Aplicações de Processos de Decisão de Markov na Composição Automática de Serviços Web

Rafael, M.A.A.[1], **Pérez-Alcázar, J.J.**[1], **Tuesta, E.F.**[1]

[1]Escola de Artes Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{monicarafael,jperez,tuesta}@usp.br

Abstract. *The automatic composition of services was initially inspired in classical planning, but this assumes a deterministic behavior not considering the uncertainty in this process. The planning based on the Markov decision processes (MDP) allows a more realistic representation because it models the domain as a stochastic system allowing unforeseen changes, such as failures or interruptions of service can be represented. A problem in the use of MDP is the definition of probabilities that are assumed in the problem of planning, therefore this work presents an alternative algorithm, which allows the update of the probabilities of transition.*

Resumo. *A composição automática de serviços foi inspirada inicialmente no planejamento clássico, porém este assume um comportamento determinístico não considerando a incerteza presente no processo. O planejamento baseado em processos de decisão de Markov (PDM) permite uma representação mais realista porque modela o domínio como um sistema estocástico permitindo que mudanças imprevistas, como falhas ou interrupções dos serviços possam ser representadas. Um dos problemas na utilização de PDMs consiste na atribuição das probabilidades que serão usadas no planejamento. Neste trabalho é apresentado um algoritmo alternativo ao usado na literatura, que permite a atualização das probabilidades de transição.*

1. Introdução

Um aspecto importante no desenvolvimento de aplicações baseadas em Serviços Web (SW) é a habilidade de selecionar e integrar de forma eficiente e efetiva serviços heterogêneos e de empresas diferentes. Se não existir um Serviço Web que possa satisfazer a funcionalidade requerida pelo usuário, deveria existir a possibilidade de combinar serviços existentes para realizar a solicitação do usuário. Fazer isso de forma manual pode ser um trabalho complexo e propenso a erros para qualquer pessoa, devido ao dinamismo e flexibilidade da Web [Digiampietri et al. 2007]. Isto tem levado à aparição de um considerável número de esforços de pesquisa sobre composição automática de serviços, tanto no nível da indústria quanto da academia [Rao and Su 2004]. Entre esses esforços podem ser destacadas as soluções baseadas em planejamento [Russel and Norvig 2003]. Este tipo de técnica usa os conceitos trabalhados na área de planejamento em IA para interpretar os processos e serviços como planos e ações. Desta maneira podem ser usados os avanços existentes na área de IA para a solução do problema de composição automática de serviços web. Estas técnicas são bastante interessantes em razão da maturidade alcançada pela área de planejamento [Long and Fox 2003].

Entretanto, para que os algoritmos de planejamento sejam aplicados, eles devem ser estendidos com características não consideradas dentro do planejamento clássico [Nau et al. 2004]. Uma dessas características refere-se à incerteza e não determinismo presente no ambiente de serviços, cujos processos ou planos de solução devem ser complexos. Neste contexto o planejamento baseado em processos de decisão de Markov (PDM) parece ser o mais adequado, pois permite a representação de aspectos não determinísticos tanto no comportamento dos serviços como da dinâmica do ambiente, sendo uma abordagem mais realista para a construção de modelos que toleram falhas, tratamento de exceções, mecanismos de recuperação, etc. Além disso, esta técnica permite a geração de planos ótimos.

Um dos problemas na utilização de PDM consiste na atribuição das probabilidades que serão usadas no planejamento, neste sentido o algoritmo *"Execute&Learn"* [Doshi et al. 2005] apresenta uma alternativa para a solução deste problema. Neste trabalho é apresentado um algoritmo alternativo denotado por *"Execute&Learn Modificado"* que mostra propriedades similares ao algoritmo apresentado por [Doshi et al. 2005], mas converge mais rapidamente às probabilidades de transição reais quando o número de estados aumenta.

