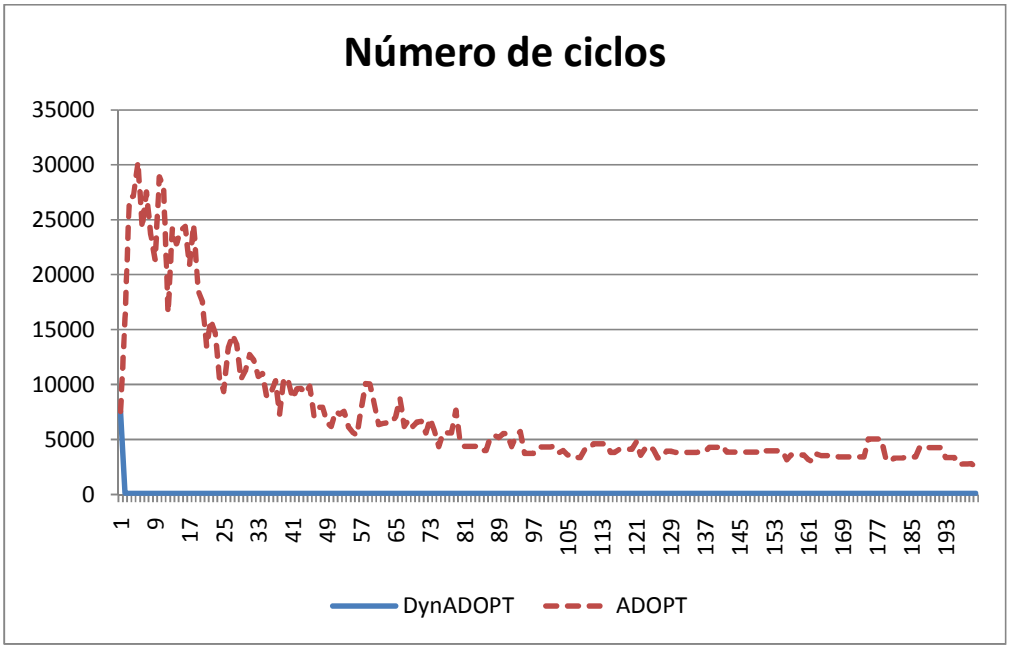

Figura 30 - Número de ciclos obtido na resolução do problema de modificação de arestas 3

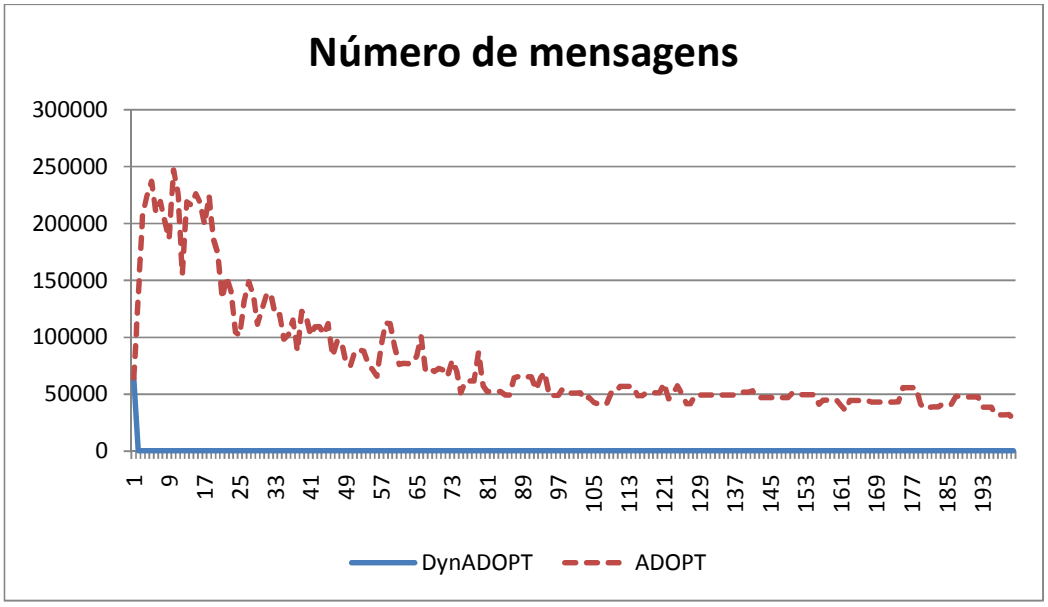

Figura 31 - Número de mensagens obtido na resolução do problema de modificação de arestas 3

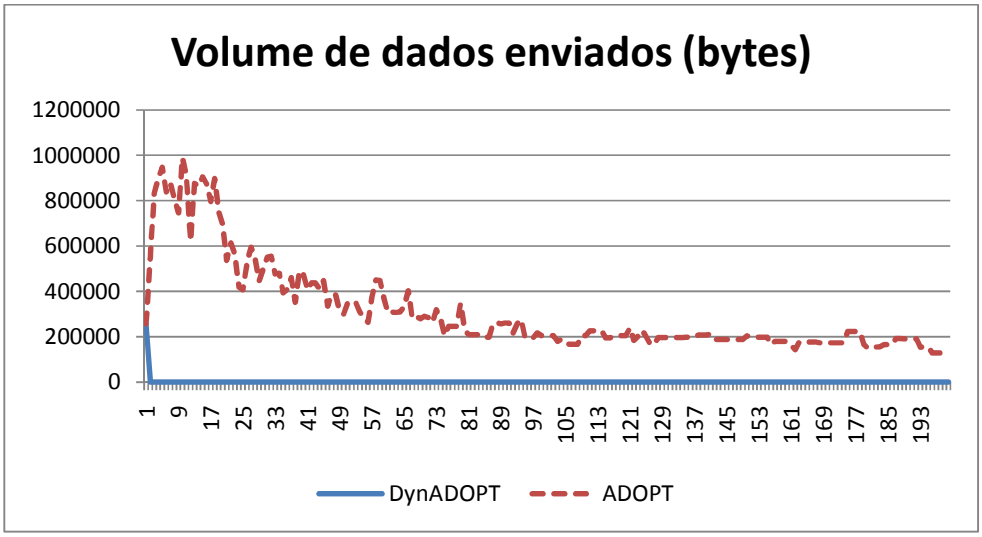

Figura 32 - Volume de dados obtido na resolução do problema de modificação de arestas 3

As Figuras 30, 31 e 32 apresentam os resultados obtidos na resolução do problema 3. Conforme mencionado anteriormente, a forte relação entre as métricas acarreta em gráficos com um alto nível de semelhança. Outra característica marcante é o número de agentes do problema não impactar em perdas de desempenho por parte de ambos os algoritmos. Por outro lado, a baixa densidade do grafo entre os demais casos apresentados acarretou em um esforço consideravelmente menor no caso 3 para ambos os algoritmos. Outro fator relevante para esta afirmação é o dinamismo mais uniforme do modelo, fazendo com que os problemas oriundos de execuções anteriores possuam um alto grau de semelhança com os antigos.