2. Conceitos Básicos

2.1. Serviços Web

Um Serviço Web (SW), de acordo com o grupo de trabalho da W3C sobre arquitetura de serviços web [Austin et al. 2002], é uma aplicação de software identificada por um URI (*Uniform Resource Identifier*), cuja descrição e transporte utilizam padrões da Internet, isto é, tecnologias baseadas principalmente em XML. Uma enorme variedade de produtos são hoje disponibilizados via SW, como por exemplo, sistemas de banco de dados, serviços empresariais e outros. Um passo importante no desenvolvimento de aplicações baseadas em SW é a habilidade de selecionar e integrar em tempo de execução e de forma eficiente e efetiva serviços heterogêneos e de diferentes empresas. Isto tem conduzido à criação de linguagens de composição de SW tais como: BPEL4WS (*"Business Process Execution Language for Web Services*") [Andrews et al. 2003] ou WSCI ("*Web Service Choreography Interface*"). O problema destas propostas, como foi dito anteriormente, é que a composição pode ser uma tarefa muito demorada e pode gerar muitos erros, se for feita de forma manual.

2.2. Composição Automática de Serviços Web

Um dos propósitos da Web Semântica consiste em permitir que os dados e serviços nela contidos possam ser interpretados automaticamente pelos "agentes de software". Especificamente, no caso dos serviços web, os modelos ontológicos (WSMO [WSMO 2005], OWL-S [OWL 2004], etc.) têm por objetivo suportar as tarefas de descoberta, chamada, composição e monitoração automática de serviços. A composição automática de SW é uma tendência recente para tratar os desafios e problemas dos SW, tais como: interoperabilidade, sincronização, e outros [Medjahed et al. 2003], incluindo a seleção e operação automática de SW, o que terá um papel importante na viabilização da Web Semântica [McIlraith et al. 2001]. A idéia principal é que os usuários especifiquem o "que" eles desejam da composição (e.g., metas ou ações a serem desenvolvidas em um

alto nível) e o sistema encarrega-se do "como", isto é, os SWs a serem utilizados, a maneira como os serviços vão interagir entre si, entre outros aspectos. Uma das técnicas que permite a composição automática de serviços web é a técnica de planejamento, como foi mencionado acima. O fato dos ambientes associados a SW serem incertos e não-determinísticos conduz a que os planejadores baseados em PDM sejam mais adequados.

2.3. Processos de Decisão de Markov (PDM)

No planejamento baseado em PDM, o domínio de planejamento é modelado como um sistema estocástico, sistema de transição de estados que representa as incertezas dos efeitos das ações, ou seja das respostas dos SW, por funções de distribuição de probabilidade. Os planos são representados como políticas que especificam a melhor ação a ser executada em cada estado, a execução de uma política resulta em um comportamento condicional e iterativo. [Nau et al. 2004].

A idéia chave dos PDMs é a representação de um problema de planejamento como um problema de otimização em que se busca a maximização da função utilidade. A distribuição de probabilidade sobre o espaço de estados, chamada também de estados de crença, é modelada através da observação parcial do domínio [White 1993].

Definição: Um PDM pode ser representado pelo conjunto Σ=*(S, A, P, R, H)* onde:

- *S*: representa o conjunto de estados (finito ou infinito).
- *A*: conjunto de ações (finito ou infinito) que representam os SW.
- *R:* $S \times A \times S: \rightarrow \Re$: Representa a função recompensa (ganho) dada pela execução da ação *a* sobre o estado *s* tendo como resultado o estado *s´*.
- *H*: Também chamado de Horizonte, indica o período sob estudo no qual deve-se determinar a política ótima.
- *P:* $S \times A \times S: \rightarrow$ *[0,1]*: Função de transição, que representa a dinâmica do ambiente, os efeitos incertos das ações nos estados do problema de planejamento.

A suposição Markoviana para esta função de transição indica que o próximo estado do sistema depende apenas do estado atual e da ação a ser executada. Assim, $P_a(s' \mid s)$ onde $a \in A$, s e $s' \in S$, denota a probabilidade de que se uma ação *a* for executada em um estado *s* o novo estado do sistema será *s´*.