O desempenho do *DynADOPT* para a problemas de modificações de restrições em *DynDCOPs* obteve resultados superiores comparados com o *ADOPT* original. Isso se deve a dois importantes fatores observados: a construção do modelo e a estratégia do algoritmo. A construção do modelo se mostra importante, pois conforme visto, um cenário com menos restrições e mais agentes apresenta uma convergência empregando menos esforços. E com isso, o grande impacto das restrições na complexidade geral do problema, sendo de vital importância o estudo e desenvolvimento de aplicações e métodos que com o auxílio de técnicas dinâmicas, amenizam o peso computacional atribuído às funções de custo.

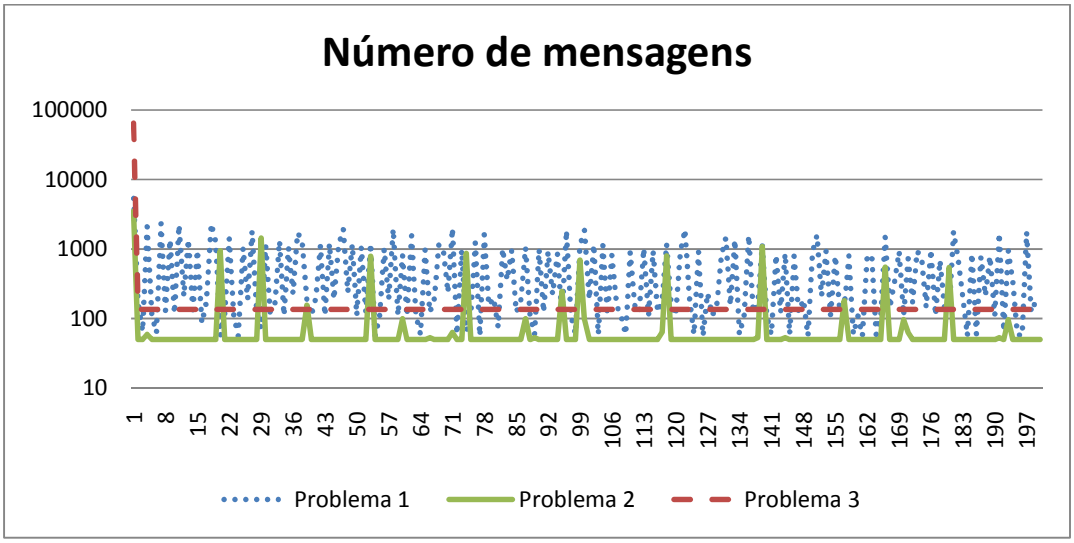

Figura 33 - Comparação entre os resultados dos números de mensagens do *DynADOPT*

Para a conclusão da análise dos casos de modificação de restrições, é apresentado um gráfico na Figura 33, contendo a comparação dos números de mensagens obtidos pelo *DynADOPT* nos problemas simulados. Nele é possível observar que a primeira execução das resoluções apresenta altas trocas de mensagens devido à ausência de informações. A partir da segunda execução o número de mensagens geradas varia dentro de um intervalo padrão para os três exemplos: entre 80 e 1300 mensagens. Outro comportamento interessante é que mesmo estabelecendo uma faixa de valores, todos os casos resultaram em curvas diferentes, apresentando uma instabilidade maior para o primeiro problema, estabilidade parcial no segundo e estabilidade no terceiro. Isso se deve ao fato da densidade dos problemas onde grafos mais densos demonstram resultados mais instáveis.

A estratégia do *DynADOPT* surge como um poderoso método para resolver problemas onde modificações nas funções de custo entre agentes se altera após sucessivas execuções. Isso se deve ao fato do mecanismo de reutilização das informações previamente obtidas em resoluções anteriores para a convergência de novos problemas derivados. Outro tipo de cenário abordado no presente documento são casos em que restrições venham a desaparecer no decorrer da simulação de um dado problema.

4.3.2. Remoção de restrições

Os casos de remoção de restrições têm como objetivo simular problemas onde no decorrer do tempo, funções de custo que conectam dois agentes deixam de existir.

Para isto foi necessária a elaboração de rotinas específicas para o tratamento deste tipo de cenário, pois o *DynADOPT* é um algoritmo que necessita de uma organização hierárquica de seus agentes. Devido a esta característica, optou-se por anular as restrições escolhidas ao invés de removê-las, suprimindo assim a necessidade da geração de novas pseudo-árvores a cada nova execução. Um detalhe importante sobre as simulações deste caso é o número variável de ciclos de resolução dos algoritmos, tendo em vista que os modelos tendem a gerar um problema que não apresenta nenhuma função de custo com pesos maiores que zero. Portanto as simulações abordadas se restringem a incluir os dados até a primeira execução de um problema sem arestas válidas. A seguir serão apresentados os resultados alcançados, divididos por casos e subdivididos em métricas para facilidade de entendimento e, ao fim dos resultados, será feita a discussão sobre os dados obtidos.

4.3.2.1. Problema de remoção de arestas 1

O primeiro caso de remoção de arestas apresentado tem como objetivo demonstrar o comportamento do *DynADOPT* em um problema com um grau relevante de complexidade tanto em sua estrutura como no dinamismo atribuído. O grafo é composto por 8 agentes conectados com densidade 0,8, onde 80% das arestas são removidas a cada resolução. Os gráficos abaixo apresentam os resultados obtidos para o problema 1.