Para resolver PDMs gerando políticas ótimas, as suposições básicas são que os estados são totalmente observáveis e alcançáveis pelo sistema [Russel and Norvig 2003]. Três tipos de soluções para PDMs são conhecidos na literatura, o método de programação linear, o método de iteração de políticas, e o método de programação dinâmica estocástica. Este último método está relacionado ao princípio de optimalidade de Bellman [Doshi et al. 2005] onde a maximização da equação 1 garante que uma política ótima seja encontrada. Neste caso, *U(s´)* representa a recompensa esperada a longo prazo quando o sistema começa no estado *s*.

$$U(s) = max_{a \in A}\{R_i(s,a) + \sum \gamma_{s' \in S} P_i(s'|s,a) U(s')\} \quad (1)$$

2.4. Uma arquitetura de solução ao problema de composição automática de SW utilizando um planejador PDM

Na figura 1 é apresentada a interação entre os componentes necessários para monitorar e compor serviços web. Os módulos utilizados no processo de composição são: a In-

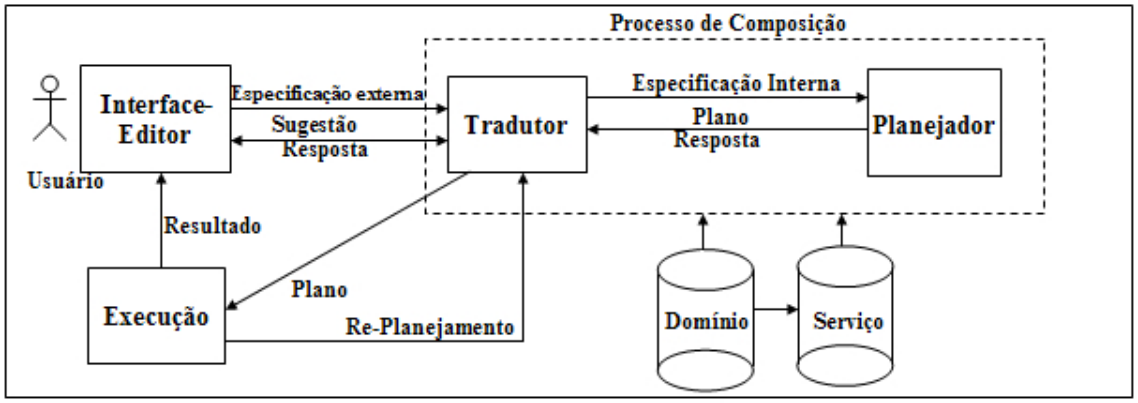

Figure 1. Arquitetura de composição de SW

terface - Editor, que recebe as requisições de serviços usando a linguagem WSML associada ao modelo ontológico WSMO [WSML 2008], essas requisições são enviadas ao tradutor; o Tradutor recebe a requisição especificada em WSML e a mapeia para a linguagem PPDDL [Younes and Littman 2004] (Linguagem de Definição de Domínio de Planejamento Probabilístico), que é reconhecida pelo Planejador; o Planejador recebe a requisição em PPDDL e encarrega-se de gerar a política ótima (serviço composto, associado), nesse processo ele utiliza as ontologias de domínio e de serviços; o Tradutor recebe a política resultante em PPDDL; o Executor recebe a plano e se encarrega de fazer o mapeamento para o plano equivalente em WSDL (Grounding [Kopecký et al. 2007]), no nível de execução, no qual será executado; por último os resultados serão apresentados ao usuário. No caso de falhas na execução do plano, um processo de re-planejamento pode ser requisitado através do tradutor.

3. Composição de Workflows Dinâmicos utilizando Processos de Decisão de Markov

Neste artigo os conceitos de *Workflows* e Serviços Web compostos serão usados como sinônimos. A proposta sobre composição de *workflows* dinâmicos utilizando PDM elaborada por Doshi, Goodwin, Akkiraju e Verma [Doshi et al. 2005] considera ambientes dinâmicos. O modelo intercala geração de *workflows* baseados em PDMs e modelos de aprendizado bayesianos.