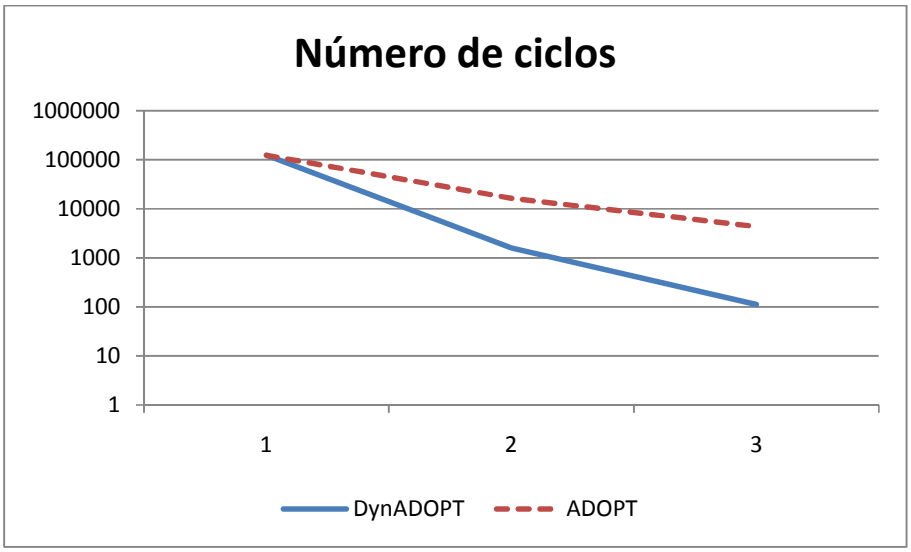

Figura 34 - Número de ciclos obtido na resolução do problema de remoção de arestas 1

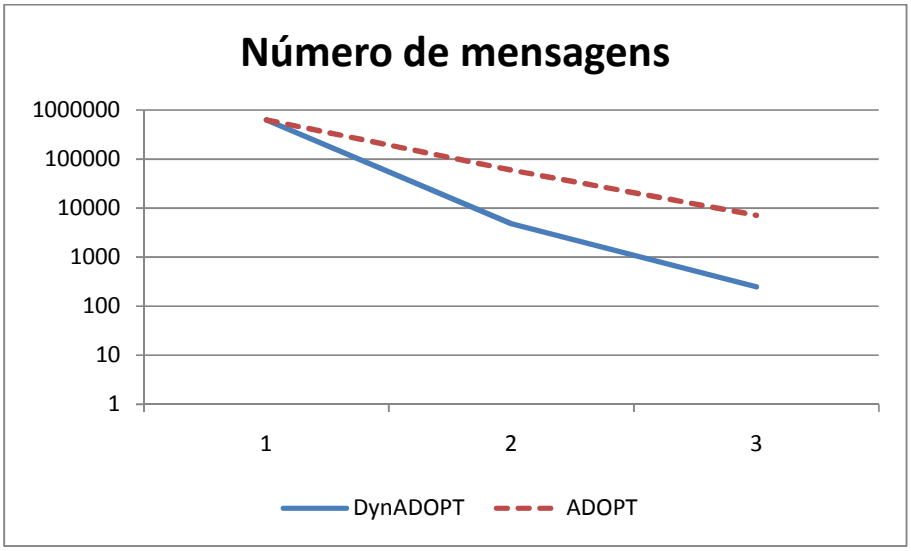

Figura 35 - Número de mensagens obtido na resolução do problema de remoção de arestas 1

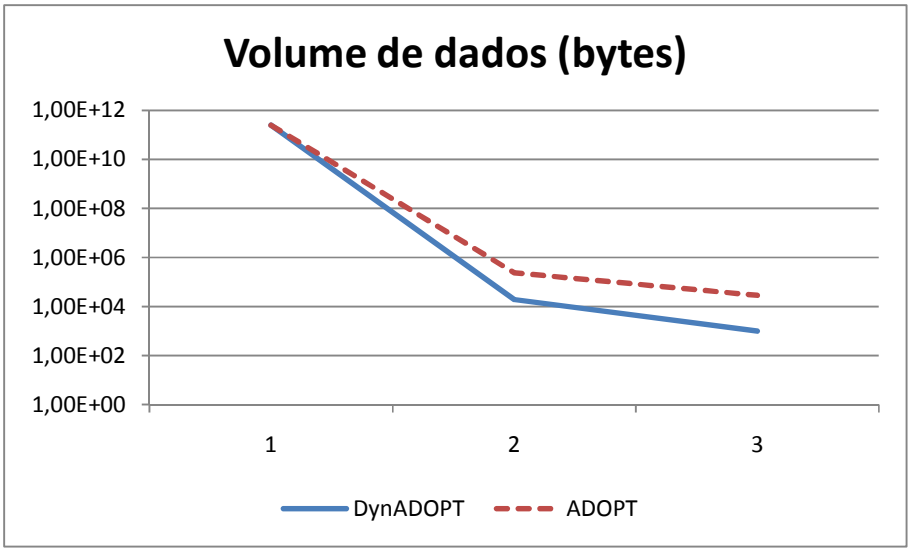

Figura 36 – Volume de dados obtido na resolução do problema de remoção de arestas 1

Analisando os gráficos das Figuras 34, 35 e 36, é possível notar uma semelhança e padrão entre as métricas de avaliação, porém não tão evidentes quanto ao encontrado nos problemas de modificação de restrições. Como se trata de um cenário com um número alto de arestas removidas, o problema apresenta apenas três sucessivas resoluções, ou seja, o grafo contém duas modificações durante a simulação. Todavia, os dados obtidos contêm características interessantes a serem observadas. A primeira delas é a redução previamente esperada do emprego de recursos computacionais. Outra tendência

presente no gráfico é que a estratégia da reutilização das informações contidas dos agentes em cenários onde restrições desaparecem impacta beneficamente no desempenho do algoritmo, reduzido o número de ciclos necessários para cada resolução, acarretando em uma quantidade menor de mensagens e volume de dados enviados. Este comportamento se torna mais evidente nos próximos cenários, onde é possível perceber a diferença de performance entre os algoritmos em problemas de remoção de arestas em grafos mais esparsos.