Para escolher um *workflow* ótimo entre vários *workflows* candidatos, o planejador deve possuir alguma estimativa da certeza com a qual seu requerimento será satisfeito. Estas estimativas são medidas do não determinismo do processo, considerado como as possíveis respostas dos SW e expressas na função de transição do PDM. Claramente uma política ótima depende destas estimativas e conseqüentemente, os *workflows* mudam conforme mudam as probabilidades.

Inicialmente o planejador é ignorante quanto aos valores das probabilidades e ele tenta entendê-las através das interações com o ambiente. De um ponto de vista Bayesiano, o planejador atribui inicialmente uma distribuição de probabilidade uniforme para cada resposta obtida na invocação dos SW. Usando algoritmos de aprendizado Bayesiano que atualizam as estimativas e as distribuições de probabilidade do planejador de acordo com as respostas às invocações dos SW obtidas durante a execução do *workflow*. Desta

maneira, intercala-se execução de *workflows* e modelos de aprendizado. Os algoritmos de atualização bayesianos permitem a convergência quase certa para as probabilidades verdadeiras.

O algoritmo *"Execute&Learn"* apresentado em [Doshi et al. 2005] consiste em executar uma política encontrada através do *"Value_Iteration"*, técnica de solução padrão para PDM [Nau et al. 2004, Russel and Norvig 2003] e de acordo com as respostas dos serviços invocados que são obtidas pela execução desta política, alterar a distribuição de probabilidade do sistema. Para executar a política gerando um *workflow* é utilizado o algoritmo *"Compose&ExecuteWorkflow"* [Doshi et al. 2005], que a partir do estado inicial e da política, intercala execução e composição até encontrar um estado meta. As alterações dos valores das funções de transição são feitas da seguinte forma:

Para cada variável aleatória presente no domínio é mantido um contador,*"exper"*. Esse contador inicialmente possui valor 1 e é incrementado cada vez que a invocação de um serviço web, que atua sobre determinada variável aleatória altera o valor da variável de *x* para *x'*. A equação 2 é utilizada para alterar a probabilidade da função de transição que possua o mesmo valor da variável aleatória que foi encontrado na resposta do serviço web.

$$T'(X = x'|a, X = x) := \frac{T(X = x'|a, X = x) \times exper + 1}{exper'} \tag{2}$$

Na equação acima "*exper´* " é o contador de experiências incrementado.

Considerando a distribuição de probabilidades sobre *X*, isto é, para que a distribuição sobre *X* some 1, as probabilidades dos outros valores de *X* são atualizados de acordo com a equação 3:

$$T'(X = y|a, X = x) := \frac{T(X = y|a, X = x) \times exper}{exper'} \tag{3}$$

O principal objetivo desse algoritmo é que as probabilidades encontradas após um número de iterações estejam o mais próximo possível das probabilidades reais. Na primeira iteração as funções de transição são inicializadas com a mesma probabilidade para cada valor assumido pela variável aleatória, o processo de solucionar o PDM, executar os serviços através da política e alterar as probabilidades de acordo com as respostas se repete até que a variação seja menor que um dado piso. A proposta dos autores acima é robusta quanto ao monitoramento de execução e recuperação perante um comportamento inesperado de SW, devido às falhas de serviço ou à dinâmica natural do mundo real, e se adapta a mudanças do ambiente. Ela tem como cenário de motivação um processo de administração e integração de negócios (*Business Process Integration and Management*) ainda que o foco não esteja nos detalhes de implementação, mas nos algoritmos propostos.

4. "Execute&Learn Modificado"

Uma análise detalhada do algoritmo de aprendizado bayesiano *"Execute&Learn"* definido em [Doshi et al. 2005] foi realizada e uma alteração foi proposta. Para simplificar a análise, foi introduzida uma modificação da notação apresentada em (2) e (3), de tal maneira que a função de transição considerada na n-ésima chamada é agora:

$$T^{(n)}(x') := \frac{T^{(n-1)}(x').n + 1_{x'}}{n+1} \tag{4}$$

Onde:

- $1_{x'}$: é o indicador do estado *x'*na n-ésima chamada.
- $T^{(n)}$*(x')*: Indica a probabilidade do estado *x'* ser visitado na n-ésima chamada.
- *n*: indica o número de chamadas realizadas e *k* o número de estados.

A equação acima trabalhada recursivamente resulta na seguinte equação:

$$T^{(n)}(x') = \frac{T^{(0)}(x')}{n+1} + \frac{(\sum_{i=1}^{n} 1_{x'_i})}{n+1} \tag{5}$$

Considerando ainda a suposição do algoritmo (distribuição uniforme para todos os estados, quando não se tem conhecimento sobre o sistema), tem-se que $T^{(0)}$ *(x')=1/k* , onde *k* é o número de estados possíveis que podem ser alcançados pela execução da ação *a* no estado *x*. Desta maneira obtém-se a seguinte equação:

$$T^{(n)}(x') = \frac{1 + k\sum_{i=1}^{n} 1_{x'_i}}{(n+1)k} \tag{6}$$

Nota-se em (6) que a verossimilhança dos dados, definida como o número de visitas ao estado *x'* ($\sum_{i=1}^{n} 1_{x'_i}$), é multiplicada por um fator que é igual ao número de estados, indicando uma mudança importante para cada nova chamada do estado *x'* em cada iteração do algoritmo. A alteração proposta no algoritmo, está baseada no trabalho de [Spiegelhalter et al. 1993] e considera mudanças parcimoniosas das probabilidades de transição diante de novas visitas aos estados.

Na proposta modificada do algoritmo *"Execute&Learn"* é mantido um indicador para cada estado, assim *1x'* é a função indicadora para o estado *x'* e denota a resposta da execução de uma ação *a* em um estado *x*, assumindo 1 caso o estado visitado seja *x'* e 0 caso contrário. Desta maneira quando um estado *x'* é obtido como resposta da invocação de um serviço em um determinado estado *x*, o valor de sua função de transição é alterada através da equação 7, onde:

$$T(x' \mid x, a) = \frac{1 + 1_{x'}}{k + \sum_{i=1}^{n} 1_{x'_i}} \tag{7}$$

- *T(x'|x,a)*: é o valor da função de transição, a probabilidade de se obter *x'* executando a ação *a* no estado *x*.
- $\sum_{i=1}^{n} 1_{x'_i}$: Soma dos indicadores de *x'*, denota a quantidade de vezes que o estado *x'* foi obtido com a execução da ação *a* em um estado *x*.
- *k*: Número de estados que podem ser alcançados pela execução da ação *a* no estado *x*.

Assim como no *"Execute&Learn"*, na primeira iteração as funções de transição são inicializadas com a mesma probabilidade, o processo de solucionar o PDM encontrando a política ótima, executar os serviços através desta política e alterar as probabilidades de acordo com as respostas se repete até que a variação seja menor que um dado piso.

5. Resultados

Para o exemplo estudado no presente trabalho foram considerados cinco cenários diferentes, cada cenário com um número diferente de estados finais possíveis em uma transição. Para cada um dos cenários foram utilizados 3 valores para δ sendo que a cada iteração foi medida uma distância definida como:

$$D = max \mid q_n - q_{n-1} \mid \quad (8)$$

- q_n = ($T_1^{(n)}$, $T_2^{(n)}$, $\cdots$, $T_k^{(n)}$): Representa o vetor de todas as transições do sistema na iteração n.
- q_{n-1} = ($T_1^{(n-1)}$, $T_2^{(n-1)}$, $\cdots$, $T_k^{(n-1)}$): Representa o vetor de todas as transições do sistema na iteração n-1.