4.3.2.1. Problema de remoção de arestas 2

O próximo caso apresentado tem como objetivo demonstrar que como a estratégia do *DynADOPT* é muito parecida com a do *ADOPT* original, tendem a obter comportamentos semelhantes em problemas de remoção de restrições. Porém graças às técnicas de reuso de informações empregadas, é possível alcançar resultados favoráveis ao *DynADOPT* no que diz respeito a ambientes com esta característica em seu dinamismo. O problema abordado nesta seção possui 8 agentes, 0,8 de densidade e 30% de arestas removidas à cada resolução.

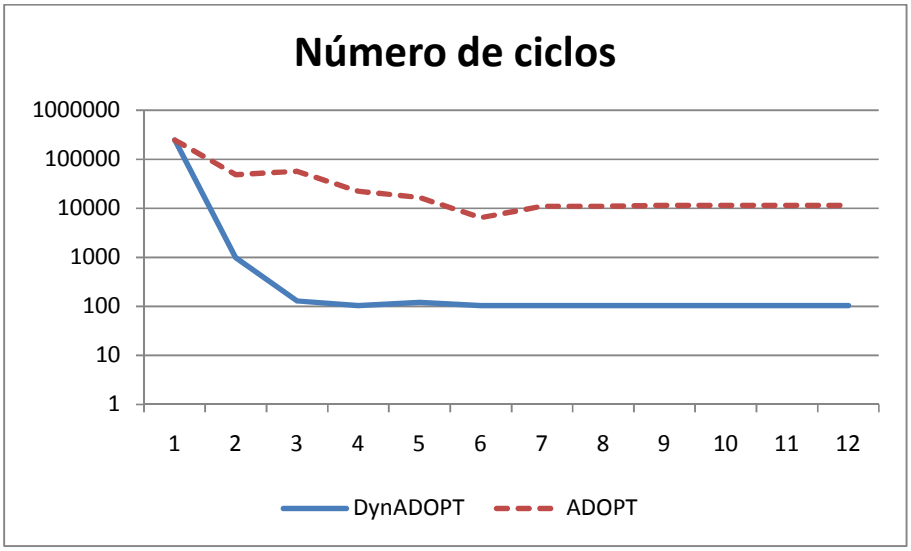

Figura 37 - Número de ciclos obtido na resolução do problema de remoção de arestas 2

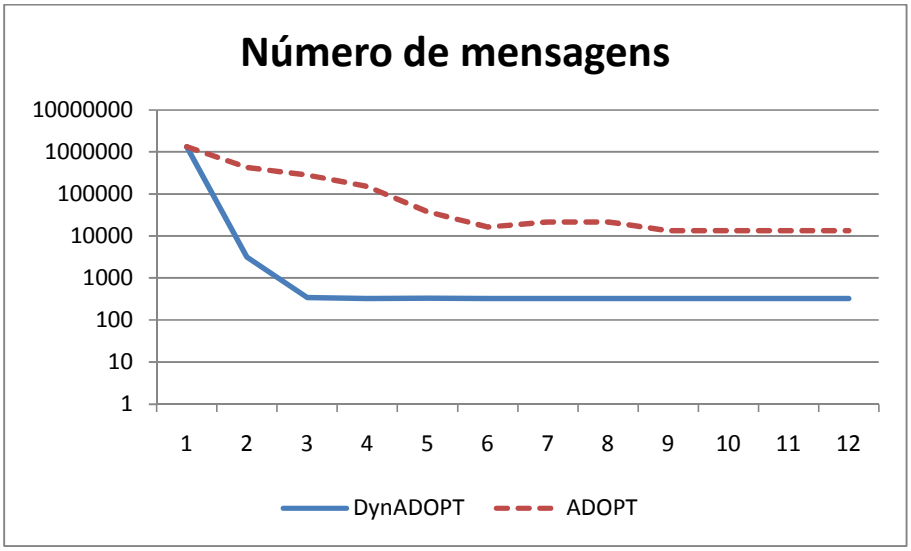

Figura 38 - Número de mensagens obtido na resolução do problema de remoção de arestas 2

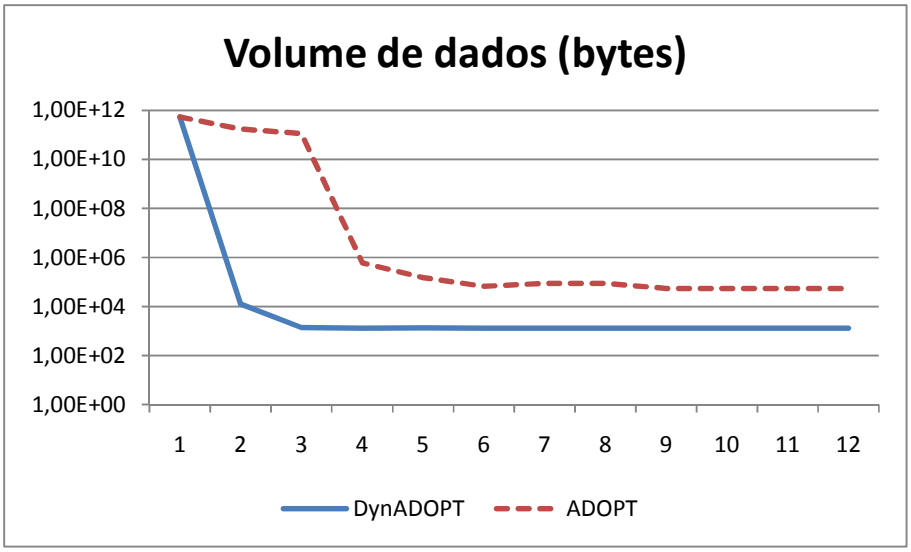

Figura 39 – Volume de dados obtido na resolução do problema de remoção de arestas 2