Para os algoritmos *"Execute&Learn"* e *"Execute&Learn Modificado"* foram realizadas 1000 simulações diferentes para cada δ e para cada cenário obtendo-se o número de iterações para cada simulação.

A tabela 1 apresenta os resultados obtidos para as simulações realizadas com os dois algoritmos, apresentando a média e o desvio padrão do número de iterações para cada δ e cenário.

Table 1. Média de Iterações com variação do valor de delta e do número possível de estados em uma transição.

Nº Estados na Transição		δ=0,001		δ=0,005		δ=0,01	
		Execute & Learn	Execute& Learn Modificado	Execute& Learn	Execute& Learn Modificado	Execute& Learn	Execute& Learn Modificado
2	Média	198,78	201,28	37,41	41,89	19,34	21,84
	Desvio	30,27	27,73	14,51	10,26	8,32	5,99
3	Média	300,85	303,4	59,97	62,03	30,1	31,79
	Desvio	25,98	25,91	13,12	9,92	9,23	5,94
5	Média	651,87	645,88	132,23	125	67,13	59,82
	Desvio	18,65	18,54	8,5	8,17	6,07	5,38
7	Média	790,41	778,8	157,53	145,52	78,76	67,09
	Desvio	10,05	9,87	4,93	4,51	3,97	3,31
9	Média	794,85	780,23	159,69	144,69	80,61	65,57
	Desvio	11,91	11,47	5,57	5,43	4,22	3,47

Observa-se que para cenários com poucos estados (2, 3) o número médio de iterações do algoritmo proposto em [Doshi et al. 2005] é menor que o número médio de iterações do algoritmo *"Execute&Learn Modificado"*. No entanto observa-se também que o desvio padrão para o caso do algoritmo *"Execute&Learn"* é sempre maior que o desvio padrão do *"Execute&Learn Modificado"*.

No caso de cenários com maior número de estados (que pode ser considerado um caso mais realista) observa-se que o número médio de iterações do algoritmo *"Execute&Learn"* é sempre maior que o número médio de iterações do algoritmo *"Execute&Learn Modificado"* e também o desvio padrão do algoritmo *"Execute&Learn"* é maior que o desvio padrão do algoritmo *"Execute&Learn Modificado"*.

A figura 2 apresenta 3 realizações das simulações considerando a distância definida na equação 8 entre o vetor de probabilidades atual do sistema e o vetor de probabilidades anterior (atualizações de acordo com as propostas *"Execute&Learn"* e *"Execute&Learn Modificado"*).

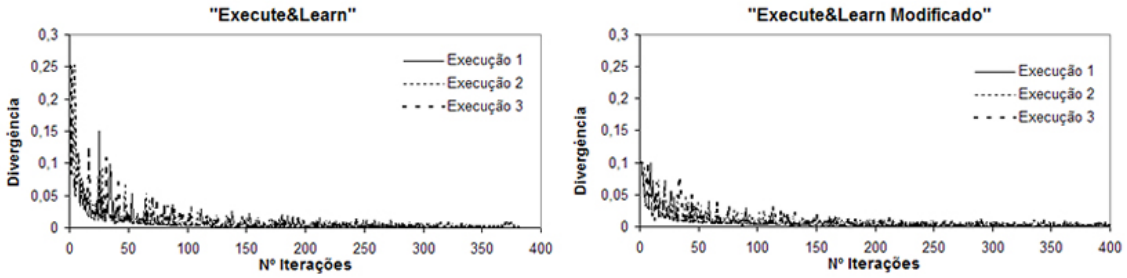

Figure 2. Divergências para *"Execute&Learn"* e *"Execute&Learn Modificado"*

Observa-se na figura 2 que o algoritmo *"Execute&Learn"* apresenta nas primeiras iterações muitos saltos, indicando alta dependência do peso atribuído à verossimilhança (como explicado anteriormente), e para o caso do algoritmo *"Execute&Learn Modificado"* estes saltos são de menor valor.

O trabalho desenvolvido por [Doshi et al. 2005] apresenta como medida de convergência a fórmula de divergência de Kullbach-Leibler definida por:

$$D(p\|q)=\sum p(x)log_2\frac{p(x)}{q(x)} \tag{9}$$

- p(x): vetor de probabilidades reais do sistema.
- q(x): vetor de probabilidades atuais do sistema.

A figura 3 apresenta 3 realizações das simulações usando a divergência definida na equação 9 entre os vetores de probabilidade real do sistema e de probabilidades atuais. A figura apresenta a divergência entre os elementos do vetor tanto para o caso da atualização segundo o algoritmo *"Execute&Learn"* como para o algoritmo *"Execute&Learn Modificado"*.

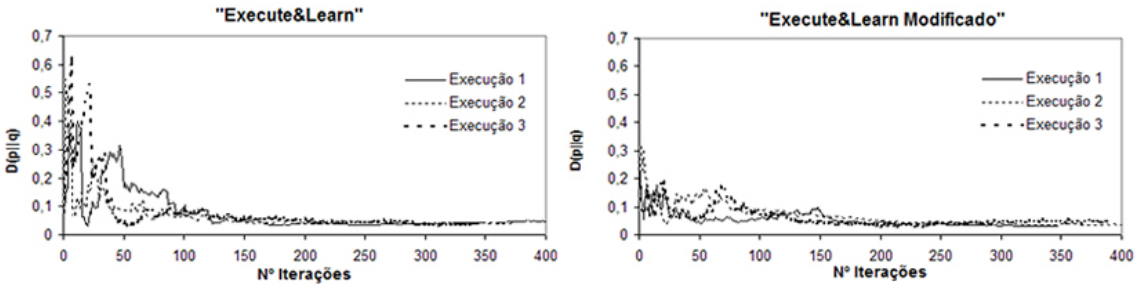

Figure 3. *Divergência de Kullbach-Leibler: "Execute&Learn"* e *"Execute&Learn Modificado"*

De maneira alternativa apresenta-se também neste trabalho a distância em variação total definida em [Serfling 1975, Serfling 1978] como:

$$D(p,q)=\frac{1}{2}\sum_{k=0}^{\infty} \mid p(x=k)-p(y=k) \mid \quad (10)$$

- p(x): vetor de probabilidades reais do sistema.
- q(x): vetor de probabilidades atuais do sistema.

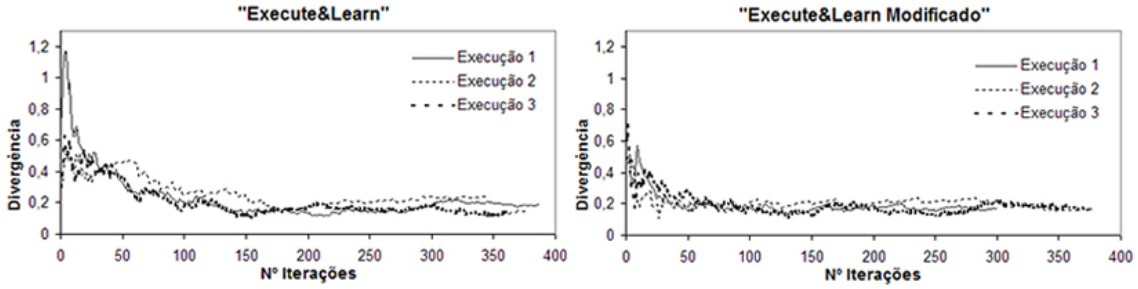

Figure 4. ***Distância em Variação Total:"Execute&Learn"* e *"Execute&Learn Modificado"***

Observa-se também nas figuras 3 e 4 a alta dependência do peso atribuído a verossimilhança para o algoritmo *"Execute&Learn"* e observa-se para o *"Execute&Learn Modificado"* um comportamento bem parcimonioso.