Os gráficos com os resultados obtidos em simulações utilizando o cenário podem ser vistos nas Figuras 37, 38 e 39. Lembrando que na resolução dos problemas de remoção de restrições, foi convencionado que as sucessivas execuções tinham como limite a primeira situação de anulação total das arestas, ou seja, a situação onde todas as funções de custo do problema não possuem nenhum peso maior que zero, É possível observar características peculiares a respeito do comportamento de ambos os algoritmos. Analisando primeiramente o *ADOPT*, é notável o volume de dados utilizado du-

rante as primeiras execuções sucedidas por uma sensível queda. Isso deve ao fato de o ADOPT necessitar de uma grande troca de mensagens devido à estratégia do algoritmo, que reporta aos agentes todas as informações globais de sua sub-árvore. A queda acentuada do volume de dados empregado na resolução mencionada refere-se à redução do número de alterações das variáveis, o que impacta diretamente na diminuição do número de ciclos e conseqüentemente de mensagens. Analisando os resultados do *DynADOPT*, observa-se um comportamento padrão para a curva dos gráficos apresentados neste caso. A explicação deste fato se deve ao reuso de informação provido pelo algoritmo, extinguindo a necessidade da troca inicial de dados entre os agentes, tornando possível a manutenção dos valores previamente encontrados. Devido ao comportamento dinâmico do problema, um determinado grafo alterado será composto apenas de restrições nulas. Logo, ambos os algoritmos tendem a estabilizar a utilização com recursos mínimos para suas resoluções. Tendo em vista os gráficos apresentados, é possível notar também uma melhora neste quesito por parte do *DynADOPT* em relação ao seu algoritmo base *ADOPT*.

4.3.2.1. Problema de remoção de arestas 3

O último caso apresentado, também sobre remoção de arestas, demonstra como mesmo tendendo a uma estabilização, a possibilidade de se utilizar informações passadas acarreta em um ganho considerável de desempenho se tratando de problemas de remoção de arestas. O problema apresentado possui um grafo com 5 agentes, completo (densidade 1) e a cada ciclo de resolução tem 30% de suas arestas removidas.

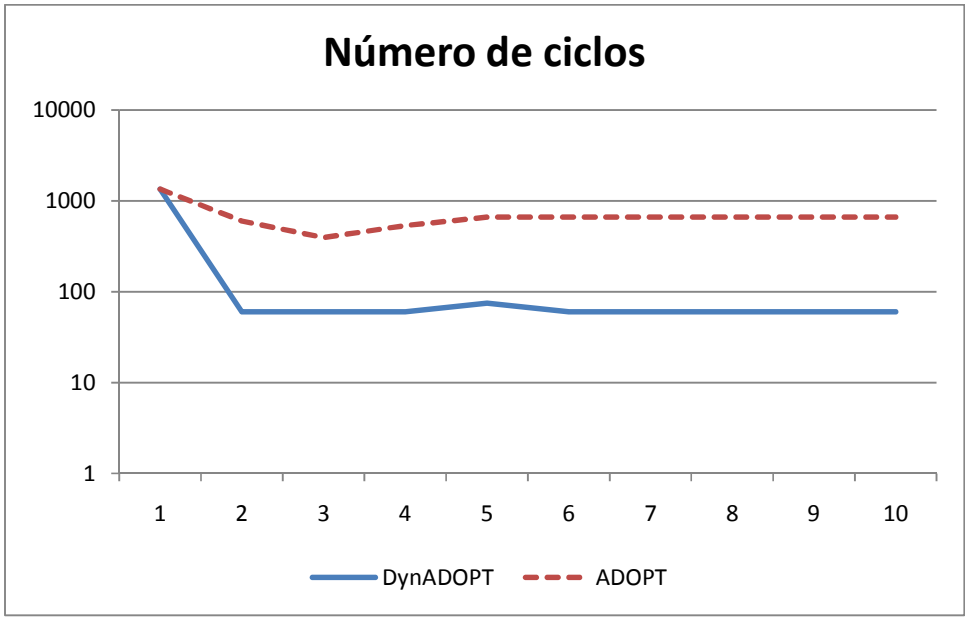

Figura 40 - Número de ciclos obtido na resolução do problema de remoção de arestas 3

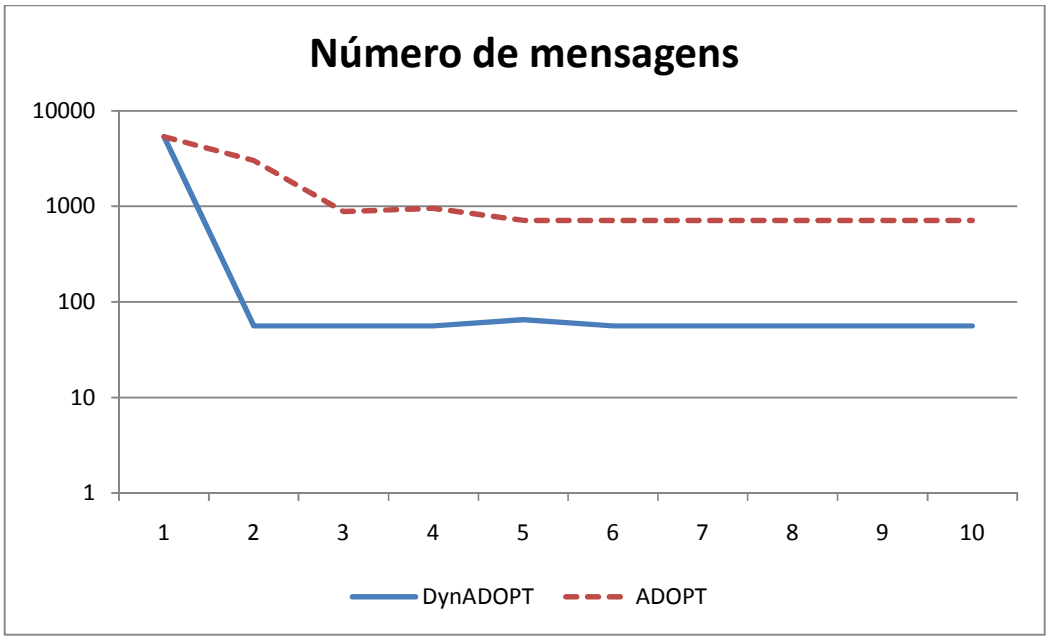

Figura 41 - Número de mensagens obtido na resolução do problema de remoção de arestas 3

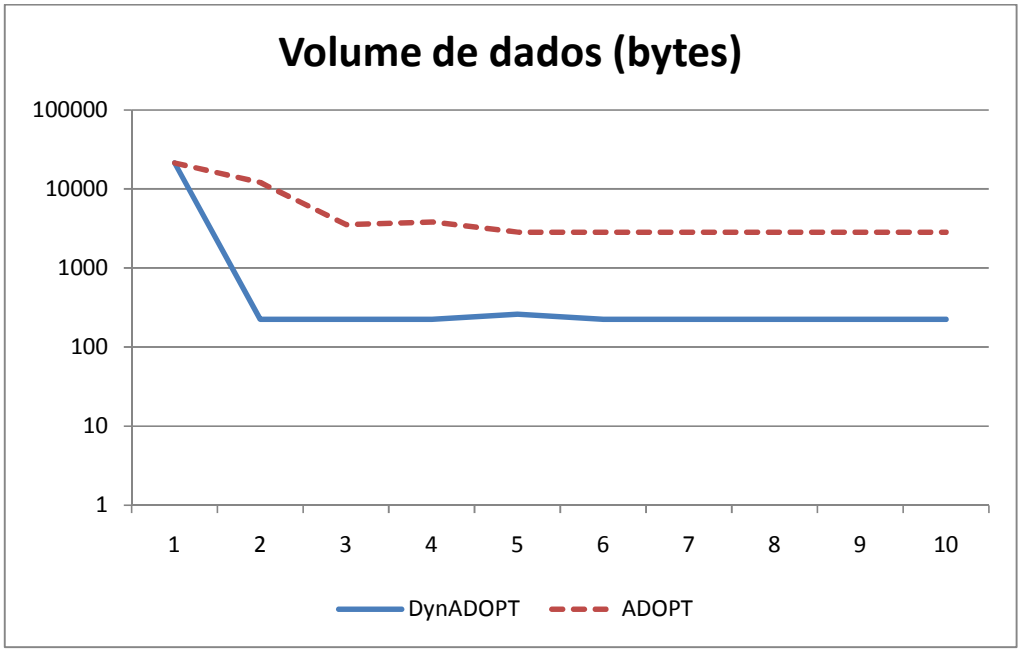

Figura 42 – Volume de dados obtido na resolução do problema de remoção de arestas 3

Os gráficos sobre o último caso apresentado estão contidos nas Figuras 40, 41 e 42. Analisando os dados obtidos, é possível observar uma tendência entre os gráficos, devido à semelhança dos sucessivos problemas, pois apenas 30% do total de arestas eram removidas a cada nova resolução. O *DynADOPT*, apresenta um comportamento semelhante ao do problema de remoção de arestas 2, onde as informações iniciais para a resolução do problema são passadas apenas uma vez, ocorrendo apenas manutenções nestes dados ao longo das demais execuções. O *ADOPT* apresentou uma suave queda devido ao dinamismo leve do modelo, e assim como o *DynADOPT* se manteve estável ao longo da simulação apresentada. Outra característica que pode ser analisada é a grande quantidade de recursos empregados pelo *ADOPT*, mesmo quando os problemas não apresentavam nenhuma aresta com custos diferentes de zero. O motivo disso é que como o os agentes do algoritmo *ADOPT* iniciam suas execuções sem nenhuma informação sobre o ambiente, é necessário que estes troquem diversas mensagens para a obtenção das informações que possibilitam a compreensão do custo total do problema, com a finalidade de alcançar a solução. O *DynADOPT* por sua vez, mantendo as informações processadas anteriormente, consegue convergir sem a necessidade de uma inicialização por parte dos agentes.

Analisando atentamente os resultados apresentados neste capítulo, é possível perceber um padrão quanto ao comportamento dos algoritmos para os diferentes casos propostos.

Em termos de desempenho, o *DynADOPT* alcançou melhores resultados em ambos os casos analisados em todos as métricas utilizadas, número de ciclos, mensagens e volume de dados enviados. Outra conclusão válida é que o algoritmo demonstrou certo grau de estabilidade em todos os modelos apresentados, com um alto grau de melhora no desempenho geral nas primeiras resoluções. Isso ocorre devido ao fato do algoritmo iniciar a simulação sem nenhum conhecimento prévio do problema.

O *ADOPT* apresentou características interessantes nos resultados alcançados. Sua convergência foi fortemente afetada por alterações nos valores nas funções de custo entre os agentes, porém demonstrou um comportamento mais estável nos casos em que restrições eram removidas do problema. Assim como o *DynADOPT*, o *ADOPT* demonstrou uma tendência a um grau de estabilidade no uso de recursos, porém muito mais demorada e com desempenho inferior ao algoritmo dinâmico.

5. CONCLUSÃO

O modelo dinâmico de otimização por restrições foi proposto como um formalismo robusto, tornando possível a modelagem natural de vários problemas reais, como agendamento de tarefas e condução de veículos automotivos. Devido às características do *DynDCOP* oriundas do *DCOP*, é possível abstrair tarefas do mundo real de maneira simples e natural, mesmo em ambientes onde relações entre variáveis podem se alterar, desaparecer ou aparecer.

As técnicas de reuso surgem como ponto altamente relevante em ambientes dinâmicos. Estas abordam meios de utilizar informações obtidas no decorrer da resolução do problema para otimizar a busca de novas soluções. Os métodos de reuso disponíveis na literatura diferem em termos de quantidade de memória, tempo computacional e qualidade da solução. Dessa forma, em um problema super-restrito, é recomendável a utilização de técnicas que aumentem a robustez ou a flexibilidade das soluções encontradas. Por outro lado, em sistemas cuja capacidade computacional é limitada, abordagens que minimizam o uso dos recursos para a busca da solução são mais interessantes.