6. Conclusões e Trabalhos Futuros

Este trabalho foi baseado na proposta desenvolvida por [Doshi et al. 2005] em que é apresentado como problema de planejamento um domínio de negócios e um algoritmo para a atualização das probabilidades de transição entre os estados chamado de *"Execute&Learn"*. Encontra-se que o esquema de atualizações proposto é dependente de um peso que multiplica a verossimilhança e que provoca saltos nas primeiras iterações.

Neste trabalho é proposta uma modificação do algoritmo *"Execute&Learn"* chamado *"Execute&Learn Modificado"* que tem como distância a priori uma Beta multivariada e aproxima as probabilidades de maneira suave.

Segundo [Zhao and Doshi 2006] os processos de decisão de Markov embora apresentem uma maneira de compor serviços não respondem bem a ambientes com muitos processos incertos, existindo o problema de explosão de estados. Portanto a continuação deste trabalho consiste em buscar soluções para esse problema e entre elas serão estudadas heurísticas [Nau et al. 2004] e processos de decisão semi-Markov [Zhao and Doshi 2006].

References

Andrews, T., Curbera, F., Dholakia, H., and et al. (2003). Business process execution language for web services version 1.1. http://www-106.ibm.com/developerworks/library/ws-bpel/ (acessado em 2009-01-22).

Austin, D., Grainger, W., Barbir, A., Ferris, C., and Garg, S. (2002). "Web services architecture requirements". Technical report, W3C.

Digiampietri, L., Pérez-Alcázar, J., and Medeiros, C. (2007). "Ai planning in web services composition: a review of current approaches and a new solution". *VI ENIA - Anais do XXVII Congresso da Sociedade Brasileira de Computação (CSBC2007, Rio de Janeiro, Brazil)*, pages 983–992.

Doshi, P., Goodwin, R., Akkiraju, R., and Verma, K. (2005). "Dynamic workflow composition: Using markov decision processes". *International Journal of Web Service Research.*, 2(1):1–17.

Kopecký, J., Moran, M., Vitvar, T., Roman, D., and Mocan, A. (2007). WSMO Grounding. http://www.wsmo.org/TR/d24/d24.2/v0.1/ (acessado em 2009-01-22).

Long, D. and Fox, M. (2003). "The 3rd international planning competition: Results and analysis". *Journal of Artificial Intelligence Research (JAIR)*, 20:1–59.

McIlraith, S. A., Son, T., and Zeng, H. (2001). "Semantic web services". *IEEE Intelligent Systems*, 16:46–53.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. (2003). "Composing web services on the semantic web". *The VLDB Journal*, 12(4):333–351.

Nau, D., Ghallab, M., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

OWL (2004). OWL Web Ontology Language. http://www.w3.org/TR/owl-features (acessado em 2009-01-22).

Rao, J. and Su, X. (2004). "A survey of automated web service composition methods". In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, SWSWPC2004, LNCS*, San Diego, USA.

Russel, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

Serfling, R. (1975). "A general poisson approximation theorem". *The Annals of Probability*, 3(4):726–731.

Serfling, R. (1978). "Some elementary results on poisson approximation in a sequence of bernoulli trial". *Society for Industrial and Applied Mathematics (SIAM).*, 20(3):567–579.

Spiegelhalter, D., Dawid, A., Lauritzen, S., and Cowel, R. (1993). "Bayesian analysis in expert systems". *Statistical Science*, 8(3):219–247.

White, D. (1993). *Markov Decision Processes*. John Wiley & Sons., 1st edition edition.

WSML (2008). WSML - Web Service Modelling Language. http://www.wsmo.org/wsml/ (acessado em 2009-01-22).

WSMO (2005). WSMO - Web Service Modelling Ontology. http://www.w3.org/Submission/WSMO/ (acessado em 2009-01-22).

Younes, H. and Littman, M. (2004). "PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects". *Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh.*

Zhao, H. and Doshi, P. (2006). "A hierarchical framework for composing nested web processes". *Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA*, pages 116–128.