Como existe uma carência em algoritmos que utilizam resoluções prévias no intuito de buscar uma solução de forma mais eficiente, o presente trabalho teve como objetivo apresentar o *DynADOPT*, um novo algoritmo baseado no *ADOPT* com a finalidade de solucionar problemas que apresentem alterações em suas funções de custo ou relaxação do grafo de restrições. A idéia principal deste método é a utilização de técnicas clássicas de resolução de *DCOP* aprimoradas com fundamentos de reuso de soluções, onde é possível armazenar o conhecimento prévio de execuções anteriores para as resoluções futuras possibilitando uma redução no esforço computacional empregado por sistemas de apoio de tomada de decisão. A escolha do *ADOPT* como algoritmo base para este projeto, se deve ao fato de seu caráter totalmente assíncrono e ao esquema de trocas de mensagens, fatores positivos para ambientes dinâmicos. Outro fator relevante para a escolha é a facilidade da adaptação de reuso de informações dentro de sua estratégia, pois graças ao seu modelo assíncrono, os agentes apenas enviam mensagens aos demais caso exista uma nova atribuição ou alteração em sua visão.

Uma metodologia bem definida de métricas de avaliação e experimentos, rotinas de geração e modificação de grafos realizados randomicamente possibilitou uma análise do comportamento dos algoritmos *DynADOPT* e *ADOPT* em ambientes dinâmicos. Através de simulações onde sucessivos grafos são originados através de alterações nos

problemas anteriores, foi possível comprovar os benefícios do *DynADOPT*, comparando-o com o algoritmo *ADOPT* original em problemas que possuem restrições dinâmicas que se alteram ou funções de custo que no decorrer do tempo deixam de existir, possibilitando diversas possibilidades de aplicações no mundo real.

Outra comprovação possível de ser analisada nos experimentos realizados é o impacto do número de restrições do problema no desempenho dos algoritmos. Problemas de alta densidade, ou seja, com um grande número de funções de custo em relação ao de agentes, requisitaram um esforço computacional maior, devido ao grande espaço de estados possível, acarretando em um número elevado de trocas de mensagens e dados entre os agentes.

Uma das deficiências do *DynADOPT* é o tratamento de casos onde exista uma nova restrição no problema dinâmico, ou seja, uma nova aresta é adicionada no grafo de restrições. Essa deficiência se deve ao fato do *DynADOPT*, assim como o *ADOPT*, trabalhar com organização de pseudo-árvores em sua estrutura com a finalidade de manter um padrão no fluxo de informação entre os agentes. (Petcu, et al., 2005) utilizam algoritmos que mantêm a topologia caso ocorram alterações na estrutura da árvore do problema. Esse passo no trabalho é de sumária importância no que diz respeito a robustez e possibilidades de aplicações de mundo real e deverá ser considerado em direções futuras desta pesquisa.

A comparação do *DynADOPT* com o *ADOPT*, um dos objetivos principais do projeto, teve como finalidade analisar o desempenho do novo algoritmo contra aquele que o originou e demonstrar a efetividade de sua nova estratégia. Uma importante extensão desta pesquisa deve produzir em trabalhos futuros uma comparação detalhada do *DynADOPT* com outros algoritmos projetados para atuar em situações dinâmicas. Estas análises não puderam ser construídas porque as abordagens dinâmicas disponíveis até o momento não são completas, como o *DynCOOA*, *DynDBA* ou até mesmo o algoritmo auto-estabilizante *S-DPOP*. Nestes casos a comparação do *DynADOPT* com esses algoritmos certamente favoreceria seus concorrentes, pois eles não garantem a melhor solução.

Do ponto de vista de aplicações práticas para a metodologia apresentada, a busca da solução ótima para o problema da condução inteligente de locomotivas, amplamente estudado no Laboratório de Agentes de Software situado na PUCPR, apresenta sem dúvida um grande potencial. Uma das propostas desenvolvidas em estudos anteriores para esse problema empregou a metodologia *DCOP* para conduzir composições por

trechos mapeados visando à redução do consumo de combustível (Leite, et al., 2009). Esta modelagem foi construída baseando-se em janelas de planejamento, ou seja, os agentes representam lacunas no tempo. Como estas janelas não podem representar grandes espaços de tempo devido principalmente a perdas expressivas de detalhes da geografia do trecho, é efetuado um conjunto de planejamentos menores que visam simular uma viagem por completo. Obviamente, cada uma destas janelas depende diretamente dos resultados obtidos nas execuções anteriores, e como as dimensões do modelo não se alteram, apenas suas restrições, um algoritmo que permitisse a utilização das informações provenientes das resoluções anteriores seria de grande interesse, no que diz respeito ao desempenho na busca por soluções ótimas com menos esforço computacional.

Em outro trabalho de mestrado (Leite, 2009) desenvolvido no laboratório de agentes de software (LAS) da PUCPR, foi apresentada uma abordagem utilizando algoritmos estáticos para resolução do problema produzindo resultados satisfatórios. Porém, a cada planejamento do sistema, uma nova execução do algoritmo é iniciada, sem levar em consideração os dados obtidos anteriormente, impactando negativamente no desempenho do sistema.

É possível citar outras aplicações onde a proposta desde trabalho pode apresentar resultados satisfatórios como a resolução do agendamento de tarefas utilizando a abordagem *DCOP* (Maheswaran, et al., 2004), e o gerenciamento de mensagens em redes de sensores (Modi, et al., 2001), onde estes podem assumir comportamentos dinâmicos durante suas respectivas existências.

REFERÊNCIAS BIBLIOGRÁFICAS

Bessière, Christian. 1991. Arc-consistency in dynamic constraint satisfaction problems. *Proceedings of AAAI-91.* 1991, pp. 221-226.

Bessière, Christian, Maestre, Arnold and Meseguer, Pedro. 2001. Distributed Dynamic Backtracking. *In International Joint Conference on AI Workshop on Distributed Constraint Reasoning.* 2001.

Bowring, Emma, et al. 2008. On K-optimal Distributed Constraint Optimization Algorithms: New Bound and Algorithms. *AAMAS.* 2008, pp. 01-12.

Collin, Z. and Dolev, S. 1994. Self-stabilizing depth-first search. *Information Processing Letters.* 1994, pp. 297-301.

Cormen, Thomas H., Leiserson, Charles E. and Rivest, Ronald L. 2003. *Introduction to Algorithms.* Boston : MIT Press, 2003. 0-262-03293-7.

Dechter, R. and Mateescu, R. 2006. AND/OR search spaces for graphical models. *Artificial Intelligence.* 2006.

Dorigo, M. and Gambadella, L. M. 1997. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation.* Abril 1997, pp. 53-66.

Ferber, J. 1999. *Multi Agent Systems An Introduction to distributed artificial intelligence.* 1999.

Freuder, E. C. 1985. A sufficient condition for backtrack-bounded search. *Journal of ACM.* 1985, 32(14), pp. 755-761.

Freuder, E. C. and Quinn, M. J. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. *Proceedings of 9th International Joint Conference on Artificial Inteligence, IJCAI-85.* 1985, pp. 1076-1078.

Hannebauer, Markus. 2002. *Autonomous Dynamic Reconfiguration in Multi-Agent Systems.* s.l. : Springer, 2002. 0302-9743.

Hattori, Hiromitsu and Ito, Takayuki. 2006. A Quick Adaptation Method for Constraint Satisfaction in a Real-time Environment. *International Journal of Computer Science and Network Security, IJCSNS.* Julho 2006, Vol. 6.

Hewitt, C. and Inman, J. 1991. DAI betwixt and between: from `intelligent agents' to open systems science. *IEEE Transactions on Systems, Man and Cybernetics.* Nov/Dez 1991, pp. 1409-1419.

Khanna, Sankalp, et al. 2009. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems. *Proceedings of the 2009 IEEE/WIC/ACM International Joint*

Conference on Web Intelligence and Intelligent Agent Technology. 2009, Vol. 2, pp. 339-346.

Leite, Allan Rodrigo. 2009. *Um Esquema para Redução de Consumo de Combustível em Sistemas de Condução Férrea Baseado em Otimização Distribuída de Restrição.* Curitiba : s.n., 2009.

Leite, Allan Rodrigo, Giacomet, Bruno and Enembreck, Fabrício. 2009. Railroad Driving Model Based on Distributed Constraint Optimization. *Proceedings of the IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology.* 2009, pp. 474-482.

Lesser, V., Ortiz, C. L. and Tambe, M. 2003. Distributed Sensor Networks: a Multiagent Perspective. *Kluwer Academic Publishers.* 2003, Vol. 9, p. 386.

Maheswaran, Rajiv T., et al. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed multi-even scheduling. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent AAMAS.* 2004, pp. 310-317.

Mailler, Roger and Lesser, Victor. 2004. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. *AAMAS.* 2004.

Mailler, Roger. 2005. Comparing Two Approaches to Dynamic, Distributed Constraint Satisfaction. *AAMAS.* Julho 25-29, 2005.

Mertens, Koenraad, Holvoet, Tom and Berbers, Yolande. 2006. The DynCOAA Algorithm for Dynamic Constraint Optimization Problems. *AAMAS.* 2006.

Modi, P. J., et al. 2001. A dynamic distributed constraint satisfaction approach to resource allocation. *Principles and Practice of Constraint Programming.* 2001.

Modi, P. J., et al. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AI Journal.* 2005, Vol. 161, pp. 149-180.

Modi, Pragnesh Jay, et al. 2003. An Asynchronous Complete Method for General Distributed Constraint Optimization. *Proc of Autonomous Agents and Multi-agents Systems.* 2003.

Morris, P. 1993. The breakout method for escaping from local minima. *Proc. of the Eleventh National Conference on Artificial Intelligence.* 1993, pp. 40-45.

Petcu, Adrian and Faltings, Boi. 2006. A partial centralization extension of DPOP. *Proc. of the Secong International Workshop on Distributed Constraint Satisfaction Problems.* Agosto 2006.

—. 2005. A Scalable Method for Multiagent Constraint Optimization. *Proceedings of the International Joint Conference on Artificial Inteligence (IJCAI).* 2005, pp. 266-271.

—. 2006. MB-DPOP: A memory-bounded extension of DPOP. *Proc. of the Second International Workshop on Distributed Constraint Satisfaction Problems, ECAI'06.* Agosto 2006.

—. 2006. *ODPOP: An Algorithm For Open/Distributed Constraint Optimization.* 2006.

—. 2007. Optimal Solution Stability in Dynamic, Distributed Constraint Optimization. *Intelligent Agent Technology, IAT'07.* Novembro 2-5, 2007, pp. 321-327.

—. 2005. S-DPOP: Superstabilizing, faultcontaining multiagent combinatorial optimization. *Proceedings of the National Conference on Artificial Inteligence , AAAI-05.* Julho 2005, pp. 449-454.

Petcu, Adrian. 2006. *Recent Advances in Dynaminc, Distributed Constraint Optimization.* Swiss Federal Institute of Technology. Lausanne, Suíça : s.n., 2006. 2006/006.

Rossi, F., Van Beek, P. and Walsh, T. 2007. *Handbook of Constraint Programming.* Eastbourne : Elsevier, 2007. 1574-6525.

Russel, Stuart and Norvig, Peter. 2003. *Artificial Intelligence - A modern approach.* Upper Saddle River : Prentice Hall, 2003. ISBN 0-13-790395-2.

Schiex, Thomas and Verfaillie, Gérard. 1993. Two Approaches to the Solution Maintenance Problem in Dynamic Constraint Satisfaction Problems. 1993.

Silaghi, M. and Yokoo, M. 2006. Nogood based Asynchronous Distributed Optimization (ADOPT ng). *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2006.

Verfaillie, Gérard and Jussien, Narendra. 2005. Constraint Solving in Uncertain and Dynamics Enviroments: a Survey. Fevereiro 12, 2005.

Verfaillie, Gérard and Schiex, Thomas. 1994. Solution Reuse in Dynamic Constraint Satisfaction Problems. *AAAI.* 1994.

Weiss, G. 1999. *Multiagent Systems - A Modern Approach to Distributed Modern Approach to Artificial Intelligence.* Cambridge, Massachusetts : MIT Press, 1999.

Yeoh, William, Ariel, Felner and Sven, Koenig. 2008. BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. *International Conference on Autonomous Agents.* 2008, pp. 591-598.

Yokoo, Makoto and Hirayama, Katsutoshi. 1996. Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems. *International Conference on Multiagent Systems.* 1996.

Yokoo, Makoto, et al. 1998. The Distributed Constraint Satisfaction Problems: Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering.* September/October 1998.